MUNICH QUANTUM TOOLKIT

The MQT Handbook

A Summary of Design Automation Tools and Software for Quantum Computing

Version 1.0.0

Chair for Design Automation
Technical University of Munich

Apr 18, 2024

Abstract

Quantum computers are becoming a reality. But designing applications for these devices requires automated, efficient, and user-friendly software tools that cater to the needs of end users, engineers, and physicists at every level of the entire quantum software stack. Many of the problems to be tackled in that regard are similar to design problems from the classical realm for which sophisticated design automation tools have been developed in the previous decades. The *Munich Quantum Toolkit (MQT)* is a collection of software tools for quantum computing developed by the Chair for Design Automation at the Technical University of Munich which explicitly utilizes this design automation expertise. Our overarching objective is to provide solutions for design tasks across the entire quantum software stack. This entails high-level support for end users in realizing their *applications* as well as efficient methods for the *classical simulation*, *compilation*, and *verification* of quantum circuits. These methods are supported by corresponding *data structures* (such as decision diagrams) and *core methods* (such as SAT encodings/solvers). Reaching towards the hardware level, we also consider tools for *quantum error correction* and *physical design*. All of the developed tools are available as open source and are hosted on github.com/cda-tum. The following sections provide a brief overview of the provided solutions for various levels in the quantum software stack.

Note: A live version of this document is available at mqt.readthedocs.io.

The MQT Handbook

I Introduction

Quantum computing has the potential to revolutionize many fields in the 21st century. Over the past decade, numerous quantum computers from multiple providers based on different qubit technologies have been made publicly available. However, the best hardware is only as good as the software available to realize corresponding applications on it—a lesson learned from the past decades of research on designing and developing classical circuits and systems. Thanks to the software tools and methods for *Electronic Design Automation (EDA)*, we can create classical systems with a staggering amount of transistors and complex functionalities that we often take for granted. These methods allow designers to efficiently and automatically handle the intricacies of such systems and optimize their performance. Compared to that, most existing software solutions for quantum computing are based on manual approaches. This is not only susceptible to errors, inefficiency, and inconsistency but also leaves decades of research on design automation methods underutilized.

The *Munich Quantum Toolkit (MQT)*, which is developed by the Chair for Design Automation at the Technical University of Munich, aims to leverage this latent potential by providing a collection of state-of-the-art design automation methods and software tools for quantum computing. Our overarching objective is to provide solutions for design tasks across the entire quantum software stack. This entails high-level support for end users in realizing their *applications* as well as efficient methods for the *classical simulation*, *compilation*, and *verification* of quantum circuits. Reaching towards the hardware level, we also consider tools for *quantum error correction* and *physical design*. In all these tools, we try to utilize *data structures and core methods* facilitating the efficient handling of quantum computations. The proposed solutions demonstrate significant improvements in efficiency, scalability, and reliability. They illustrate the immense benefits of leveraging expertise in classical circuit and system design rather than starting from scratch. All tools developed as part of the MQT are made available as open-source packages on github.com/cda-tum.

In the following, we briefly summarize the main tools (covering classical simulation, compilation, and verification of quantum circuits as well as benchmarking). We particularly focus on how to use the tools, but additionally provide references and links that offer detailed descriptions of the underlying methods as well as summaries of corresponding case studies and evaluations demonstrating the benefits.

II Classical Simulation of Quantum Circuits

Performing a quantum computation (commonly described as a quantum circuit) entails evolving an initial quantum state by applying a sequence of operations (also called gates) and measuring the resulting system. Eventually, the goal should obviously be to do that on a real device. However, there are several important reasons for simulating the corresponding computations on a classical machine, particularly in the early stages of the design: As long as no suitable devices are available (e.g., in terms of scale, feasible computation depth, or accuracy), classical simulations of quantum circuits still allow one to explore and test quantum applications, even if only on a limited scale. However, also with further progress in the capabilities of the hardware platforms, classical simulation will remain an essential part of the quantum computing design process, since it additionally allows access to *all* amplitudes of a resulting quantum state in contrast to a real device that only probabilistically returns measurement results. Moreover, classical simulation provides means to study quantum error correction as well as a baseline to estimate the advantage of quantum computers over classical computers.

