Call: HORIZON-CL4-2022-QUANTUM-01-SGA

Topic: HORIZON-CL4-2022-QUANTUM-01-SGA

Funding Scheme: HORIZON Research and Innovation Actions (RIA)

Deliverable No. 5.1

Low Level Interface – Phase 1 Implementation

Grant Agreement no.: 101113946

Project Title: **OpenSuperQPlus100 -** Open Superconducting Quantum Computers (OpenSuperQPlus)

Contractual Submission Date: 31/08/2024

Actual Submission Date: 29/07/2024

Responsible partner: P08: Qruise (QRU)

Funded by the European Union

Grant agreement no.	101113946
Project full title	**OpenSuperQPlus100 -** Open Superconducting Quantum Computers (OpenSuperQPlus)

Deliverable number	**D5.1**
Deliverable title	**Low Level Interface – Phase 1 Implementation**
Type[1]	R
Dissemination level[2]	SEN
Work package number	WP5
Author(s)	Anurag Saha Roy
OpenSuperQPlus reviewers	
Keywords	Quantum Software Stack, Interface, API

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union.

Neither the European Union nor the granting authority can be held responsible for them.

[1] **Type**: Use one of the following codes (in consistence with the Description of the Action):

R:	Document, report (excluding the periodic and final reports)
DEM:	Demonstrator, pilot, prototype, plan designs
DEC:	Websites, patents filing, press & media actions, videos, etc.

[2] **Dissemination level**: Use one of the following codes (in consistence with the Description of the Action)

PU:	Public, fully open, e.g. web
SEN:	Sensitive, limited under conditions of the Grant Agreement

Table of Contents

Summary ... 4
1 Introduction ... 4
2 Description of Activities ... 5
2.1 Architecture ... 5
2.2 Supported Instructions ... 5
2.3 Site specific customisations ... 6
3 Results & Discussion ... 6
3.1 Example usage ... 6
3.1.1 Amplitude Rabi Experiment ... 6
3.1.2 Two Qubit CZ Gate with Discriminator ... 7
3.1.3 Arbitrary Pulse Gates ... 8
3.2 Experiment Library ... 8
4 Conclusion ... 9
5 Next steps (if applicable) ... 9

Summary

Based on the Low-Level Interface Phase 1 Definition (Milestone 3), an implementation was undertaken to integrate the bring-up software developed by Qruise with the control electronics and firmware provided by Zurich Instruments. This document outlines the result of the implementation task, demonstrating the key results and outcomes of the integration process and then following that up with the upcoming next steps. The Deliverable was already completed at the time of the most recent progress meeting (June 17-18 2024, Budapest) and this report closely follows the results presented to the WP5 partners at that meeting.

1 Introduction

In between the quantum computing hardware and the end-user running experiments on this hardware, there lies a sophisticated stack of control electronics, firmware and middleware responsible for converting high-level user routines to the low-level instructions. A significant amount of effort in WP5 (Enabling Software) is dedicated to building a robust interface at this level which allows for both flexibility of efficient pulse level control required for calibration and characterization experiments as well as the abstractions required for more high-level algorithmic tasks.

The Low-Level Interface Phase 1 Definition document goes into more details outlining these different user types and how that impacts the design of the software and its architecture. In this document we highlight the implementation of this interface and how we used it to build a library of bring-up experiments.

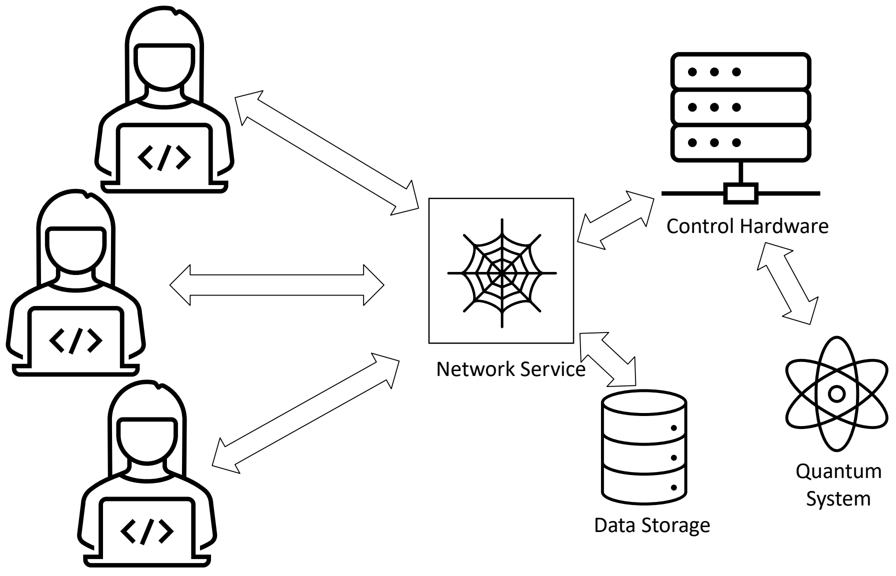


Figure 1: Architecture of Low-Level Interface

2 Description of Activities

2.1 Architecture

The Low-Level Interface mentioned above is implemented using the architecture shown below:

