Towards System-Level Quantum-Accelerator Integration

Ralf Ramsauer
Technical University of Applied Sciences Regensburg
Regensburg, Germany
ralf.ramsauer@oth-regensburg.de

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg
Siemens Foundational Technologies
Regensburg/Munich, Germany
wolfgang.mauerer@othr.de

***Abstract*—Quantum computers are often treated as experimental add-ons that are loosely coupled to classical infrastructure through high-level interpreted languages and cloud-like orchestration. However, future deployments in both, high-performance computing (HPC) and embedded environments, will demand tighter integration for lower latencies, stronger determinism, and architectural consistency, as well as to implement error correction and other tasks that require tight quantum-classical interaction as generically as possible.**

We propose a vertically integrated quantum systems architecture that treats quantum accelerators and processing units as peripheral system components. A central element is the *Quantum Abstraction Layer* (QAL) at operating system kernel level. It aims at real-time, low-latency, and high-throughput interaction between quantum and classical resources, as well as robust low-level quantum operations scheduling and generic resource management. It can serve as blueprint for orchestration of low-level computational components "around" a QPU (and inside a quantum computer), and across different modalities.

We present first results towards such an integrated architecture, including a virtual QPU model based on QEMU. The architecture is validated through functional emulation on three base architectures (x86_64, ARM64, and RISC-V), and timing-accurate FPGA-based simulations. This allows for a realistic evaluation of hybrid system performance and quantum advantage scenarios. Our work lays the ground for a system-level co-design methodology tailored for the next generation of quantum-classical computing.

I. Introduction

Contemporary quantum computers often remain close to physical laboratory setups, and quantum processing units (QPUs) are connected to classical host systems through loosely coupled, high-level interfaces. This has enabled rapid experimental progress, but imposes significant limitations for future applications, particularly in scenarios that demand scalable, low-latency, and reproducible quantum-classical interaction.

QPUs will never operate in isolation, but will act as accelerators that are tightly embedded within heterogeneous classical systems [1], [2]. This hybrid nature necessitates a system architecture that reflects and supports the duality from the ground up. Future quantum computing deployments will span a wide range of performance classes: from embedded devices based on compact solid-state technologies, such as diamond NV centres, to quantum accelerators deployed as extension cards in workstations, and further to large-scale high-performance quantum computing (HPQC) clusters in data centres (see Fig. 1). These classes particularly differ in latency constraints and control requirements. A system architecture must abstract and support all of them without locking into specific technologies.

Previous efforts, such as QDMI, advocate for more integrated quantum-classical systems by standardised user-level interfaces. While this is a promising direction, no such approach does, to the best of our knowledge, take the role of the operating system kernel into account. We argue that kernel-level integration is not a mere optimisation, but a necessity.

There are several reasons for this. First, in analogy with classical accelerators like GPUs, abstraction through kernel-space drivers enables the decoupling of vendor-specific hardware implementations from standardised user-space APIs. This is essential for software portability and long-term ecosystem evolution. Second, kernel-level interaction supports deterministic scheduling, real-time control, and system-wide resource management. Such properties are required in embedded control environments and HPC workloads alike. Third, it allows for hardware abstraction layers that are both extensible and technology-agnostic, facilitating support for diverse quantum hardware types, including further emerging quantum technologies such as quantum sensing or quantum communication.

In contrast to, for example, network-based control interfaces, our design promotes tight coupling of quantum control electronics to the host system via standard interconnects. This aims at precise synchronisation, reduced latency, and unified orchestration under operating system control.

We introduce an integrated QC device model implemented as a virtual accelerator for early-stage validation. This model, embedded within a full-stack software environment, dispatches quantum programs from user space through a hardware abstraction interface. It enables iterative hardware/software co-design and early functional testing. The architecture is realised using reconfigurable hardware (FPGA) to simulate time-accurate behaviour and measure key metrics such as communication latency and scheduling overheads. This allows for empirical evaluation of quantum systems under near-realistic integration conditions. Our main contributions are:

- A tightly coupled quantum-classical system architecture that addresses key requirements for real-time control, portability, and scalability.

