Tools for Quantum Computing Based on Decision Diagrams

ROBERT WILLE, Johannes Kepler University Linz and Software Competence Center Hagenberg GmbH (SCCH)
STEFAN HILLMICH and LUKAS BURGHOLZER, Johannes Kepler University Linz

With quantum computers promising advantages even in the near-term NISQ era, there is a lively community that develops software and toolkits for the design of corresponding quantum circuits. Although the underlying problems are different, expertise from the design automation community, which developed sophisticated design solutions for the conventional realm in the past decades, can help here. In this respect, decision diagrams provide a promising foundation for tackling many design tasks such as simulation, synthesis, and verification of quantum circuits. However, users of the corresponding tools often do not have a proper background or an intuition about how these methods based on decision diagrams work and what their strengths and limits are. In this work, we first review the concepts of how decision diagrams can be employed, e.g., for the simulation and verification of quantum circuits. Afterwards, in an effort to make decision diagrams for quantum computing more accessible, we then present a visualization tool for quantum decision diagrams, which allows users to explore the behavior of decision diagrams in the design tasks mentioned above. Finally, we present decision diagram-based tools for simulation and verification of quantum circuits using the methods discussed above as part of the open-source Munich Quantum Toolkit (MQT)—a set of tools for quantum computing developed at the Technical University of Munich and the Johannes Kepler University Linz and released under the MIT license. More information about the corresponding tools is available at https://github.com/cda-tum/. By this, we provide an introduction of the concepts and tools for potential users who would like to work with them as well as potential developers aiming to extend them.

CCS Concepts: • **Hardware → Electronic design automation**; **Quantum computation**; **Emerging tools and methodologies;**

Additional Key Words and Phrases: Decision diagrams, quantum circuit simulation, quantum circuit verification

ACM Reference format:
Robert Wille, Stefan Hillmich, and Lukas Burgholzer. 2022. Tools for Quantum Computing Based on Decision Diagrams. *ACM Trans. Quantum Comput.* 3, 3, Article 13 (June 2022), 17 pages.
https://doi.org/10.1145/3491246

1 INTRODUCTION

Quantum computers are steadily improving in terms of their computational power to the extent that first computations are being performed that are no longer feasible on conventional machines

Authors' addresses: R. Wille, Technical University of Munich Arcisstraße 21 80333 Munich GERMANY, and Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, 4232, Austria; email: robert.wille@tum.de; S. Hillmich, and L. Burgholzer, Johannes Kepler University Linz Altenberger Straße 69 4040 Linz AUSTRIA; emails: {stefan.hillmich, lukas.burgholzer}@jku.at.
Updated author affiliation: R. Wille, Technical University of Munich.

2643-6817/2022/06-ART13 $15.00
https://doi.org/10.1145/3491246

[3, 48]. Achieving these milestones is only possible through interdisciplinary efforts by physicists, mathematicians, computer scientists, and many others. Just as in the design of conventional circuits and systems, the development of design automation tools for quantum computing will be one of the key factors for the success of the technology.

In the 1980s, decision diagrams were proposed as a data structure for efficient representation and manipulation of Boolean functions [6]. Following this development, a multitude of decision diagrams such as BDDs, FBDDs, KFDDs, MTBDDs, and ZDDs emerged (see, e.g., References [4, 7, 16, 18, 31, 44]), which established them as a core asset in the design of today's circuits and systems. Due to their success in the past, the use of decision diagrams has also been proposed in the domain of quantum computing [26, 30, 35, 41–43, 50]. In particular for design tasks such as *simulation* [24, 41, 42, 53], *synthesis* [1, 34, 39, 52], and *verification* [13, 25, 38, 43] of quantum circuits, they found great interest recently.

Despite the vast knowledge on design automation that may be exploited in quantum computing, there is still a huge gap to bridge between the quantum computing community and the design automation community. In fact, many promising techniques have hardly reached the core of the quantum computing community until now and are not yet established—despite showing promising results. Due to the interdisciplinarity of the field, quite often, users of the corresponding tools are hardly familiar with the underlying concepts and, understandably, have a hard time getting a proper intuition about how these tools work.

With this work, we aim at narrowing the aforementioned gap. We show that decision diagrams are a promising tool for design automation in the quantum realm. After defining and reviewing decision diagrams for quantum computing, we show how an easy-to-use visualization can help to develop an intuition on the behavior of decision diagrams. The software we present focuses on simulation as well as verification of quantum circuits and offers different styles to accommodate various use cases. Subsequently, we introduce our decision diagram-based tools for simulation and verification as part of the open-source Munich Quantum Toolkit (MQT)—a set of corresponding tools developed at the Technical University of Munich and the Johannes Kepler University Linz. While each tool (including ours) certainly has strengths and weaknesses, we offer complementary approaches for many of the problems that need to be tackled when designing quantum circuits. By making our tools publicly available as open-source, we also provide other researchers with the option to incorporate the underlying methods into their existing tools. In fact, this already motivated "players" like IBM and Atos to include, e.g., the simulation approach based on decision diagrams into their tools.

The remainder of this work is structured as follows. In Section 2, we review decision diagrams and their applications in quantum computing. Section 3 illustrates how the application of decision diagrams in simulation and verification of quantum circuits can be visualized to help the interested user to develop an intuition. In Section 4, we detail how to approach simulation and verification with the MQT from a user's perspective. Section 5 gives insight on how the MQT is organized and where interested developers can start to extend the available tools with their own methods. Finally, we conclude this work in Section 6.

2 DECISION DIAGRAMS AND THEIR APPLICATIONS

State vectors and operation matrices of a quantum system are exponential in size with respect to the number of qubits—quickly rendering the representation of a system state or the construction of a system matrix an extremely difficult task. *Decision diagrams* have been proposed as an efficient way for representing and manipulating quantum functionality [1, 13, 24–26, 30, 34, 35, 38, 39, 41–43, 50, 52, 53]. While they are still exponential in the worst-case, decision diagrams have been shown to lead to very compact representations in many cases. In the following, we review how

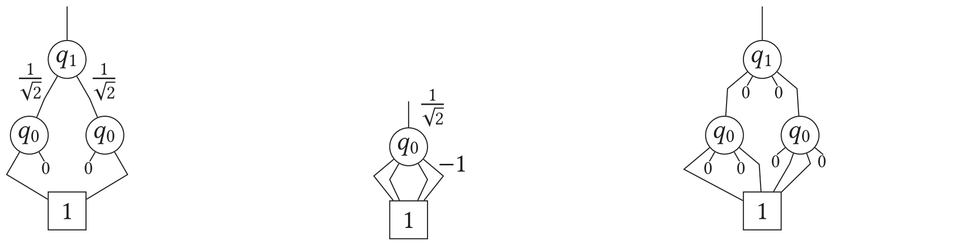
$$[\alpha]^\top = [\alpha_{00\ldots0}, \ldots, \alpha_{11\ldots1}]^\top$$

$$[\alpha_{00\ldots0}, \ldots, \alpha_{01\ldots1}]^\top \qquad [\alpha_{10\ldots0}, \ldots, \alpha_{11\ldots1}]^\top$$

$$[\alpha_{000\ldots0}, \ldots, \alpha_{001\ldots1}]^\top \quad [\alpha_{010\ldots0}, \ldots, \alpha_{011\ldots1}]^\top \quad [\alpha_{100\ldots0}, \ldots, \alpha_{101\ldots1}]^\top \quad [\alpha_{110\ldots0}, \ldots, \alpha_{111\ldots1}]^\top$$

$$\vdots$$

Fig. 1. Decomposition of a state vector.



