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

Convenor: **PAUL Alexandra MME**

HAL_Instructions

Document type	Related content	Document date	Expected action
Meeting / Document for discussion	Meeting: VIRTUAL 17 Jun 2026	2026-05-30	

Description

Dear members,

Please find attached an input for the WI "HAL".

Best regards,

Simon Del Nin

Instructions supported by the HAL

Date of submission:	2026-05-28
Submitted by:	Rob F.M. van den Brink (Rob.vandenBrink@Delft-Circuits.com)
Supported by:	Michele Amoretti (University of Parma) Abel Martínez Suárez (CTIC) Niels Neumann (TNO)
Expected action:	Vote
Expected date:	2026-06-17 (JTC22/WG3 meeting)
WG3-Project:	HAL

Abstract

Document N239 gives the latest Draft (V03) for a Technical Specification (TS) about the Hardware Abstraction Layer. Chapter 6 is aimed for instructions and gates, and the content on "instructions" is still to be elaborated.

But the amount of text dedicated to each of them justifies a change in structure of the overall draft TS. This means that text on "instructions" and "gates" deserve separated chapters (new 6 and 7), and that moving associated parts of the existing text in chapter 5 to the new chapters 6 or 7 will improve the overall readability of the draft TS.

This contribution proposes literal text for inclusion into a chapter 6 and 7 of the HAL, and is mainly dedicated to instructions. It denotes (in red) which parts of chapter 5 should be moved to where.

Literal text proposal

Start of literal text proposal

5. Overview

This original chapter will be reduced to the present text up to first section.

- The original content of section 5.1 is incorporated in the new chapter 6
- The original content of section 5.2 is to be moved to the new chapter 7
- The original content of section 5.3 is incorporated in the new chapter 6
- The original content of section 5.4 remains in chapter 5

6. Quantum Instructions

A quantum instruction is a request or order to perform an executable step within a quantum program or task. Instructions are encoded, such as in a human-friendly ascii

format (as common in assembly languages) or -preferably- in a computer-friendly binary format (as in byte-code representations). These instructions have to be executed in a specific order, as dictated by some quantum algorithm.
A single instruction is encoded as a sequence of bits or bytes representing the associated operator and zero or more operands. In principle, instructions are executed one-by-one, but instructions can consist of multiple parallel quantum operations on distinct qubits.

Higher layers may push these instructions into a buffer within the HAL or the HAL pulls these instructions out from a buffer within higher layers. This enables the HAL to pause the execution of an instruction flow for a while to give priority to other tasks. For instance, to execute instructions in another flow or to perform a warm-start calibration of the hardware.
It depends on the kind of instruction if the HAL can simply hand it over to the control software layer, if the HAL should translate it first into something that will be understood by the control software layer, or if the HAL can execute the instruction itself.

The HAL supports a variety of instructions, organised as:

- Initialisation instructions
- Inquiry instructions, about
  - capabilities of hardware
  - organisation of qubits
  - supported gates
  - qubit or register performance
- Execution instructions, about
  - preparation/initialisation
  - check pointing
  - selecting libraries
  - pulse-level instructions to change the state
  - gate-level instructions to change the sate
  - measurement-level to readout the state
- Fault handling instructionsI

These instructions are explained below, and the text is (obviously) quite similar to the specification of an ISA in standard TSxxxxxx about Cryogenic Solid State Quantum computing.

6.1 Initialisation instructions

Initialisation instructions are to prepare the setup for quantum computing tasks. These instructions are typically sent at the beginning of a calculation sequence by higher layers, such as the HAL. But they may also be re-sent as often as needed. At least the following instructions should be supported by the HAL:

- "*def shapes*": construct a data structure with predefined wave shapes for pulses.
- "*def bases*": construct a data structure with predefined bases to measure individual qubits.
- "*def registers*": group the qubits into registers and allocate indices to individual qubits. This grouping can be defined from higher layers, such as the HAL, or selected from one or more predefined register grouping.
- "*def meta instructions*": store a pre-defined sequence of instructions into a data structure, callable via a single instruction name, so that it can be played-back as if it was a new (user definable) instruction.
- "*set mapping*": construct qubit topological mapping.
- "*set lanes*": allocate numbering to lanes of pulsing channels.

- "*run calibration*": call a calibration from a set of predefined calibration procedures, to prepare a setup.

It might be possible that future systems will allow for a user-definable grouping of qubits to arbitrary registers. In that case the definitions of lane indices and mapping will also change.

6.2 Inquiring instructions

Inquiring capabilities is a mechanism for higher layers to identify relevant information about the setup, without performing any calculation. A quantum compiler or interpreter should have full knowledge about this in order to optimise generated code. To prevent that each compiler is targeted only to a single system, it could start with inquiring these capabilities via the HAL.