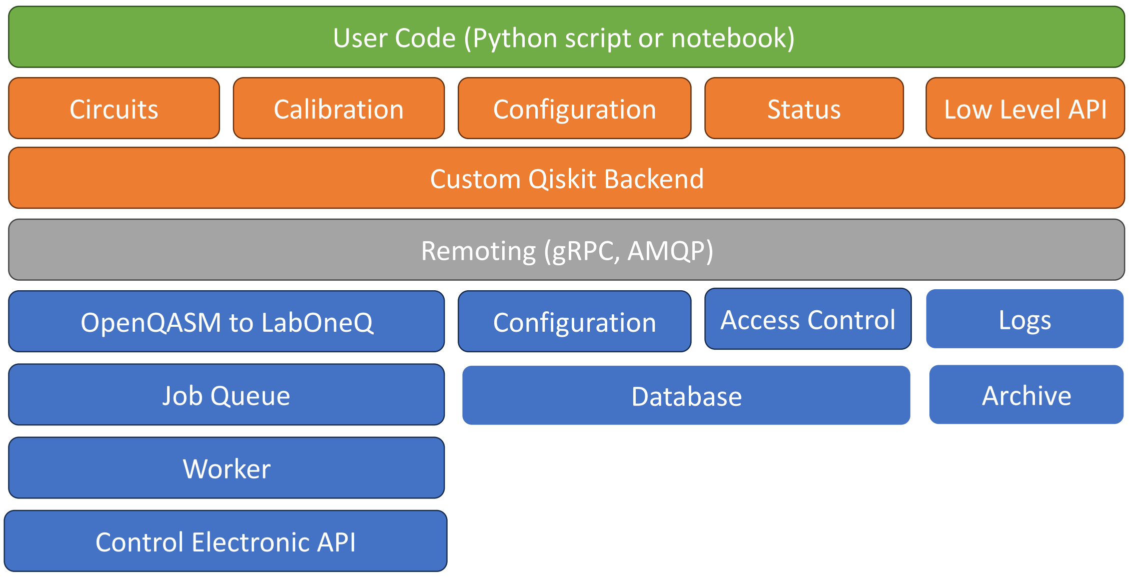


Figure 2: Block Diagram of modules in Low-Level Interface

The key detail to be highlighted in this implementation is the abundant use of open source standards and technologies. For example, we use the Qiskit and OpenQasm 2/3 API for instruction definition, gRPC for network calls, RabbitMQ for job queuing, xarray as data-format etc. This use of open source standards is crucial for various reasons as we note below:

1. Easy interoperability with other software and hardware, both within and beyond the OpenSuperQPlus project
2. No vendor lock-in for data as might be the case with proprietary database solutions
3. Flexibility to swap out different parts of the implementation with alternative solutions that follow the same standards
4. Rich support ecosystem through extensive user-forums and community engagement

During the implementation period, the interface was developed and thoroughly tested on superc-conducting quantum computing hardware installed within the premises of project coordinator Forschungszentrum Jülich. This already served as a preliminary testing ground for this software stack as it underwent regular feedback from users engaged in calibration and characterization of the quantum computer.

2.2 Supported Instructions

As described in the interface document, the phase 1 implementation has support for the following different instructions:

- **Quantum Gates:** Basic single and multi-qubit operations.
- **Measurement Operations:** Instructions to measure the state of qubits and return the results at various levels of pre-processing

- **Reset Operations:** To reset qubits to a known state, usually the basis state.
- **Barrier:** A synchronization point in the circuit, preventing certain types of optimizations across the barrier.
- **Custom Pulse Gates:** User-defined pulses described as an arbitrary waveform to be played on the AWG
- **Arbitrary LabOneQ subroutines:** Only available for elevated access users to perform highly optimized native LabOneQ operations (LabOneQ being Zurich Instruments proprietary software)

2.3 Site specific customisations

While the above sections outline the general implementation of the low-level interface as described in the interface document, deploying this to the sites of the hardware partners in this project will still require customisations tailored to their individual stack. This low-level interface implementation bridges the gap between the Qruise software and the Zurich Instruments firmware, but every quantum computing lab often makes further customisations to their control stack, both in terms of developing some in-house software infrastructure as well as some non-Zurich Instruments control electronics (home grown or off-the-shelf). WP5 (Enabling Software) is dedicated to providing abstractions around these local customisations to ensure end-users do not have to worry about such details and their programs can be portable between different hardware partners. With the general implementation complete, future efforts will be focussed on making site specific customisations to deploy and integrate this software in partner labs.

