Efficient *and* Correct Compilation of Quantum Circuits

(Overview Paper)

Robert Wille Stefan Hillmich Lukas Burgholzer
Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
{robert.wille, stefan.hillmich, lukas.burgholzer}@jku.at
http://iic.jku.at/eda/research/quantum/

***Abstract*—High-level descriptions of quantum algorithms do not take the restrictions of physical hardware into account. Therefore actually executing an algorithm in the form of a quantum circuit on a quantum computer requires compiling it for the desired target architecture first. The compilation of quantum circuits depends on efficient methods to be feasible for all but the trivial instances. To this end, different compiling methods have been introduced in the past, but room for improvement still exists. Moreover, just an efficient compilation process itself is not sufficient—the resulting circuits must be correct as well. In this summary paper, we review how existing compilation approaches can be optimized by utilizing heuristic search algorithms or exact reasoning engines. Furthermore, we review how the correctness of the obtained results can be verified afterwards by clever data structures such as decision diagrams. This illustrates core steps of a compilation flow which can generate minimal or close-to-minimal results for many instances and, additionally, guarantees correctness throughout the process.**

I. Introduction

Quantum computing offers a new model of computation that promises to significantly outperform classical computing at specific tasks. These tasks include integer factorization (Shor's algorithm [1]), database search (Grover's search [2]), quantum chemistry [3], Boson sampling [4], and many more. An increasing number of big players such as Google, IBM, and Microsoft as well as start-ups like Rigetti and IonQ are currently investing in the field in order to bring quantum computing closer to reality.

Despite the plethora of promising applications, the capability of physical quantum computers is not quite there yet—the phase we are currently in is often referred to as *Noisy Intermediate-Scale Quantum Computing* (NISQ [5]). At the moment, only up to around 50 qubits are available, which additionally suffer from comparatively high error rates and short decoherence times. Due to the small number of qubits, quantum error correction methods are not yet feasible.

The shortcomings of the hardware can be mitigated to some extent by carefully selecting algorithms and utilizing an appropriate design flow. Since quantum algorithms are typically formulated on an abstraction level agnostic of the restrictions imposed by physical hardware, one or more compilation steps are required in order to execute them on the actual hardware. This includes decomposing the algorithmic description into gates from the supported quantum operation library (such as Clifford+T) and adhering to possible connectivity constraints imposed by the architecture.

To this end, dedicated compilation flows are required. They have to satisfy two major properties: On the one hand, they have to be *efficient*, i.e., they should be able to decompose a given algorithm and to address the connectivity constraints using as few elementary operations as possible. On the other hand, they have to be *correct*, i.e., it should be possible to check whether the resulting quantum circuit indeed realizes the initially given quantum functionality.

In this work, we review existing compilation methods and how they can be optimized using heuristic as well as exact methods. We particularly focus thereby on the so-called mapping problem which addresses the connectivity constraints mentioned above. Afterwards, we review methods that can be used to verify whether the obtained results are correct, i.e., whether they indeed realize the desired functionality. To this end, we focus on methods based on decision diagrams, which are particularly suited for this task. All methods will be illustrated by means of examples. Furthermore, references for further reading and a more in-depth treatment will be provided.

The remainder of this paper is structured as follows: Section II provides a brief overview of quantum computing and architectural constraints of physical devices. Section III discusses the problem of mapping and reviews efficient solutions for that. Afterwards, Section IV covers verification for quantum computing, especially the problem of equivalence checking, which can be used to check whether the results determined from the compilation are correct. Finally, Section V concludes the paper.

II. Background

This section provides the background in quantum computations and architectural constraints of quantum computers which are necessary to keep this work self-contained.