Most of these capabilities are hard-coded in the control software layer by the vendor, who has full knowledge about the hardware. In such cases, the HAL only needs to relay queries and replies between layers above and below the HAL. A conversion between uniform and proprietary interfaces might be the only functionality that the HAL should add to offer this functionality.

6.2.1 Inquiring the organisation of qubits

Qubits are usually grouped into one or more quantum registers. A quantum register is a system comprising multiple qubits, each with its own index. All available qubits can be fixed members of a single register, can be spread out over multiple (smaller) registers, or can be spread out over a user-definable grouping into registers.
If multiple registers are being used, each of them has a unique identifier to address the individual registers. When the setup has this capability, it should support instructions on how they can modify these registers.

The HAL should support the handling of several queries about how qubits are organised and how they perform. This includes information about:

- *Width*: The HAL should report the number of available qubits for each quantum register and how they are organised within these registers. It can also specify whether all qubits are part of a single quantum register or if they are allocated to multiple (smaller) registers. The use of multiple registers may occur when using modular hardware architectures.
- *Depth*: The HAL may report on qubit performance by means of the maximum depth for circuits of gates that can be executed before the calculated result becomes unreliable. This value is related to the coherence time of the implementation and other imperfections of the underlying hardware.
- *Connections*: The HAL should report an adjacency matrix for each quantum register to indicate which qubits are edge-connected. For instance, when a register has N qubits, then this adjacency matrix $\mathbf{C}$ has a size of $N \times N$. The default of each element in this matrix is false, but if qx and qy are the indices of two adjacent qubits, then $\mathbf{C}(qx,qy)=\mathbf{C}(qy,qx)=$true. Matrix $\mathbf{C}$ is a symmetric matrix since $\mathbf{C}(k,r)=\mathbf{C}(r,k)$.

Queries about the *depth* of circuits are described below in the section on *qubit or register performance*.

The HAL should therefore support at least the following capability instructions:

- "*Get register set*": Return a list with identifiers of each of the supported quantum registers.
- "*Get register width*": Return the number of available qubits for each individual quantum register.
- "*Get adjacency matrix*": Returns an "adjacency matrix" for each individual quantum register, to specify which qubits in the register are edge-connected.
- "*Get lane matrix*": Returns a "lane matrix" for each individual quantum register, to specify for each pulse generator to which qubit it can be connected. For instance, when *Np* pulse generators can be connected to *Nq* qubits in that register, then the associated lane matrix **L** has size *Np*×*Nq*. The default of each element in this matrix is false, but if pulse generator px can be connected with qubit qy then **L**(px,qy)=true. Returning an empty matrix means that this limitation does not exist.

Different kind of qubits can be distinguished when inquiring system specifications, such as:

- Physical qubit: A noisy quantum system in which a two-dimensional Hilbert space can be encoded.
  - Data qubit: data qubits are physical qubits used for storing and processing quantum information.
  - Measurement qubits: physical qubits used for stabiliser measurements of quantum error-correcting codes.
  - Communication qubit: physical qubits specifically designed to perform entanglements to other communication qubits on non-local registers. They are also entangled to other qubit types within the quantum computers and are primarily used to mitigate information for distributed quantum operations.
- Logical qubit: An error-corrected quantum system whose state lies in a two or higher-dimensional Hilbert space.

6.2.2 Inquiring supported gates

A native gate is a gate that can be executed within a "single pulse interval" using one or several simultaneous pulses. If a gate requires multiple pulses in sequence, then it is called a compound gate. If a compiler has full knowledge about which gates are native and which are compound, then it can optimise a compiled program in terms of minimal execution time.
The HAL may also support shortcuts for commonly used (sequences) of native gates as if they were a single gate. Those shortcuts are called primitive gates. An example may be the well-known Pauli gates that are implemented as (a sequence of) rotations. All primitive gates that are not native are assumed to be compound gates.
The HAL should support at least the following gate inquiry instructions:

- "*Get primitive gates*": Returns an identifier list of all gates that are understood by the ISA. This includes both single qubit as well as multi qubit gates. Examples are:
  - Rx(a), Ry(b), Rz(c), X, Y, Z, etc
  - Rxx(a), Rxy(a), cX, cZ, SWAP, cNOT, etc
- "*Get native gates*": Return an identifier list of a subset of primitive gates that can be executed within a single pulse interval.
- "*Get gate matrix*": Returns for each primitive gate the associated matrix defining the operation.

- "*Get gate duration*": Returns for each primitive gate the operation time.

