Barnes et al. *EPJ Quantum Technology* (2023) 10:36
https://doi.org/10.1140/epjqt/s40507-023-00192-z

EPJ Quantum Technology
a SpringerOpen Journal

RESEARCH Open Access

Optimising the quantum/classical interface for efficiency and portability with a multi-level hardware abstraction layer for quantum computers

Kenton M. Barnes[1], Anton Buyskikh[1], Nicholas Y. Chen[1], Gabriel Gallardo[1], Marco Ghibaudi[1], Matthew J.A. Ruszala[1*], Daniel S. Underwood[1], Abhishek Agarwal[2], Deep Lall[2], Ivan Rungger[2] and Nikolaos Schoinas[2]

*Correspondence:
matthew.ruszala@riverlane.com
[1]Riverlane, St Andrew's House, 59 St Andrew's St, Cambridge, CB2 3BZ, United Kingdom
Full list of author information is available at the end of the article

Abstract

Steady progress is being made in the development of quantum computing platforms based on different types of qubit technologies. Each platform requires bespoke strategies to maximise the efficiency of the quantum/classical interface when operating close to the qubits. At a higher level, however, a shared interface allowing portability of quantum algorithms across all the available quantum platforms is preferred. Striking the right balance between portability and performance of the algorithm as implemented on quantum hardware remains a major challenge for this field. Here, we propose a quantum hardware abstraction layer (QHAL) providing a multi-level intermediate representation of the quantum stack. A collaborative effort between software specialists and quantum hardware developers operating on four major qubit technologies (photonics, silicon, superconducting and trapped ions) led to the identification of a minimum common set of instructions and metadata allowing the QHAL to interact efficiently with multiple platforms. Access to the stack from the higher levels increases latency yet minimises the amount of hardware architecture parameters to be handled by the algorithm developer, thus simplifying code development and reducing security threats from misuse or malicious access for hardware developers. Access to the stack from the lowest—closest to the qubits—level provides the highest hardware responsiveness, suitable for algorithms requiring minimum latency for data and instruction transfer. With respect to existing quantum assembly languages, the QHAL extends further down in the stack by defining an application-binary interface to interact with the quantum hardware. By defining a standard representation of the quantum stack, a common reference framework is provided to both software and hardware developers which would ensure future integration of their R&D efforts.

1 Introduction

Developing uniform quantum computing is a remarkable scientific task. In recent years the global quantum computing market has grown rapidly, with bigger and smaller quantum developers, both in hardware and software, entering the picture. On a quantum hardware level, there are several types of technologies deemed promising to support qubits, with a handful of those under development in academic and commercial labs. Each technological platform also requires a bespoke control system to efficiently manipulate and read out the qubits, typically through sequences of optical or electric pulses that are unique to the technology considered. As we move away from the qubits towards higher ends of the quantum stack, the degree of specificity for the interactions between the quantum and classical units decreases. Within the growing quantum community, developers of control software, error correction schemes, compilers and many more pieces of the quantum stack puzzle pursue several different directions in search of the most highly performing solutions. While such diversity is allowing for many approaches to be evaluated, one risk of uncoordinated development is that optimal solutions identified for each of these elements may lose performance when interfaced with the rest of the stack. Early coordination efforts among different players operating in hardware and software development, leading further along the road to standardisation of the interfaces connecting the various components, may enable all the different pieces to interface in a versatile way and combine in an end-to-end, feasible solution.

Standardised interfaces between software and hardware can offer several benefits to the relevant parties, with the main one being inter-operability. By clearly defining the boundaries between those two worlds, hardware and software developers can focus their R&D efforts in optimising particular layers of the stack independently, and without worrying whether they will be able to interface between them, as long as the new advanced features adhere to the agreed standards. As a result, targeted R&D efforts can be maximised instead of being duplicated among the different parties wasting resources and time in the broader context.

At the same time, standardisation is a means to balance the quantum ecosystem; naturally, large organisations and their choices in terms of e.g. qubit technology have a bigger footprint, inevitably affecting any adjacent work carried out by smaller parties. A standardised interface between software and hardware will allow all qubit technologies to be seen as equal in terms of merit and potential and will attract similar attention from developers working across the stack. However, standardisation also carries the risk of realising suboptimal interfaces that sacrifice quantum hardware performance for the sake of generalisation. This opens the door to an interesting problem: how to strike the right balance between portability and efficiency?

