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

Convenor: **Lefebvre Catherine Mme**

Proposal for adding ISA_functionality

Document type	Related content	Document date	Expected action
Meeting / Document for discussion	Meeting: VIRTUAL 27 Nov 2024	2024-11-07	**INFO**

Description

Dear expert,

Please find attached a proposal for adding ISA functionality linked with the Work Item project on Cryogenic solid state quantum computing.

This document will be discussed during our next meeting.

Kind regards,

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

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

CEN-CLC-JTC 22_N##_Proposal_for_adding_ISA_functionality

Document type	**Meeting**	**Document date**	**Expected action**
Contribution	JTC22-WG3-014	2024-28-05	For decision

Title	Text proposal for ISA in the control software layer
Authors	Juan Carlos Boschero (TNO), juan.boschero@tno.nl Rob van den Brink (Delft Circuits), Rob.vandenBrink@Delft-circuits.com
Organisation	TNO
Representing	NEN
Work Item number	N/A
Work Item title	Cryogenic solid state quantum computing
Summary	This document proposes literal text about the ISA functionality for inclusion into chapter "8 Control Software". The present text is focussed on the section on "functional descriptions", but summarizes some of the topics for the section on "functional requirements" as well
Motivation	The legacy document proposes a software driver layer that includes the ISA without a description. This proposal adds a detailed description as well as a diagram showing the ISA/software layer operating in relation to other layers with user input.
Details (also next page)	See next pages. • Add section 8.1 • Add section 8.2

Start of literal text proposal

8. Layer 4 – Control Software

The control software refers to the software systems and tools designed to manage, coordinate and optimize operations dictated by higher level languages. Thus, the software plays a crucial role in translating higher-level quantum assembly instructions into executable instructions that can be processed by quantum processors.

This layer may include an instruction set architecture (ISA), error correction and calibration functionalities (as seen in Figure 8.1).

- ***ISA*** (Instruction set architecture) refers to a lower-level method of defining operations on a quantum computer. Instead of defining specific gates, this layer defines gates (or other instructions) as operations, using pulses pulsed for a certain time, on specific qubits. An example of an instruction set architecture is pulse level programming where a user can specify wave pulses on qubits instead of gates. This requires knowledge of the system's control equipment as well as the topology and qubit nature.
- ***Error correction*** refers to all low-level techniques to enable error-robust physical operations. Error correction is mitigated by the implementation of strategies to detect and correct potential errors that can occur during the computation.
- ***Calibration*** refers to low-level methods to stabilize the hardware by continuous monitoring of hardware performance to maintain optimal operation.

8.1 Functional Descriptions

8.1.1 Instruction Set Architecture

The aim of an instruction set architecture (ISA) is to convert a sequence of machine-specific instructions from higher layers into commands for the control electronics, to control individual qubits. As such the ISA has full awareness of the underlying quantum hardware and its topology.
Due to the ISA's knowledge on the quantum hardware, it also has the responsibility of handling the execution timings and scheduling of individual instructions, such that higher layers should only know their sequence.

On input, the ISA may receive for instance instructions from higher layers to change the quantum state of qubits (gate-instructions, read-out qubits (measurement instructions)) or any other instructions to interact with all available qubits. These instructions can be fed to the ISA as for instance (binary) machine instructions, as (ascii) human readable instructions, or as function calls. Instructions intended for controlling one or two qubits may be fed one by one to the ISA, but it is more efficient if an ISA can handle many of them in parallel as a "vector" of instructions to interact with an ensemble of qubits simultaneously.

Higher layers may push these instructions into a buffer within the ISA each time the ISA signals to be ready for it. Alternatively, an ISA may also poll these instructions out from a buffer within higher layers each time the execution of a previous group of instruction has completed. This includes the polling of requests and instructions given by many users. In all cases, it requires a well-defined interface (API) with the above layer(s), as well as a well-defined instruction set language (such as *OpenPulse* [1] or *Pulser* [2]).

An ISA may handle gate-level instruction as well as pulse-level instructions. Both may be mixed in a single compilation pass for bypassing specific gate instructions, which can be parsed in the SDK by the user. Gate level instructions are considered to be any set of operations that can be parsed onto universal gate based quantum computers regardless of the hardware while pulse level instructions are operations that are heavily dependent on the system's physical architecture. The ISA will thus support instructions to specify the exact waveform of a pulse to be fired to a specific qubit, as well as an ensemble of pulses where each pulse has its own waveform and relative delay.

The common way of sending instructions to the ISA are via higher level layers such as the asembler or programming layer. Alternatively a user may be allowed by the communication module to access the control software layer directly or via the hardware abstraction layer, and supply ISA readable instructions directly.

On output, the ISA sends commands to the control hardware, to fire for instance pulses to qubits or to read-out their response via a measurement. This requires that the ISA is fully responsible of the timing of all these commands.

