QDMI – Quantum Device Management Interface: Hardware-Software Interface for the Munich Quantum Software Stack

Robert Wille*†, Ludwig Schmid*, Yannick Stade*, Jorge Echavarria‡, Martin Schulz‡§, Laura Schulz‡, Lukas Burgholzer*
*Chair for Design Automation, Technical University of Munich, Munich, Germany
†Software Competence Center Hagenberg GmbH, Hagenberg, Austria
‡Leibniz Supercomputing Centre, Garching, Germany
§Chair of Computer Architecture and Parallel Systems, Technical University of Munich, Munich, Germany
{robert.wille, ludwig.s.schmid, yannick.stade, martin.w.j.schulz, lukas.burgholzer}@tum.de
{laura.schulz, jorge.echavarria}@lrz.de

***Abstract*—Quantum computing is a promising technology that requires a sophisticated software stack to connect end users to the wide range of possible quantum backends. However, current software tools are usually hard-coded for single platforms and lack a dynamic interface that can automatically retrieve and adapt to changing physical characteristics and constraints of different platforms. With new hardware platforms frequently introduced and their performance changing on a daily basis, this constitutes a serious limitation. In this paper, we showcase a concept and a prototypical realization of an interface, called the *Quantum Device Management Interface* (QDMI), that addresses this problem by explicitly connecting the software and hardware developers, mediating between their competing interests. QDMI allows hardware platforms to provide their physical characteristics in a standardized way, and software tools to query that data to guide the compilation process accordingly. This enables software tools to automatically adapt to different platforms and to optimize the compilation process for the specific hardware constraints. QDMI is a central part of the Munich Quantum Software Stack (MQSS)—a sophisticated software stack to connect end users to the wide range of possible quantum backends. QDMI is publicly available as open source at https://github.com/Munich-Quantum-Software-Stack/QDMI.**

I. Motivation

Quantum utility—the ability to solve useful problems with quantum computing—crucially depends on the quality of the quantum software stack used to realize potential applications. Such a stack consists of various layers of software tools and must be able to connect the end users (usually domain experts from the respective application areas such as material simulation, machine learning or optimization) with the wide range of quantum backends developed by physical experimentalists across a broad spectrum of technologies, including superconducting qubits, ion traps, and neutral atoms.

In general, these software tools act as *compilers* between levels of abstraction. A good quantum compiler should produce an implementation that is executable on and optimized for the targeted quantum platform. This requires upfront and direct communication of the *constraints* that need to be satisfied and the *objective functions* that should be optimized for. In a spirit of rapid prototyping and exploration, the quantum computing community has settled on rather simplistic and abstract constraints and objectives in the past. So far, the main constraints that have been considered were the limited gate set and the limited connectivity of existing quantum platforms—resulting in the de facto standard optimization criterion that aimed to minimize the number of gates in the resulting circuit (e.g., see [1]–[6]). However, pushing the boundaries further towards practical applications of quantum computing requires far more sophisticated considerations due to the diversity and complexity of quantum devices and algorithms. Different quantum devices have different architectures, gate sets, error rates, topology, calibration, and noise models or provide fundamentally different operational capabilities such as qubit shuttling [7], [8].

Different quantum algorithms have different requirements, objectives, and trade-offs [9]–[12]. In addition, these factors can vary over time and depend on the environmental conditions as well as the state of the device. This needs a way to enable efficient communication and optimization between quantum compilers and quantum devices that encapsulates and reflects the knowledge base of the people developing said software and hardware. After all, quantum computers are likely to be used as accelerators for classical computing platforms and, hence, need to be tightly integrated into the rest of the ecosystem and workflows [13]. Such a communication and optimization process would require a common language and a standardized interface that both parties can understand and use. This would allow the people developing software tools to query relevant information and feedback about devices, and the people developing the hardware to provide guidance, express limitations, and offer suggestions in a standardized and automated machine-readable form.

In this paper, we showcase the *Quantum Device Management Interface* (QDMI) as a central part of the *Munich Quantum Software Stack* (MQSS) that addresses this problem. The MQSS is a project of the *Munich Quantum Valley* (MQV) initiative and is jointly developed by the *Leibniz Supercomputing Centre* (LRZ) and the *Chair for Design Automation* (CDA) as well as the *Chair of Computer Architecture and Parallel Systems* (CAPS) at the *Technical University of Munich* (TUM). It provides a comprehensive compilation and runtime infrastructure for on-premise and remote quantum devices, support for modern compilation and optimization techniques, and enables both current and future high-level abstractions for quantum programming. This stack is designed to be capable of deployment in a variety of scenarios via flexible configuration options, including stand-alone scenarios for individual systems, cloud access to a variety of backends as well as tight integration into HPC environments supporting quantum acceleration. Within the MQV, a concrete instance of the MQSS is deployed at the LRZ for the MQV, serving as a single access point to all of its quantum devices via multiple compatible access paths, including a web portal, command line access via web credentials as well as the option for hybrid access with tight integration with LRZ's HPC systems. It facilitates the connection between end-users and quantum computing platforms by its integration within HPC infrastructures, such as those found at the LRZ.