Intermediate representations (IRs), across different levels of the stack, can offer a compromise between uniformity and specificity needs, with the ultimate goal of speeding up development across the entire quantum stack. Currently there are intermediate representations such as Microsoft's QIR [1] or the MLIR [2] quantum dialect designed for use in quantum compilers like qcor, XACC [3], and the Q# compiler. Their main advantage is being seamlessly integrated and able to communicate with other widely accepted platforms for programming quantum computers such as high-level quantum programming languages like Q# or Silq, quantum assembly languages like OpenQASM [4] or Quil [5], and software development kits (SDKs) like Qiskit, Cirq, and ProjectQ. SDKs offer applica-

tion programming interfaces (APIs) through which a user can submit quantum programs to cloud-connected quantum computers. In principle, all these languages and SDKs are interoperable by the ingestion and emission of OpenQASM or another IR.

Yet there still does not exist a hardware-friendly, low level of abstraction to guarantee the portability of programs and high-level tools across technologies. Low-level management of the quantum/classical interface and optimisation of latency and other key performance is delegated to individual hardware developers, who develop bespoke solutions without considering platform interoperability. However, optimisation of the low-level quantum stack will rapidly become a hard problem as the complexity of the control system grows to address advanced functionalities like quantum error correction (QEC). By making their low-level stack compatible with cross-platform solutions, hardware manufacturers may benefit from third-party innovations that would significantly reduce in-house costs for research and development. Rather than an intermediate representation, a low-level hardware abstraction layer is needed to address this issue.

Here we report a multi-level Quantum Hardware Abstraction Layer (QHAL), a quantum-classical interface designed to ensure interoperability between the low-level quantum hardware and control systems and the higher-level stack components. As a result of a collaborative effort involving quantum hardware developers encompassing four leading qubit technologies—superconducting (SEEQC, Oxford Quantum Circuits), silicon (Hitachi), trapped ions (Oxford Ionics, Universal Quantum) and photonics (Duality Quantum Photonics)—we identified a series of standard commands and metadata that can be implemented on most devices. An optimal bitstring representation readable by the control system hardware has also been defined for these instructions, in order to minimise latency while allowing broad qubit control and efficient interfacing with virtually any quantum computing platform able to understand such binaries.

By providing this set of common features, the QHAL extends the IR pipeline and its benefits much deeper into the hardware stack than otherwise possible. The QHAL also provides hardware manufacturers the ability to offer different levels of hardware access to different users, tailoring the balance between latency and security to the user requirements and level of trust. In fact, security and intellectual property (IP) protection in a booming ecosystem is another major concern for all quantum companies. Any standardisation approach should ensure that hardware companies can choose which information to expose without compromising their IP, nor stall the co-development efforts.

The QHAL architecture is a first attempt to provide a standardised interface across different qubit platforms. As research and development evolution in quantum hardware may fundamentally change the way to operate qubits in the future, we expect QHAL to evolve accordingly. As such, the architecture has built-in flexibility allowing the key sets of instructions and information crossing the quantum-classical interface to expand. The latest releases of the QHAL and its related documentation are reported in an online repository for public access on GitHub [6].

2 A quantum hardware abstraction layer (QHAL)

We aimed at developing a QHAL architecture offering an effective low-level of abstraction and portability across technologies by following two main design guidelines. First, abstraction of the hardware details should not hamper fast interaction (minimised latency) with the qubits when this is required for highly demanding algorithms such as the execution

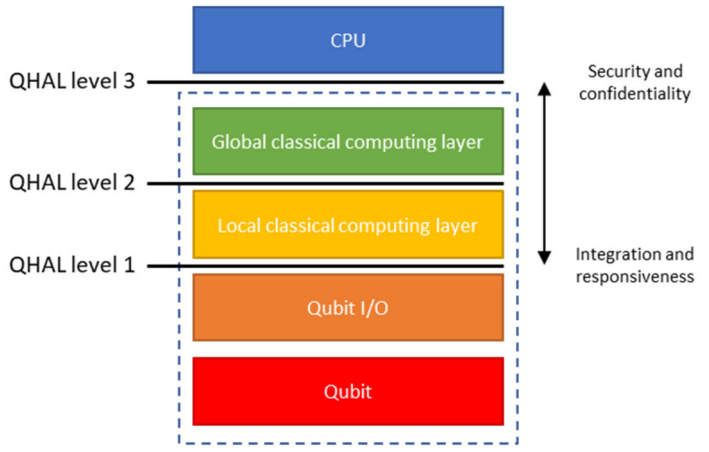

Figure 1 *The three levels of access to the QPU system stack (dashed box) provided by QHAL.* Moving closer to the qubits from QHAL level 3 to QHAL level 1, a higher integration with fast logic components reduces latency and ensures faster responsiveness. As a tradeoff, security threats increase due to the increasing amount of privileged information required to interact with the hardware

of error correction cycles. At the same time, it should allow the hardware manufacturer to offer access to different types of resources, depending on the user level of trust and needs. Similar to the hierarchical protection domain mechanisms adopted for classical CPU architectures [7], the most privileged users should have access to the largest amount of information and the fastest level of interaction with the hardware resources, whereas low-trust users should be allowed to communicate with the machine with a reduced level of control. This is to minimise security threats from malicious behavior aiming at exposing hardware details protected by trade secrets or IP, or aiming at damaging hardware functionality.