(a) $|\varphi\rangle = 1/\sqrt{2}\,[1, 0, 0, 1]^\top$ (b) Hadamard gate (c) Controlled-NOT gate

Fig. 2. Decision diagram representations for quantum states and operations.

decision diagrams for quantum computing work and how they can be applied to the problems of *quantum circuit simulation* and *quantum circuit verification.*

2.1 Decision Diagrams

The state of an n-qubit system is represented by a state vector of size 2^n—an exponential representation. However, the inherent tensor product structure of many quantum states and redundancies in their description provide ground for a more compact representation. To this end, a given state vector α with its complex amplitudes α_i for $i \in \{0, 1\}^n$ is decomposed into sub-vectors as illustrated in Figure 1 until only individual complex numbers remain.

This gives rise to a decision diagram structure with n levels of nodes (labeled q_{n-1} to q_0) and the individual amplitudes as its leaves. Each node q_i has exactly two successors—indicating whether the corresponding path leads to an amplitude where qubit q_i is in the state $|0\rangle$ or $|1\rangle$, respectively. During these decompositions, equivalent sub-vectors can be represented by the same node—allowing for sharing and, hence, a reduction of the complexity of the representation. Further compaction is achieved by introducing edge weights on all levels and employing normalization schemes,[1] to unify sub-vectors only differing by a common factor and further exploit possible redundancies. The amplitude of a given basis state can then be reconstructed from the multiplication of the edge weights along the path from the root node to the terminal node. An example illustrates the idea.

Example 1. Consider the state $|\varphi\rangle \equiv 1/\sqrt{2}\,[1, 0, 0, 1]^\top$. Recursively splitting this vector into sub-vectors results in a decision diagram as shown in Figure 2(a). It consists of three nodes (the terminal node is usually not counted towards a decision diagram's size). The two paths leading from the root edge to the terminal node encode the states $|00\rangle$ and $|11\rangle$, respectively—each with an amplitude of $1/\sqrt{2} \cdot 1 = 1/\sqrt{2}$. Sub-vectors composed solely of 0 entries are typically denoted by 0-stubs to reduce visual clutter, while edge weights equal to 1 are frequently omitted.

[1]Normalization can be performed by, e.g., dividing the weight of the outgoing edges of a node by the norm of the vector containing both edge weights and multiplying this factor to the incoming edges—allowing for efficient sampling from the resulting decision diagram [24].

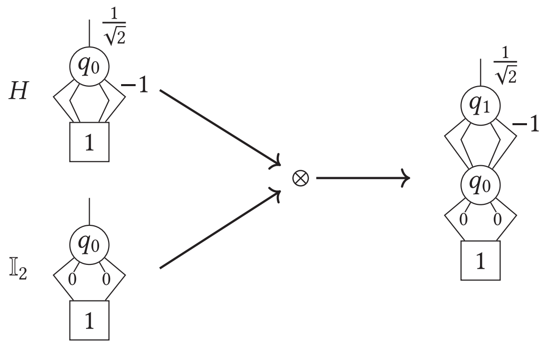


Fig. 3. Creation of $H \otimes \mathbb{I}_2$ using decision diagrams.

A similar construction is employed for representing quantum operations, i.e., the corresponding unitary matrices. Instead of two successors in the decomposition of vectors, each node in a decision diagram representing the (unitary) matrix U of an operation has four successors—corresponding to four equally sized sub-matrices U_{ij} as in

$$U = \begin{bmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{bmatrix}.$$

At each level l, this splitting corresponds to the action of U depending on the value of the qubit q_l, i.e., U_{ij} describes how the rest of the system is transformed given that q_l is mapped from $|j\rangle$ to $|i\rangle$ for $i, j \in \{0, 1\}$.

Example 2. Consider the single-qubit Hadamard operation and the two-qubit controlled-NOT operation, i.e.,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{and} \quad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \text{respectively.}$$

Their corresponding representations as decision diagrams are shown in Figures 2(b) and 2(c), respectively. To this end, the first (second) edge points to the node corresponding to the upper-left (upper-right) sub-matrix U_{00} (U_{01}), while the third (fourth) edge points to the lower-left (lower-right) sub-matrix U_{10} (U_{11}).

Matrices of individual gates have to be extended to the full system size using tensor products before being applied to the current state of a system. This extension is a natural operation on decision diagrams. Given two decision diagrams representing matrices U and V, their tensor product $U \otimes V$ is obtained by just replacing the terminal node in the decision diagram of U with the root node of V's decision diagram (and potentially relabeling the nodes).

Example 3. The 2×2 matrix of the Hadamard gate was extended to a 4×4 representation by computing the tensor product with the 2×2 identity matrix $\mathbb{I}_2$. Figure 3 now illustrates this process using decision diagrams.

Decision diagrams allow one to not only efficiently represent quantum states and matrices but also manipulate them. In the following, we illustrate their application for two particularly important design tasks in quantum computing: *quantum circuit simulation* and *quantum circuit verification.*

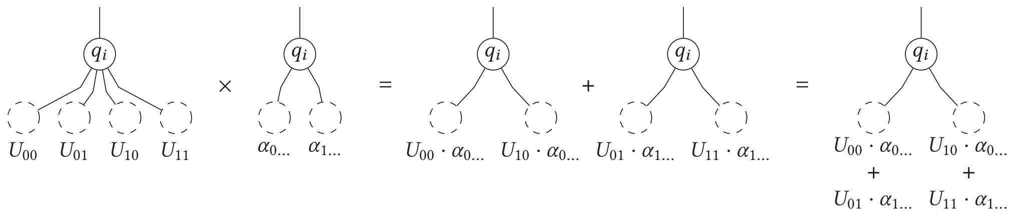

q_i
U_{00}
U_{01}
U_{10}
U_{11}
$\times$
$\alpha_{0\ldots}$
$\alpha_{1\ldots}$
$=$
$U_{00}\cdot\alpha_{0\ldots}$
$U_{10}\cdot\alpha_{0\ldots}$
$+$
$U_{01}\cdot\alpha_{1\ldots}$
$U_{11}\cdot\alpha_{1\ldots}$
$=$
$U_{00}\cdot\alpha_{0\ldots} + U_{01}\cdot\alpha_{1\ldots}$
$U_{10}\cdot\alpha_{0\ldots} + U_{11}\cdot\alpha_{1\ldots}$

Fig. 4. Recursive structure of multiplication and addition using decision diagrams.

