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

Convenor: **PAUL Alexandra MME**

Nxx_HAL_Libraries

Document type	Related content	Document date	Expected action
Project / Draft	Meeting: VIRTUAL 11 Feb 2026	2026-02-06	**INFO**

Description

Dear members,

Please find attached a document with the extending HAL functionalities with libraries.

Best regards,

Extending HAL functionalities with libraries

Draft 01

Date of submission:	2026-02-xx
Submitted by:	Rob F.M. van den Brink (Rob.vandenBrink@Delft-Circuits.com) Michele Amoretti Juan Boschero
Expected action:	Vote
Expected date:	2026-02-11 (JTC22/WG3 meeting)
WG3-Project:	Hardware Abstraction Layer

---

1. Abstract

Document N201 gives a first Draft for a Technical Specification (TS) about the hardware abstraction layer. This first draft is currently a preliminary table of contents only.

The present contribution proposes literal text for inclusion into chapter 7 of that first draft, which is dedicated to the concept of libraries. This text is not complete, but already good enough to make a step forward

2. Literal text proposal

Start of literal text proposal

7. Libraries

The HAL should support a software mechanism to let its functionality grow with future needs, and/or to improve existing functionalities. Rather than hard-coding vendor-specific logic in the HAL, the HAL should be able to load vendor-provided plugins (preferably in runtime) offering these new/improved functionalities.
This approach supports the integration of multiple vendor packages concurrently, and a convenient adaptation of the HAL to different hardware stacks. All without recompilation of higher layers. It also allows vendors to innovate independently while maintaining interoperability, and to provide users with a unified interface that abstracts hardware complexity.

The common software approach for offering this flexibility is the use of "plug-in" libraries. To make this happen the HAL should not only define the supported mechanism, but also a common way of internal interfacing to let the library interwork with other parts of the HAL. This means two aspects:

- A software mechanism to let libraries run, preferably by reusing existing mechanisms for *dynamic linking* of the underlying operating system. Examples are the use of the Dynamic Linking Library (DLL) mechanism within Windows, or the use of Dynamic Shared Objects mechanism within Linux.
- A mechanism to tell the HAL that a library is operational, what its characteristics are (name, version, ..) and to intercept instructions that are to be handled by the inserted library.

An example of the latter is the interception of query instructions about capabilities so that new capabilities can be reported. Another example is that instructions for modifying the state of qubits become less error-prone after installing a new library with an improved Quantum Error Correction mechanism.
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. Plugins may be versioned and may be validated for integrity, ensuring compatibility and security.

Currently, most quantum software stacks rely on *static linking* in stead of *dynamic linking.* This means that vendor-specific logic and libraries are to be linked into the system at compile-time, and are to be implemented in the same programming language as the rest of the stack. This monolithic approach may simplify initial deployment but limits flexibility, especially as the ecosystem grows to include multiple hardware vendors and diverse programming environments.
The use of static linking is not excluded by the HAL, but in the future the use of dynamic linking will become increasingly important. By loading vendor plugins and libraries at run-time, 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.

The sub sections hereafter will discuss a few powerful libraries that are currently foreseen. These sub sections are examples only and may grow with future developments.

7.1 Library for Fault Tolerant Quantum Computing (FTQC)

This section is left for further study

7.2 Library with Optimised Circuits

The library mechanism allows the HAL to offer vendor-specific optimised circuits to higher layers, which are fully tailored to the involved hardware in lower layers. For example, an optimised QFT (Quantum Fourier Transform) circuit provided by a hardware vendor is expected to be the best possible implementation of the QFT algorithm for the hardware of that vendor. Here, best means the one that can be executed with the minimum amount of execution time. Thus, it is expected that the optimised circuit is composed by *native* gates supported by the underlying hardware. We remind that the name *native* gate refers to an operation for changing

the quantum state of a register by means of a “single” physical action on one or more qubits simultaneously, as explained in chapter 5.

Circuits that are frequently used as building blocks for applications, are:

- Quantum Fourier Transform - QFT
- Phase Estimation
- Grover Operator
- Logical operators (AND, OR, XOR, etc.)
- Bit-Flip Oracle Gate
- Diagonal Gate
- (circuit for preparing a) Graph State
- Phase Oracle

These circuits are suitable candidates for being provided in a library as vendor-specific optimised circuits.

7.3 Library for Mapping and Routing

This section is left for further study

End of literal text proposal