Figure 1 shows a schematic representation of the Quantum processing unit (QPU) system stack, together with the three access levels we identified for the QHAL architecture. The fastest and lowest level—QHAL level 1—is located at the interface with the input-output (I/O) hardware controlling the qubits. Such close interaction guarantees a very short latency in transferring instructions and data. Keeping latency below the qubit decoherence time enables mid-circuit measurements and circuit branching dependent on such measurements, both essential conditions for the implementation of quantum error correction algorithms as well as some NISQ-era algorithms such as the holographic variational quantum eigensolver [8].

QHAL level 2 provides access to the local classical computing layer managing the quantum hardware, for instance through field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs). The classical-to-quantum latency here is expected to be comparable with the qubit decoherence time. Users accessing this level can input a single circuit or small batches of circuits, and act upon their results. While the controlling hardware at this level cannot perform circuit updates upon mid-circuit measurements, action at the end of single-circuit measurements still ensures fast operation and minimisation of idle times for the quantum hardware. Examples of algorithms that can be run with this type of hardware interaction are the iterative/Bayesian phase estimation [9, 10] and the accelerated variational quantum eigensolver (aVQE) [11].

QHAL level 3 is the slowest access point to the quantum system stack, with latency typically larger than the qubit decoherence time. Interfacing through global classical logics (global FPGAs or CPU) allows users to input large batches of static circuits and collect measurements output at the end of the batch. The capabilities offered by this level of access are comparable to those currently provided by most quantum hardware vendors through cloud access.

The multi-level architecture seamlessly lends itself to the definition of a hierarchical set of access privileges that each hardware vendor can customise according to their security requirements. QHAL level 1 users operate at the lowest level of control, thus requiring the highest level of hardware information and potentially exposing the hardware to high risk of resource misuse or malicious attacks; as such, hardware manufacturers may decide to share this access only with authenticated and trusted users or internal developers. In contrast, applications executed at a QHAL level 3 require a less direct interaction with the hardware and a reduced amount of hardware information, thus reducing potential vulnerabilities by limiting user capability.

In the following sections, we will describe two unique features of the QHAL: the ability to expose system metadata upstream and a hardware-friendly command set, which enable the QHAL to achieve efficient qubit control and minimise latency.

2.1 The role of metadata

The QHAL architecture provides two degrees of flexibility, with respect to the type of hardware technology it is interfacing with and to the type of privilege/level of trust of the user. Such flexibility is implemented through the definition of a set of metadata available to the user at each QHAL level. Metadata (Table 1) expose information related to those hardware resources/parameters that are essential for the users to understand whether the platform is suitable to their problems and to execute algorithms, and at the same time hide

Table 1 *QHAL metadata*. Table of metadata exposed when using the QHAL, the QHAL levels at which they are required and accompanying notes

Metadata	Description	Level 1 required	Level 2 required	Level 3 required	Notes
NUM_QUBITS	Number of qubits available	Yes	Yes	Yes	–
MAX_DEPTH	Maximum depth of an executable circuit	Yes	Yes	Yes	Expressed as max number of universal gates (Level 3) or native gates (levels 1-2)
NATIVE_GATES	List of native gates	Yes	Yes	No	–
CONNECTIVITY	Connectivity matrix of the qubits	Yes	Yes	No	Required for the correct compilation of circuits
GATE_TIMES	Execution time of the native gates	Yes	No	No	Advanced users can also infer this metric through execution of test/benchmarking algorithms
ERROR_RATE	Average error rate for one- and two-qubit native gates	No	No	No	Advanced users can also infer this metric through execution of test/benchmarking algorithms

details or trade secrets specific to the hardware implementation. Importantly, although the metadata set is shared across the three QHAL levels, access to each individual parameter can be limited according to the specific QHAL level or user privilege. As such, the vendor can choose to expose only a subset of the full metadata at the higher QHAL levels when being used by untrusted parties. Additionally, individual hardware manufacturers may decide to apply further access rules, making some of these metadata accessible only to internal developers. For instance, it is required that the metadata returns a list of native gates in a specified order, so when "ERROR RATE" metadata is requested, a list of error rates corresponding to the hardware native gates in the same order will be returned. This information can be optionally made available by the manufacturers at a desired QHAL level or only to specific users. Note that the error quantification method has not been standardised at this stage.

Table 1 provides a list of essential metadata identified in the first release [6] Metadata requests are encoded into a single 64-bit integer, whereas results are encoded in one or multiple 64-bit integers as appropriate for the requested parameter. We refer the readers to the QHAL Specifications [6] for more details and examples of metadata requests and responses.