2.2 Simulation of Quantum Circuits

The simulation of a quantum circuit G consisting of quantum operations $g_0, \ldots, g_{m-1}$ entails the consecutive calculation of the matrix-vector product between the state vector $|\varphi\rangle$ and the current operation matrix U_i (corresponding to gate g_i) until all operations have been applied. The following example sketches how matrix-vector multiplication is realized on decision diagrams.

Example 4. The multiplication of a (unitary) matrix U and a (state) vector $|\varphi\rangle$ can be broken down into sub-computations as follows:

$$\begin{bmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{bmatrix} \cdot \begin{bmatrix} \alpha_{0\ldots} \\ \alpha_{1\ldots} \end{bmatrix} = \begin{bmatrix} (U_{00}\alpha_{0\ldots} + U_{01}\alpha_{1\ldots}) \\ (U_{10}\alpha_{0\ldots} + U_{11}\alpha_{1\ldots}) \end{bmatrix}.$$

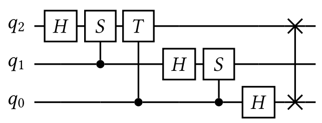
Now, the U_{ij}-submatrices precisely correspond to the four successors of the matrix's root node, while the $\alpha_{i\ldots}$-subvectors correspond to the two successors of the state vector's root node. This is illustrated in Figure 4. Thus, by recursively decomposing these sub-computations further until only operations on complex numbers remain, an efficient scheme for matrix multiplication using decision diagrams is devised.[2]

Measuring the resulting state, i.e., sampling from the corresponding decision diagram, can be efficiently conducted by a randomized single-path traversal of the decision diagram [24]. At each node, the squared magnitude of the left (right) successor gives the probability of the qubit associated to the node being $|0\rangle$ ($|1\rangle$), while the probability of an individual basis state is the product of all probabilities along the path. In contrast to quantum computations on real quantum computers, measurements of classically simulated quantum states can be conducted non-destructively, i.e., they can be repeated on the same state without having to repeat the whole calculation.

2.3 Verification of Quantum Circuits

To realize a conceptual quantum algorithm on an actual device, the algorithm's description is transformed through various levels of abstraction—including steps usually called compilation, synthesis, transpilation, mapping, and/or similar. To this end, several methods have been proposed [28, 29, 32, 40, 45, 51, 54]. During this process, it is of utmost importance to guarantee that the resulting circuit is still functionally equivalent to the original algorithm. The functionality of a quantum circuit is described by the unitary system matrix U arising from the matrix-matrix multiplications of the individual gate descriptions. Thus, the equivalence of two circuits can be reduced to the comparison of their system matrices. In the following, we illustrate this verification scenario using the *Quantum Fourier Transform (QFT)* (a popular building block in many quantum algorithms) [33] as an example.

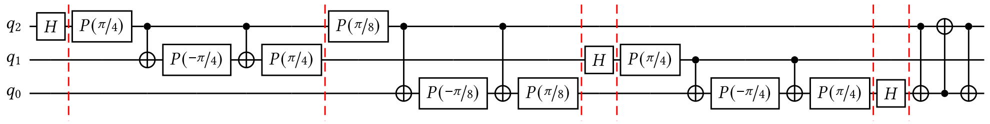
[2]Decision diagram packages employ several implementation techniques to further exploit possible redundancies and to reduce the number of computations necessary, see, e.g., [50].



(a) Three-qubit QFT

$$\frac{1}{\sqrt{8}}\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7\\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6\\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5\\ 1 & \omega^4 & 1 & \omega^3 & 1 & \omega^4 & 1 & \omega^4\\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3\\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2\\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega\end{bmatrix}$$

(b) Functionality



(c) Compiled circuit

Fig. 5. The Quantum Fourier Transformation and its functionality.

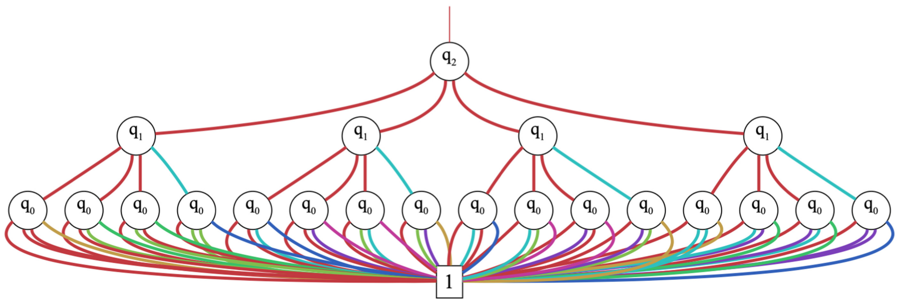


Fig. 6. Decision diagram for the functionality of the three-qubit QFT.

Example 5. The circuit of the three-qubit QFT is shown in Figure 5(a). It consists of Hadamard gates, controlled phase gates, i.e., rotations with an angle that is a certain fraction of π (e.g., $S = p(\pi/2)$, $T = p(\pi/4)$) controlled on the value of another qubit and a SWAP gate (which swaps the value of the two qubits indicated by $\times$). The latter two types of gates are not native to any current quantum computer and, thus, need to be compiled into sequences of gates that are supported. Figure 5(c) shows one possible compiled version of the abstract QFT circuit. Both circuits realize the functionality described by the 8×8 matrix shown in Figure 5(b), where $\omega = e^{i\pi/4} = \sqrt{i} = {}^{1+i}/\sqrt{2}$.

While conceptually simple, the exponential size of the involved matrices quickly renders straightforward techniques based on matrices infeasible. Decision diagrams are a prominent candidate for checking the equivalence of two circuits, since they (1) can represent quantum functionality compactly in many cases and (2) still offer a canonical representation (with respect to a given variable order and normalization scheme). Thus, the equivalence of two decision diagrams can be concluded by comparing their root pointers and the corresponding edge weights in most implementations.

Example 6. Constructing the decision diagrams for the two circuits shown in Figure 6(a) and Figure 6(c), respectively, results in a decision diagram as shown in Figure 6 in both cases.[3] Hence, both circuits are considered equivalent.

[3]For sake of readability, edge weights are not explicitly annotated to this decision diagram anymore. Instead, a color code is used (also explained in detail later in Section 3 and Figure 7(b)).

As seen in the previous example, decision diagrams can still grow exponentially large in the worst case—again presenting a severe obstacle for verifying the correctness of circuits. However, as recently shown in Reference [13], this complexity can be drastically reduced in many cases by exploiting the inherent reversibility of quantum operations. The general idea is as follows: If two quantum circuits G and G' are equivalent, then concatenating the first circuit G with the inverse G'^{-1} of the second circuit would realize the identity function $\mathbb{I}$. The potential now lies in the order in which the operations from either circuit are applied. Whenever a strategy can be employed so that the respective gates from G and G' are applied such that the yielded decision diagrams remain close to the identity representation, the entire procedure can be conducted efficiently with low memory usage (e.g., in case of verifying the results of compilation flows [10]).