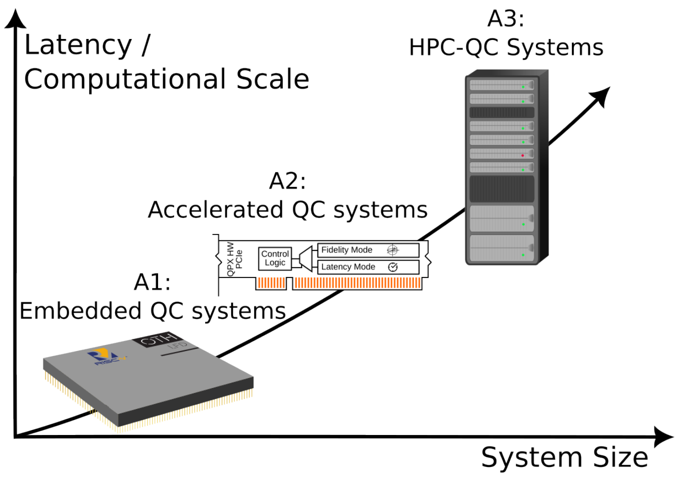


Fig. 1. Overview of integration scenarios. We particularly focus on A1 and A2, and consider integration of multiple instances of A2 into A3.

- A working quantum device model using virtualised hardware for early architectural exploration and co-design.
- We outline a methodology to transition from emulation to time-accurate FPGA-based simulation to analyse real-world integration constraints.

II. State of the Art

Integration of quantum hardware into classical systems spans multiple abstraction layers—from high-level algorithm design [3], [4] to low-level qubit control via AWGs, RF generators, and FPGAs [5]. Standardised interfaces and layered architectures are essential to bridge this gap. Prior work has encoded low-level quantum instructions as binary streams [6], [7], [8], [9], while efforts like QDMI [10] offer hardware-agnostic abstractions. Full-stack frameworks have been proposed for hybrid systems [11], [12], yet complete end-to-end implementations with integrated hardware simulation remain limited.

The RISC-V architecture allows us to make systemic modifications [13], and study connection of external hardware components via FPGA instantiations. While the latter are established techniques for controlling QPUs for control and readout [14], [15], [16], quantum-classical architectures have only recently received consideration via quantum instruction sets [8], [17], [18] or micro-architectures [9], [19], [20], [21], [22]. Integrating QPUs into HPC environments has been discussed more intensively [23], [24], [25], [26], [27], as summarised in Ref. [28].

III. Systems Architecture

A. Overview and Design Rationale

a) General Assumptions: In designing a future-proof architecture for quantum-classical systems, we start from a set of fundamental assumptions shaped by practical deployment constraints and technology readiness levels. A significant body of research has explored quantum extensions to classical instruction set architectures (ISAs), introducing new instruction formats or opcodes to incorporate quantum-specific semantics directly into classical compute pipelines. While these approaches are conceptually appealing and valuable for long-term hardware/software co-design, we argue that such models are premature for the initial stages of quantum computing adoption.

In contrast, our architectural perspective is driven by integration pragmatism. We believe that early quantum computing systems can only be accepted at scale if they seamlessly integrate into the classical computing environments that are already in use. In other words, quantum computing components must first become plug-in extensions to existing compute systems—rather than requiring invasive or fundamental changes to their core processor architectures. Crucially, these devices are typically not accessed directly by user-level applications or libraries. Instead, it is the operating system's task to abstract and communicate with a QPU controller, a dedicated intermediary that resides either on the same hardware card or in a tightly coupled component, that provides means to offload quantum-specific execution, transpilation, translation, and optimisation tasks. By (optionally) consolidating them in a programmable controller, the system achieves a high degree of flexibility and autonomy. This architectural choice reflects the layered nature of the hybrid stack:

General-purpose computation (*e.g.*, application logic, orchestration, scheduling logic, classical pre-/post-processing) is executed on the host CPU. Special-purpose operations, such as pulse generation, qubit manipulation, and physical measurement handling, reside in the quantum control hardware. Intermediate control logic *may* be offloaded to the QPU controller, potentially based on flexible and extensible compute architectures such as RISC-V.