2.2 Core and additional commands

Together with metadata (Table 1), a core set of commands (Table 2) has been defined for each QHAL level. We have adopted a RISC (reduced instruction set computer)-type approach to identify a minimum set of instructions that allow full control over qubit functionality (across different qubit platforms) while maximising speed. An additional set of optional commands specific to each qubit technology has also been defined for advanced operations where appropriate. Both the core and optional command sets can be flexibly extended to adapt to future developments in the use of quantum resources.

Table 2 *QHAL core commands*. Table outlining the QHAL core commands, their parameters, descriptions, and to which QHAL level they are relevant to

Command	Parameters	Description	QHAL level
NOP	None	Performs no operation	All
State prepare	qubit address, state value	Prepare specific qubit to a known state	All
State prepare all	state value	Prepare all qubits to a |0> state	All
Qubit measure	qubit address	Return the measured state of a qubit	All
Rx, Ry, Rz	qubit address, angle	Perform qubit rotation around the X, Y or Z axis of the Bloch sphere on a qubit	All
X,Y,Z,H,S,T	qubit address	Perform the corresponding gate operation on a qubit	All
CNOT	qubits addresses	Perform a Controlled-NOT operation	3
Start of session	Type of session	Define the destination of the received commands (emulator, hardware, simulator)	3,2
End of session	None	Close a session	3,2
Set Page Qubit 0	Offset for the qubit index (0)	Modify the offset used in the qubit index computation	All
Set Page Qubit 1	Offset for the qubit index (1)	Modify the offset used in the qubit index computation	All

For each QHAL level, latency requirements and expected modality of resource usage determine the type and format of required commands. Users accessing the quantum stack at QHAL level 3 should be able to run large batches of circuits back-to-back, while keeping their batch separate from those of the previous and next users; hence, they require commands to execute gates from a universal gate set as well as section commands to delimit the binary sequences corresponding to their batch. In this high latency setting where classical logic acts upon measurements occurring at the end of the whole batch execution, compilation, transpilation and buffering steps can be performed offline before transmitting the commands to the target hardware. In QHAL level 2 with intermediate latency—where classical logic acts upon single-circuit measurements and selects the next circuit to execute—users require commands to define gates from a native gateset, as well as parallel compilation for branching statements and circuit repetition/reloading. The commands received by the user are then transpiled to the relevant hardware representation before being transmitted to the target hardware. The commands and measurements flow in QHAL level 1 has the lowest latency, with the classical logic being potentially able to act upon mid-circuit measurements. Here, commands include the execution of gates from a native gate set. Conversion to hardware control sequences, loading of the sequences and feedback of measurements to classical logic must occur within the qubit decoherence time.

An initial selection of core commands is reported in Table 2. It includes commands to execute single-qubit operations, two-qubit operations and control commands required for advanced functionalities. We notice that, at this stage, a single two-qubit operation (CNOT) has been included in the core commands for execution at QHAL level 3 on a universal gateset, whereas implementation at QHAL levels 2 and 1 of two-qubit operations on native gatesets is part of the hardware platform-specific list of optional commands. In fact, each technology leverages unique properties and physical interactions to implement two-qubit gates in the most effective way. As such, at this stage the adoption of a single type of native two-qubit gate across different technologies would lead to suboptimal execution (i.e., the gate will affect the total fidelity of the chain of commands) or to an impossible request (i.e. the gate cannot be implemented). Similarly, instructions for conditional execution (FOR/IF) are also part of the list of optional commands, since their format and limits are platform specific. A future push towards standardisation of the classical-to-quantum interface would motivate hardware vendors to adapt the format of native two-qubit instructions, as well as of these other internal features, to the QHAL structure, making such commands available to users in the core set across the whole multilevel structure. Within the core control commands, a "start of session" command is used to specify whether the user session should be executed on a simulator, real quantum hardware or on proprietary quantum emulators made available by the hardware vendors.

The QHAL is also expected to return basic responses to notify users at QHAL levels 3 and 2 whether the input circuit has been completed successfully or whether errors (such as sending an invalid command or an incorrect computation in the quantum computer) have occurred. Additional platform-specific error codes can be included by each hardware provider. In order to keep latency at its lowest possible value, these responses are not expected to be received at QHAL level 1 access.

A bitstring representation of the above-mentioned quantum instructions has been defined in order to be directly interpreted by low-level control system hardware. The choice of representation can strongly affect the overall communication speed. In particular,

throughout this work, we focused on two important elements impacting latency: the commands transfer process from the user application to the quantum control stack and the role of commands format on parsing speed.