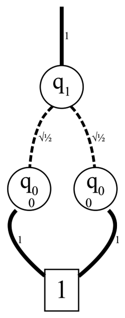
Example 7. Consider again the two circuits realizing the QFT shown earlier in Figure 5. Then, their equivalence can be concluded by (1) starting with a decision diagram resembling the identity, (2) applying one gate from the circuit shown in Figure 5(a), and, (3) applying all gates from the circuit shown in Figure 5(c) up to the next barrier (indicated by dashed lines in Figure 5(c)). The resulting decision diagram resembles the identity and, hence, the equivalence of both circuits can be concluded. Conducting the verification in this fashion only requires a maximum of 9 nodes (as opposed to 21 nodes for building the entire system matrix).

3 VISUALIZING DECISION DIAGRAMS IN DESIGN AUTOMATION

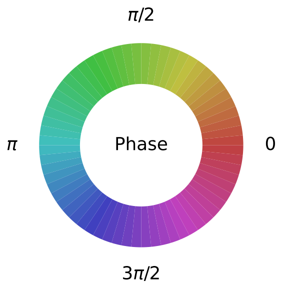
In the previous section, we have shown that decision diagrams provide a promising basis for important design tasks such as simulation and verification. However, users of the corresponding tools often do not have a corresponding background or an intuition about how these methods based on decision diagrams work and what their strengths and limits are. In an effort to make decision diagrams for quantum computing more accessible, we developed a tool that visualizes quantum decision diagrams and allows to explore their behavior when used in the design tasks covered above. This is similar to tools such as Quirk [36] but with a distinct focus on decision diagrams. To keep the effort of using the visualization as small as possible, the tool has been implemented as a web tool that can be used simply by accessing https://www.cda.cit.tum.de/app/ddvis/. In the following, we illustrate how our tool (1) visualizes decision diagrams for vectors and matrices, (2) can be used for simulating quantum circuits, and (3) can be used for verifying quantum circuits.

3.1 Styling Decision Diagrams

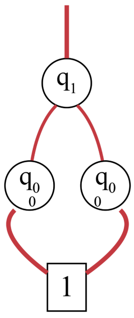
To provide the most accessible user interface possible, the tool offers several options for customizing how decision diagrams are visualized. Figure 7 illustrates the available styles for decision diagrams representing vectors, i.e., states of a quantum system. The “classic” mode (see Figure 7(a)) offers a look and feel that is most similar to what is found in research papers (as, e.g., shown throughout Section 2). Edges with a corresponding weight not equal to 1 are drawn using dashed lines and 0-stubs are retracted into the nodes themselves. Since the explicit annotation of edge weights quickly requires lots of space and leads to unreadable decision diagrams, there is also an option for removing these edge labels. Instead the magnitude of an edge weight can be reflected by the thickness of the line, while its complex phase can be color-coded using the HLS color wheel shown in Figure 7(b). Examples using this color code are shown in Figures 6 and 7(c). In addition to the above, the tool provides a more “modern” look for the decision diagram nodes, where the connection to the underlying state vector is expressed in a more straightforward fashion (shown later in Figure 8). Such a “modern” look is also available for matrices (shown later in Figure 9). These should allow less accommodated users to more easily grasp the concept of decision diagrams. In



(a) "Classic" mode

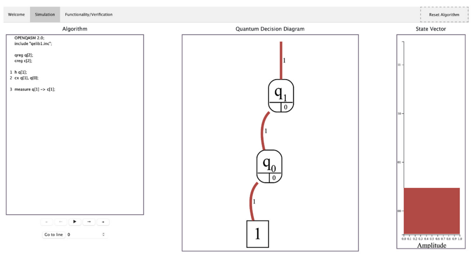


(b) HLS color wheel for phase encoding



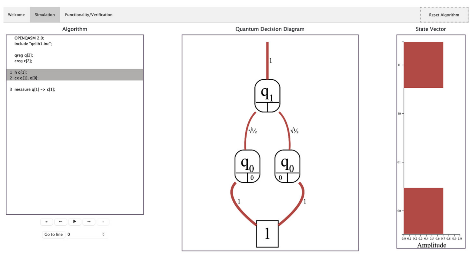
(c) Colored edge weights

Fig. 7. Visualization options for vector decision diagrams.



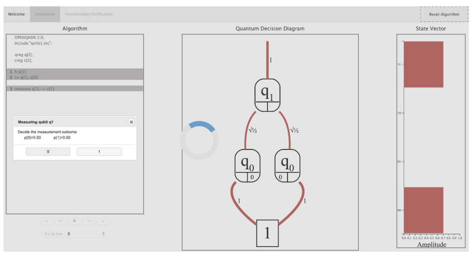
```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[2];

h q[1];
cx q[1], q[0];

measure q[1] -> c[1];
```

(a) Initial state $|00\rangle$



```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[2];

h q[1];
cx q[1], q[0];

measure q[1] -> c[1];
```

(b) Resulting state $1/\sqrt{2}\,|00\rangle + 1/\sqrt{2}\,|11\rangle$

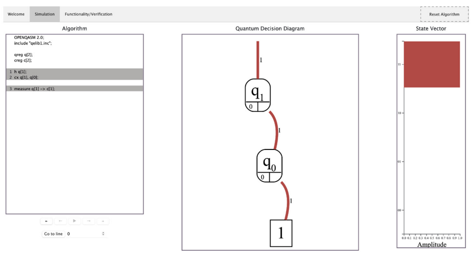


```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[2];

h q[1];
cx q[1], q[0];

measure q[1] -> c[1];
```

Measuring qubit q1

Decide the measurement outcome
p(0)=0.50 p(1)=0.50

(c) Measuring qubit q_0



```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[2];

h q[1];
cx q[1], q[0];

measure q[1] -> c[1];
```

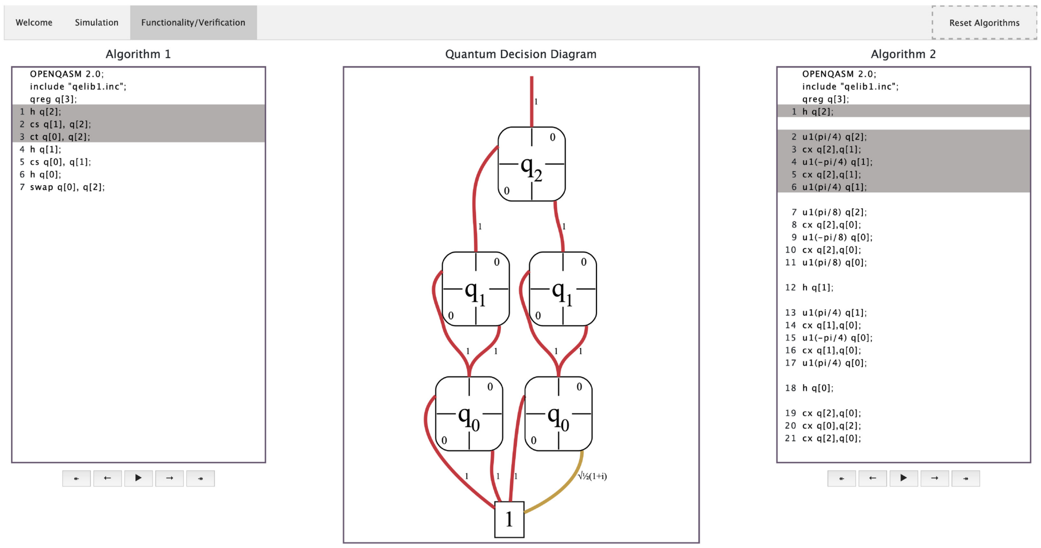
(d) Post-measurement state $|11\rangle$

Fig. 8. Visualizing the simulation of a circuit creating and measuring a Bell state.

the following, we provide a deeper look into the individual features of the tool and how simulation and verification of quantum circuits can be conducted using the tool.