The classical simulation of quantum circuits is commonly conducted by performing consecutive matrix-vector multiplication, which many simulators realize by storing a dense representation of the complete state vector in memory and evolving it correspondingly (see, e.g., [1, 2, 3, 4, 5]). This approach quickly becomes intractable due to the exponential growth of the quantum state with respect to the number of qubits—quickly rendering such simulations infeasible even on supercomputer clusters. Simulation methodologies based on decision diagrams [6, 7, 8] are a promising complementary approach that frequently allows reducing the required memory by exploiting redundancies in the simulated quantum state.

The *MQT* offers the classical quantum circuit simulator *DDSIM* that can be used to perform various quantum circuit simulation tasks based on using decision diagrams as a data structure. This includes strong and weak simulation, approximation techniques, noise-aware simulation, hybrid Schrödinger-Feynman techniques, support for dynamic circuits, the computation of expectation values, and more [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20].

Example 1. *Consider the following listing that builds the quantum circuit for generating a three-qubit GHZ state:*

```python
from qiskit import QuantumCircuit

circ = QuantumCircuit(3)
circ.h(2)
circ.cx(2, 1)
circ.cx(1, 0)
circ.measure_all()
```

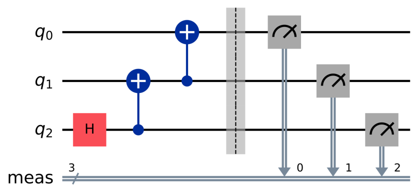

Fig. 1. Quantum circuit for generating a three-qubit GHZ state.

The following listing demonstrates how to simulate this circuit using DDSIM as a backend for IBM Qiskit:

```python
from mqt.ddsim import DDSIMProvider

provider = DDSIMProvider()
backend = provider.get_backend("qasm_simulator")
result = backend.run(circ, shots=10000).result()
result.get_counts()
```

```
{'000': 4989, '111': 5011}
```

MQT DDSIM
Code: https://github.com/cda-tum/mqt-ddsim
Python Package: https://pypi.org/p/mqt.ddsim
Documentation: https://mqt.readthedocs.io/projects/ddsim

III Compilation of Quantum Circuits

In today's digital world, creating computer programs has become a crucial element of software development. With the advent of high-level programming languages such as C++ or Python, the development process has become simpler and more efficient. These languages enable developers to produce code that is more human-readable and understandable without having to worry about the underlying hardware's low-level features. But before these programs can be executed on a computer, they must be translated into machine code that the computer can process. This procedure is known as *compilation*, and it entails converting high-level code into a binary format that the computer's processor can directly execute. By making it easier for more people to create computer programs, this has enabled the development of complex software applications that can run on many different platforms such as desktops, laptops, mobile phones or embedded devices.

Just as in classical computing, the design of quantum circuits and the development of quantum algorithms are fundamental in the development of quantum computing applications. Quantum circuits are analogous to classical functions or programs in that they are a sequence of quantum gates that perform specific operations on quantum bits or qubits instead of classical bits. Similarly to classical processors, quantum processors can only execute a certain set of native instructions, and they might further limit the qubits on which these operations might be applied. Thus, any high-level quantum circuit (describing a quantum application) must be *compiled* into a representation that can be executed on the targeted device. Most importantly, the resulting quantum circuit must only use gates that are native to the device on which it shall be executed. If the device only has limited connectivity between its qubits, it must only apply gates to qubits that are connected on the device. Naturally, the efficiency of this compilation process is critical because it can have a significant impact on the performance of the resulting quantum program. Inefficient compilation can lead to longer execution times, higher error rates, and reduced accuracy in the final result. Therefore, developing efficient compilation methods for quantum programs is essential to overcome the challenges of quantum computing and realize the potential of this technology.

In the following, we mainly focus on the *quantum circuit mapping* task. This is a crucial step in the compilation flow, as it directly affects the feasibility and performance of the quantum circuit on a given device. It involves finding a way to map the qubits of a quantum circuit to the qubits of a quantum device, while respecting the limited connectivity constraints of the device and minimizing the overhead of additional gates. In most cases, it is not possible to statically define a mapping of the circuit's qubits to the device's qubits such that all gates of the circuit conform to the connectivity limitations of the device. Consequently, this mapping has to change dynamically throughout the circuit. This can be accomplished by using *SWAP* gates that allow the position of two logical qubits on the architecture to be interchanged. However, since any additional gate increases the error rate and, hence, reduces the accuracy of the computation, it is vital to keep the number of additionally added gates as low as possible. It has been shown that even this small part in the compilation flow is an NP-complete problem [21].