With respect to the transfer process, we have investigated the most hardware-universal yet optimal data format that works across different types of communication channels. The definition of a command size of 64 bits originates from its broad applicability. In fact, CPU registers consist of 64 bits [12]; similarly, FPGA internal buses and external memory interfaces tend to favor a 64 bits wide implementation [13]. Notice that transmission through non-standard interfaces, that some hardware vendors may have developed internally, have not been considered, as this would hamper future standardisation efforts for the QHAL architecture. Limiting the register length to 64 bits instead of 128 bits, however, imposes constraints on the maximum field size for qubit indexing to be dedicated within each command. Within such constraints, our strategy to maximise qubit addressability is based on the assumption that, within the same session, parallel operations will be executed within a patch, or "page", of a maximum of 2^{10} qubits, whereas different execution sessions can address up to 2^{36} different pages. At the beginning of each session, a dedicated control command ("Set Page Qubit" in Table 2) provides the base offset of the qubit indices (encoded in binary format in a field size of 36 bits) that will be used for the duration of the session, whereas the relative offset field (size of 10 bits) is provided within each one-qubit or two-qubits command. The actual qubit index is expressed by the sum of the base and relative offsets, thus allowing indexing of up to 2^{46} different qubits.

Parsing speed, represented by how quickly the command decoding logic can convert the received command into a sequence of low-level directives, strongly depends on the command format. We performed a sequence of format optimisations with the goal of obtaining

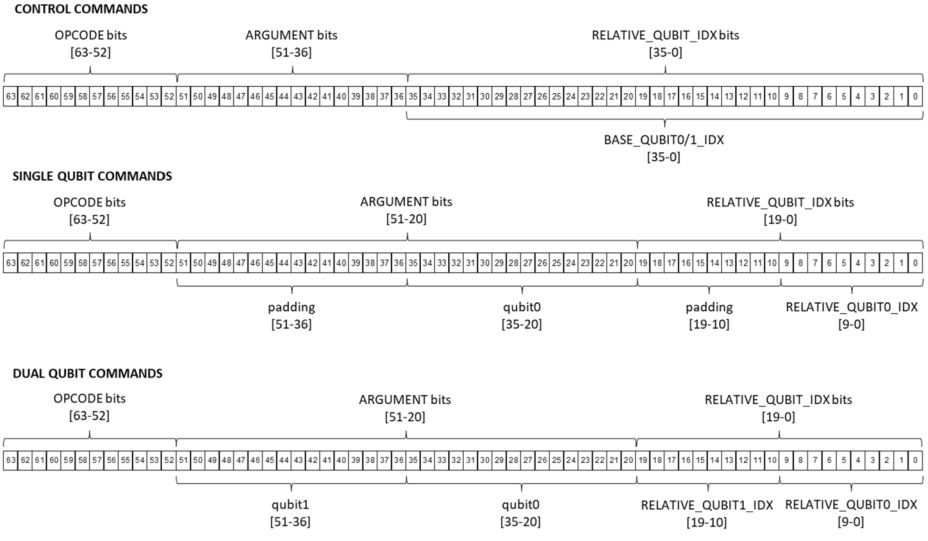

Figure 2 *Bit allocation in the 64-bits long command strings*. A fixed OPCODE field of 12 bits is used to identify the command to execute. Control commands (top string) use 16 bits for the command argument, whereas this field extends to 32 bits for single qubit (middle string) and dual qubit (bottom string) commands. The RELATIVE_QUBIT_IDX field is used to represent the relative index of qubit 0 and 1 (10 bits each) in single qubit and dual qubit commands, whereas it corresponds to the base qubit index in the control commands "Set Page Qubit 0" and "Set Page Qubit 1". In OPCODEs corresponding to single or dual qubit commands, the leftmost bit always flags whether the instruction is a one-qubit or two-qubit gate

a rapid-to-decode logic that relies on ultrafast checks to decide what to execute. For example, the most significant bits—considering a big-endian format—are used to flag whether the command represents a one-qubit or two-qubit gate instruction. We also adopted a fixed-length command identifier (12 bits), which allows the use of lookup tables for the decoder logic thus minimising latency. Figure 2 summarises the bits allocation for each type of command.

2.3 Testing scenarios

Figure 3 shows three different scenarios proposed to test the QHAL on real quantum hardware or emulators provided by our hardware developer partners. These approaches aim at performing a simple Rabi flop operation—the quantum "hello world" to check qubit control and readout across different technologies.

