QUANTUM MACHINES

USE CASE

# Quantum Orchestration for Quantum Networks

Run groundbreaking experiments with simple code, using the Quantum Orchestration Platform. Have a look at how quickly you can program the OPX+ to run entanglement and distillation protocols in this real-world use case.

# ENTANGLEMENT DISTILLATION FOR QUANTUM NETWORKS

Quantum networks are one of the most spectacular endeavors towards a future based on quantum technologies, with advancements promised for communication, computation, metrology, and many more [1]. One fundamental building block for quantum networks is the ability to generate high-quality quantum entanglement shared between remote nodes while keeping unavoidable errors and sources of decoherence in check.

One way to purify entanglement is entanglement distillation [2], which offers a trade-off of many not-too-impure entangled states with fewer high-fidelity ones. This protocol is extremely promising in increasing fidelity via local operations, but it has demanding experimental requirements and calls for complex control systems with real-time capabilities. Such demands have harshly limited its experimental explorations, producing strong friction for the entire field.

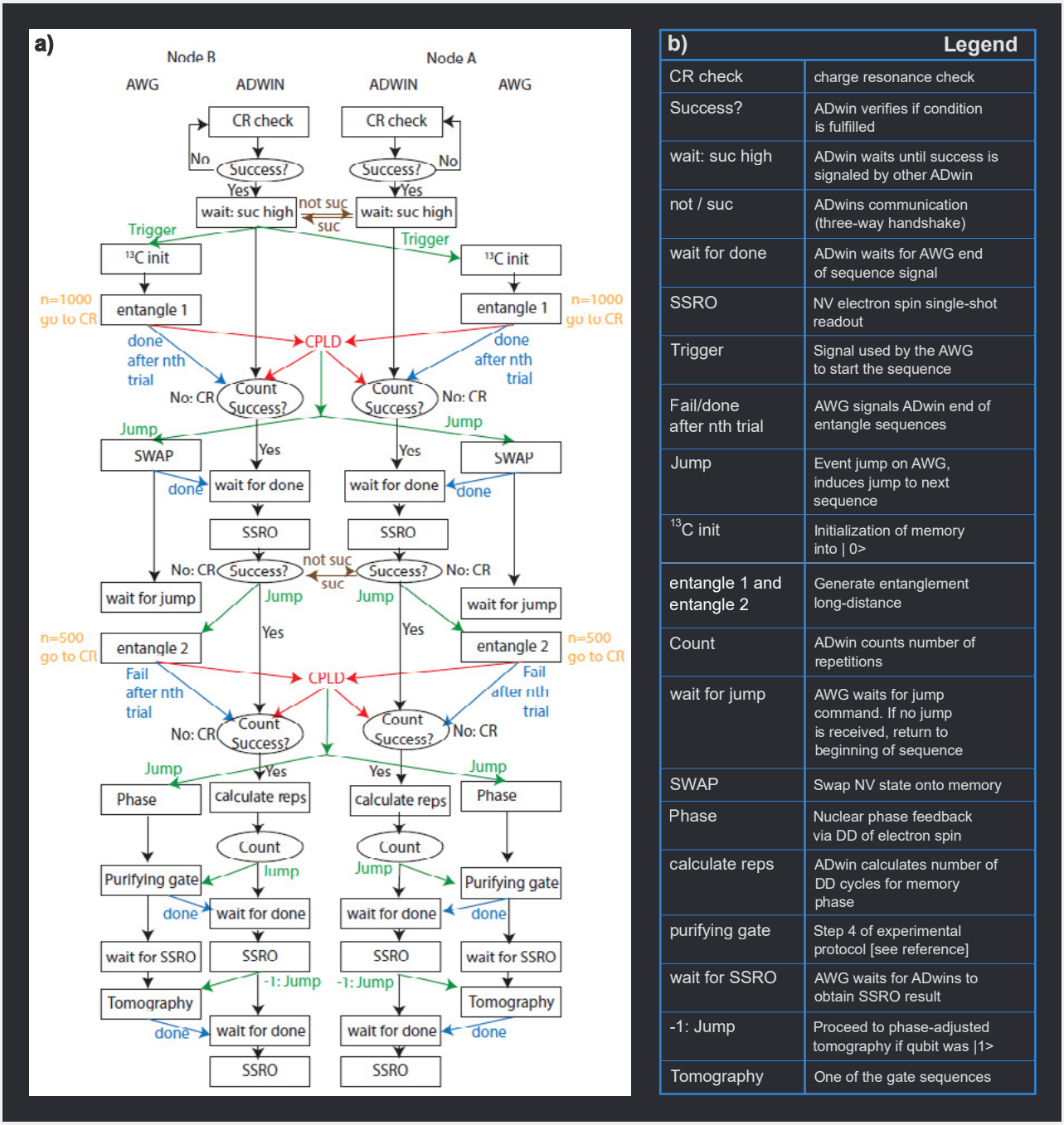
# DISTILLATION ON REMOTE ELECTRON-NUCLEAR 2-QUBIT NODES

The groundbreaking work from Kalb et al. [3] was the first demonstration of entanglement distillation on remote electron-nuclear 2-qubit nodes, which include all elements of a rudimentary but practical quantum network (communication qubits and memory qubits). By combining generation, storage, and processing of high-fidelity distilled entangled states, the authors paved the way for the upscaling that quantum networks needed to unlock their full potential.

