CEN/CLC/JTC 22/WG 3 "Quantum Computing and Simulation"
WG Secretariat: **AFNOR**
Convenor: **Lefebvre Catherine Mrs**

Contribution_RB_and_MA_LayerModel_HardwareAbstractionLayer

Document type	Related content	Document date	Expected action
Meeting / Document for discussion	Meeting: VIRTUAL 28 Sep 2023	2023-09-22	

Contribution on Layer Model Draft 01 from Rob F.M. van den Brink and Michele Amoretti

Functional description of Layer 7; Hardware abstraction layer

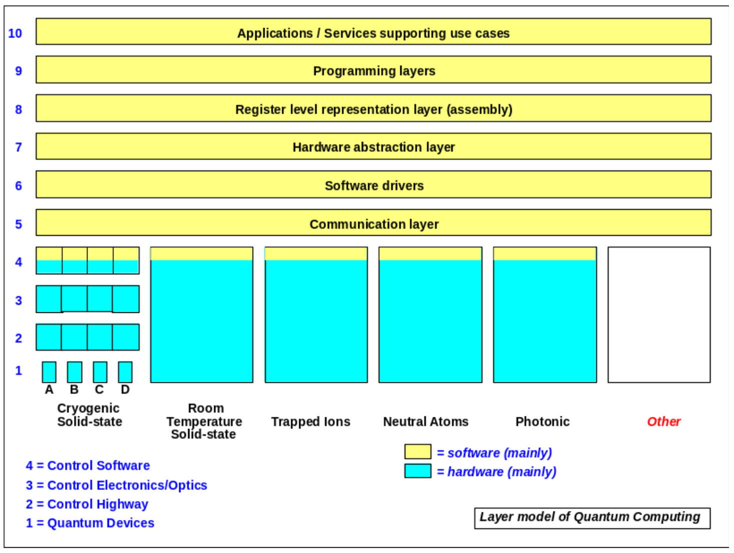
Date of submission:	2023-09-21
Submitted by:	Rob F.M. van den Brink (Rob.vandenBrink@Delft-Circuits.com) Michele Amoretti (michele.amoretti@unipr.it)
Expected action:	Vote
Expected date:	2023-09-28 (JTC22/WG3)
WG3-Project:	Layer Model

1. Abstract

Contribution N19 (to JTC22/WG3) has proposed a first draft of a TR “Layer model for quantum computing” with functional descriptions of each layer. Layer 7 is dedicated to the “hardware abstraction layer”.
This contribution proposes a functional description of that layer, for inclusion in the draft of the aimed Technical Report (TR).

2. Proposal

The layer model that has been described in the FGQT Roadmap Document has several layers in a stack, and the low-level layers are subdivided into multiple hardware stacks. One of these layers is the “*Hardware abstraction layer”*, or simply HAL, to abstract away the quantum computer implementation details while keeping the performance. A functional description of that layer is being proposed here. The intro is taken from the originating FGQT Roadmap Document and extended with further details.



Start of literal text proposal

X. Layer 7: Hardware abstraction layer

The aim of the Hardware Abstraction Layer is to inform higher layers with capabilities and limitations supported by the underlying hardware. Layers above the HAL can use this information to hide many implementation-specific details to higher layers by offering a more unified interface.
Not all quantum computers make use of the same paradigm. Annealing quantum computers behave differently from gate-based quantum computers, and therefore their HALs might be different as well.
The HAL can therefore provide information about the underlying architecture, such as for instance being “gate-based” or “annealing”.

X.1 HAL for gate-based quantum computers

A gate-based quantum computer processes a sequence of instructions to change the state of a quantum register with many qubits before the resulting state is queried by measurements. A convenient graphic representation of such a sequence has the appearance of a circuit where the elements seem to operate on one or more qubits simultaneously. Due to this convenient graphic representation, these instructions are called gates.

X.1.1. Organization of qubits

Quantum register: A quantum register is a system comprising multiple qubits. The HAL supports instructions to operate on such a register, for initializing, changing and querying its state.

Width: The HAL can specify the number of available qubits and how they are organized in these registers. It can also specify if all qubits are part of a single quantum register or if are they are allocated to multiple (smaller) registers. The use of multiple registers may occur when using modular hardware architectures.

Depth: The HAL can specify the maximum depth for circuits of gates that can be executed before the calculated result becomes unreliable. This value is related to coherence time of the implementation and other imperfections of underlying hardware.

Connections: The HAL can also provide 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 C has size NxN. The default of each element in this matrix is "false", but if qx and qy are the indices of two adjacent qubits then C(qx,qy)=C(qy,qx)="true". Matrix C is therefore a symmetric matrix, since C(k,r)=C(r,k).

The HAL can provide additional information about the underlying architecture.