This insight leads to a peripheralisation approach: we assume that first-generation quantum computing accelerators will be implemented as classical peripherals—either embedded directly via memory-mapped I/O (MMIO) interfaces (*e.g.*, in compact setups like diamond-based systems) or as more general-purpose PCIe-based accelerator cards for workstations and HPC nodes (see Fig. 1). Depending on the actual quantum technology, the actual quantum core may be physically located on the card (as is conceivable with diamond technologies) or housed externally (*e.g.*, cryogenic platforms). In the latter case, the interface card acts as a high-speed low-latency bridge, interfacing with control electronics such as arbitrary waveform generators (AWGs), microwave electronics, or photonics modules.

b) Hardware Aspects: Despite significant physical differences across quantum hardware technologies, such as superconducting qubits, trapped ions, spin qubits, or NV centers in diamond, there exist structural commonalities in how these systems are controlled, measured, and calibrated. These tasks typically involve the generation of shaped control pulses, precise timing coordination, and synchronous acquisition of analogue or digital readout signals. Although the specific waveform characteristics, timing constraints, and physical wiring vary between modalities, the logical structure of control is strikingly similar across platforms. This observation enables a generalised control architecture, which can be tailored through parametrisation rather than restructured from scratch for each technology.

At a systems level, most quantum control stacks follow a layered pattern comprising digital signal generation (via CPUs or FPGAs), waveform synthesis (often via AWGs or DACs), and classical feedback control. By focusing our design on configurable interfaces and exchange formats, rather than hardwiring to a specific technology, we enable a hardware abstraction layer that is general-purpose yet extensible, forming the basis for a reusable and adaptable stack.

These quantum accelerator cards inherently require special-purpose embedded control compute to manage configuration, calibration, and runtime orchestration of quantum operations. The design of such control processors must address stringent requirements on latency, determinism, and low-level signal fidelity. To meet these constraints, custom architectural modifications for the special-purpose control processors are often necessary-for example, to ensure precise timing alignment or direct hardware-level waveform control. RISC-V presents a particularly suitable architecture in this context, due to its openness, modularity, and extensibility. Its structure enables the implementation of domain-specific extensions tailored to the needs of quantum-classical interfacing, such as tightly coupled scheduling logic or application-specific peripheral interfaces. This makes RISC-V an ideal foundation for the special-purpose embedded control compute required in tightly integrated quantum systems.

Despite variations in physical deployment, the host interface abstraction (*e.g.*., MMIO vs. PCIe) remains conceptually uniform. This stable abstraction layer enables portability, system-level validation, and standardisation across performance classes and quantum technologies.

To rapidly prototype and validate our architectural ideas, we emulate the accelerator interface using virtualised hardware in QEMU. This model supports detailed interaction studies, early software stack development, and architectural experimentation. Communication between host and accelerator follows state-of-the-art systems techniques, such as MSI-X based interrupt delivery to ensure scalable, low-latency event signalling from device to host, DMA-based memory access to enable efficient bidirectional data transfers with minimal CPU intervention.

Looking forward, our architecture envisions support for multi-QPU configurations enabling peer-to-peer quantum communication and entanglement distribution, which are essential for scalable quantum networks and distributed quantum computing paradigms. To facilitate flexible resource sharing and isolation, we consider incorporating virtualisation interfaces, such as, for example Single Root I/O Virtualization (SR-IOV), to provide multiple virtual quantum accelerator instances on shared physical hardware.

This enables a single physical quantum accelerator device to present multiple virtual functions, allowing concurrent and isolated access by different host system components or virtual machines. In high-performance computing (HPC) environments, this capability facilitates efficient resource sharing and improved utilisation of costly quantum hardware by multiple users (tenants / virtual guests) or applications without compromising performance. SR-IOV reduces overhead by enabling direct device access from user space, bypassing the hypervisor or kernel layers for critical I/O operations, thereby minimising latency and maximising throughput. This low-overhead virtualisation is essential for HPC workloads that require strict timing guarantees and high data transfer rates, making SR-IOV an attractive approach for integrating quantum accelerators into large-scale classical-quantum hybrid HPC systems.