6.2.3 Inquiring qubit or register performance

Inquiring performance information is a mechanism to enable higher layers to infer the maximum circuit depth and other quality parameters. This means the number of time slices that can be executed before the calculation becomes unreliable. Examples are specifying the error of primitive gates, expressing a fidelity for each gate, or simply specifying a maximum circuit depth. This value is related to coherence time of the implementation and other performance parameters of each connection between qubits. Details about these mechanisms are still to be elaborated, but they should facilitate higher layers, such as compilers, to identify one or more of the following performance parameters:

- "*Get gate performance*" such as:
  - Coherence times, e.g. (T1, T2), where "T1" refers to state decoherence time and "T2" refers to phase decoherence time.
  - Fidelity for certain gates.
  - Qubit pulse error rates.
  - Read out error to indicate how accurate a measurement can be
- "*Get instruction duration*": List of durations for each identified instruction (operations, pulses, read out times, etc.). Note that this is a generalisation of the afore mentioned "get gate duration" instruction
- "*Get qubit properties*": List of properties of each identified qubit e.g. (T1, T2, qubit type, position coordinates, etc.)

6.2.4 Inquiring supported libraries

The functionality of the HAL can be extended by plug-in libraries, as further explained in chapter ZZ. The HAL shall support instructions to identify which libraries are available and what their capabilities are. This includes:

- "*Get libraries*": Returns an identifier list, version info, etc of all libraries that are available in the HAL.

The kind of capabilities supported by each library is library-dependent, so query instructions of these capabilities are to be handled by the involved library. If a specific library supports circuits that are optimised for the involved hardware, it may report which gates it can emulate. In that case it may respond to:

- "*Get library gates*": Returns an identifier list of all gates that are emulated in the library and can be called as if it was a primitive gate.

6.3 Execution instructions

Execution instructions are instructions that are meant to change the state of qubits for calculation or measurement purposes, or to prepare for such actions.

6.3.1 Preparation and initialisation instructions

- "*Reset calculation*": call a procedure for re-calibration the setup from a set of pre-defined calibration procedures. One should only adjust for small changes in the setup due to drift or other imperfections of the setup, which can be done in a reasonably short period of time. As such it does not refer to a full calibration as described in "initialisation instructions".
- "*Reset qubits*": Change the states of all qubits of a selected register into predefined ones, which can be identified as their "|0>" state.
- "*Reset flags*": Set all binary flags to "false" of a selected conditional register.
- "*Set flag X*": Set an individual binary flag "X" of a selected conditional register to a selected value.
- "*Select Library*": Activate a particular library, such that it can preprocess instructions. For instance, to emulate a gate that is not supported by the ISA or to emulate Fault Tolerant Quantum Computing. The concept of libraries is further explained in chapter XX.

6.3.2 Check-point instructions

The HAL virtualises the "simultaneous" handling of multiple jobs by concurrent execution of multiple workloads. This mechanism is further explained in chapter YY. The main consequence of that is that the HAL has to pause a running job after some time, in order to execute other jobs or tasks.
Interrupting a running job at random point may destroy the quantum state of the qubits and will destroy a current quantum calculation. Therefore, pausing a job (without destroying it) must be done at carefully defined moments. Such moments occur when the HAL detects an instruction for resetting the state of a full quantum register. But in many other cases this detection needs assistance from a compiler or user that inserts dedicated checkpoints in the instruction sequence.

A *checkpoint instruction* is a no-operation instruction that informs the HAL that it is not harmful to interrupt a running job. Since the HAL raises a pausing flag to higher layers, the paused job can execute dedicated instructions (when needed) to undo the harm when it starts running again.

6.3.3 Selecting library instructions

The functionality of the HAL can be extended by plug-in libraries, as further explained in chapter ZZ. This includes an extension of emulated gates or the virtualisation of a Fault Tolerant Quantum Computer with less qubits and more reliability.
If such a library is detected via an associated inquiry instructions and a higher layer would like to take advantage of some of these libraries, then they can be selected by proper instructions.
It will cause that the library performs extra processing on (some of) the instructions, and/or that the instruction set is extended with extra instructions.

6.3.4 Pulse level instructions

Control electronics can fire multiple pulses in parallel to change the state of qubits, and before they are actually fired by the electronics, the HAL must support hardware-specific instructions to prepare these pulses for each qubit individually. However, the available hardware limits the number of pulses that can be fired simultaneously. When more

qubits are to be controlled than this limit, a switching matrix is to be used to reach all qubits of interest. They can offer so called "lanes" to connect a qubit to a particular pulse generator. To save hardware such a switching matrix may support only a restricted set of lanes, and the HAL should be aware of such limitations.
The HAL should therefore support the following (low-level) instructions:

- "*select qubits*": Prepare for the connection of a selected set of qubits with available lanes, through which pulses will be transported thereafter. The actual connection may be delayed until a "wait" or "fire" instruction is handled.
- "*select pulses*": Prepare for each lane of interest a predefined pulse that is to be transported thereafter. The actual firing of pulses may be delayed until a "fire" instruction is handled.
- "*fire pulses*": Fire an ensemble of the selected pulses through the selected lanes for a specified duration, when the selection of lanes and pulses is completed. This instruction will actually execute a specific native gate by changing its state into a specific phase. During this duration, the ISA can prepare for other lanes and pulses, to be used for firing a next ensemble of pulses.
- "*wait*": impose a selected amount of delay before a next ensemble of pulses can be fired. During this duration, the ISA can prepare for other lanes and pulses, to be used for firing a next ensemble of pulses.