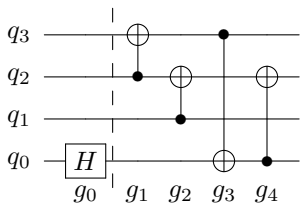
A. Quantum Computation

Computations in the quantum realm utilize *quantum bits* (*qubits*) [6], which can assume more states than just 0 and 1 known from conventional logic. In fact, while the basis states (denoted by $|0\rangle$ and $|1\rangle$ in Dirac-notation) are the same, a qubit $|\psi\rangle$ can be in any linear combination $|\psi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$ described through *amplitudes* $\alpha, \beta \in \mathbb{C}$ with $|\alpha|^2 + |\beta|^2 = 1$. If both amplitudes α and β are non-zero, the qubits state is also referred to as being in *superposition*. A second exploitable quantum effect is called *entanglement*,

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

$$CNOT = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

(a) Quantum operations



(b) Quantum circuit

Fig. 1: Quantum operations and circuit

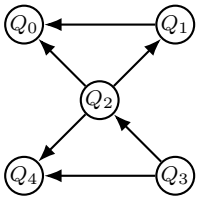


Fig. 2: Coupling map of IBM QX4

where operations on one qubit influence the state of another qubit. On a quantum computer, the values of α and β cannot be directly observed. Instead, a measurement of a qubit collapses the state of the qubit back to one of the basis states $|0\rangle$ (with probability $|\alpha|^2$) or $|1\rangle$ (with probability $|\beta|^2$).

In general a quantum system consists of an ensemble of n qubits—spanning a 2^n-dimensional complex state space, i.e., $|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$ with $\alpha_x \in \mathbb{C}$ such that $\sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1$, which is commonly represented as the 2^n-dimensional complex *state vector* $[\alpha_x]_{x\in\{0,1\}^n}$. The current state of a quantum system $|\psi\rangle$ can be manipulated using quantum operations, which are defined through unitary $2^n \times 2^n$ matrices[1]. With the exception of measurements, quantum operations are therefore inherently reversible.

Example 1. *Two common quantum operations with their matrix representations are shown in Fig. 1a. The Hadamard transformation H sets a qubit into superposition, e.g., it transforms $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. The controlled-NOT $CNOT$ operates on two qubits and negates the* target qubit*, iff the chosen* control qubit *is in the state $|1\rangle$.*

A quantum algorithm is commonly described as a sequence of quantum operations applied to the qubits of a quantum system. Quantum circuit diagrams allow to visualize any such sequence. Here, the qubits are represented by horizontal lines, while quantum operations are placed on the qubits and are applied from left to right. In this context quantum operations are also referred to as *quantum gates*.

Example 2. *Fig. 1b illustrates a quantum circuit consisting of a single Hadamard gate and four CNOTs. In case of the CNOTs, the black dot denotes the control qubit, while the $\oplus$-symbol denotes the target qubit. The gates are labeled $g_0, \ldots, g_4$ and are applied to the qubits $q_0, \ldots, q_3$.*

B. Quantum Computer Architectures

The high-level descriptions of quantum algorithms commonly do not satisfy the restrictions imposed by actual physical devices. Hence, the quantum circuit representing the algorithm has to be compiled to a version executable on the hardware. The remainder of this section focuses on the IBM QX architectures [7] as a representative example.

[1]A matrix U is unitary if $UU^\dagger = U^\dagger U = \mathbb{I}$, i.e., its inverse U^{-1} is given by the conjugate transpose $U^\dagger$ [6].

All quantum computers available today only offer a restricted set of quantum gates that can be applied to the devices' qubits. This does not mean these quantum computers are not universal—in fact, rather "small" sets of libraries such as Clifford+T are sufficient to approximate any quantum state arbitrarily close [6].

Example 3. *Quantum computers made by IBM share the same set of supported quantum operations. They provide the single-qubit U gate, which allows for arbitrary single-qubit operations. Additionally, IBM architectures support the two-qubit $CNOT$ gate. As these operations form a superset of the Clifford+T gate set, universal quantum computations may theoretically be performed on IBM's hardware.*