Additionally, achieving ultra-low latency and high-throughput interaction between the classical control units and quantum hardware is critical for real-time pulse generation and dynamic error correction protocols. Our design anticipates such tightly coupled co-processing capabilities to enable effective implementation of these time-sensitive quantum-classical feedback loops.

c) Software Aspects: To enable broad applicability and system-level flexibility, our architecture supports multiple data exchange formats, with a focus on QIR (Quantum Intermediate Representation) and pulse-level instructions. The rationale for this dual support lies in the complementary roles of the two representations. Pulse-level interfaces represent the lowest level of control, offering maximal flexibility for fine-grained manipulation of quantum operations. This level is particularly important for experimental setups, calibration procedures, or future extensions toward quantum-classical co-design at the physical layer.

On the other hand, QIR provides a hardware-agnostic, circuit-level representation of quantum programs that abstracts from physical details. In our model, we support direct ingestion of QIR on the accelerator, based on the assumption that the card itself is best suited to perform transpilation and optimisation, as it has full knowledge of its internal topology, constraints, and calibration data. This implies the need for on-card control compute, which we explicitly respect in our design. The QIR pathway is optional: cards may accept other formats or perform compilation on the host side if desired.

Future research will explore intelligent caching strategies to minimise repeated transpilation times and accelerate execution in long-running or multi-user workloads. The format interface is designed to be extensible, allowing additional quantum IRs or device-specific formats to be integrated without architectural changes.

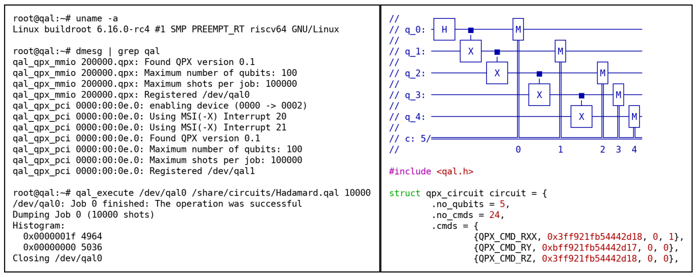
d) Systems Software Aspects: To maintain compatibility with emerging and broadly accepted standards, the software architecture includes a QDMI-style interface embedded into the kernel driver, allowing integration with higher-level frameworks and runtime systems.

The OS driver-level abstraction must ensure that quantum accelerators can be managed like conventional hardware resources, enabling multi-process access, secure context isolation, and system-level orchestration independent of the scale of the system, from embedded environments to HPC-like environments. Linux as basis for implementing our kernel-level driver infrastructure allows for integration with existing HPC and embedded system software.

The quantum accelerator is exposed as character device (*e.g.*, `/dev/qal0`), offering an abstraction that aligns with the standard device models It is responsible for receiving quantum execution sequences from user-space applications, queuing and scheduling them according to defined policies, and tracking execution state. Interaction between user space and driver is based on `ioctl` calls, providing a flexible mechanism for command invocation and control. Future revisions will add support for `mmap`-based interfaces to enable direct hardware-level access to DMA buffers, allowing low-latency and OS-bypassed communication paths for applications with determinism requirements. We detail Two instantiation s below: a QEMU-based virtual model for rapid iteration and HW/SW co-design, and a time-accurate FPGA-based hardware-in-the-loop system for empirical latency and integration analysis.