The *MQT* offers the quantum circuit mapping tool QMAP that allows one to generate circuits which satisfy all constraints given by the targeted architecture and, at the same time, keep the overhead in terms of additionally required quantum gates as low as possible. More precisely, different approaches based on design automation techniques are provided, which are generic and can be easily configured for future architectures. Among them is a heuristic, scalable solution for arbitrary circuits based on informed-search algorithms [22, 23] as well as a solution for obtaining mappings ensuring minimal overhead with respect to SWAP gate insertions [24, 25].

MQT offers many more methods for various compilation tasks, such as Clifford circuit synthesis [26, 27], determining optimal sub-architectures [28], compiler optimization [29], or compilation techniques for different architectures [30, 31, 32, 33, 34].

Example 2. *Assume we want to perform the computation from Fig. 1 on a five-qubit IBM quantum computer described by the coupling map shown in Fig. 2.*

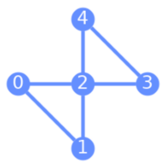

Fig. 2. Generic five-qubit IBM device.

Then, mapping the circuit to that device merely requires the following lines of Python and results in the circuit shown in Fig. 3.

```python
from mqt.qmap import compile
from qiskit.providers.fake_provider import Fake5QV1

backend = Fake5QV1()
circ_mapped, results = compile(circ, backend)
```

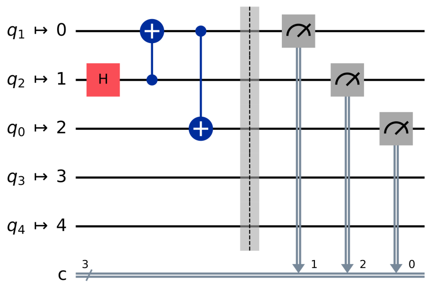

Fig. 3. Quantum circuit from Fig. 1 mapped to the five-qubit device shown in Fig. 2.

MQT QMAP

Code: https://github.com/cda-tum/mqt-qmap

Python Package: https://pypi.org/p/mqt.qmap

Documentation: https://mqt.readthedocs.io/projects/qmap

IV 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. As described in the previous section, even individual compilation tasks can be highly complex. Consequently, checking whether the original functionality is indeed maintained throughout all these different abstractions becomes increasingly relevant in order to guarantee a consistent and error-free compilation flow. This is similar to the classical realm, where descriptions at various levels of abstraction also exist. These descriptions are verified using design automation expertise—resulting in efficient methods for verification to ensure the correctness of the design across different levels of abstraction [35]. However, since quantum circuits additionally employ quantum-physical effects such as superposition and entanglement, these methods cannot be used out of the box in the quantum realm. Accordingly, verification of quantum circuits must be approached from a different perspective. At first glance, these characteristics of quantum computing make verification much harder as for classical circuits and systems. In fact, equivalence checking of quantum circuits has been proven to be a computationally hard problem [36].

At the same time, quantum circuits possess certain characteristics that offer remarkable potential for efficient equivalence checking that is not available in classical computing. More precisely, consider two quantum circuits $G = g_1, \ldots, g_m$ and $G' = g'_1, \ldots, g'_n$ whose equivalence shall be checked. Due to the inherent reversibility of quantum operations, the inverse of a quantum circuit can easily be computed by taking the complex conjugate of every gate and reversing the sequence of the gates in the circuit, i.e., $G'^{-1} = (g'_n)^\dagger, \ldots, (g'_1)^\dagger$. If two circuits are equivalent, this allows for the conclusion that $G \cdot G'^{-1} = I$, where I is the identity function. Since the identity has the most compact representation for most data structures representing quantum functionality (e.g., linear with respect to the number of qubits in case of decision diagrams), the equivalence check can be simplified considerably. Even complex circuits can be verified efficiently, if one manages to apply the gates of both circuits in a sequence that keeps the intermediate representation "close to the identity". Within the MQT, several methods and strategies were proposed that utilize this characteristic of quantum computations. Eventually, this led to solutions that can verify the results of whole quantum compilation flows (such as IBM's Qiskit) in negligible runtime—something we never even managed for classical circuits and systems.