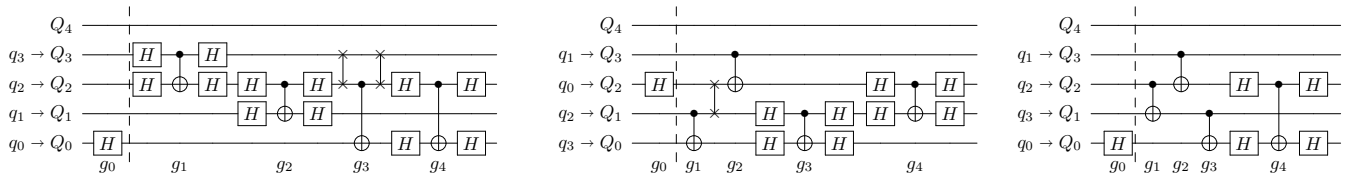
In addition to restricting the natively available gates, quantum computers may also impose certain (hardware-dependent) connectivity constraints. These typically limit the possible interactions between qubits in the form of a oupling map.

Example 4. *Consider the IBM QX4 architecture with five qubits, whose coupling map is shown in Fig. 2. There, the nodes Q_i indicate the physical qubits and an arrow from Q_c to Q_t indicates that a CNOT with control Q_c and target Q_t may be applied. In order to distinguish an algorithm's logical and the hardware's physical qubits, logical qubits are denoted with lower-case letters q_i by convention.*

Such connectivity constraints only apply to certain types of quantum computers, e.g., those based on superconducting qubits [8]—as opposed to hardware based on trapped ions [9].

III. Compilation

Compilation of quantum circuits necessitates a multitude of different tasks in order to conduct a given quantum computation on a specific physical device. As already mentioned in the previous section, algorithmic building blocks have to be *decomposed* into elementary gates provided by the underlying architecture. Then, the resulting circuit's logical qubits have to be *mapped* to the device's physical qubits in a way that all coupling constraints imposed by the architecture are satisfied. Furthermore, several pre- and post-mapping *optimizations* may be applied in order to reduce gate count and/or increase computation fidelity. In the following, we consider the *mapping problem* as one of the most important parts of the compilation problem.



(a) Naïve approach (b) A^* approach (c) Minimal approach

Fig. 3: Solutions G' for the compilation problem

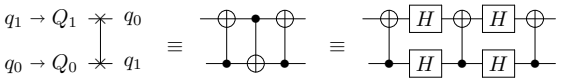


Fig. 4: Decomposition of a SWAP operation

A. The Mapping Problem

For the mapping problem, we assume that all building blocks of the quantum algorithm are already decomposed into the specific gate set provided by the target architecture (for this purpose, several methods have been proposed in the literature, see, e.g. [10]–[13]). Then, in order to satisfy all connectivity constraints imposed by the device's coupling map (see Section II-B), the algorithms logical qubits have to be accordingly assigned to the architecture's physical qubits. Since there are typically no constraints on the application of single qubit gates, these need not be considered during the mapping process. In general, such a mapping cannot be determined in a static fashion, i.e., by fixing an initial assignment, but the mapping has to change dynamically in all but the trivial cases. Typically, this is realized by inserting SWAP operations[2], which allow to exchange the logical qubits the operation acts on (see Fig. 4).

However, tackling this problem in a naive fashion, i.e., by considering gate after gate and resolving arising issues through appropriate SWAP insertions, does not yield a feasible procedure and frequently incurs large overheads.

Example 5. *Consider again the circuit from Fig. 1b and assume that the target architecture is IBM QX4 with the coupling map given in Fig. 2. Choosing the initial mapping $q_i \to Q_i$ yields a situation where none of $g_1, \ldots, g_4$ satisfy the coupling constraints. Adding SWAP gates in a naive fashion (i.e., by adding SWAP gates to "move" the qubits so that they eventually satisfy the constraints) may lead to a result as shown in Fig. 3a—and, hence, a circuit G' with substantially more gates.*

B. Proposed Solutions

Determining a solution which satisfies the constraints and, at the same time, keeps the cost at a minimum has been proven an NP-complete problem [14], [15]. Accordingly, efficient approaches are key to tackle this problem. In the following paragraphs, we will briefly review two different solutions to the mapping problem, namely a heuristic one based on the A^* search algorithm (introduced in [16]) and a minimal one utilizing reasoning engines (introduced in [17])[3].