X.1.2. The concept of native gates

The HAL can specify a list of "native gates" supported by the underlying hardware. 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. An example is a single pulse composition that cannot be broken down into two or more shorter pulse compositions. In other words, if a gate can be divided into two or more shorter independent sequential physical actions, it is not native.

As a result, a native gate can be executed in the minimum amount of execution time. Knowledge about which gates are native is relevant information for compilers that try to optimize a circuit with respect to execution time.
Gates that can only be implemented by a sequence of two or more native gates are called "compound" gates.

The boxed example below illustrates for a specific case that the single qubit gates X, Y, Rx(a), Ry(b) are all native for that implementation, while the gates Z and Rz(c) are compound gates. A similar example can be elaborated with dual qubit gates. For a specific implementation, a gate like CNOT may turn out to be compound as well when it cannot be implemented with one native dual qubit gate.

Example

The concept of native gates can be explained by the following example. Assume that a specific hardware implementation supports a mechanism to rotate a qubit via a "single" pulse composition that can be controlled with two real parameters "a" and "b". Assume that the definition of this rotation function equals:

```
RN(a,b) = [ cos(a/2),              -j*exp(-j*b)*sin(a/2)]
          [-j*exp(j*b)*sin(a/2),                cos(a/2)]
```

Then some of the well known gates can be implemented via:

```
Rx(a) = [   cos(a/2), -j*sin(a/2)] = RN(a,0)
        [-j*sin(a/2),    cos(a/2)]

Ry(b) = [   cos(b/2),   -sin(b/2)] = RN(a,pi/2)
        [   sin(b/2),    cos(b/2)]

Rz(c) = [exp(-j*c/2),       0    ] = RN(pi,0) * RN(pi,-c/2)       * exp(j*pi)
        [          0,  exp(j*c/2)]


X     = [ 0,  1] = Rx(pi) * exp(j*pi/2) = RN(pi,0)                * exp(j*pi/2)
        [ 1,  0]

Y     = [ 0, -j] = Ry(pi) * exp(j*pi/2) = RN(pi,pi/2)             * exp(j*pi/2)
        [+j,  0]

Z     = [ 1,  0] = Rz(pi) * exp(j*pi/2) = RN(pi,pi) * RN(pi,pi/2) * exp(-j*pi/2)
        [ 0, -1]
```

In this hardware implementation, Rx(a), Ry(b), X, Y can be considered as native gates. The gates Rz(c) and Z are to be combined from two sequential native gates, so they are compound. Knowledge about which gates are native is relevant for quantum algorithms that try to find an optimal circuit representation in terms of execution time.

X.1.3 Concept of primitive gates

A compiler or interpreter does not always know how to convert well-known gates into a smart combination of native gates for any possible set of native gates. In those cases, a fall-back situation should be supported by the HAL in terms of predefined solutions for well-known gates like Rx(a), Ry(b), Rz(c), X, Y, Z, H, S, T, CNOT, etc.
Therefore, the HAL can specify a list of "primitive gates" that it can emulate by a sequence of one or more native gates.

X.1.4 Concept of measurement

The HAL supports instructions to query 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 register. Note that the state will be collapsed after such a query. The HAL also supports instructions to read out the bits in this register and/or to use these bits for instructing controlled gates.
If the hardware supports it, the HAL can also offer instructions to specify the basis for these measurements.

X.1.5 Interfacing considerations

A preferred way of communicating with the HAL is by means of binary instructions, preferably common for all quantum computing implementations. Therefore, it requires a list of binary commands for letting the HAL report capabilities and limitations of the underlying hardware, and for executing all aforementioned instructions.
Such an interface may also offer a convenient format for instructing a simulator that emulates a quantum computer with a limited set of qubits.

X.2 HAL for annealing quantum computers

EDITORIAL NOTE: Contributions are invited for adequate text that should fit here.

End of literal text proposal

3. Left for further study

Contributions on topics that should be added to the above literal text are invited. Topics for consideration are:

- HALs for annealing/adiabatic quantum computers?
- Is there a better wording for defining the concept of “native gate” ?
- Do quantum simulators need a more dedicated HAL or is the generic gate-based HAL description adequate for them as well?
- Is a quantum repeater a device that should be covered by this layer model for quantum computers or should it be kept out of the present layer model document since it is another type of device that deserves a document on its own?
- What additional information should a HAL provide about the underlying architecture?

When needed the above proposed text can be extended with relevant explanations and references to associated documents.

4. Suggestions for further reading

- https://arxiv.org/abs/2201.08825
- https://arxiv.org/abs/2305.06687
- https://www.ibm.com/quantum/roadmap
- https://riverlane.github.io/QHAL_internal/dev/index.html
- https://www.gsma.com/aboutus/workinggroups/resources/ig-14-quantum-hardware-abstraction-layer-for-quantum-computing-and-networking