3 Results & Discussion

3.1 Example usage

In the examples below, we show code snippets that perform various common tasks that might be undertaken by different types of users of quantum computing hardware both within and beyond the OpenSuperQPlus project.

3.1.1 Amplitude Rabi Experiment

Amplitude Rabi is a standard experiment in QPU bring-up to identify the pulse required for flipping a qubit to the excited state.

```
# map qubit configuration
QUBIT_INDEX = 0
qpu = backend.qc_configuration.qpu
QC_QUBIT = qpu.qubits[QUBIT_INDEX]

sweep = tuple(np.linspace(0, 0.3, 151))
qc = QuantumCircuit(1, 1)

x = Parameter("x180_amplitude")
qc.append(QcAcquireLoop(2**10), [0])  # works like shots
qc.append(QcSweep(x, sweep), [0])

qc.rx(x, 0)
qc.measure(0, 0)
qc.barrier()

qc.append(QcCloseLoop(), [0])
qc.append(QcCloseLoop(), [0])


job = backend.run(
    qc,
    qubits=[QUBIT_INDEX],
    dry_run=DRY_RUN,
    meas_level="integration",
    thermalize=True,
    qpu_update=qpu,
)

data = job.result().results[0].data
data = getattr(data, f"qubit{QUBIT_INDEX}_readout0")
```

3.1.2 Two Qubit CZ Gate with Discriminator

The CZ gate is a workhorse of modern transmon-based superconducting quantum hardware. Additionally, pre-calibrated discriminators allow users to have bitstrings as outputs, instead of requiring to manually discriminate between raw signals from the readout pulse.

```
QUBIT_INDEXES = [1,2] # Q2,Q3 are coupled
qc = QuantumCircuit(2,2)

qc.rx(np.pi / 2, 0)
qc.rx(np.pi / 2, 1)
qc.cz(0, 1)
qc.rx(-np.pi / 2, 1)

qc.measure_all()

job = backend.run(
    [qc],
    qubits=QUBIT_INDEXES,
    meas_return="single",
    meas_level="integration",
    thermalize=True,
    discrimination_level="label",
    no_svg=True,
    shots=4096,
)

data = job.result().results[0].data

d =[getattr(data, f"qubit{i}_readout0").values for i in QUBIT_INDEXES]

labels = np.stack(d, axis=1)

counts = defaultdict(int)
for label in labels:
    label = tuple(label)
    counts[label] += 1
```

