Verifying Results of Quantum Circuit Compilation Flows

Lukas Burgholzer[*] Robert Wille[*†]
[*]Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[†]Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria
lukas.burgholzer@jku.at robert.wille@jku.at
https://iic.jku.at/eda/research/quantum/

Quantum computers aim to change the way we tackle certain problems in the future. Numerous quantum computing applications with a near-term perspective (e.g., for finance, chemistry, machine learning, optimization) and with a long-term perspective (i.e., cryptography, database search) are currently investigated. However, in order to realize those, a multitude of (computationally complex) design tasks have to be conducted—eventually forming a process called *quantum circuit compilation*. This results in descriptions of quantum algorithms at various abstraction levels which may significantly differ in their basis operations and structure. This is similar to the classical realm where, e.g., descriptions at the *Electronic System Level*, the *Register Transfer Level*, and the *Gate Level* exist. All this substantially changes the circuit description during the design flow. At the same time, all these steps should obviously preserve the originally intended functionality of the quantum circuit. In order to verify whether two quantum circuits indeed realize the same functionality, *equivalence checking* is usually conducted. In this brief summary paper, we provide an overview on how equivalence checking can be conducted for quantum circuits and, by this, how the results of corresponding compilation flows can be verified.

Quantum Circuit Compilation

In the first part of this summary paper, we review the compilation flow as it got established in the recent years. Initially, quantum algorithms (often provided in terms of a *quantum circuit*) are described in a way which is agnostic of the device they are planned to be executed on. Similar to the conventional realm, where a compiler transforms high-level code into (machine-executable) assembly, a conceptual quantum algorithm needs to be *compiled* to a representation that conforms to all the restrictions imposed by the targeted device. This is commonly done in three steps:

1) *Synthesis:* Quantum computers built today typically only feature a limited (but universal) set of supported operations (called gates). Usually, this gate set consists of arbitrary single-qubit gates and a certain two-qubit gate (such as the CNOT). Thus, the gates of the original quantum circuit first have to be *synthesized* to the gate-set supported by the targeted device. Most importantly, since devices typically only support up to two-qubit gates, any gate acting on more than two qubits has to be decomposed into "elementary" gates.
2) *Mapping:* After the first step, the circuit just contains elementary gates supported by the device. However, current quantum computers (at least those based on superconducting qubits) only feature a rather limited connectivity between their physical qubits—only allowing two-qubit gates to be applied to certain pairs of qubits. Thus, the quantum circuit has to be *mapped* to the target architecture so that a mapping between the circuit's *logical* and the device's *physical* qubits is established which satisfies all connectivity constraints. In most cases, it is not possible to define such a mapping which conforms to *all* connectivity limitations globally. In these cases, the logical-to-physical qubit mapping is changed dynamically throughout the circuit by adding so-called $SWAP$ gates into the circuit—effectively allowing to change the mapping of logical qubits to physical qubits so that all operations can be executed while, at the same time, all connectivity constraints are satisfied.
3) *Optimization:* After the previous step, circuits are ready to be executed on the targeted devices. However, the previous steps significantly increased the size of these circuits. Since today's physical qubits are inherently affected by noise—leading to rather short coherence times and limited fidelity of the individual operations—and error correction is not yet an option, this severely impacts the achievable performance. Thus, several *optimizations* may be employed to reduce the circuit's size and, hence, improve the actual performance on the quantum computer.

Example 1. *An example of a quantum circuit G with 16 gates acting on three qubits is shown in Fig. 1a. This sequence of operations describes a small instance of the famous* Grover *search algorithm [1]. The small boxes with identifiers correspond to operations applied to single qubits such as X and H gates. Moreover, there are* multiple-controlled *X operations, where an X operation is only applied to a* target *qubit (denoted by $\oplus$) if all of its* control *qubits (denoted by $\bullet$) are in state $|1\rangle$. An exemplary device architecture (that of the IBMQ London quantum computer) is shown in Fig. 1c. It consists of five (physical) qubits and supports arbitrary single-qubit gates, while a $CNOT$ operation may only be applied to connected qubits. If G shall be executed on this system, the Toffoli gate (the two-controlled NOT) first has to be realized using only the supported elementary gates. One possible synthesized version is shown in Fig. 1b. Mapping the resulting circuit to the architecture might result in a circuit $\tilde{G}$ as shown in Fig. 1d—using only a single $SWAP$ operation. Applying several optimizations to $\tilde{G}$ allows to further eliminate nine single-qubit gates and results in the* optimized *circuit G' shown in Fig. 1e.*

