CEN/CLC/JTC 22/WG 3 "Quantum Computing and Simulation"

Convenor: **Paul Alexandra Mme**

Hardware_Abstraction_Layer_draft

Document type	Related content	Document date	Expected action
Project / Draft	Meeting: VIRTUAL 25 Nov 2025	2025-11-24	**INFO**

Description

Dear members,

Please find a first draft regarding the possible new WI on "Hardware Abstraction Layer".

Kind regards,

CEN/CLC/JTC 22/WG 3 **N** ##

CEN/CLC/JTC 22/WG 3 "Quantum Computing and Simulation"
WG Secretariat: **xxNSBxx**
Convenor: **xxWGCHAIRxx**

CEN-CLC-JTC 22_##_Hardware_Abstraction_Layer_FR

Document type	**Meeting**	**Document date**	**Expected action**
Contribution	JTC22-WG3-014	2025-06-06	For decision

Title	Proposal for addressing the hardware abstraction layer
Authors	Juan Carlos Boschero (juan.boschero@tno.nl), Rob F.M. van den Brink (rob.vandenbrink@delft-circuits.com), Michele Amoretti (michele.amoretti@unipr.it), Valentin Torggler (v.torggler@parityqc.com), Michael Fellner (m.fellner@parityqc.com)
Organisations	TNO, Delft Circuits, University of Parma, Parity QC
Representing	NEN, UNI, DIN, ASI
Work Item number	N/A
Work Item title	Hardware Abstraction Layer; functional requirements
Summary	This work provides an initial table of content, augmented with functionalities and descriptions of the tasks the HAL may offer.
Motivation	
Details (also next page)	

Contents

1. Scope and Objectives

The HAL functions as the intermediary between the Assembly Layer and the Control Software Layer, providing a standardized interface for compilers and higher-level software while concealing vendor-specific hardware complexities. Its primary role is to enable portability of quantum algorithms across diverse hardware platforms and streamline integration within modular quantum computing environments. The HAL translates low-level instructions, ideally in binary, from higher layers into proprietary instructions or function calls for lower layers, such as the hardware backend controlled by its associated software layer. It also manages resource allocation and scheduling, while providing detailed hardware information, including qubit topology, supported gate sets, and error metrics. Beyond these core tasks, the HAL incorporates advanced capabilities, including virtualization to ensure multi-user isolation, configuration of quantum error correction mechanisms, and adaptive behavior based on real-time hardware feedback. Additionally, the HAL maintains connectivity with external systems through the Communication Unit, supporting cloud-based execution and hybrid computing workflows.

The HAL supports similar functionalities as the Instruction Set Architecture (ISA) within the Control Software Layer, but the way these functions are invoked differs significantly.

- The ISA may offer a proprietary interface to handle instructions and/or function calls, while the HAL should offer a standardized interface to handle a sequence of standardized instructions and translate them for execution by the ISA
- The ISA typically exposes a fixed set of low-level instructions and queries that are tightly coupled to the hardware model, designed for direct execution on a specific backend without runtime adaptation. The HAL may generalize them into a common set of instructions. Furthermore, the HAL may generalize them into a set of logical instructions for a fault-tolerant quantum computing architecture in which quantum information is encoded into a quantum error correction code. An exemplary set of logical instructions may comprise only T gates and Pauli measurements, for instance.

In addition, the HAL offers a dynamic invocation model to support the concept of dynamically linked libraries. Different from static linking, which occurs at compile time of a program, dynamic linking occurs at execution time. This capability allows for extendable functionalities, such as:

- standard interfacing and associated instruction translation via vendor-specific plugins dedicated to a specific hardware back-end. This prevents hardcoding vendor-specific logic, in each HAL implementation.
- error correction and fault-tolerant quantum computation strategies, offered by third parties as plug-in, and tailored to a variety of hardware back-ends
- optimized circuit libraries, possibly offered by third parties as well, that can be called without recompiling code in higher layers. For instance, calling a Quantum Fourier Transform via a single QFT instruction

As the ecosystem evolves, this dynamic approach will be critical for supporting multiple vendors, diverse programming languages, and advanced features.

The HAL also offers virtualization and resource management to bring the illusion to each user of exclusive access to the hardware back-end.