3.2 Simulation of Quantum Circuits

The simulation feature of the tool provides a settings panel, an algorithm box for entering or loading a quantum algorithm/circuit, and an interactive decision diagram box that displays the current system state in terms of a decision diagram, as well as the state vector currently represented by the decision diagram. Loading quantum algorithms/circuits into the tool is as easy as drag-and-dropping an algorithm/circuit file (in either *.qasm* [15] or *.real* [46] format) into the corresponding algorithm box, or starting to enter your own description using one of the templates provided in the "Example Algorithms" list. Once a valid algorithm/circuit has been entered/loaded in the algorithm box, the simulation can be controlled using the navigation buttons below it:



```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
h q[2];
cs q[1], q[2];
ct q[0], q[2];
h q[1];
cs q[0], q[1];
h q[0];
swap q[0], q[2];
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
h q[2];

u1(pi/4) q[2];
cx q[2],q[1];
u1(-pi/4) q[1];
cx q[2],q[1];
u1(pi/4) q[1];

u1(pi/8) q[2];
cx q[2],q[0];
u1(-pi/8) q[0];
cx q[2],q[0];
u1(pi/8) q[0];

h q[1];

u1(pi/4) q[1];
cx q[1],q[0];
u1(-pi/4) q[0];
cx q[1],q[0];
u1(pi/4) q[0];

h q[0];

cx q[2],q[0];
cx q[0],q[2];
cx q[2],q[0];
```

Fig. 9. Visualizing the verification of the QFT circuits shown in Figure 5.

- →/← : Go one step forward or backward. Can be used to step through the simulation.
- ↠/↞ : Go straight to the end (or the next special operation; see below) or back to the beginning.
- ▹/|| : Start/Pause a slide show where the simulation advances step-by-step in an automated fashion.

Some operations are considered *special operations*, since they do not directly correspond to the application of a unitary matrix:

- Barrier statements (e.g., *"barrier q;"*) can be used as breakpoints when stepping forward with ↠.
- Measurement operations (e.g., *"measure q[0] -> c[0];"*) collapse the state of a qubit to one of its basis states. Whenever a qubit is about to get measured and it has a non-zero probability of being in either $|0\rangle$ or $|1\rangle$ (i.e., it is in superposition), a pop-up dialog appears that displays the probabilities for obtaining $|0\rangle$ and $|1\rangle$, respectively. Upon choosing one of the options, the decision diagram is irreversibly collapsed. Measurements also act as breakpoints due to their non-unitary (and, thus, non-reversible) nature. The tool supports OpenQASM's classically-controlled operations, where a certain gate is only applied if some classical bits obtained from measurements are set.
- Reset operations (e.g., *"reset q[0];"*) discard a qubit and initialize it to $|0\rangle$ as if it were a new qubit. Mathematically, this corresponds to taking the partial trace of the whole state and, then, setting the qubit to $|0\rangle$. However, the partial trace maps pure states to mixed states and can thus in general not be represented by the same kind of decision diagram used for representing state vectors. The tool supports resets in a probabilistic fashion (similar to measurements). Whenever a reset operation is encountered where the considered qubit has a non-zero probability of being in either $|0\rangle$ or $|1\rangle$, a pop-up dialog appears that displays the probabilities for obtaining $|0\rangle$ and $|1\rangle$, respectively. Upon choosing one of the options, the other decision diagram branch is discarded and the remaining branch is set as the $|0\rangle$ branch. Resets also act as breakpoints due to their non-unitary (and, thus, non-reversible) nature.

Example 8. Figure 8 illustrates the process of simulating the quantum circuit that creates and measures a Bell state. This circuit consists of two qubits, a Hadamard gate, and a controlled-NOT gate—followed by a measurement on both qubits. The first screenshot (Figure 8(a)) shows the initial quantum circuit and its state $|00\rangle$. Then, the two gates are applied—yielding the resulting state $1/\sqrt{2}\,|00\rangle + 1/\sqrt{2}\,|11\rangle$ (Figure 8(b)). If now the first qubit is to be measured, then there is a 50 % chance of it being either $|0\rangle$ or $|1\rangle$ (Figure 8(c)). Assume, that the measurement outcome is $|1\rangle$. Then, the value of the second qubit is completely determined due to the entanglement of both qubits—resulting in the final state $|11\rangle$ (Figure 8(d)).

3.3 Verification of Quantum Circuits

The verification feature of the tool provides a similar settings panel and decision diagram box as the simulation tab, but now features two algorithm boxes. In case only one algorithm/circuit is loaded in the left (right) algorithm box, the tool can be used to build the (inverse) functionality of the corresponding circuit.

Example 9. Creating the QFT circuit shown in Figure 5(a) in the left algorithm box and applying all the operations precisely yields the decision diagram shown in Figure 6.

Once a valid algorithm/circuit has been entered in each of the algorithm boxes, their equivalence can be checked by successively applying operations from both circuits (using the corresponding →/↠ controls) and checking whether the final result resembles the identity. As for the simulation, *Barrier* statements can be used as breakpoints when stepping through both algorithms/circuits. In contrast to simulation, *Measurement*, *Reset*, and *Classically-Controlled Operations* are currently not supported due to their non-unitary nature.

Example 10. Consider again the two circuits realizing the three-qubit QFT shown in Figures 5(a) and 5(c). Figure 9 shows how the tool is used to verify the equivalence of both circuits. The first three gates have already been applied from the left circuit, while six operations have been applied from the right circuit. The corresponding decision diagram in the middle only slightly differs from the identity (by the right node labeled q_0). Continuing the computation as discussed in 7 eventually allows to verify the equivalence of both circuits while remain close to the identity throughout the whole process.

4 DECISION DIAGRAMS WITHIN THE MUNICH QUANTUM TOOLKIT

The decision diagram-based methods discussed in Sections 2.2 and 2.3, respectively, have been implemented as part of the open-source Munich Quantum Toolkit (MQT; formerly JKQ [47]). MQT offers an interface for users to leverage the power of design automation for quantum computing as a black box: The user provides the input and does not have to develop a deeper understanding of the methods. In the following, we show how the MQT can be applied to two of the fundamental design problems in quantum computing: simulation (Section 4.1) as well as verification (Section 4.2) of quantum circuits.