Such outstanding demonstration resulted from a complex and carefully orchestrated protocol, a constellation of advanced instruments, all meticulously tuned and timed for flawless operation at the shortest timescales. Various components are used to produce the intricate protocol shown in Fig. 1, including acousto-optical modulators (AOMs) to provide the correct optical pulses, TCSPC electronics for time tagging and temporal filtering, AWGs and microprocessors for real-time waveform generation, and many more.

## LET'S RUN THE DISTILLATION PROTOCOL WITH THE OPX+

OPX+ replaces many traditional tools such as AWGs, time-taggers, etc. It offers a unified FPGA-based platform optimized for quantum control that can orchestrate experiments and take measurement-based decisions dynamically and in hardware time. All the novel functionalities of the OPX+ are accessible using our easy-to-use python-based programming language, QUA, which is compiled directly to FPGA assembly. This powerful combination of quantum-dedicated hardware and intuitive software makes even the most complex experiments seem like a first-year programming exercise. To demonstrate this, let's see how to perform the Kalb et al. [3] protocol using the Quantum Orchestration Platform.




| b) | Legend |
|---|---|
| CR check | charge resonance check |
| Success? | ADwin verifies if condition is fulfilled |
| wait: suc high | ADwin waits until success is signaled by other ADwin |
| not / suc | ADwins communication (three-way handshake) |
| wait for done | ADwin waits for AWG end of sequence signal |
| SSRO | NV electron spin single-shot readout |
| Trigger | Signal used by the AWG to start the sequence |
| Fail/done after nth trial | AWG signals ADwin end of entangle sequences |
| Jump | Event jump on AWG, induces jump to next sequence |
| $^{13}$C init | Initialization of memory into \| 0> |
| entangle 1 and entangle 2 | Generate entanglement long-distance |
| Count | ADwin counts number of repetitions |
| wait for jump | AWG waits for jump command. If no jump is received, return to beginning of sequence |
| SWAP | Swap NV state onto memory |
| Phase | Nuclear phase feedback via DD of electron spin |
| calculate reps | ADwin calculates number of DD cycles for memory phase |
| purifying gate | Step 4 of experimental protocol [see reference] |
| wait for SSRO | AWG waits for ADwins to obtain SSRO result |
| -1: Jump | Proceed to phase-adjusted tomography if qubit was \|1> |
| Tomography | One of the gate sequences |

***Fig. 1*** ***a)*** *Logical blocks diagram used by Kalb et al. [3] to demonstrate entanglement distillation on remote electron-nuclear 2-qubit nodes. Adapted from [3] with the editor's permission.* ***b)*** *Legend of terms used.*

# MACROS IN QUA

To write any complex sequence, we first break it into manageable blocks. QUA allows us to create reusable macros and libraries. For instance, here's how to program the SSRO (single-shot readout [4]) block, a sequence that experimentalists perform on a day-to-day basis:

```
0       def SSRO(system,SSRO_threshold):
1           times_internal = declare(int, size=100)
2           counts_internal = declare(int)
3           play("laser", f"qubit{system}")
4           align(f"qubit{system}",f"laser_red_A1_{system}")
5           play("laser_on", f"laser_red_A1_{system}")
6           measure("SSRO_readout", f"qubit{system}", \
7       None, time_tagging.analog(times_internal, 300, \
        counts_internal))
8           return (counts_internal < SSRO_threshold)
```

***QUA macro for running a single-shot readout sequence.***

This macro defines two real-time variables (a vector to record the timestamps and an integer for the total number of counts), triggers the readout laser, opens a measurement window in the ADC input of the OPX+, and time-tags and counts the pulses generated by a photodiode. The macro returns a boolean, which is True when the number of counts is smaller than a threshold. Similar to Python, a QUA macro can accept parameters which, in this case, are the name of the system (1 or 2) and the threshold value. This lets us reuse the same macro to address different qubits.

Similar macros exist for routine protocols, such as SSRO or XY8 blocks. See below for the full QUA code, or visit our Quantum Sensing with NV center page for more on the XY8 sequences.