2. Standardized Instruction Set

For further study. But see how the ISA is specified within the Control Software Layer, to get a good idea: See CEN-CLC-JTC 22-WG 3_N175_CryogenicSolidState_Draft08.pdf

3. Dynamic Linking

The HAL may implement a (dynamic) linking mechanism, analogous to DLLs or Dynamic Shared Objects in classical computing, enabling modularity and extensibility across diverse quantum hardware platforms. Rather than hardcoding vendor-specific logic, the HAL loads vendor-provided plugins at runtime using standardized interfaces. Each plugin may encapsulate proprietary instruction translation, hardware control routines, and optimized algorithm libraries, allowing the HAL to dispatch calls dynamically based on the active hardware backend. This approach supports multiple vendor packages concurrently, enabling seamless switching between hardware stacks without recompilation of higher layers. For example, when a compiler requests execution of a quantum circuit, the HAL identifies the target hardware, loads the corresponding plugin, and links any required algorithm libraries provided by third parties. Plugins may be versioned and may be validated for integrity, ensuring compatibility and security. This architecture allows vendors to innovate independently while maintaining interoperability, and it provides users with a unified interface that abstracts hardware complexity while leveraging vendor-specific optimizations dynamically.

Currently, most quantum software stacks rely on static linking, where vendor-specific logic and libraries are compiled into the system at build time. This approach simplifies deployment but limits flexibility, especially as the ecosystem grows to include multiple hardware vendors and diverse programming environments.

In the future, dynamic linking will become increasingly important. By loading vendor plugins and libraries at runtime, the HAL can support heterogeneous hardware platforms without requiring recompilation of higher layers. This dynamic approach enables seamless integration with multiple programming languages, facilitates rapid updates, and allows vendors to innovate independently while maintaining interoperability.

3.1 Example: Fault Tolerant Quantum Computing Library

The HAL is responsible for abstracting FTQC and QEC functionalities so that higher layers can utilize error correction without requiring hardware-specific knowledge. It may support multiple QEC schemes, such as surface codes, concatenated codes, or other vendor-defined approaches, and provide a standardized interface for configuration. Through this interface, the HAL exposes critical parameters including error rates, thresholds, and syndrome extraction capabilities, enabling compilers and schedulers in higher layers to make informed decisions about circuit optimization and resource allocation. The HAL should allow dynamic QEC configuration based on real-time hardware feedback, such as changes in qubit fidelity or calibration status, ensuring that error correction strategies remain adaptive and efficient. By centralizing QEC control and reporting, the HAL ensures consistent execution across diverse hardware platforms, while maintaining transparency for higher layers that need to optimize performance without managing low-level error handling and control.

Error correction functionality introduces logical qubits, which are constructed from multiple physical qubits to provide greater robustness against noise and operational errors.

- *Physical qubits* represent the raw hardware-level quantum states, which are highly susceptible to decoherence and gate errors.
- *Logical qubits* combine many physical qubits into a single error-protected unit, or logical qubit, allowing the use of the qubit for robust computation.

To support QEC, the control software layer must handle additional queries similar to those defined by the ISA, such as operations for syndrome extraction, error decoding, and logical gate application. This requires a flexible library design, often implemented through dynamic components and plugins, to accommodate evolving error correction strategies.

QEC can be implemented in both static and dynamic forms:

- *Static QEC* applies a fixed error-correcting code throughout execution. This is simpler to implement but less adaptable to changing error profiles or hardware conditions.
- *Dynamic QEC* by contrast, adjusts error correction strategies in real time based on observed error rates, resource availability, and hardware feedback. This requires a flexible architecture capable of loading different QEC routines at runtime, similar to how the HAL uses dynamic linking for vendor plugins.

While QEC does not necessarily need to reside within the HAL, future best practices may favor integrating QEC capabilities into the HAL to streamline resource management, scheduling, and hardware-aware optimizations.

3.2 Example: Optimized Circuit Libraries

For further study

3.3 Example: Mapping & Routing Libraries

For further study (might be part of QEC)

3.4 Example: Instruction translation

For further study. Should also translate proprietary interfaces of the control software layer into a standardized instruction interface.

4. Virtualization & Resource Management

The HAL incorporates virtualization and resource management to enable multi-user operation while preserving the illusion of exclusive access for each user. The resource manager dynamically allocates qubits, memory buffers, and timing channels based on user requirements and system constraints. It also implements scheduling policies that optimize for different parameters, throughput, and latency, allowing concurrent execution of multiple workloads without compromising performance. Virtualization extends beyond qubit allocation to include isolation of error correction resources and control channels, ensuring that each virtual instance behaves as an independent quantum system. By abstracting these complexities, the HAL provides a seamless experience for higher layers, enabling cloud-based quantum services and hybrid computing environments to scale securely and efficiently.