The A^* algorithm [23] is a state-space search algorithm. To this end, (sub-)solutions of the considered problem are represented by state nodes. Nodes that represent a complete solution are called *goal nodes* (multiple goal nodes may exist). The main idea is to determine the cheapest path (i.e., the path with the lowest cost) from the root node (representing an empty solution) to a goal node (representing a complete solution). Since the search space is typically exponential, sophisticated mechanisms are employed in order to consider as few paths as possible. This A^*-algorithm was one of the first heuristic methods that has been utilized to address the mapping problem [16].

Example 6. *Fig. 3b illustrates the mapped circuit G' resulting from the A^*-approach. As can be seen, this circuit requires fewer SWAP and H operations and, even it is not optimal in the number of gates, this result is commonly attained after a short runtime.*

A minimal mapped circuit can be obtained by finding the cheapest of all possible mappings – again, an $\mathcal{NP}$-complete problem [14], [15]. To this end, the mapping task can be formulated as a problem of *Boolean satisfiability* (SAT) for which efficient reasoning engines tackling this complexity exists (e.g., [24], [25]). The relationship between logical and physical qubits is encoded in Boolean variables which are then constrained in order to only allow valid assignments conforming with the targeted architecture and ruling out invalid solutions (details on that are provided in [17]). Passing these variables and constraints to a reasoning engine in combination with a cost metric (e.g., number of additionally inserted gates) allows to determine an optimal solution to the mapping problem.

Example 7. *Fig. 3c provides the mapped circuit G' determined by the reasoning engine as a result of the SAT encoding. Notably it does not require a single SWAP operation, the only change necessary was the addition of four Hadamard operations to account for the direction in the coupling map.*

Both approaches are extremely flexible when it comes to adjusting to different architectures or cost metrics, as was already shown in [26], [27] for the heuristic A^* algorithm, allowing for a broad applicability of these mapping techniques.

[2]In case a CNOT with control Q_i and target Q_j is to be applied, but there is only an edge from Q_j to Q_i in the coupling map, instead of applying a SWAP operation the CNOT can be surrounded with four Hadamard operations, effectively switching control and target

[3]Several solutions exist that address the similar problem of nearest neighbor optimization [18]–[22].

IV. Verification

The design task verification comprises multiple levels from the specification of an algorithm down to the actual execution on physical hardware. In this section, we consider *equivalence checking* as a representative part of verification. The problem of equivalence checking addresses the question whether two given circuits G and G' do realize the same function—an important question during/after the compilation process. While the problem has been intensively studied in the past [28], [29], huge improvements may be achieved by utilizing efficient data structures and exploiting some properties unique to quantum computing.

A. Decision Diagrams

Decision diagrams have been successfully utilized to drastically reduce the required memory to represent state vectors in quantum computing [30]–[33]. Strong simulation approaches based on decision diagrams have recently moved into the spotlight since they can significantly outperform array-based simulators in cases where redundancies can be exploited—in extreme cases leading to an improvement in runtime by from 30 days to 2 minutes [34], [35].

The main idea of decision diagrams is to identify redundancies in the state vector and provide compaction by sharing structures. The vector is split in half into two sub-vectors. This process is repeated until the sub-vectors contain only single elements, i.e., one split for every qubit. If identical sub-vectors occur in the process, this redundancy is exploited by re-using (sharing) the same structure in the resulting decision diagram.

More precisely, consider a quantum system with qubits $q_{n-1}, q_{n-2}, \ldots, q_0$, whereby q_{n-1} represents the most significant qubit. Then, the first 2^{n-1} entries of the corresponding state vector represent the amplitudes for the basis states with q_{n-1} set to $|0\rangle$; the other entries represent the amplitudes for states with q_{n-1} set to $|1\rangle$. This decomposition is represented in a decision diagram structure by a node labeled q_{n-1} and two successors leading to nodes representing the two sub-vectors. By convention, the left (right) edge indicates the 0-successor (1-successor). The sub-vectors are recursively decomposed further until vectors of size 1 (i.e., complex numbers) result. During this decomposition, equivalent sub-vectors can be represented by the same nodes—reducing the complexity of the representation. Then, instead of having a terminal node for every distinct value in the state vector, common factors of the amplitudes are stored in the edge weights. The value can be reconstructed by multiplying the edge weights along the desired path in the decision diagram.