6.3.5 Gate level instructions

An alternative to pulse-level instructions that is a bit more hardware agnostic is the use of gate-level instructions. The ISA in the control software layer has full knowledge on what (sequence of) lanes and pulses are needed to execute primitive gates. This knowledge frees higher layers of the burden to know the exact implementation of lanes and required pulses. As such, gate level execution instructions can be regarded as higher in abstraction than pulse level execution instructions.
The HAL should therefore support the following instructions:

- "*select gates*": Prepare for an ensemble of gates to be executed simultaneously on selected qubits. The actual connection may be delayed until a "wait" or "fire" instruction is handled. This includes the selection of potential conditional gates.
- "*fire gates*": Fire a sequence of simultaneous pulse-delay combinations to the selected qubits in order to execute the selected native gates.

6.3.6 Measurement level instructions

The HAL should support instructions to examine the state of one or more qubits in a quantum register by means of a measurement. The answer will be returned as a binary string stored in a dedicated bit-register. Note that a state will always collapse after such a query. The HAL should also support instructions to read out the bits in this state-register and/or to use these bits for instructing controlled gates. If the hardware supports it, a HAL may also provide instructions to specify or change the basis for these measurements.

The HAL should therefore support the following instructions:

- "*select basis*": prepare for an ensemble of selected qubits the pulse from the "basis library", that are needed to measure the state of these qubits in a specific basis.

- "*fire measurements*": Fire a sequence of simultaneous pulse-delay combinations to measure the selected qubits in a specific basis, detects their response and store the results in the associated conditional bits.
- "*read state register*: Read selected conventional bits in a state register that are set by the results of an associated measurement instruction.

6.4 Fault handling instructions

These instructions are about classical means to handle all kinds of "classical errors". But to avoid confusion with "quantum errors" the word "fault" is consistently used here to denote those "classical errors".
Each time a fault is raised, the setup should increment the associated fault counter. This enables higher layers to read-out these fault counters and to perform proper fault handling.
Each time a fault occurs, the HAL can raise an interrupt to higher layers to inform them about a fault that has occurred. These interrupts can be masked individually to prevent them being raised.
Examples of such fault counters are:

- Raise exceeding pulse duration time.
- Raise parameter overflow/underflow faults (when phase is out of bounds, index of libraries, lanes or qubits are wrong, etc.).
- Raise pulse timing faults (in the event that pulses are too close together, pulsed at too short times, etc.).
- Raise runtime faults (in the event that a channel is used to pulse a qubit that is not contained in channel lane, etc.).
- Raise memory faults (instruction is too long).
- Raise connection faults (if during an active hybrid session server disconnects).
- Raise adjacency faults, if direct entanglement is requested between two non-adjacent qubits.
- Raise lane faults, if a pulse has to be sent to a qubit while the associated lane cannot be set-up.

Most of these faults are raised by the ISA in the control software layer and they may bypass the HAL to inform layers above the HAL directly.

The HAL should therefore support readout and (re)setting instructions of these fault counters and flags, including:

- "*Set fault mask all*": force for all faults that the HAL (or ISA) will raise an interrupt to higher layers when a fault occurs.
- "*Set fault mask X*": force for fault "X" that the HAL (or ISA)will raise an interrupt to higher layers when such a fault "X" occurs.
- "*Get fault last*": returns the identifier (or index) of the last fault being encountered.
- "*Get fault all*": returns an array with the content of all fault counters and flags.
- "*Get fault X*": returns the content of fault counter or flag "X".
- "*Reset fault all*": Resets all fault counters and flags to zero or false.
- "*Reset fault masks*": Resets all masks to false, so that no fault will raise an interrupt to higher layers.
- "*Reset fault X*": Resets fault counter or flag "X" to zero or false.

7. Quantum gates

A quantum gate is a quantum operation that transforms the quantum state of one or more qubits into another state. It can be considered as an elementary calculation (native, primitive or predefined in a library) within a quantum computation sequence.

7.1 Queries about supported gates

NOTE: This section will contain the present text in section 5.2 in draft03 of the HAL. This covers:

- reporting the definition of each gate being supported
- reporting if gates are native or primitive
- reporting emulated gates

7.2 Predefined quantum gates

NOTE: This section will contain the present text in section 6.2 in draft03 of the HAL. This covers:

- naming conventions
- definition of single qubit gates
- definition of double qubit gates
- definition of multiple qubit gates

End of literal text proposal