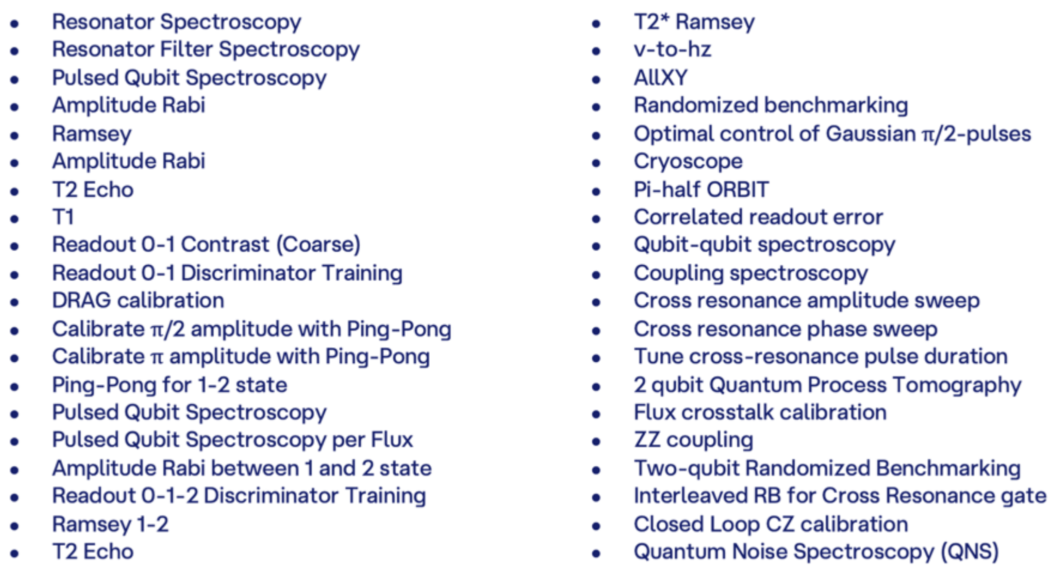
3.1.3 Arbitrary Pulse Gates

The ability to run arbitrary pulse waveforms is crucial not just for system calibration and custom gate definitions but also for algorithms such as the variational quantum eigensolver.

```
circ = QuantumCircuit(1, 1)
# Should be of the form [[I_0, Q_0], ..., [I_N, Q_N]] for N samples
# Sampling rate is 2.4 GSa/s
# We play a constant waveform of 20 nanosecond
q1 = qiskit_backend.qc_configuration.qpu.qubits[0]
rabi_amp = q1.x180_amplitude
rabi_amp_arr = np.linspace(0, rabi_amp, 21)
waveform_arr = [r * (np.arange(20) + 1j * np.arange(20)) for r in rabi_amp_arr]
# The QcPulseGate below acts on 1 qubit with the waveform given as params
# Please set dry_run to False when connected to device
# Applying the gate to qubit [0]
for gate_waveform in waveform_arr:
    pulse_gate = QcPulse(gate_waveform)
    circ.append(pulse_gate, [0])
    circ.measure(0, 0)
job = qiskit_backend.run(
    circ,
    qubits=[0],
    shots=1 << 16,
    meas_return="cyclic",
    meas_level="integration",
    dry_run=True,
)
job_result = job.result()
```

3.2 Experiment Library

We used this low-level interface implementation to create a library of bring-up experiments that will be used for MS14 “Bring-Up Development Version”. This was a fantastic opportunity for dogfooding the software as we used it extensively to program a comprehensive catalogue of calibration and characterisation experiments that span in complexity from very low-level spectroscopy sweeps to high level randomized circuit type benchmarks (both single and two qubit operations). We outline below a list of these experiments as it demonstrates the flexibility and extensibility of the software:



- Resonator Spectroscopy
- Resonator Filter Spectroscopy
- Pulsed Qubit Spectroscopy
- Amplitude Rabi
- Ramsey
- Amplitude Rabi
- T2 Echo
- T1
- Readout 0-1 Contrast (Coarse)
- Readout 0-1 Discriminator Training
- DRAG calibration
- Calibrate π/2 amplitude with Ping-Pong
- Calibrate π amplitude with Ping-Pong
- Ping-Pong for 1-2 state
- Pulsed Qubit Spectroscopy
- Pulsed Qubit Spectroscopy per Flux
- Amplitude Rabi between 1 and 2 state
- Readout 0-1-2 Discriminator Training
- Ramsey 1-2
- T2 Echo
- T2* Ramsey
- v-to-hz
- AllXY
- Randomized benchmarking
- Optimal control of Gaussian π/2-pulses
- Cryoscope
- Pi-half ORBIT
- Correlated readout error
- Qubit-qubit spectroscopy
- Coupling spectroscopy
- Cross resonance amplitude sweep
- Cross resonance phase sweep
- Tune cross-resonance pulse duration
- 2 qubit Quantum Process Tomography
- Flux crosstalk calibration
- ZZ coupling
- Two-qubit Randomized Benchmarking
- Interleaved RB for Cross Resonance gate
- Closed Loop CZ calibration
- Quantum Noise Spectroscopy (QNS)

Figure 3: List of Bring-Up Experiments

4 Conclusion

This report outlines the result of the implementation of the Low-Level Interface Phase 1 description. We elaborate on the various design choices behind this implementation, outline future work on customisation and present the results of using this software implementation for various kinds of use-cases at different abstraction levels. The Deliverable has been fully completed well in schedule and we are now focussed on deploying and integrating this software with quantum computing hardware from project partners.

5 Next steps (if applicable)

The immediate next steps involve the deployment and integration of this software interface at a partner site to test, improve and demonstrate its functionality. This is the task that culminates into MS14 “Bring Up Development Version”. As described in Section 2.3, the general implementation is followed by site-specific customisations that vary depending on the software and hardware infrastructure at partner labs. We are currently working with Chalmers as the first site where the interface implementation reported here will be installed and then will be used for the bring-up of their QPU. This will also be a fertile ground for extensive user feedback to understand how to improve both the interface definition and the interface implementation for phase 2.