B. Virtual Quantum Accelerator Model

To enable rapid prototyping, functional validation, and iterative development, we have implemented a virtual quantum accelerator using QEMU, a widely adopted system emulator. This allows us to evaluate architectural concepts and software interfaces prior to hardware availability, while maintaining tight alignment with real-world system constraints.



```
root@qal:~# uname -a
Linux buildroot 6.16.0-rc4 #1 SMP PREEMPT_RT riscv64 GNU/Linux

root@qal:~# dmesg | grep qal
qal_qpx_mmio 200000.qpx: Found QPX version 0.1
qal_qpx_mmio 200000.qpx: Maximum number of qubits: 100
qal_qpx_mmio 200000.qpx: Maximum shots per job: 100000
qal_qpx_mmio 200000.qpx: Registered /dev/qal0
qal_qpx_pci 0000:00:0e.0: enabling device (0000 -> 0002)
qal_qpx_pci 0000:00:0e.0: Using MSI(-X) Interrupt 20
qal_qpx_pci 0000:00:0e.0: Using MSI(-X) Interrupt 21
qal_qpx_pci 0000:00:0e.0: Found QPX version 0.1
qal_qpx_pci 0000:00:0e.0: Maximum number of qubits: 100
qal_qpx_pci 0000:00:0e.0: Maximum shots per job: 100000
qal_qpx_pci 0000:00:0e.0: Registered /dev/qal1

root@qal:~# qal_execute /dev/qal0 /share/circuits/Hadamard.qal 10000
/dev/qal0: Job 0 finished: The operation was successful
Dumping Job 0 (10000 shots)
Histogram:
  0x0000001f 4964
  0x00000000 5036
Closing /dev/qal0
```

```
#include <qal.h>

struct qpx_circuit circuit = {
        .no_qubits = 5,
        .no_cmds = 24,
        .cmds = {
                {QPX_CMD_RXX, 0x3ff921fb54442d18, 0, 1},
                {QPX_CMD_RY, 0xbff921fb54442d17, 0, 0},
                {QPX_CMD_RZ, 0x3ff921fb54442d18, 0, 0},
```

Fig. 2. Illustration of QPX in a virtualised RISC-V environment. Left: Userspace shell within QEMU guest executing a sample quantum circuit via `/dev/qal0`. Right: Corresponding high-level source (Hadamard.c) generated using Qiskit and compiled into QAL binary format. This demonstrates end-to-end execution through the system stack.

Our *Quantum Peripheral Extension* (QPX) acts as drop-in device model for quantum accelerator modalities. QPX supports MMIO-based embedded integration and PCIe-based peripheral accelerator configurations, covering a wide spectrum of possible quantum-classical co-design deployments—from embedded edge systems to high-performance compute environments. We validate and test our device model on three different classical host architectures: x86_64, riscv64 and arm64.

The device model provides an interface stack, including:

- DMA-based memory access for high-throughput, low-latency data exchange,
- Support for interrupt signalling via IRQ and MSI(-X),
- A command protocol supporting for the internal QAL,
- Ongoing integration of QIR (Quantum Intermediate Representation) to facilitate compiler-side integration and future-proofing against emerging quantum software toolchains.

The backend uses libquantum as simulation engine for design validation and ensuring functional correctness from APIs via kernel drivers to device logic. This has several benefits:

- Hardware/Software Co-Design: Short development cycles allow architectural feedback from quantum algorithm developers to be directly integrated into interface design.
- Early integration testing: Enables system-level validation of driver functionality, kernel interaction, and execution model semantics before hardware availability.
- Technology abstraction: Since the model is decoupled from specific quantum hardware implementations, it supports experimentation with technology-agnostic system designs.
- Reproducibility [29] and CI integration using automated test pipelines.

However, QPX cannot be timing accurate, as simulating QPUs efficiently is impossible. It is not suitable for latency and runtime analysis.

C. Time-Accurate FPGA Simulation

To complement the functional validation provided by the QEMU-based virtual model, we implement a timing-accurate quantum accelerator prototype on a *digital twin* FPGA platform. This approach enables cycle-accurate and time-aware simulation of the quantum-classical interface, which is critical for evaluating latency, throughput, and real-time behaviour. These key factors are essential for determining potential quantum advantage in hybrid systems.

The FPGA model mirrors the architectural concepts validated in QEMU, supporting the same PCIe and MMIO interfaces, DMA-based data transfer, and interrupt handling mechanisms (legacy IRQ, MSI, MSI-X). This ensures a seamless transition from software simulation to hardware prototyping, facilitating direct comparison and cross-validation between both platforms.