B. DD-based Equivalence Checking

Decision diagrams are ideal for equivalence checking, since the resulting representations are canonic (as proven in [36]), i.e., simply creating decision diagrams for two circuits G and G' and comparing them afterwards solves the problem (as done in [37], [38]). Besides that, further improvements can be achieved.

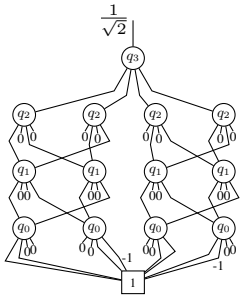


Fig. 5: DD for circuits G and G' from Fig. 1b and Fig. 3

For example, in case two circuits G and G' are indeed equivalent, it certainly holds that $G \cdot G'^{-1} = \mathbb{I}$, i.e., multiplying G with the inverse of G' results in the identity $\mathbb{I}$[4].

The full potential of this idea is utilized by exploiting the associativity of the matrix multiplications involved in computing $G \cdot G'^{-1}$, i.e., starting from a decision diagram representing the identity $\mathbb{I}$, gates from G and G'^{-1} are applied successively either from the left (G) or from the right (G'^{-1}):

$$\begin{aligned} G \cdot G'^{-1} &= (g_0 \ldots g_{m-1}) \cdot (g'^{-1}_{m'-1} \cdots g'^{-1}_0) \\ &\equiv (U_{m-1} \cdots U_0) \cdot (U'^{-1}_0 \cdots U'^{-1}_{m'-1}) \\ &\equiv (U_{m-1} \cdots U_0) \rightarrow \mathbb{I} \leftarrow (U'^{-1}_0 \cdots U'^{-1}_{m'-1}) \\ &= G \rightarrow \mathbb{I} \leftarrow G'^{-1}. \end{aligned}$$

However, determining when to apply gates from G and when to apply gates from G'^{-1} is not always obvious. But whenever a "good" strategy for a selection of gates can be employed, equivalence checking of two equivalent quantum circuits can be conducted very efficiently and compactly using decision diagrams.

Example 8. *Consider the two circuits G and G' from Fig. 1b and Fig. 3c. Conducting the multiplications in an alternating fashion frequently results in a situation where the application of a gate from G is immediately reverted by the application of gates from G'^{-1}, thus effectively keeping the involved decision diagram close to the identity. By this, instead of* 13 *nodes (as shown in Fig. 5), no more than* 7 *nodes are required during the equivalence check.*

V. Conclusions

In this work, we discussed methods towards an efficient and correct compilation flow for quantum circuits. To this end, we showed how to conduct efficient mapping utilizing reasoning engines such as satisfiability solvers to attain minimal solution as well as an A^*-based heuristic for close-to-minimal solutions. However, efficiency by itself is not sufficient, correctness is required, too. In this regard, we showed how the use of clever data structures such as decision diagrams can reduce the runtime significantly in many cases.

Acknowledgment

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

[4]The identity is the best case for decision diagrams due to its recursively equal sub-parts, allowing for a representation linear in the number of qubits.

REFERENCES

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Jour. of Comp.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Symp. on Theory of Computing*, 1996, pp. 212–219.

[3] B. P. Lanyon, J. D. Whitfield, G. G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri *et al.*, "Towards quantum chemistry on a quantum computer," *Nature chemistry*, vol. 2, no. 2, p. 106, 2010.

[4] M. Tillmann, B. Dakić, R. Heilmann, S. Nolte, A. Szameit, and P. Walther, "Experimental boson sampling," *Nature photonics*, vol. 7, no. 7, p. 540, 2013.

[5] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[6] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[7] "IBM Quantum Experience," https://quantum-computing.ibm.com, accessed: 2020-01-22.

[8] J. M. Gambetta, J. M. Chow, and M. Steffen, "Building logical qubits in a superconducting quantum computing system," *npj Quantum Information*, vol. 3, no. 1, p. 2, 2017.