The *MQT* offers the quantum circuit equivalence checking tool QCEC which encompasses a comprehensive suite of efficient methods and automated tools for the verification of quantum circuits based on the ideas outlined in [37, 38, 39, 40, 41, 42, 43, 44, 45]. By this, an important step towards avoiding or substantially mitigating the emerge of a verification gap for quantum circuits is taken, i.e., a situation where the physical development of a technology substantially outperforms our ability to design suitable applications for it or to verify it.

Example 3. *Verifying that the quantum circuit from Fig. 3 has been correctly compiled to the architecture from Fig. 2, i.e., whether it still implements the functionality of the circuit shown in Fig. 1, merely requires the following lines of Python:*

```python
from mqt.qcec import verify

result = verify(circ, circ_mapped)
print(result.equivalence)
```

```
equivalent
```

MQT QCEC
Code: https://github.com/cda-tum/mqt-qcec
Python Package: https://pypi.org/p/mqt.qcec
Documentation: https://mqt.readthedocs.io/projects/qcec

V Benchmarking Software and Design Automation Tools for Quantum Computing

Tools like the ones proposed above are key in order to support end users in the realization of their quantum computing applications. And, thankfully, a huge variety of tools has been proposed in the past—with many more to come. However, whenever such a quantum software tool is proposed, it is important to empirically evaluate its performance and to compare it to the state of the art. For that purpose, proper benchmarks are needed. To provide those, MQT Bench is proposed, which offers over $70,000$ benchmarks on various abstraction levels (depending on what level the to-be-evaluated software tool operates on). Having all those benchmarks in a single repository enables an increased comparability, reproducibility, and transparency. To make the benchmarks as accessible as possible, MQT Bench comes as an easy-to-use website that is hosted at www.cda.cit.tum.de/mqtbench/ and as a Python package available on PyPI.

Example 4. *A larger version of the quantum circuit from Fig. 1 can easily be obtained programmatically from the MQT Bench Python package as follows:*

```
from mqt.bench import get_benchmark

circ = get_benchmark("ghz", circuit_size=8, level="alg")
```

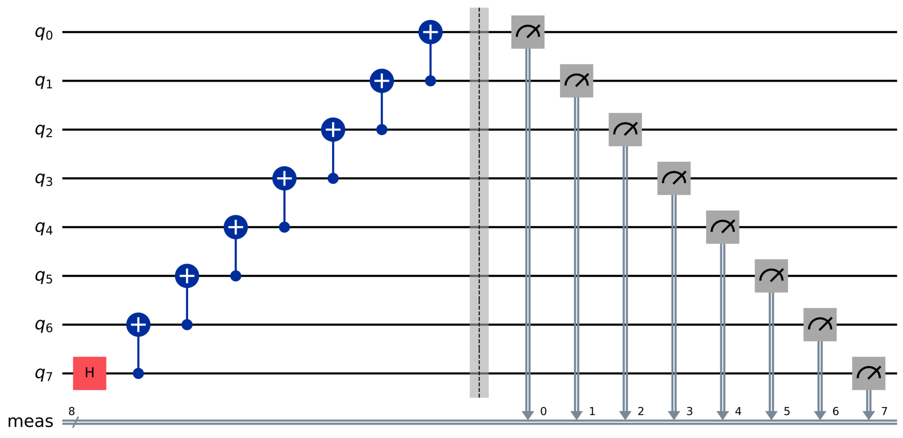

Fig. 4. Larger version of the circuit from Fig. 1 obtained via MQT Bench.

This circuit can then be used to evaluate the performance of a quantum software tool, e.g., to test how well it can simulate the circuit or how well it can compile it to a given architecture.

MQT Bench
Code: https://github.com/cda-tum/mqt-bench
Python Package: https://pypi.org/p/mqt.bench
Documentation: https://mqt.readthedocs.io/projects/bench

VI Open-Source Implementations

All tools that have been developed as part of the *MQT* are publicly available on github.com/cda-tum. Many of these tools are powered by MQT Core, which forms the backbone of the entire toolkit. It features a comprehensive intermediate representation for quantum computations as well as a state-of-the-art decision diagram package for quantum computing and a high-performance ZX-calculus library.