The ability to operate in two modes is crucial:

1) Fidelity Mode. The FPGA emulates the quantum hardware (at obvious exponential cost), running quantum circuits as per the virtual device model, verifying correctness of the implementation.
2) Latency Mode. The FPGA mimics timing behaviour of real future quantum hardware components or external devices, providing real-time control and data exchange, enabling in-depth latency profiling and operational testing under realistic conditions.

Reconfigurability of FPGAs enables a flexible, yet precise platform that balances rapid development cycles with high-fidelity performance metrics. This allows for exploring

- quantum-classicalcommunication latencies,
- impact of the system / user-level software stack while interacting with real hardware,
- real-time constraints for quantum pulse-level control,
- overall system behaviour under realistic load conditions.

The model bridges between virtual prototypes and physical hardware, serving as tool for iterative co-design of hardware and software components, and to fine-tune architectural parameters for early identification of performance limitations.

Ultimately, the combination of QEMU virtualisation and FPGA-based hardware-accurate simulation provides a comprehensive framework for the design, validation, and optimisation of tightly integrated quantum-classical systems, driving progress towards practical and scalable quantum accelerators.

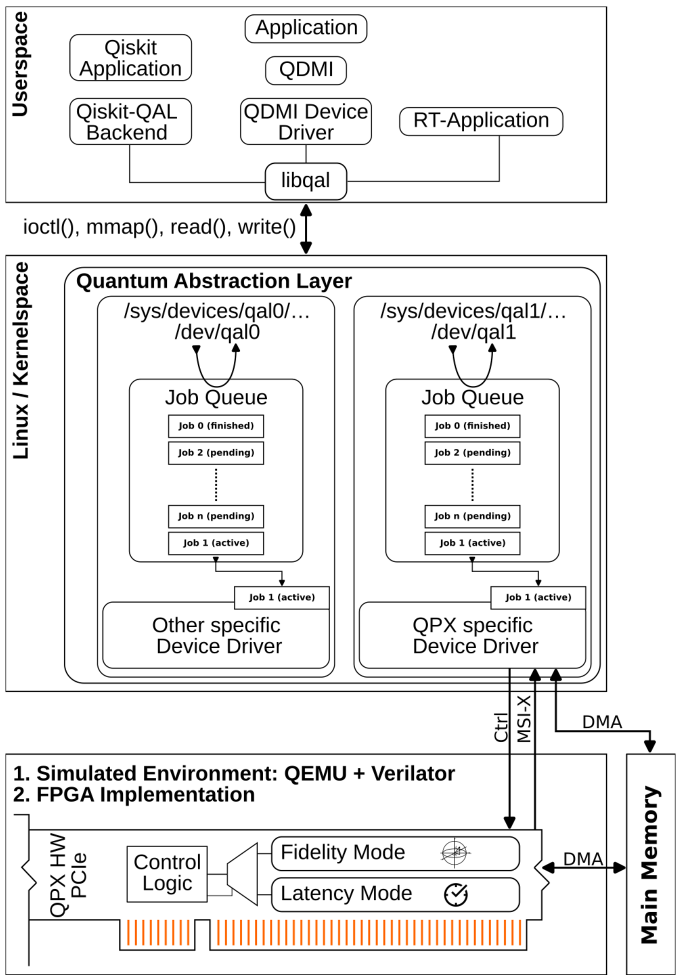


Fig. 3. Overview of the prototypical architecture for classical-quantum integration. User space applications interface with standard quantum programming frameworks (*e.g.*, Qiskit, QLM, Qrisp, . . .), or directly via libqal, a thin user space library providing access to kernel-level quantum device functionality. The Linux kernel hosts a generic QAL subsystem responsible for managing and scheduling quantum sequences, offering abstractions prioritisation and state tracking. This layer connects to device-specific drivers, exemplified by our QPX model, which defines a concrete quantum hardware interface.

IV. Prototype Implementation and Roadmap

The current system is a functionally complete, yet deliberately lean and modular prototype. All essential components of the classical/quantum integration stack are realised virtually.