4.1 Simulation of Quantum Circuits

MQT offers a decision diagram-based quantum circuit simulator called *DDSIM*. A recent case study [20] showed that decision diagram-based quantum circuit simulation outperforms array-based approaches whenever the (intermediate) quantum states have redundancy that can be exploited for a compact representation as decision diagram. DDSIM is able to simulate quantum circuits defined in *.qasm* [15], *GRCS* [5], and *.real* [46] format alongside parameterized instances of Grover's algorithm [19], Shor's algorithm [37], and the Quantum Fourier Transformation [33].

Using the standalone executable for MQT DDSIM requires cloning the GitHub repository https://github.com/cda-tum/ddsim and building the tool as in the following listing.[4] Alternatively, a pre-compiled Python package can be installed from PyPI with `pip install mqt.ddsim`.

```
$ git clone https://github.com/cda-tum/ddsim
$ cd ddsim
$ cmake -DCMAKE_BUILD_TYPE=Release -S . -B build
[...]
$ cmake --build build --config Release --target ddsim_simple
[...]
$ ./build/ddsim_simple --help
[displays available commands]
```

Example 11. Simulating Grover's algorithm with a two qubit oracle using the MQT DDSIM simulator can be conducted as follows:

```
$ ./ddsim_simple simulate --simulate_grover 2 --shots 1000 --ps
{
  "measurements": {
    "000": 503,
    "100": 497
  },
  "non_zero_entries": 2,
  "statistics": {
    "simulation_time": 0.125837,
    "measurement_time": 0.000180,
    "benchmark": "grover_2",
    "shots": 1000,
    "n_qubits": 3,
    "applied_gates": 16,
    "max_nodes": 8,
    "seed": 0
  }
}
```

Here, the parameters define the number of measurements to be performed on the final quantum state (`--shots 1000`) and cause the simulator to print statistics (`--ps`). The output is formatted according to the JSON standard and, hence, machine readable for further processing.

Simulations of a given *.qasm* or *.real* file with the simulator can be started with the parameter `--simulate_file <filename>.<extension>` (the respective format is derived from the extension). The full set of parameters can be listed via `./ddsim_simple --help`. This includes

- advanced techniques such as emulation [55], which enable significant speedups for certain quantum algorithms,
- weak simulation [24], which more faithfully mimics a physical quantum computer, and
- approximating simulation [23, 49], which enables a finely controlled tradeoff between accuracy and runtime.
- noise-aware simulation [21, 22], which allows to classically simulate the effects of noise on the execution of a quantum circuit.
- hybrid Schrodinger-Feynman simulation [8], which allows to exploit parallelization to significantly speedup simulation for certain use cases.

[4]The building process requires a C++17-compatible compiler and CMake version $\geq$3.14.

4.2 Verification of Quantum Circuits

Compiling quantum algorithms results in different representations of the considered functionality, which significantly differ in their basis operations and structure but are still supposed to be functionally equivalent. Consequently, checking whether the originally intended functionality is indeed maintained throughout all these different abstractions becomes increasingly relevant to guarantee an efficient, yet correct design flow. Existing solutions for equivalence checking of quantum circuits suffer from the complexity of the underlying problem—which has been proven to be QMA-complete [27]. Most notably, the need to represent matrices that require an exponential amount of memory with respect to the number of qubits quickly makes such approaches infeasible. However, quantum mechanical characteristics, such as the inherent reversibility of quantum operations and the compact representation of the identity with decision diagrams, provide potential for more efficient equivalence checking of quantum circuits.

MQT offers a *quantum circuit equivalence checking (QCEC)* tool [14] that explicitly exploits these characteristics based on the ideas outlined in References [11–13] and offers a strategy especially suited for verifying compilation results [10], as well as dedicated random stimuli generation schemes [9]. Similarly to our simulator, MQT QCEC can be obtained from the GitHub repository https://github.com/cda-tum/qcec and can subsequently be used by building the `qcec_app` CMake target. However, there is an easier way for users to get started with the MQT QCEC tool. Specifically, the tool is also provided as a Python package, which can be easily installed with `pip install mqt.qcec` and also provides native integration with IBM Qiskit.

Example 12. Verifying that a quantum circuit has been compiled correctly merely requires the following lines of Python:

```python
from mqt.qcec import Configuration, Strategy, verify
from qiskit import QuantumCircuit, transpile

# create your quantum circuit
qc = <...>

# append measurements to save output mapping of physical to logical (qu)bits
qc.measure_all()

# compile circuit to appropriate backend using some optimization level
qc_comp = transpile(qc, backend=<...>, optimization_level=<0 | 1 | 2 | 3>)

# set dedicated verification strategy
config = Configuration()
config.strategy = Strategy.compilationflow

# verify the compilation result
result = verify(qc, qc_comp, config)
```

A complete list of the available methods as well as additional configuration options can be listed via `help(qcec.Configuration)` in Python or `qcec_app --help` when using the commandline app.

5 DEVELOPER'S PERSPECTIVE

The design automation tools described in the previous sections are quite powerful by themselves; nonetheless, there is always room for improvement and additional features. Because of this, the visualization and the MQT is open source and developers are kindly invited to extend or

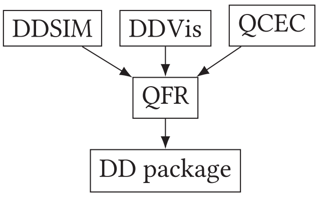


Fig. 10. Structure of the MQT.

modify the methods at their own discretion. This section gives a brief overview of how the tools work internally and serves as a starting point for the interested developer wanting to take a deeper dive.

The dependency relations of the MQT discussed here are illustrated in Figure 10 and the source code of the individual application can be found on GitHub as individual repositories via https://github.com/cda-tum/. Each application depends on a library referred to as *Quantum Functionality Representation (QFR)*, which handles the input and output of files describing quantum functionalities. Additionally selected quantum algorithms such as Grover's search and Shor's algorithm are directly integrated as classes—allowing to programatically construct the respective quantum circuits for simulation, compilation, or verification with parameters controlling, e.g., the number qubits. The QFR itself depends on a package providing the functionality for representing and manipulating quantum states and operations via decision diagrams [50, 53].

The DD package has options that are set during compile time to enable more aggressive compiler optimizations and influence the later execution in simulation and verification. These fall primarily into one of the following categories:

- *Cache sizes:* The underlying routines storing and operating on decision diagrams use different caches for nodes, edges, and complex numbers. There is a tradeoff between larger cache sizes and better data locality. Developers can adjust these values according to their needs.
- *Floating point representation:* Across all individual projects, the generally used floating point datatype by default is `double` (i.e., 64 bit on most platforms), as defined through the `fp` alias (in `include/dd/Definitions.hpp`). Depending on the required precision, the developer may change this to `float` (less precision with faster execution) or `long double` (higher precision but slower execution). If the precision is changed, then this should be reflected in the `TOLERANCE` (defined in `include/dd/ComplexTable.hpp`), which mitigates effects caused by the fundamentally limited precision in the representation of complex numbers [50].