Depending on the access level, the same Rabi flop test must be performed in different ways. At QHAL level 3, the application generates command objects that are subsequently surrounded by Section delimiters (via, for example, a code factory). On the receiving end, within the quantum experiment, these commands are parsed, buffered and converted into low-level instructions ready to be executed. A standard web interface like a Representational State Transfer (REST) can provide sufficient abstraction to send these objects to the experimental setup over the internet. Moving to the QHAL level 2, we need to reduce latency during both the generation and extraction of commands. A protocol

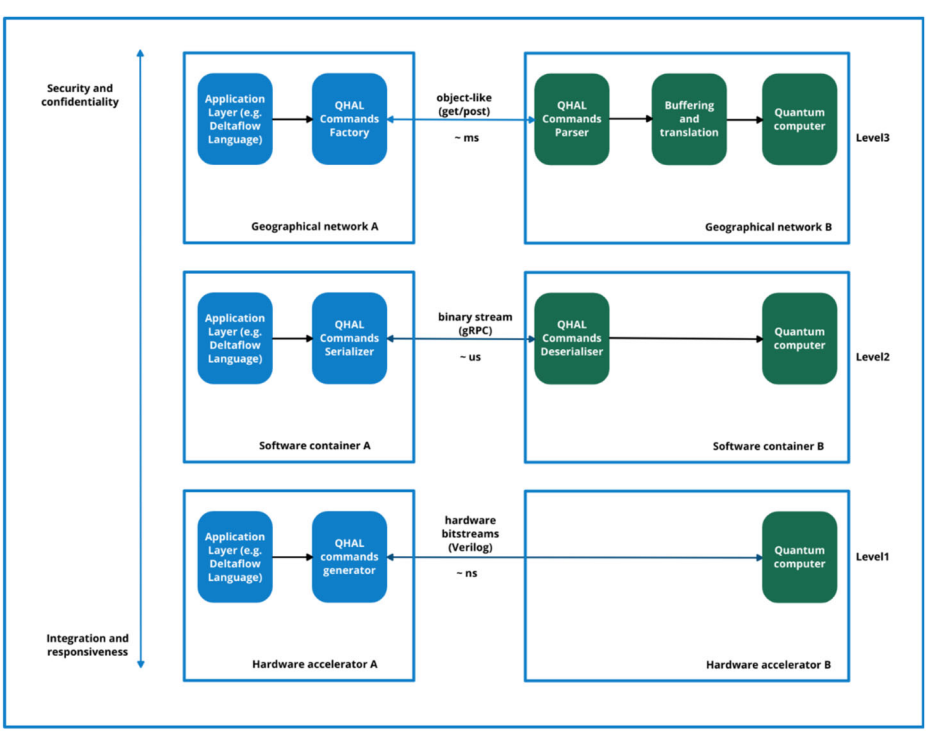

Figure 3 *Testing scenarios for the QHAL*. Steps to transmit a commands sequence from the application layer to the quantum computer at QHAL level 3 (top), 2 (middle) and 1 (bottom). At QHAL level 3, communications occur through the geographical network infrastructures of the Internet. At QHAL level 2, communications occur through software containers—abstractions at the application layer packaging code and dependencies together. At QHAL level 1, communications occur through hardware accelerators, which perform specific functions with greater efficiency than if they had been completed on a generic CPU

like gRPC (a high-performance Remote Procedure Call) executed in a local network or as inter-process communication can meet the more stringent timing requests. Given that the QHAL level 2 operates via native gates, the translation phase is removed, and additional latency is shaved off. Finally, at QHAL level 1 the maximum speed in response can be achieved via a direct implementation, in this case by running within the control hardware (FPGA or ASIC) of the experiment.

Preliminary tests have been performed using simulators and real hardware made available by the hardware manufacturers members of the consortium. For instance, control commands to simulators and hardware for all four hardware technologies (photonics, silicon, superconducting and trapped ions) have been delivered through a QHAL level 3 connection; in these cases, the application layer and QHAL command factory were implemented locally where the commands were then converted into low-level instructions and processed on the simulator/hardware.

At this stage, tests for communications through QHAL level 2 have also been achieved on a trapped-ion quantum computer, but QHAL level 1 access has not been attempted yet. In fact, unlocking maximum benefit from operating at level 1—which is, processing of QHAL commands on the fly during circuit execution—requires advanced hardware engineering capabilities, such as the possibility to translate the binary-coded commands into the analog pulse sequences fed to the qubits with low enough latency, as well as the possibility to implement mid-circuit measurements. Until such capabilities are developed, the practical advantage of operating at the lowest QHAL level cannot be fully appreciated; yet, we may expect that future implementations of smarter, more programmable control solutions will enable QHAL integration and real-time processing deeper in the control system.

3 Discussion and conclusions

One of the ideas central to the design of the 3-level QHAL is that different algorithms and applications will require different amounts of access to the intermediate feedback available from the quantum computer when the application is running. For simple applications, one may not need any intermediate access to the results. For algorithms such as aVQE [11], one needs access to the outcome of a circuit execution to determine the next circuit to run. In other cases, such as certain implementations of Shor's algorithm [14] and quantum phase estimation [10], access to the results of qubit measurement is required to determine what gate needs to be run *in the same circuit*. This realisation led to the identification of a minimum number of levels for the QHAL architecture to provide the most effective interface to each type of algorithms.