A full cut-through system is demonstrated based on QEMU (see Fig. 2), including userspace interaction via the `libqal` library, kernel-level job scheduling through a dedicated device node, and a virtual quantum accelerator model (QPX). The kernel interface presently uses ioctl-based command handling for task management and device control. Upcoming versions will include direct memory-mapped access to DMA regions for low-latency, trap-free [30] interaction.

We validated our stack on three major processor architectures: x86_64, ARM64, and RISC-V 64, ensuring portability and architectural agnosticism across mainstream and emerging compute platforms. These evaluations included both MMIO-based and PCIe-based configurations, highlighting the versatility of the QPX model and confirming correct behaviour of the stack across different execution environments.

While the QEMU-based simulation back-end does not yet support timing-accurate analysis, it provides an environment for functional testing, rapid iteration, and architectural prototyping. Insights guide the refinement of an FPGA-based hardware implementation for cycle-accurate evaluation of control latencies and communication paths (at the expense of result correctness).

In summary, the current system validates the feasibility of our integration model and forms an baseline for subsequent research. It enables experimentation with interface semantics, abstraction strategies, and system-level integration patterns in a controlled and especially reproducible environment. It represents a step towards a broader vision of a technology-agnostic, tightly integrated classical-quantum computing architecture.

An important open question that can be empirically addressed concerns division of responsibilities between kernel and user space. While present interfaces largely follow a basic pattern (*e.g.*, `QDMI_device_job_{submit, check/wait, get_results}`) it remains an open issue which abstractions and operations should be delegated to kernel-level logic, and which are more suitably handled in user space.

This becomes increasingly relevant as systems scale. Especially quantum error correction requires intricate interaction between CPUs, accelerators and QPUs. Until such mechanisms are fully refined and can be abstracted away, we expect contradicting requirements between low latency, involved compute, and flexibility to require system-global architectural decisions. Our architecture provides the necessary foundation to explore these trade-offs empirically and systematically.

V. Conclusion & Future Work

We have introduced a system-level architecture that integrates quantum accelerators with classical computing environments. Our approach is grounded in the assumption that hybrid classical-quantum systems use quantum devices operating as tightly coupled peripherals rather than stand-alone units, and that the inner working of such stand-alone units features similarities across modalities that can benefit from a standard base platform. We have shown that seamless integration is possible at the kernel level without invasive modifications.

Our prototypical architecture has been validated across multiple host architectures (x86_64, arm64, and riscv64), and has demonstrated a vertical system cut-through from user space to simulated hardware. It establishes a basis for architectural experimentation and refinement, particularly for challenges like error correction that require the coordinated interaction between multiple computing entities.

Future work will focus on refining an FPGA-based implementation to enable time-accurate (and necessarily result-incorrect) evaluation and analysis of latencies, which is essential for understanding practical quantum advantage under realistic conditions. We will explore questions related to the placement of abstractions across user and kernel space, refine the interaction with emerging interface standards such as QDMI, and investigate advanced offloading strategies for transpilation and scheduling. Furthermore, virtualisation capabilities and support for distributed QPU systems will be explored to enable scalable and shareable quantum-classical infrastructures. The open and modular nature of our design also supports the release of technology-agnostic components as open source, thereby contributing to the broader ecosystem and encouraging future standardisation efforts.

References

[1] C. Carbonelli et al., "Challenges for quantum software engineering: An industrial application scenario perspective," in *Quantum Software: Aspects of Theory and System Design*. Springer, 2024.

[2] T. Yue et al., "Challenges and opportunities in quantum software architecture," in *Software Architecture: Research Roadmaps from the Community*. Springer, 2023.

[3] A. A. Khan et al., "Software architecture for quantum computing systems—a systematic review," *SSRN Journal*, 2023.

[4] S. Thelen, H. Safi, and W. Mauerer, "Approximating under the influence of quantum noise and compute power," in *IEEE QCE*, vol. 02, 2024, pp. 274–279.

[5] T. Guo, X. Guo, and M. Schulz, "An FPGA-based quantum control system with a runtime configurable signal generator," in *IEEE QCE*, 2024.