MQT Core
Code: https://github.com/cda-tum/mqt-core
Python Package: https://pypi.org/p/mqt.core
Documentation: https://mqt.readthedocs.io/projects/core

All tools have been mainly implemented in C++, but strive to be as user-friendly as possible for the community. Hence, push-button solutions are provided through Python bindings, pre-built Python wheels are available for all major platforms and Python versions, and all tools integrate natively with IBM's Qiskit. All tools are actively maintained and well documented.

VII Conclusions

Design automation tools and software have been crucial for the development of classical circuits and systems. They enable faster and more reliable design cycles, reduce human errors, and allow for complex and large-scale designs. In the domain of quantum computing, the corresponding design automation methods (which have been developed over the past decades) remain heavily underutilized. The *Munich Quantum Toolkit (MQT)* makes substantial contributions towards leveraging this latent potential. For many important design tasks, several methods and tools have been proposed that explicitly use design automation expertise while, at the same time, considering characteristics of quantum computing.

Overall, the *MQT* demonstrates the immense benefits of leveraging existing knowledge and expertise in classical circuit and system design for the development of quantum software. By applying design automation methods to various tasks, the MQT significantly advances the state of the art in quantum computing and improves the efficiency, scalability, and reliability of quantum software solutions. In the absence of appropriate tools and software, the potential of quantum computers may remain untapped despite their immense computational capabilities. The *MQT* highlights the importance of design automation methods in shaping the future of quantum computing.

Acknowledgments

We thank everyone that contributed to the development of the *Munich Quantum Toolkit*. Special thanks go to Alwin Zulehner, Stefan Hillmich, Thomas Grurl, Hartwig Bauer, Sarah Schneider, Smaran Adarsh, and Alexander Ploier for their specific contributions in the past.

The Munich Quantum Toolkit has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001318), the Bavarian State Ministry for Science and Arts through the Distinguished Professorship Program, as well as the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus.

References

[1] Thomas Häner and Damian S. Steiger. 0.5 petabyte simulation of a 45-Qubit quantum circuit. In *Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*. 2017. doi:10.1145/3126908.3126947.

[2] Jun Doi, Hitomi Takahashi, Rudy Raymond, Takashi Imamichi, and Hiroshi Horii. Quantum computing simulator on a heterogenous HPC system. In *Int'l Conf. on Computing Frontiers*, 85–93. 2019. doi:10.1145/3310273.3323053.

[3] Tyson Jones, Anna Brown, Ian Bush, and Simon C. Benjamin. QuEST and high performance simulation of quantum computers. In *Scientific Reports*. 2018. doi:10.1038/s41598-019-47174-9.

[4] Gian Giacomo Guerreschi, Justin Hogaboam, Fabio Baruffa, and Nicolas P D Sawaya. Intel Quantum Simulator: a cloud-ready high-performance simulator of quantum circuits. *Quantum Science and Technology*, 5(3):034007, 2020. doi:10.1088/2058-9565/ab8505.

[5] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. Full-state quantum circuit simulation by using data compression. In *Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*. 2019. doi:10.1145/3295500.3356155.

[6] George F. Viamontes, Igor L. Markov, and John P. Hayes. Improving gate-level simulation of quantum circuits. *Quantum Information Processing*, 2(5):347–380, 2003. doi:10.1023/B:QINP.0000022725.70000.4a.

[7] Robert Wille, Stefan Hillmich, and Lukas Burgholzer. Decision Diagrams for Quantum Computing. In *Design Automation of Quantum Computers*. 2023. doi:10.1007/978-3-031-15699-1_1.

[8] Alwin Zulehner and Robert Wille. Advanced simulation of quantum computations. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD)*, 2019. doi:10.1109/TCAD.2018.2834427.

[9] Alwin Zulehner and Robert Wille. Matrix-Vector vs. Matrix-Matrix multiplication: Potential in DD-based simulation of quantum computations. In *Design, Automation and Test in Europe (DATE)*. 2019. doi:10.23919/DATE.2019.8714836.