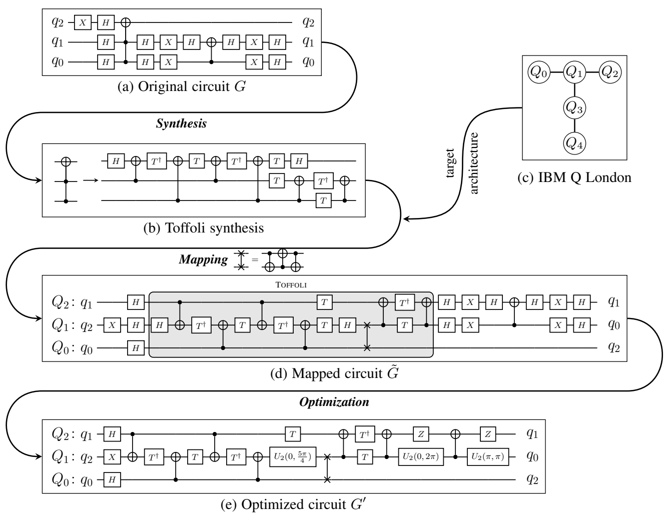

Figure 1: Exemplary illustration of the IBM Qiskit compilation flow

Verification of Quantum Circuits

Naturally, it is of utmost importance that the results of such compilation flows are correct, i.e., that the compiled quantum circuit still realizes the originally intended functionality. This motivates the development of methods for *verification* or, more precisely, *equivalence checking* of quantum circuits.

Checking the equivalence between two quantum circuits boils down to checking whether they indeed realize the same functionality. Mathematically, the quantum gates g_i of an n-qubit quantum circuit G are defined by $2^n \times 2^n$ unitary matrices U_i. Consequently, the functionality of a quantum circuit G with gates $g_0, \ldots, g_{m-1}$ is described by a unitary matrix U, which is obtained by consecutively multiplying the unitary matrix representations U_i of each gate g_i in reverse order, i.e., $U = U_{m-1} \cdots U_0$. Thus, checking the equivalence of two circuits G and G' amounts to building and comparing the circuits' system matrices U and U'. While conceptually simple, the exponential size of the involved matrices quickly renders many direct approaches infeasible. In fact, equivalence checking of quantum circuits has even been proven to be QMA-complete in the general case [2].

In the recent past, some methods addressing this problem have been proposed [3]–[8]. In the following, we review approaches (and a resulting verification flow) introduced in [3] which rests on the following observations:

- Quantum circuits are inherently reversible. Because of that, 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 $\mathbb{I}$, i.e., $G \cdot G'^{-1} = \mathbb{I}$. Hence, checking whether G and G' are in fact equivalent can be conducted by starting with the identity and, then, applying operations of G and (inverted) operations of G' in a particular order until all operations have been applied (written as $G \rightarrow \mathbb{I} \leftarrow G'$). Whenever the final result again resembles the identity, the circuits are considered equivalent. If all the operations can be applied so that the respective intermediate computations remain as close to the identity as possible, substantial improvements can be achieved since the identity constitutes the best case for most representations of quantum functionality (e.g., linear in the number of qubits for decision diagrams). However, how to determine the "perfect" order of applications in order to keep the respective intermediate representation close to the identity, remains a notoriously difficult task. It has been shown in [9], that utilizing knowledge about the compilation flow allows to design a strategy which keeps the respectively occurring intermediate representations close to the identity in an almost perfect fashion. By this, the exponential complexity is frequently reduced to a linear or close-to-linear complexity—substantially reducing the runtime for verifying the results of compilation flows.
- Moreover, even in the case where the two considered quantum circuits are not equivalent, quantum characteristics can be exploited. In fact, due to the inherent reversibility of quantum operations, even small differences in quantum circuits frequently affect the *entire* functional representation. Hence, it may not always be necessary to check the complete functionality, but it is highly likely that the simulation of both computations with a couple of arbitrary input states (i.e., considering only a small part of the whole functionality) will already provide a counterexample showing the non-equivalence. This is in stark contrast to the classical realm, where the inevitable information loss introduced by many logic gates and the resulting masking effects often require a complete consideration of *all* possible input states or sophisticated schemes for constraint-based stimuli generation, fuzzing, etc. In [10], several different (random) stimuli generation schemes have been proposed that offer a trade-off between error-detection rate and efficiency.