Support for additional "hardcoded" algorithms or file formats should be integrated into the QFR, so the tools for simulation, verification, and visualization can access these new features. We recently added support for importing IBM Qiskit `QuantumCircuit` objects from Python to the QFR library—allowing to integrate our tools with Qiskit through Python bindings. In the future, this could be extended to integrate with other popular quantum software frameworks.

Internally, the simulator can be used as follows to simulate the *.qasm* file `shor_115_2.qasm` and conduct 1,000 measurements:

```
QuantumComputation qc1("shor_115_2.qasm");
QFRSimulator sim(qc1);
sim.Simulate();
auto samples = sim.MeasureAllNonCollapsing(1000);
```

This reads a file `shor_115_2.qasm` into the QFR and uses the resulting `QuantumComputation` object to construct the simulator instance. The actual simulation process is handled in the `Simulate` method, where developers can start optimizing for their specific problem by creating their own simulator sub-class. For the simulation of files each instruction in the input file is translated into the corresponding decision diagram (for unitary operations) and applied to the quantum state. Non-unitary operations such as measurements require separate handling in the program. Other paradigms such as *approximating simulation* [23, 49], *weak simulation* [24], noise-aware simulation [22], and hybrid Schrodinger-Feynman simulation [8] are also supported and easy to extend.

The equivalence checking methodology described in References [9–13] is readily extendable and offers lots of freedom for adapting to specific scenarios. Developers wanting to implement their own equivalence checking strategies can get started at `ImprovedDDEquivalenceChecker.hpp`. There, the `Proportional` strategy for example is realized in the following way:

```
int ratio1 = std::max(1, qc1.getNops()/qc2.getNops());
int ratio2 = std::max(1, qc2.getNops()/qc1.getNops());
while (it1 != end1 && it2 != end2) {
	for (int i=0; i<ratio1 && it1!=end1; ++i, ++it1)
		applyGate(*it1, results.result, perm1, LEFT);
	for (int i=0; i<ratio2 && it2!=end2; ++i, ++it2)
		applyGate(*it2, results.result, perm2, RIGHT);
}
```

In `CompilationFlowEquivalenceChecker.hpp`, the dedicated compilation flow verification strategy can easily be extended to anticipate further optimizations, or adapt to compilation flows different than IBM Qiskit [2]. Currently, QCEC supports different notions of equivalence, i.e., "unitary equivalence," "equivalence up to global phase," and "equivalence of measurement outcomes." In the future, the methodology could also be extend to support further notions of equivalence. This includes "equivalence up to relative phase," "equivalence up to permutation of the outputs," or (even more general) "equivalence up to relabeling of the qubits."

The MQT decision diagram visualization tool DDVis is designed as a NodeJS application that builds on top of our QFR library. It uses the `dd::export2Dot()` functionality of the DD package to generate a GraphViz [17] representation of the considered decision diagrams, which is, in turn, rendered using `d3-graphviz`. At the moment, it provides basic support for visualizing the simulation and verification of quantum circuits. The simulation part might be extended in the future to accommodate the approximation capabilities of the MQT DDSIM simulator. Similarly, the verification part might be extended to automatically conduct one of the available verification schemes from the MQT QCEC tool and provide an indicator whether the two circuits are considered equivalent at any particular time.

For more details on the implementations, we refer to the documentations in the respective repositories at https://github.com/cda-tum/, which include links to pre-compiled packages on the PyPI service.

6 CONCLUSIONS

In an effort to bridge the gap between the design automation and the quantum community and to make decision diagrams for quantum computing more accessible, we provided a summary on how decision diagram techniques can be used for the design of quantum circuits. Additionally, we presented an easily accessible tool that visualizes quantum decision diagrams as well as their applications and covered the usage of the Munich Quantum Toolkit from two perspectives: First, for users, who want to solve their problems, but not necessarily develop a deeper understanding of the tools and how they work. Second, for developers, who want to enhance or integrate the tools to tackle their specific problems in quantum computing. More information about the corresponding tools is available at https://github.com/cda-tum/. We sincerely hope that our efforts help interested researchers to learn and adopt these techniques.

REFERENCES

[1] Afshin Abdollahi and Massoud Pedram. 2006. Analysis and synthesis of quantum circuits by using quantum decision diagrams. In *Proceedings of the Design, Automation and Test in Europe Conference*. 317–322.

[2] Héctor Abraham et al. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://github.com/Qiskit/qiskit/blob/master/Qiskit.bib.

[3] Frank Arute et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574 (2019), 505–510.

[4] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. 1993. Algebraic decision diagrams and their applications. In *Proceedings of the International Conference on CAD*. 188–191.

[5] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. 2018. Characterizing quantum supremacy in near-term devices. *Nat. Phys.* 14, 6 (April 2018), 595–600.

[6] Randal E. Bryant. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* C-35, 8 (1986), 677–691.

[7] Randal E. Bryant. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.* 24, 3 (1992), 293–318.

[8] Lukas Burgholzer, Hartwig Bauer, and Robert Wille. 2021. Hybrid schrödinger-feynman simulation of quantum circuits with decision diagrams arXiv:2105.07045. https://arxiv.org/abs/2105.07045.

[9] Lukas Burgholzer, Richard Kueng, and Robert Wille. 2021. Random stimuli generation for the verification of quantum circuits. In *Proceedings of the Asia and South Pacific Design Automation Conference*.

[10] Lukas Burgholzer, Rudy Raymond, and Robert Wille. 2020. Verifying results of the IBM Qiskit quantum circuit compilation flow. In *Proceedings of the International Conference on Quantum Computing and Engineering*. 356–365.

[11] Lukas Burgholzer and Robert Wille. 2020. Improved DD-based equivalence checking of quantum circuits. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 127–132.

[12] Lukas Burgholzer and Robert Wille. 2020. The power of simulation for equivalence checking in quantum computing. In *Proceedings of the Design Automation Conference*.

[13] Lukas Burgholzer and Robert Wille. 2021. Advanced equivalence checking for quantum circuits. *IEEE Trans. CAD Integr. Circ. Syst.* 40, 9 (2021), 1810–1824.

[14] Lukas Burgholzer and Robert Wille. 2021. QCEC: A JKQ tool for quantum circuit equivalence checking. *Softw. Impacts* 7 (2021), 100051.

[15] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. 2017. Open quantum assembly language. arXiv:1707.03429. https://arxiv.org/abs/1707.03429.

[16] Rolf Drechsler, Andisheh Sarabi, Michael Theobald, Bernd Becker, and Marek A. Perkowski. 1994. Efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams. In *Proceedings of the Design Automation Conference*. 415–419.

[17] Emden R. Gansner and Stephen C. North. 2000. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exp.* 30, 11 (2000), 1203–1233.

[18] Jordan Gergov and Christoph Meinel. 1994. Efficient boolean manipulation with OBDD's can be extended to FBDD's. *IEEE Trans. Comput.* 43, 10 (1994), 1197–1209.