[9] D. Kielpinski, C. Monroe, and D. J. Wineland, "Architecture for a large-scale ion-trap quantum computer," *Nature*, vol. 417, no. 6890, p. 709, 2002.

[10] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 6, pp. 818–830, 2013.

[11] D. M. Miller, R. Wille, and Z. Sasanian, "Elementary quantum gate realizations for multiple-control Toffolli gates," in *Int'l Symp. on Multi-Valued Logic*, 2011, pp. 288–293.

[12] K. Matsumoto and K. Amano, "Representation of quantum circuits with Clifford and $\pi/8$ gates," *arXiv preprint arXiv:0806.3834*, 2008.

[13] R. Wille, M. Soeken, C. Otterstedt, and R. Drechsler, "Improving the mapping of reversible circuits to quantum circuits using multiple target lines," in *Asia and South Pacific Design Automation Conf.*, 2013, pp. 85–92.

[14] M. Siraichi, V. F. Dos Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Int'l Symp. on Code Generation and Optimization*, 2018, pp. 113–125.

[15] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Symp. on Combinatorial Search*, 2018.

[16] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2018.

[17] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Design Automation Conf.*, 2019, p. 142.

[18] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quantum Information Processing*, vol. 10, no. 3, pp. 355–377, 2011.

[19] R. Wille, A. Lye, and R. Drechsler, "Optimal SWAP gate insertion for nearest neighbor quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2014, pp. 489–494.

[20] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler, "Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits," in *Asia and South Pacific Design Automation Conference*, 2016, pp. 292–297.

[21] W. Hattori and S. Yamashita, "Quantum circuit optimization by changing the gate order for 2D nearest neighbor architectures," *Int'l Conf. of Reversible Computation*, pp. 228–243, 2018.

[22] R. Wille, A. Lye, and R. Drechsler, "Exact reordering of circuit lines for nearest neighbor quantum architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1818–1831, 2014.

[23] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: https://doi.org/10.1109/TSSC.1968.300136

[24] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

[25] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler, "SWORD: A SAT like prover using word level information," in *VLSI of System-on-Chip*, 2007, pp. 88–93.

[26] A. Zulehner and R. Wille, "Compiling SU(4) quantum circuits to IBM QX architectures," in *Asia and South Pacific Design Automation Conf.*, 2019, pp. 185–190.

[27] A. Zulehner, H. Bauer, and R. Wille, "Evaluating the flexibility of A* for mapping quantum circuits," in *Workshop on Reversible Computation*, 2019, pp. 171–190.

[28] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Checking equivalence of quantum circuits and states," in *Int'l Conf. on CAD*, 2007, pp. 69–74.

[29] P. Niemann, R. Wille, and R. Drechsler, "Equivalence checking in multi-level quantum systems," in *Int'l Conf. of Reversible Computation*, 2014, pp. 201–215.

[30] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Improving gate-level simulation of quantum circuits," *Quantum Information Processing*, vol. 2, no. 5, pp. 347–380, 2003.

[31] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2016.

[32] A. Abdollahi and M. Pedram, "Analysis and synthesis of quantum circuits by using quantum decision diagrams," in *Design, Automation and Test in Europe*, 2006, pp. 317–322.

[33] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "An XQDD-based verification method for quantum circuits," *IEICE Trans. Fundamentals*, vol. 91-A, no. 2, pp. 584–594, 2008.

[34] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 848–859, 2019.

[35] A. Zulehner and R. Wille, "Matrix-vector vs. matrix-matrix multiplication: Potential in DD-based simulation of quantum computations," in *Design, Automation and Test in Europe*, 2019.

[36] P. Niemann, R. Wille, and R. Drechsler, "On the "Q" in QMDDs: Efficient representation of quantum functionality in the QMDD data-structure," in *Int'l Conf. of Reversible Computation*, 2013, pp. 125–140.

[37] S. Yamashita and I. L. Markov, "Fast equivalence-checking for quantum circuits," in *Int'l Symp. on Nanoscale Architectures*, 2010, pp. 23–28.

[38] L. Burgholzer and R. Wille, "Improved DD-based equivalence checking of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2020.