Importantly, a distinctive feature of QHAL as a quantum assembly language and application programming interface is its lower-level position within the quantum stack with respect to existing quantum assembly languages. In fact, QHAL goes one step further than other IRs by defining an Application-Binary Interface, as opposed to an API. This defines bitstring representations of quantum instructions which can be interpreted by control system hardware; as such, any quantum computer which can understand QHAL binaries will be able to execute QHAL programs. Quantum compilers, tools, and programs adopting this common interface will thus be shareable across the industry. In addition, the bitstring representation has been optimised for minimised latency, which is a critical parameter for the successful operation of key tools of the quantum stack.

One application that demands ultra-low latency is, for instance, quantum error correction (QEC). In QEC, measurements of auxiliary qubits need to be rapidly transferred to a decoder, which then computes the errors that have afflicted qubits in the system. For its successful implementation, there needs to be fast communication between a qubit control system and a decoder so that the decoding loop occurs well within the decoherence time of qubits, otherwise new errors would be introduced faster than they can be detected and corrected. For this application, specifying corrective actions following the detection of errors using existing quantum assembly languages or intermediate representations, which would need to be written out, transferred, parsed and processed, would likely introduce too much time overhead. In contrast, QHAL specifies every quantum instruction as a 64-bit word, which can be encoded, transmitted, and interpreted quickly by hardware. This, combined with low-latency methods to translate the binary-coded commands into pulse sequences, may provide a viable strategy to perform quantum error correction. Yet, the effectiveness of this approach, which aims at preserving platform transferability, with respect to alternative, platform-tailored solutions for fast quantum error correction, will require further assessment.

The QHAL also lends itself to natural integration in compiler applications deep within the control system which can complement existing languages and IRs, such as QASM and QIR. While there will always be a need for hardware-specific transpilation and optimisation in the backend, the tools which perform and enable these operations can be shared across the industry.

This is not an exhaustive list of QHAL applications. Classical accelerators, classical programs, and quantum programs can be qubit-agile, so long as they and the underlying quantum hardware implements the QHAL. This fosters the sharing of technology and accelerates the advancement of the field.

Throughout the journey to develop the QHAL a number of lessons have been learnt. Firstly, in the rapidly changing and developing quantum computing ecosystem, for the QHAL to remain relevant it will need to be an evolving specification that will change as the requirements of hardware manufacturers, software developers and end users advance. One example of a change which has been implemented since the first iteration of the QHAL is the development of an OpenQASM-to-QHAL compiler to increase user adoption by supporting interoperability with other standards [6]. The purpose of this compiler is to enable easier access to the benefits of the QHAL without the user having to convert circuits which they may already have defined in OpenQASM themselves. Other improvements under discussion aim to allow the hardware providers to expose custom, device-specific, operations to the users via the metadata. This will enable the running of applications such as boson sampling which use operations that might be too specific to define in the cross-platform QHAL specifications.

As discussed throughout this report, security with regards to access is a topic which is of utmost importance to any hardware manufacturer who employs the QHAL. A pain point for hardware manufacturers is allowing low-level access (i.e., QHAL level 1) into their quantum stack in a secure way. They do recognise today's benefits for operating at this level (such as allowing users to run mid-circuit measurements) and the ever-increasing value that minimising latency will have in the future as the industry continues to make advances and move out of the NISQ era and closer toward the fault tolerant era of quantum computing. To do this in a portable, qubit agile manner, such as how the QHAL has

been designed, then low-level access will be required. As such, ensuring that threats aiming at damaging the hardware uptime or at exposing IP are prevented by the QHAL architecture are a top priority to ensure broad adoption from hardware manufacturers. At the same time, any strategy adopted to increase security must be cleverly devised to minimise impact on performance upon integration of QHAL to the quantum hardware. Other concerns around supportability and maintainability of a QHAL level 1 implementation also need to be addressed. For this low-level implementation to gain traction within the industry it needs to ensure smooth compatibility with any future update or replacement of the control system. Moreover, partitioning of the low-level systems must be carefully designed to avoid any negative impact of the QHAL on the correctness of operations in such a complex environment.

Further work remains to be done to understand whether the QHAL will be able to address all the present and future challenges of the quantum computing industry. Yet, the fast progress shown by the consortium demonstrates that gathering a team of different organisations with complementary specialisations is an effective work strategy to address a problem that affects the entire quantum computing ecosystem. By dividing the workload between hardware and software specialists and sharing knowledge, hardware manufacturers can focus on the evolution of their platforms while software experts can maximise the functionality of the hardware available through software development and coordinate future interoperability efforts.

