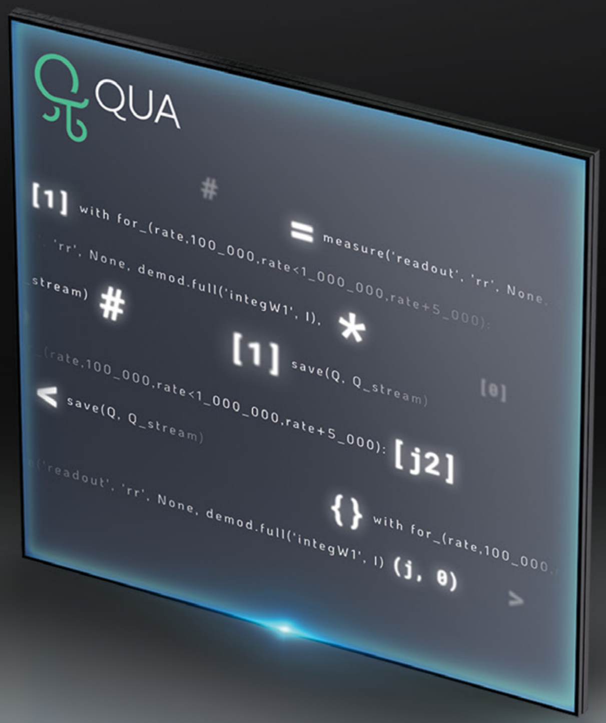
QUANTUM MACHINES

QUA

QUA

Quantum pulse-level programming language: implement the protocols of your wildest dreams, as easily as writing pseudocode


QUA
[1] with for_(rate,100_000,rate<1_000_000,rate+5_000):
= measure('readout', 'rr', None,
'rr', None, demod.full('integW1', I),
stream) #
*
[1] save(Q, Q_stream)
[0]
(rate,100_000,rate<1_000_000,rate+5_000): [j2]
< save(Q, Q_stream)
('readout', 'rr', None, demod.full('integW1', I) (j, 0)
{} with for_(rate,100_000,
>

Accelerate your quantum research and development to unrivaled speeds

Bring out the best of your qubits

Code 1,000 Qubits as Easy as One

Quantum Complete

Native quantum pulse-level operations

```
0  play(‘pi_pulse’, ‘qubit1_xy’)
1  measure(‘readout_pulse’, ‘qubit12’, raw_data, demod.full(‘cos’, result))
```

Turing Complete

Comprehensive classical processing of classical variables

```
2  assign(a, Math.cos(x) * Math.sqrt(y))  # a = cos(x)*sqrt(y)
3  assign(b, Math.abs(z) + Math.ln(w))    # b = abs(z)*ln(w)
```

Comprehensive control flow

```
4  while(...), for(...), if/else(...), switch-case      # nest them !!
```

Quantum Classical

Quantum operations based on classical variables and calculations

```
5  play(‘pi_pulse’*amp(a), ‘q1_xy’, duration=b,         # a from line 2
6        chirp=(Math.cos(a) * Math.exp(b), ‘Hz/nsec’))  # b from line 3
7  measure(‘readout_pulse’, ‘QPC’,                      # signal integration
8            integration.full(‘weights’, result))
9  measure(‘trigger’, ‘laser’,                          # time-tagging
10           time_tagging.analog(timestamps, length, counts))
```

Classical calculations based on quantum measurements

```
11 assign(state_estimation,
12        0.5 * (1 + result * (alpha+beta*C)))    # result from lines 7-8
13 assign(error_syndrome,
14        ancilla_result[0] & ~ancilla_result[1]) # vector of errors
```

Control flow based on classical variables based on quantum measurements

```
15 if_(error_estimation):
16     play(‘pi’*amp(Math.ArgMax(all_states)), ‘q1_xy’) # analog feedback
17 while_(error_syndrome == 0):
18     do_something()                          # user-defined QUA macro
```

Comprehensive Timing Control

Absolute timing control (relative to global time-stamp) and relative timing control (sync and async multi-threading)

