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

Convenor: **PAUL Alexandra MME**

HAL_resource_management

Document type	Related content	Document date	Expected action
Meeting / Document for discussion	Meeting: Delft (Netherlands) 29 Apr 2026	2026-04-23	

Description

Dear members,

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

Best regards,

Simon Del Nin

CEN/CLC/JTC 22/WG 3 **N 238**

CEN/CLC/JTC 22/WG 3 "Quantum Computing and Simulation"
WG Secretariat: **AFNOR**
Convenor: Alexandra Paul

CEN-CLC-JTC 22_##_HAL_resource_management

Document type	**Meeting**	**Document date**	**Expected action**
Contribution	JTC22-WG3-014	2026-04-23	For decision

Title	HAL Resource Management
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) Abel Martínez Suárez (abel.martinez@fundacionctic.org), Alejandra Ruiz (alejandra.ruiz@tecnalia.com)
Organisations	TNO, Delft Circuits, University of Parma, Parity QC, CTIC, Tecnalia
Representing	NEN, UNI, DIN, ASI, UNE
Work Item number	N/A
Work Item title	Hardware Abstraction Layer; functional requirements
Summary	This work provides the contributions to sections 3 and 4 of the HAL document. Notably, introduces the concept of partition sequences.
Motivation	This section provides detailed specifications for resource management
Details (also next page)	

7. Resource Management

The HAL incorporates resource management dynamically by allocating 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 ensures that each virtual instance behaves as an independent quantum system. By abstracting these complexities, the HAL provides a seamless experience for higher layers, enabling multi-user operation, cloud-based quantum services and hybrid computing environments to scale securely and efficiently.

A job refers to an instruction flow that contain the complete execution of an application of a single user. This could be for instance a single fixed circuit that has to run many times or a hybrid quantum application involving sequential or interleaved quantum program calls. This enables execution time to be shared across multiple users, as jobs can be temporarily paused to allow execution of another user's job. Therefore the resource manager has the following means to cut in jobs

- ***Pausing*** a job to hand-over execution time to another user. Such an interruption will not destroy the quantum calculation.
- ***Breaking*** a job when it takes too long before a convenient moment is found to pause the job. Such an interruption will destroy parts of a quantum calculation. and the resource manager should raise a fault to a higher layer so that it can redo the broken calculation at a later moment in time .
- ***Killing*** a job when it exceeds agreed allocated time or resource quota, or a higher layer has explicitly asked for it.

To facilitate the pausing of a job, the resource manager must have a robust mechanism to identify when it can pause a job, without destroying ongoing quantum calculations.

7.1 The concept of partition sequences

To enable fair and efficient multi-user operation, the resource manager within the HAL supports the concept of *partition sequences.* These are logical blocks of instructions treated as atomic units after which a quantum calculation can be paused without harm and without compromising correctness or isolation between jobs from different users.

For example, if a user runs a job with 100,000 shots, or executions of a circuit, then a partition can be allocated to each shot. Assume that hardware calibration is required every 1,000 shots, then the resource manager can pause the job after 1000 partitions, allow a lower layer to perform a warm-start calibration, and then resume execution from the next partition. Something similar occurs when jobs from multiple users are running "simultaneously".

This mechanism is known as *snap shotting*. Snap shotting enables partitions from different users' partition sequences to be interleaved and scheduled according to priority, thereby ensuring fairness and maximizing throughput under *fair-share scheduling.* When the resource manager decides to hand-over execution time to another job it waits a reasonable amount of time for detecting the end of a partition. If detected in time, it will pause that job, but if it takes too long the resource manager will break that job, as explained before. Pausing means that the execution context of a job is preserved, but they relinquish control of the hardware until the scheduler returns them to the top of the priority queue.

This decision making is controlled by priority, derived from the number of concurrent users waiting for execution time or from an interrupt request received from a lower layer. For instance when a lower layer needs to run a warm start calibration or even a system restart.

Figure 7-1 shows an example of how the resource manager within the HAL may process partition sequences from multiple users. It depends on the priority of other jobs how many partition sequences may run without being paused. Sometimes it can be many partitions, and sometimes pausing will occur directly after a first partition.

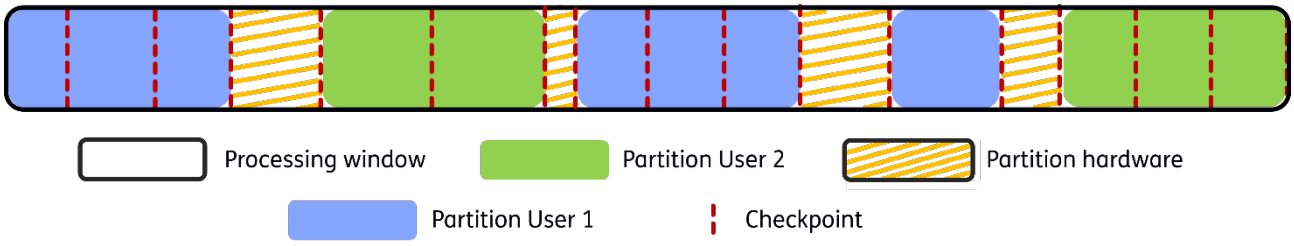

Figure 7-1: Representation of a processing window that can be subdivided into scheduled partitions with checkpoints for multiple users.

This mechanism combines dynamic scheduling and resource management, allowing concurrent workloads to share hardware efficiently while maintaining isolation and performance guarantees. Although the HAL's current focus is on abstractions for quantum hardware, its architecture intentionally does not exclude classical high-performance computing components. The HAL is conceptualized as a collection of extensible libraries, and nothing prevents the inclusion of HPC-oriented modules, for example, routing to GPU clusters for simulation or pre-processing, or coordinating with distributed classical workflows. However, these capabilities remain outside the present scope of the HAL's core definition.

Future iterations may integrate HPC more deeply as hybrid quantum-classical pipelines become increasingly important, but for now, the HAL concentrates on providing robust, secure, and scalable interfaces to quantum hardware itself.

7.2 The concept of checkpointing

It may be obvious that the resource manager cannot always automatically detect when a user completes a subprocess or reaches a natural yield point marking the end of a partition. There are exceptions, such as when instructions are encountered that reset all quantum registers. These *checkpoint instructions* are useful for indicating the start of a new partition. However, if they do not occur frequently enough, the resource manager must interrupt the job and defer further handling to higher layers.

The solution for preventing that is the use of *checkpointing*. These are explicit no-operation instructions, with the only purpose to indicate where a job can be paused. In other words, to demarcate the end of a partition.

It is the responsibility of a higher layer, such as a compiler, to insert sufficient checkpoint instructions. Otherwise it must accept the risk that a job will suddenly be broken by the resource manager and that (parts of) the quantum calculation gets lost.

Natural places where a compiler can insert checkpoint instructions is at loop iterations, measurement batches, and synchronization points.