Acknowledgements
The authors would like to thank Innovate UK for funding the NISQ.OS project (48482) which brought the consortium members together. The work discussed in this paper has been achieved by the collaboration and input from many people across a number of organisations. As such, the authors would like to thank and acknowledge the following people and organisations for their contributions; Paul Gleichauf and Hugo Vincent from Arm, Oliver Thomas and Naomi Solomons from Duality Quantum Photonics, Normann Mertig from Hitachi, Tobias Lindstrom from National Physical Laboratory, Tom Harty, Chris Ballance, and Vera Schafer from Oxford Ionics, the Oxford Quantum Circuits team, Ksenija Brankovic, Joe Donlan, Gianmarco Girau, Maria Maragkou, Luigi Matiradonna, Alex Moylett, Leonie Mueck, Brendan Reid, and Robin Sterling from Riverlane, Jerome Javelle and the wider SEEQC UK team, and Can Nur and Louise Aherne from Universal Quantum.

Funding
The work outlined in this manuscript was funded by Innovate UK as part of the NISQ.OS project (48482).

Abbreviations
API, application programming interface; ASIC, applicationspecific integrated circuits; aVQE, accelerated variational quantum eigensolver; CNOT, controlled NOT; CPU, central processing unit; FPGA, fieldprogrammable gate arrays; I/O, inputoutput; IP, intellectual property; IR, intermediate representation; NISQ, noisy intermediatescale quantum; OPCODE, operation code; QEC, quantum error correction; QHAL, quantum hardware abstraction layer; QPU, quantum processing unit; R&D, research and development; RISC, reduced instruction set computer; RPC, remote procedure call; SDK, software development kits.

Availability of data and materials
All supporting data can be found within the material referenced within the manuscript.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
All authors consent to the publication of this manuscript.

Competing interests
The authors declare no competing interests.

Author contributions
Kenton M. Barnes, Anton Buyskikh, Nicholas Y. Chen, Gabriel Gallardo, Marco Ghibaudi, Daniel S. Underwood, Abhishek Agarwal, Deep Lall, Ivan Rungger, and Nikolaos Schoinas developed the QHAL. Matthew J. A. Ruszala, Abhishek Agarwal, Deep Lall, Ivan Rungger, and Nikolaos Schoinas wrote the main manuscript. Matthew J. A. Ruszala prepared the figures. All authors reviewed the manuscript.

Author details
[1]Riverlane, St Andrew's House, 59 St Andrew's St, Cambridge, CB2 3BZ, United Kingdom. [2]National Physical Laboratory, Teddington, TW11 0LW, United Kingdom.

Received: 20 December 2022 Accepted: 31 August 2023 Published online: 13 September 2023

References

1. Geller A. Introducing quantum intermediate representation (QIR). Q# Blog. 2020. https://devblogs.microsoft.com/qsharp/introducing-quantum-intermediate-representation-qir/.
2. McCaskey A, Nguyen T. A MLIR dialect for quantum assembly languages. 2021. https://doi.org/10.48550/arXiv.2101.11365.
3. McCaskey AJ et al. XACC: a system-level software infrastructure for heterogeneous quantum-classical computing. 2019. https://doi.org/10.48550/arXiv.1911.02452.
4. Cross AW et al. OpenQASM 3: a broader and deeper quantum assembly language. ACM Trans Quantum Comput. 2022;3(3):1–50. https://doi.org/10.1145/3505636.
5. Smith RS et al. A practical quantum instruction set architecture. 2017. https://doi.org/10.48550/arXiv.1608.03355.
6. QHAL—quantum hardware abstraction layer. qhal.npl.co.uk.
7. Schroeder MD, Saltzer JH. A hardware architecture for implementing protection rings. Commun ACM. 1972;15(3):157–70. https://doi.org/10.1145/361268.361275.
8. Foss-Feig M et al. Holographic quantum algorithms for simulating correlated spin systems. Phys Rev Res. 2021;3(3):033002. https://doi.org/10.1103/PhysRevResearch.3.033002.
9. Kitaev A et al. Classical and quantum computation. Providence: Am. Math. Soc.; 2002. www.ams.org. https://doi.org/10.1090/gsm/047.
10. Somma RD. Quantum eigenvalue estimation via time series analysis. New J Phys. 2019;21(12):123025. https://doi.org/10.1088/1367-2630/ab5c60.
11. Wang D et al. Accelerated variational quantum eigensolver. Phys Rev Lett. 2019;122(14):140504. https://doi.org/10.1103/PhysRevLett.122.140504.
12. Grimes JD et al. The Intel I860 64-bit processor: a general-purpose CPU with 3D graphics capabilities. IEEE Comput Graph Appl. 1989;9(4):85–94. https://doi.org/10.1109/38.31467.
13. Xilinx. DDR4 controller. https://www.xilinx.com/products/intellectual-property/ddr4.html.
14. Monz T et al. Realization of a scalable Shor algorithm. Science. 2016;351(6277):1068–70. https://doi.org/10.1126/science.aad9480.

Publisher's Note