If a pulse is to be applied to a qubit, the ISA may calculate its characteristics on the fly, such as waveform / pulse-shape and magnitude. But it may also read predefined characteristics from a library created by other software units, stored somewhere in the control software layer or in the control electronics layer.
In all cases, it requires a well-defined interface (API) with the control electronics(s) as well as a well-defined command-set.

Figure 8.1 illustrates from an example work-flow of how an ISA contextually may interact with other functionalities in the stack. When multiple users access the quantum computer, the communication module verifies at which layers they may access the full stack.

- If a (production) user may only access the stack from the top of the *assembly layer* (or higher layers), the assembler compiler/interpreter will then convert assembly instructions into hardware-abstracted ones for the *hardware abstraction layer*. These instructions are then compiled into machine-specific code by the *control software layer*, where the instruction set architecture optimally schedules the user's program. It also determines the program's placement in relation to other users' tasks, ensuring efficient execution across the system.
- If a more dedicated user/designer may also access the stack from the top of the control software layer, he should be fully aware of the hardware-specific aspects of the quantum computer and generate machine-specific instructions for the ISA himself.

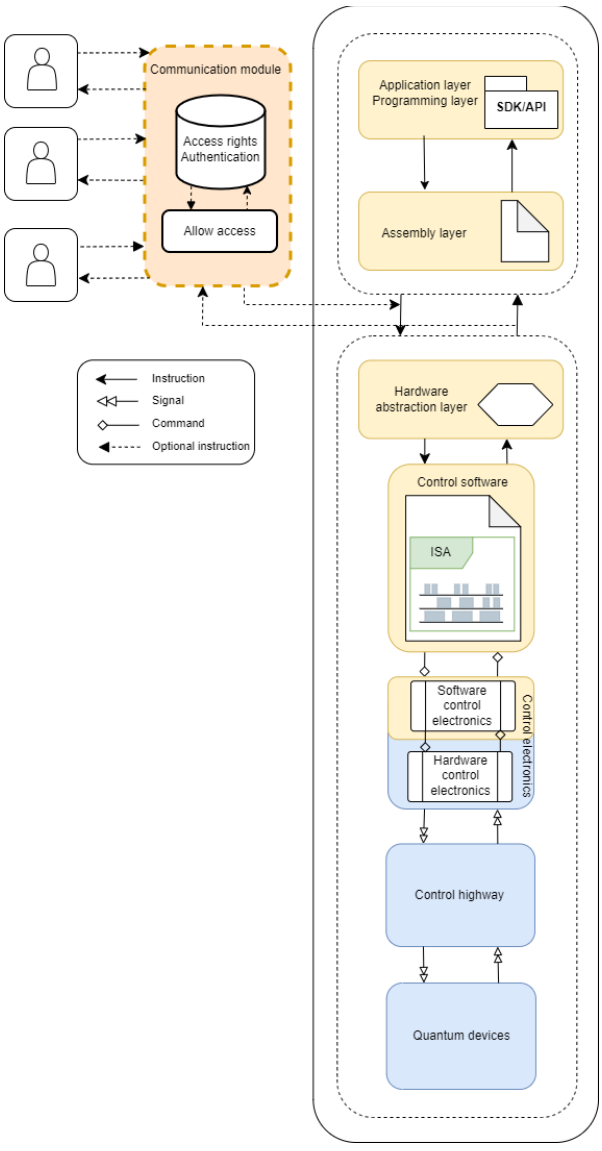

Figure 8.1 Workflow of instruction set architecture functionality. The colored boxes denote the layer (with a corresponding title) and the arrows show the different information type exchanged.

Editor's Note/questions:

- *Should ISA be responsible for knowing priority of users for program scheduling?*
- *Hardware abstraction layer local or non-local?*
- *Instruction, signal and commands correct?*
- *Assembly layer obfuscates many specific commands that may be relevant to the ISA (compiling/decompiling), how do we deal with this?*

8.1.2 Calibration

Editor's Note: Contributions are invited to fill-in this section

8.1.3 Error Correction

Editor's Note: Contributions are invited to fill-in this section

8.2 Functional Requirements

8.2.1 Instruction Set Architecture

Editor's Note: Contributions are invited to fill-in this section. Topics for considerations are:
The instruction set architecture requires:

- Hardware specifications including:
    - Topology of the backend
    - Time parameters of pulses
    - Possible waveforms/pulse shapes
    - Accessible qubits (global vs local pulses)
    - Scheduling restrictions i.e minimum time per sequence
    - Qubit dead times
- Feedback from communication module between device backend and API
- Pre-compilation error handling supplied to user
- Compilation of quantum assembly code to instruction set architecture

Bibliography

[1] Cross, Andrew, et al. "OpenPulse Grammar — OpenQASM Live Specification Documentation." *Openqasm.com*, 2019, openqasm.com/language/openpulse.html. Accessed 6 Sept. 2024.

[2] "Pulser — Pulser 0.19.0 Documentation." *Readthedocs.io*, Pulser Development Team, 2022, pulser.readthedocs.io/en/stable/. Accessed 6 Sept. 2024.

End of literal text proposal