Pseudocode vs QUA Code - STOP Algorithm (AWS)

```
initialize: $t=(d-1)/2$; $n_{diff}=0$; countSyn=1;
 SynRep=1; $n_{diff}$Increase=0;
while $test=0$ do
    if $n_{diff}=t$ then
        test=1
    end
    Measure the error syndrome $s_j$.
        Store the error syndrome $s_j$-1
        from the previous round in
        synPreviousRound and the
        current syndrome $s_j$ in
        synCurrentRound.;
    if $countSyn>1$ then
        if $synPreviousRound=synCurrentRound$
        then
            SynRep=SynRep+1;
            $n_{diff}$Increase=0;
        else
            SynRep=0;
            if $n_{diff}Increase=0$ then
                $n_{diff}=n_{diff}+1$;
                $n_{diff}$Increase=1;
            else
                $n_{diff}$Increase=0;
            end
        end
    end
    if $SynRep=t-n_{diff}+1$ then
        test=1;
    end
    countSyn=countSyn+1;
end
```

∗ arXiv:2012.04108

```
while_(test == False):
    if_(n_diff == t):
        assign(test, 1)
    assign_vec(synPrevRound, synCurrRound)              # vector processing
    measure_ZL_syndrome(logical_qubit, synCurrRound) # QUA macro
    if_(countSyn > 1):
        if_(synCurrRound == synPrevRound):
            assign(SynRep, SynRep + 1)
            assign(n_diffInc, 0)
        else_():
            assign(SynRep, 0)
            if_(n_diffInc == 0):
                assign(n_diff, n_diff + 1)
                assign(n_diffInc, 1)
            else_():
                assign(n_diffInc, 0)
    if_(SynRep == t - n_diff + 1):
        assign(test, 1)
    assign(countSyn, countSyn + 1)
```

Key Benefits

Comprehensive
Quantum and Classical

QUA unifies universal quantum operations in their 'raw' format, at the pulse level, with universal classical operations used in classical processing (Turing complete) and comprehensive control flow.

Expressive
As Advanced as It Gets

Code the most advanced programs and run them on hardware with best possible performance.

Natively describe your most challenging experiments, from complex AI-based multi-qubit calibrations to quantum error correction.

Scalable
Grows with You

QUA scales with you to enhance your quantum algorithms and experiments - today and tomorrow. With QUA, coding 1,000 qubits is as easy as coding one qubit. QUA removes limitations in implementing protocols, from the simplest to the most complex.

Quantum Computing

Quantum Sensing

Hybrid Quantum - Classical Algorithmics

Quantum Technologies Research & Development

Quantum Simulations

Quantum Communication

Quantum Firmware Development

QUA is a first-of-its-kind pulse-level programming language that integrates classical processing into the lowest levels of quantum programming in an unprecedented way. It unifies universal quantum operations in their 'raw' format – all the pulse-level stuff used to control and measure qubits – with universal classical operations used in classical processing – all the good stuff you know from Python, Matlab, Java, etc.

Run Your Quantum Experiments with Ease

- Randomized and cross-entropy benchmarking
- Multi-qubit active reset
- Quantum error correction (e.g. cat codes, surface code)
- From Rabi, Ramsey and spectroscopy to neural-net-based calibrations
- Qubit state tracking and qubit stabilization
- Real-time atom sorting
- Bayesian estimation-based adaptive sensing
- Multi-node entanglement distillation
- [Your Next Dream Protocol Here!]

QUANTUM MACHINES

If you wish to learn more:

info@quantum-machines.co

Read and watch short demos in our blog

quantum-machines.co/blog

Quantum Control Systems

OPX+

- All-in-one quantum controller
- Real-time processing and ultra-fast analog feedback
- Diversified qubit technologies

Octave

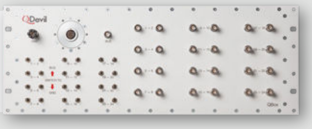
- Auto-calibrated IQ mixing and local oscillator system
- Up/down conversion signals
- Extends the OPX+ range to 18 GHz

QDAC-II

- Advanced signal generation
- Ultra-low noise, high stability
- High bandwidth, many channels

QBox

- Breakout box for DC wiring



About Quantum Machines

Quantum Machines (QM) accelerates the realization of practical quantum computing that will disrupt all industries. Our comprehensive portfolio includes state-of-the-art control systems and cryogenic electronic solutions that support multiple quantum processing unit technologies. QM's OPX family of quantum controllers leverages unique Pulse Processing Unit (PPU) technology to deliver unprecedented performance, scalability, and productivity.

Easily programmable at the pulse-level or gate-level (OpenQASM3), OPX runs even the most complex quantum algorithms right out of the box – including quantum error correction, multi-qubit calibration, mid-circuit frequency tracking, and more. With hundreds of deployments, Quantum Machines' products and solutions have been widely adopted by national and academic research labs, HPC centers, quantum computer manufacturers, and cloud service providers. For more information, please visit quantum-machines.co.