[6] A. McCaskey and T. Nguyen, "A MLIR dialect for quantum assembly languages," in *IEEE QCE*, 2021.

[7] Y. Stade, L. Burgholzer, and R. Wille, *Towards supporting qir: Thoughts on adopting the quantum intermediate representation*, 2024. arXiv: 2411.18682 [quant-ph].

[8] X. Fu et al., "eQASM: An executable quantum instruction set architecture," in *IEEE HPCA*, 2019.

[9] X. Guo, K. Qin, and M. Schulz, "HiSEP-Q: A highly scalable and efficient quantum control processor for superconducting qubits," in *IEEE ICCD*, 2023.

[10] R. Wille et al., "QDMI - quantum device management interface: Hardware-software interface for the munich quantum software stack," in *IEEE QCE*, 2024.

[11] A. Elsharkawy, X. Guo, and M. Schulz, "Integration of quantum accelerators into hpc: Toward a unified quantum platform," in *IEEE QCE*, 2024.

[12] F. Zhang et al., "A classical architecture for digital quantum computers," *ACM TQC*, 2023.

[13] J. Bachrach et al., "Chisel: Constructing hardware in a scala embedded language," in *Proc. 49th Design Automation Conference*, 2012.

[14] C. A. Ryan et al., "Hardware for dynamic quantum computing," *Rev. Sci. Instrum.*, vol. 88, no. 10, p. 104 703, 2017.

[15] Y. Xu et al., "Qubic: An open-source fpga-based control and measurement system for superconducting quantum information processors," *IEEE Transactions on Quantum Computing*, 2021.

[16] X. Guo et al., "Design of an fpga-based neutral atom rearrangement accelerator for quantum computing," in *Proc. DATE*, 2025, pp. 1–6.

[17] X. Fu et al., "An experimental microarchitecture for a superconducting quantum processor," in *50th Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2017.

[18] S. Batabyal and L. Sharma, "A quantum pipeline for an executable quantum instruction set architecture," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 294–299.

[19] A. Butko et al., "Understanding quantum control processor capabilities and limitations through circuit characterization," in *IEEE ICRC*, 2020.

[20] L. Stefanazzi et al., "The QICK (Quantum Instrumentation Control Kit): Readout and control for qubits and detectors," *Rev. Sci. Inst.*, vol. 93, no. 4, 2022.

[21] M. Zhang et al., "Exploiting different levels of parallelism in the quantum control microarchitecture for superconducting qubits," in *54th Annual IEEE/ACM Int. Sym. on Microarchitecture*, 2021.

[22] M. O. Tholén et al., "Measurement and control of a superconducting quantum processor with a fully integrated radio-frequency system on a chip," *Rev. Sci. Inst.*, vol. 93, no. 10, 2022.

[23] T. S. Humble et al., "Quantum computers for high-performance computing," *IEEE Micro*, vol. 41, no. 5, 2021.

[24] K. Wintersperger, H. Safi, and W. Mauerer, "Qpu-system co-design for quantum hpc accelerators," in *Proc. 35th GI/ITG ARCS*, Gesellschaft für Informatik, 2022.

[25] H. Safi, K. Wintersperger, and W. Mauerer, "Influence of HW-SW-Co-Design on quantum computing scalability," in *IEEE QSW*, 2023.

[26] M. N. Farooqi and M. Ruefenacht, "Exploring hybrid classical-quantum compute systems through simulation," in *Proc. IEEE QCE*, 2023.

[27] P. Seitz et al., "Toward a unified hybrid hpcqc toolchain," in *IEEE QCE*, 2023.

[28] A. Elsharkawy et al., "Integration of quantum accelerators with high performance computing – a review of quantum programming tools," *ACM Transactions on Quantum Computing*, 2025.

[29] W. Mauerer and S. Scherzinger, "1-2-3 reproducibility for quantum software experiments," in *Proc. IEEE SANER*, 2022.

[30] R. Ramsauer et al., "Look mum, no VM exits! (almost)," in *OSPERT@ECRTS*, 2017.