[10] Stefan Hillmich, Igor L. Markov, and Robert Wille. Just like the real thing: Fast weak simulation of quantum computation. In *Design Automation Conference (DAC)*. 2020. doi:10.1109/DAC18072.2020.9218555.

[11] Stefan Hillmich, Richard Kueng, Igor L. Markov, and Robert Wille. As accurate as needed, as efficient as possible: Approximations in DD-based quantum circuit simulation. In *Design, Automation and Test in Europe (DATE)*. 2020. doi:10.23919/DATE51398.2021.9474034.

[12] Stefan Hillmich, Alwin Zulehner, and Robert Wille. Concurrency in DD-based quantum circuit simulation. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2020. doi:10.1109/ASP-DAC47756.2020.9045711.

[13] Stefan Hillmich, Alwin Zulehner, Richard Kueng, Igor L. Markov, and Robert Wille. Approximating decision diagrams for quantum circuit simulation. *ACM Transactions on Quantum Computing*, 3(4):1–21, 2022. doi:10.1145/3530776.

[14] Thomas Grurl, Jürgen Fuß, and Robert Wille. Considering decoherence errors in the simulation of quantum circuits using decision diagrams. In *International Conference on Computer Aided Design (ICCAD)*. 2020. doi:10.1145/3400302.3415622.

[15] Thomas Grurl, Richard Kueng, Jürgen Fuß, and Robert Wille. Stochastic quantum circuit simulation using decision diagrams. In *Design, Automation and Test in Europe (DATE)*. 2021. doi:10.23919/DATE51398.2021.9474135.

[16] Thomas Grurl, Jurgen Fuß, and Robert Wille. Noise-aware quantum circuit simulation with decision diagrams. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD)*, 42(3):860–873, 2023. doi:10.1109/TCAD.2022.3182628.

[17] Lukas Burgholzer, Hartwig Bauer, and Robert Wille. Hybrid Schrödinger-Feynman simulation of quantum circuits with decision diagrams. In *Int'l Conf. on Quantum Computing and Engineering*. 2021. doi:10.1109/QCE52317.2021.00037.

[18] Lukas Burgholzer, Alexander Ploier, and Robert Wille. Exploiting arbitrary paths for the simulation of quantum circuits with decision diagrams. In *Design, Automation and Test in Europe (DATE)*. 2022. doi:10.23919/DATE54114.2022.9774631.

[19] Lukas Burgholzer, Alexander Ploier, and Robert Wille. Simulation paths for quantum circuit simulation with decision diagrams: What to learn from tensor networks, and what not. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD)*, 2022. arXiv:2203.00703, doi:10.1109/TCAD.2022.3197969.

[20] Lukas Burgholzer, Rudy Raymond, Indranil Sengupta, and Robert Wille. Efficient construction of functional representations for quantum algorithms. In *Conference on Reversible Computation (RC)*. 2021. doi:10.1007/978-3-030-79837-6_14.

[21] A. Botea, A. Kishimoto, and Radu Marinescu. On the complexity of quantum circuit compilation. In *Int'l Symp. on Combinatorial Search*. 2018. doi:10.1609/socs.v9i1.18463.

[22] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD)*, 2019. doi:10.1109/TCAD.2018.2846658.

[23] Stefan Hillmich, Alwin Zulehner, and Robert Wille. Exploiting Quantum Teleportation in Quantum Circuit Mapping. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 792–797. 2021. doi:10.1145/3394885.3431604.

[24] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Design Automation Conference (DAC)*. 2019. doi:10.1145/3316781.3317859.

[25] Lukas Burgholzer, Sarah Schneider, and Robert Wille. Limiting the search space in optimal quantum circuit mapping. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2022. doi:10.1109/ASP-DAC52403.2022.9712555.

[26] Tom Peham, Nina Brandl, Richard Kueng, Robert Wille, and Lukas Burgholzer. Depth-optimal synthesis of Clifford circuits with SAT solvers. In *IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2023. arXiv:2305.01674, doi:10.1109/QCE57702.2023.00095.

[27] Sarah Schneider, Lukas Burgholzer, and Robert Wille. A SAT encoding for optimal Clifford circuit synthesis. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2023. doi:10.1145/3566097.3567929.