[19] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Theory of Computing*. 212–219.

[20] Thomas Grurl, Jürgen Fuß, Stefan Hillmich, Lukas Burgholzer, and Robert Wille. 2020. Arrays vs. decision diagrams: A case study on quantum circuit simulators. In *Proceedings of the 50th IEEE International Symposium on Multiple-Valued Logic (ISMVL'20)*. IEEE, 176–181.

[21] Thomas Grurl, Jürgen Fuß, and Robert Wille. 2020. Considering decoherence errors in the simulation of quantum circuits using decision diagrams. In *Proceedings of the International Conference on CAD*.

[22] Thomas Grurl, Richard Kueng, Jürgen Fuß, and Robert Wille. 2021. Stochastic quantum circuit simulation using decision diagrams. In *Proceedings of the Design, Automation and Test in Europe*.

[23] Stefan Hillmich, Richard Kueng, Igor L. Markov, and Robert Wille. 2021. As accurate as needed, as efficient as possible: Approximations in DD-based quantum circuit simulation. In *Proceedings of the Design, Automation and Test in Europe*.

[24] Stefan Hillmich, Igor L. Markov, and Robert Wille. 2020. Just like the real thing: Fast weak simulation of quantum computation. In *Proceedings of the Design Automation Conference*.

[25] Xin Hong, Yuan Feng, Sanjiang Li, and Mingsheng Ying. 2021. Equivalence checking of dynamic quantum circuits. arXiv:2106.01658 [quant-ph]. https://arxiv.org/abs/2106.01658.

[26] Xin Hong, Xiangzhen Zhou, Sanjiang Li, Yuan Feng, and Mingsheng Ying. 2020. A tensor network based decision diagram for representation of quantum circuits. arXiv:2009.02618 [quant-ph]. https://arxiv.org/abs/2009.02618.

[27] Dominik Janzing, Pawel Wocjan, and Thomas Beth. 2005. "Non-identity check" is QMA-complete. *Int. J. Quant. Inf.* 3, 3 (2005), 463–473.

[28] Lingling Lao, Hans van Someren, Imran Ashraf, and Carmen G. Almudever. 2020. Timing and resource-aware mapping of quantum circuits to superconducting processors. arXiv:1908.04226 [quant-ph]. https://arxiv.org/abs/1908.04226.

[29] Atsushi Matsuo, Wakaki Hattori, and Shigeru Yamashita. 2019. Reducing the overhead of mapping quantum circuits to IBM Q system. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (2019).

[30] D. Michael Miller and Mitchell A. Thornton. 2006. QMDD: A decision diagram structure for reversible and quantum circuits. In *Proceedings of the International Symposium on Multi-Valued Logic*.

[31] Shin-ichi Minato. 1993. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the Design Automation Conference*. 272–277.

[32] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*. 1015–1029.

[33] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information*. Cambridge University Press.

[34] Philipp Niemann, Robert Wille, and Rolf Drechsler. 2014. Efficient synthesis of quantum circuits implementing clifford group operations. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 483–488.

[35] Philipp Niemann, Robert Wille, David Michael Miller, Mitchell A. Thornton, and Rolf Drechsler. 2016. QMDDs: Efficient quantum function representation and manipulation. *IEEE Trans. CAD Integr. Circ. Syst.* 35, 1 (2016), 86–99.

[36] Quirk authors. 2021. *Quirk: Quantum Circuit Simulator*. Retrieved August 2, 2021 from https://algassert.com/quirk.

[37] Peter W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comp.* 26, 5 (1997), 1484–1509.

[38] Kaitlin N. Smith and Mitchell A. Thornton. 2019. Quantum logic synthesis with formal verification. *IEEE Int. Midw. Symp. Circ. Syst.* (2019), 73–76.

[39] Mathias Soeken, Robert Wille, Christoph Hilken, Nils Przigoda, and Rolf Drechsler. 2012. Synthesis of reversible circuits with minimal lines for large functions. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 85–92.

[40] Bochen Tan and Jason Cong. 2020. Optimal layout synthesis for quantum computing. In *Proceedings of the International Conference on CAD*.

[41] George F. Viamontes, Igor L. Markov, and John P. Hayes. 2003. Improving gate-level simulation of quantum circuits. *Quant. Inf. Process.* 2, 5 (2003), 347–380.

[42] Lieuwe Vinkhuijzen, Tim Coopmans, David Elkouss, Vedran Dunjko, and Alfons Laarman. 2021. LIMDD a decision diagram for simulation of quantum computing including stabilizer states. arXiv:2108.00931 [quant-ph]. Retrieved from https://arxiv.org/abs/2108.00931.

[43] Shiou-An Wang, Chin-Yung Lu, I.-Ming Tsai, and Sy-Yen Kuo. 2008. An XQDD-based verification method for quantum circuits. In *IEICE Trans. Fundam.* 91, 2 (2008). 584–594.

[44] Ingo Wegener. 2000. Branching programs and binary decision diagrams: Theory and applications. *SIAM J. Comput.* (2000).

[45] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Proceedings of the Design Automation Conference* (2019).

[46] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W. Dueck, and Rolf Drechsler. 2008. RevLib: An online resource for reversible functions and reversible circuits. In *Proceedings of the International Symposium on Multi-Valued Logic*. 220–225.

[47] Robert Wille, Stefan Hillmich, and Lukas Burgholzer. 2020. JKQ: JKU tools for quantum computing. In *Proceedings of the International Conference on CAD*.

[48] Han-Sen Zhong et al. [n.d.]. Quantum computational advantage using photons. arXiv:2012.01625. https://arxiv.org/abs/2012.01625.

[49] Alwin Zulehner, Stefan Hillmich, Igor L. Markov, and Robert Wille. 2020. Approximation of quantum states using decision diagrams. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 121–126.

[50] Alwin Zulehner, Stefan Hillmich, and Robert Wille. 2019. How to efficiently handle complex values? Implementing decision diagrams for quantum computing. In *Proceedings of the International Conference on CAD*.

[51] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2019. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Trans. CAD Integr. Circ. Syst.* 38, 7 (2019), 1226–1236.

[52] Alwin Zulehner and Robert Wille. 2018. One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic. *IEEE Trans. CAD Integr. Circ. Syst.* 37, 5 (2018), 996–1008.

[53] Alwin Zulehner and Robert Wille. 2019. Advanced simulation of quantum computations. *IEEE Trans. CAD Integr. Circ. Syst.* 38, 5 (2019), 848–859.

[54] Alwin Zulehner and Robert Wille. 2019. Compiling SU(4) quantum circuits to IBM QX architectures. In *Proceedings of the Asia and South Pacific Design Automation Conference.* 185–190.

[55] Alwin Zulehner and Robert Wille. 2019. Matrix-vector vs. matrix-matrix multiplication: Potential in DD-based simulation of quantum computations. In *Proceedings of the Design, Automation and Test in Europe Conference.*

Received April 2021; revised August 2021; accepted August 2021