These observations complement each other in many different ways. Trying to keep $G \rightarrow \mathbb{I} \leftarrow G'$ close to the identity proves very efficient in case two circuits are indeed equivalent—provided a "good" strategy can be employed. Conducting simulations with appropriately-chosen stimuli on the other hand allows to quickly detect non-equivalence even in cases where both circuits only differ slightly. Combining both ideas naturally leads to an equivalence checking flow as illustrated in Fig. 2.

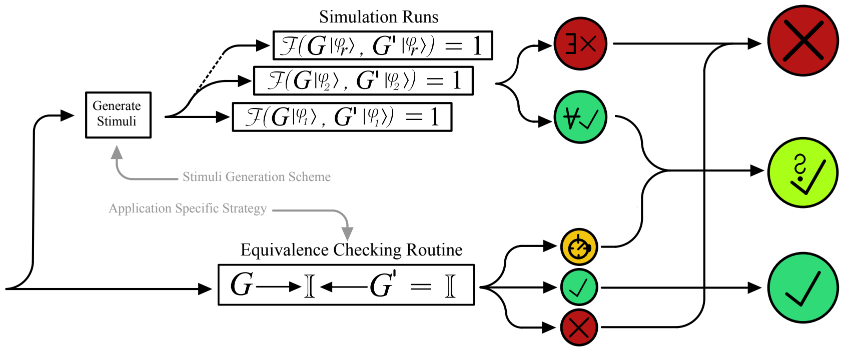
Simulation Runs
$\mathcal{F}(G|\varphi_r\rangle, G'|\varphi_r\rangle) = 1$
$\mathcal{F}(G|\varphi_2\rangle, G'|\varphi_2\rangle) = 1$
$\mathcal{F}(G|\varphi_1\rangle, G'|\varphi_1\rangle) = 1$
Generate Stimuli
Stimuli Generation Scheme
Application Specific Strategy
Equivalence Checking Routine
$G \rightarrow \mathbb{I} \leftarrow G' = \mathbb{I}$
∃✗
∀✓

Figure 2: Equivalence checking flow

Here, a limited number of $r \ll 2^n$ simulation runs with stimuli chosen according to some generation scheme (e.g., random computational basis states) is started in parallel with the $G \rightarrow \mathbb{I} \leftarrow G'$ equivalence checking routine according to some strategy (e.g., a strategy tailored towards verifying compilation flow results). Should any of these simulations (with stimulus $|\varphi_i\rangle$) yield different outputs in both circuits (i.e., a fidelity $\mathcal{F}(G\,|\varphi_i\rangle\,, G'\,|\varphi_i\rangle) \neq 1$) or should the equivalence checking routine be able to determine a final result not resembling the identity, the non-equivalence of the circuits under consideration has been shown and all other executions can be aborted. On the other hand, the equivalence checking routine either manages to establish whether both circuits are equivalent or, in case it times out, leaves an indication (although no proof) that the circuits are likely to be equivalent, due to the fact that even small errors frequently effect the entire functionality.

QCEC - A tool for Quantum Circuit Equivalence Checking

The methodology for verifying the results of quantum circuit compilation flows described above is available as an open-source software package called *QCEC* (available at https://github.com/iic-jku/qcec). It is mainly developed in C++, runs under any major operating system, and also provides Python bindings (and native integration with IBM Qiskit) in order to be as accessible as possible to its community. After getting the tool using `pip install jkq.qcec`, verifying that a quantum circuit has been compiled correctly by IBM Qiskit merely requires the following lines of Python:

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

# create your quantum circuit and append measurements to save output mapping
qc = <...>
qc.measure_all()

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

# verify the compilation result
result = verify(qc, qc_comp, strategy="compilationflow")
```

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101001318). It has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria as well as by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proc. of the ACM*, 1996.

[2] D. Janzing, P. Wocjan, and T. Beth, ""Non-identity check" is QMA-complete," *Int. J. Quantum Inform.*, 2005.

[3] L. Burgholzer and R. Wille, "Advanced equivalence checking for quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2021.

[4] R. Duncan *et al.* "Graph-theoretic simplification of quantum circuits with the ZX-calculus." arXiv: 1902.03178. (2019).

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

[6] S.-A. Wang *et al.*, "An XQDD-based verification method for quantum circuits," in *IEICE Trans. Fundamentals*, 2008, pp. 584–594.

[7] P. Niemann *et al.*, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2016.

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

[9] L. Burgholzer, R. Raymond, and R. Wille, "Verifying results of the IBM Qiskit quantum circuit compilation flow," in *Int'l Conf. on Quantum Computing and Engineering*, 2020.

[10] L. Burgholzer, R. Kueng, and R. Wille, "Random stimuli generation for the verification of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2021.