[28] Tom Peham, Lukas Burgholzer, and Robert Wille. On Optimal Subarchitectures for Quantum Circuit Mapping. *ACM Transactions on Quantum Computing*, 2023. arXiv:2210.09321, doi:10.1145/3593594.

[29] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. Compiler Optimization for Quantum Computing Using Reinforcement Learning. In *Design Automation Conference (DAC)*. 2023. arXiv:2212.04508, doi:10.1109/DAC56929.2023.10248002.

[30] Daniel Schoenberger, Stefan Hillmich, Matthias Brandl, and Robert Wille. Using Boolean Satisfiability for Exact Shuttling in Trapped-Ion Quantum Computers. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2024. doi:10.1109/ASP-DAC58780.2024.10473902.

[31] Daniel Schoenberger, Stefan Hillmich, Matthias Brandl, and Robert Wille. Towards Cycle-based Shuttling for Trapped-Ion Quantum Computers. In *Design, Automation and Test in Europe (DATE)*. 2024.

[32] Daniel Schoenberger, Stefan Hillmich, Matthias Brandl, and Robert Wille. Shuttling for Scalable Trapped-Ion Quantum Computers. 2024. arXiv:2402.14065.

[33] Ludwig Schmid, Sunghey Park, and Robert Wille. Hybrid Circuit Mapping: Leveraging the Full Spectrum of Computational Capabilities of Neutral Atom Quantum Computers. In *Design Automation Conference (DAC)*. 2024.

[34] Ludwig Schmid, David Locher, Manuel Rispler, Sebastian Blatt, Johannes Zeiher, Markus Müller, and Robert Wille. Computational Capabilities and Compiler Development for Neutral Atom Quantum Processors - Connecting Tool Developers and Hardware Experts. *Quantum Science and Technology*, 2024. doi:10.1088/2058-9565/ad33ac.

[35] Rolf Drechsler. *Advanced Formal Verification*. Springer, 2004. doi:10.5555/1024203.

[36] Dominik Janzing, Pawel Wocjan, and Thomas Beth. “Non-identity check” is QMA-complete. *International Journal of Quantum Information*, 03(03):463–473, 2005. doi:10.1142/S0219749905001067.

[37] Lukas Burgholzer and Robert Wille. Advanced equivalence checking for quantum circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD)*, 2021. doi:10.1109/TCAD.2020.3032630.

[38] Lukas Burgholzer and Robert Wille. Improved DD-based equivalence checking of quantum circuits. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2020. doi:10.1109/ASP-DAC47756.2020.9045153.

[39] Lukas Burgholzer and Robert Wille. The power of simulation for equivalence checking in quantum computing. In *Design Automation Conference (DAC)*. 2020. doi:10.1109/DAC18072.2020.9218563.

[40] Lukas Burgholzer, Richard Kueng, and Robert Wille. Random stimuli generation for the verification of quantum circuits. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2021. doi:10.1145/3394885.3431590.

[41] Lukas Burgholzer, Rudy Raymond, and Robert Wille. Verifying results of the IBM Qiskit quantum circuit compilation flow. In *Int'l Conf. on Quantum Computing and Engineering*. 2020. doi:10.1109/QCE49297.2020.00051.

[42] Tom Peham, Lukas Burgholzer, and Robert Wille. Equivalence checking of parameterized quantum circuits: Verifying the compilation of variational quantum algorithms. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2023. doi:10.1145/3566097.3567932.

[43] Tom Peham, Lukas Burgholzer, and Robert Wille. Equivalence checking of quantum circuits with the ZX-Calculus. *jetcas*, 2022. doi:10.1109/JETCAS.2022.3202204.

[44] Tom Peham, Lukas Burgholzer, and Robert Wille. Equivalence checking paradigms in quantum circuit design: A case study. In *Design Automation Conference (DAC)*. 2022. doi:10.1145/3489517.3530480.

[45] Robert Wille and Lukas Burgholzer. Verification of Quantum Circuits. In Anupam Chattopadhyay, editor, *Handbook of Computer Architecture*, pages 1–28. Springer Nature Singapore, Singapore, 2022. doi:10.1007/978-981-15-6401-7_43-1.