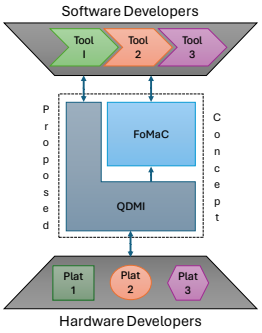

Fig. 1: Illustration of the proposed concept

II. The Proposed Concept

Fig. 1 illustrates the proposed concept to bridge the gap between quantum software developers and hardware experts based on a standardized interface, the *Quantum Device Management Interface* (QDMI), and a software library, the *FoMaC* library (*Figures of Merit and Constraints*), to facilitate dynamic communication and optimization between software tools and quantum hardware platforms. QDMI enables hardware platforms to provide detailed characteristics and constraints of their devices in a standardized manner, allowing software layers to query this data effectively. The FoMaC library, leveraging QDMI, abstracts this low-level information into actionable metrics for software tools, optimizing quantum computing applications based on real-time hardware data.

This approach not only simplifies the integration of new hardware platforms into the quantum software stack but also supports continuous adaptation to hardware changes. By providing a common language and standardized interface, QDMI and the FoMaC library aim to harmonize the objectives of software and hardware developers, facilitating the development of more efficient and adaptable quantum computing applications.

III. Technical Realization

Quantum circuits, typically represented in intermediate formats like OpenQASM, MLIR dialects, or QIR, undergo a series of transformations during compilation before execution on a quantum platform as illustrated in Fig. 2. Traditionally, this compilation process has been static, with platform-specific optimizations hardcoded into the compiler. However, this approach lacks flexibility and fails to account for dynamic changes in hardware characteristics. To address these limitations and to enable dynamic interaction between quantum software tools and hardware platforms, we introduce QDMI. QDMI is provided as a collection of C header files to allow fast integration into an HPC environment and consists of four main components:

1) *QDMI Core*: Provides core functionality to manage sessions as well as to open and close connections to devices.
2) *QDMI Control*: Enables the control of the quantum devices. One can submit quantum circuits, control the job queue, and readout measurement results.
3) *QDMI Device*: Provides device handling functionality, e.g., device calibration or checking the device status.
4) *QDMI Query*: Allows querying properties of the device, e.g., supported gates, error rates, gate duration, etc.

This interface supports querying a wide range of data, from static information like qubit count and gate sets to dynamic data such as current environmental conditions. This data is accessible through a key/value interface and can be used by:

1) The *FoMaC* library, which abstracts low-level hardware details into higher-level metrics for software tools,
2) Software layers for selecting the most suitable platform and optimization strategies based on current hardware characteristics.

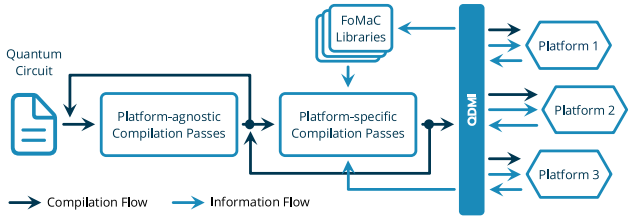

Fig. 2: Overall flow.

The adoption of QDMI by hardware vendors simplifies the process of obtaining device information, eliminating the need for custom solutions. Additionally, the *FoMaC* library, powered by QDMI, provides a standardized way to compute and utilize important metrics like expected fidelity, enhancing the efficiency of quantum compilers and tools by leveraging real-time hardware data. QDMI is publicly available as open source at https://github.com/Munich-Quantum-Software-Stack/QDMI.

Acknowledgments

The authors acknowledge funding from the Munich Quantum Valley initiative, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. R.W., L.S., Y.S., and L.B. acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 101001318) and the MILLENION project (Grant Agreement No. 101114305).

References

[1] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.

[2] Y. Li, Y. Zhang, M. Chen, *et al.*, "Timing-Aware Qubit Mapping and Gate Scheduling Adapted to Neutral Atom Quantum Computing," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3768–3780, 2023.

[3] 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*, 2019.

[4] S. Sivarajah, S. Dilkes, A. Cowtan, *et al.*, "Tket⟩: A retargetable compiler for NISQ devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014 003, 2021.

[5] M. Amy and V. Gheorghiu, "Staq—A full-stack quantum processing toolkit," *Quantum Science and Technology*, vol. 5, no. 3, p. 034 016, 2020.

[6] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, "A software methodology for compiling quantum programs," *Quantum Science and Technology*, vol. 3, no. 2, p. 020 501, 2018.

[7] D. Bluvstein, H. Levine, G. Semeghini, *et al.*, "A quantum processor based on coherent transport of entangled atom arrays," *Nature*, vol. 604, no. 7906, pp. 451–456, 2022.

[8] W. K. Hensinger, "Quantum computer based on shuttling trapped ions," *Nature*, vol. 592, no. 7853, pp. 190–191, 2021.

[9] B. Poggel, N. Quetschlich, L. Burgholzer, *et al.*, "Recommending Solution Paths for Solving Optimization Problems with Quantum Computing," in *Int'l Conf. on Quantum Software*, 2023. arXiv: 2212.11127.

[10] T. Lubinski, S. Johri, P. Varosy, *et al.*, "Application-oriented performance benchmarks for quantum computing," *IEEE Transactions on Quantum Engineering*, vol. 4, pp. 1–32, 2023.

[11] A. Montanaro, "Quantum algorithms: An overview," *npj Quantum Information*, vol. 2, no. 1, p. 15 023, 2016. arXiv: 1511.04206.

[12] B. Bauer, S. Bravyi, M. Motta, and G. K.-L. Chan. "Quantum algorithms for quantum chemistry and quantum materials science." arXiv: 2001.03685. (2020), preprint.

[13] M. Schulz, M. Ruefenacht, D. Kranzlmuller, and L. B. Schulz, "Accelerating HPC With Quantum Computing: It Is a Software Challenge Too," *Computing in Science & Engineering*, vol. 24, no. 4, pp. 60–64, 2022.