Having written macros for each block in the sequence, all that is left to do is program the control flow. With QUA, this is just a simple programming exercise. QUA is particularly simple when working with repeat-until-success protocols, thanks to the `align()` command, which we will now explore in more detail.

The entire entanglement distillation protocol [3] runs in QUA with less than 300 lines of code. Most of this code consists of macro definitions that will be reused for other experiments and can be adapted from code snippets you will find in our open-source libraries. Once we program all basic operations into macros, the entire sequence demonstrated by Kalb et al. [3] is compressed in less than 50 lines of code, and runs with the lowest possible latencies. Additionally, as the authors suggest, the sequence can readily be improved by including steps such as active reset, which are straightforward to implement on OPX+ without complications to the experimental setup.

## REPEAT-UNTIL-SUCCESS PROTOCOLS

The sequence we are implementing is acting on two different nodes, each consisting of an NV center and a nuclear spin. Each node is controlled independently for most of the sequence, but we want the sub-sequences to align at certain stages so that the protocol continues only when both nodes are ready. Using traditional equipment would be challenging because of the abundance of repeat-until-success blocks, especially since we cannot predict in advance how long it will take each node to reach a checkpoint. Fig. 1

depicts these as the “wait for done” blocks.
With QUA, we synchronize sequences with a simple `align('element_1', 'element_2')` command. This tells the FPGA to wait on all commands addressing either `element_1` or `element_2` until they have both reached that part of the program. Evaluation happens in real-time, so we do not need to know in advance how long it will take either node to be ready. An example of how such synchronization will look in QUA is:

```
0         Nuclear_spin_init(1, counts1_total,a1,t1)
1         Nuclear_spin_init(2, counts2_total,a2,t2)
2         align("qubit1", "qubit2")
```

***QUA code to run initialization macros for two different nuclear qubits in parallel and then align the sequence temporally, waiting for both spins to have completed initialization.***

Where `Nuclear_spin_init()` is a macro that initializes a memory qubit (system), composed of pulse applications and XY8 blocks.

The initialization macros `Nuclear_spin_init()` on the two qubits run in parallel on different cores. Thanks to the `align()` command, the FPGA knows that both qubits must be initialized at this point in the sequence to continue. We do not need to program ADwin microprocessors and tangle our setup further because QUA and OPX+ were made with such quantum experiments in mind.

Several steps in our example protocol call for a repeat-until-success flow. Therefore, there is no way to predict how many times the subroutine will run when beginning the experiment: maybe after one try and maybe after a thousand. We can implement this kind of flow with a simple `while()` statement, as we would in Python:

```
0         with while_((entangled == False) & (N < N_max)):
1             entangle()
2             assign(N,N+1)
```

***QUA Code***

This executes the entanglement protocol, by running the `entangle()` macro until the `entangled` variable equals `True`, in a while loop that runs on the pulse processing unit (PPU), on hardware time. We also include a counter to keep track of the number of attempts to quit after a maximum is reached. Additionally, we know how long it took to exit the loop, which allows for phase tracking and correction, another useful feature of the OPX+.

# COMPLETE PHASE CONTROL

Not only does the OPX+ enable decision-making during an experiment, it also takes care of the problem of the relative phase with no need for user intervention. It does that by generating its pulses with automatic hardware-level tracking of the phase accumulation of each output frequency. By default, all pulses are performed in the rotating frame of the qubit.

Additionally, the OPX+ allows switching between an unlimited number of frequencies while preserving the rotating frame of each tone. Such switching is crucial when the same output channel is driving different transitions. This is done in QUA by using the `update_frequency('system','frequency')` command as in the `Nuclear_spin_init()` macro you previously saw.
We can see an example of the switching by running a simple code in QUA:

```
0         With program() as Phase_Example_QUA:
1             reset_phase('output1')                ## Reset Phases
2             reset_phase('output2')
3             update_frequency('output1', 21168452)   ## Set initial frequency
4             update_frequency('output2', 21168452)
5             play('pi', 'output1', duration = 50)    ## Play pulses on outputs
6             play('pi', 'output2', duration = 150)
7             update_frequency('output1', 52849685)   ## Update output1 frequency
8             play('pi', 'output1', duration = 50)
9             update_frequency('output1', 21168452)   ## Update output1 frequency back
10            play('pi', 'output1', duration = 50)
```

***QUA code example to show the phase tracking capabilities of the OPX+. The results of the output measurements of an OPX+ running this code are in Fig. 2.***

This code plays pulses with an arbitrary frequency out of two OPX+ outputs. Then, one of the frequencies is changed and changed back. In Fig. 2, we show what an observing oscilloscope would measure at the outputs. Thanks to the OPX+ phase tracking, once the frequency of `output1` is restored to the original frequency, the resulting output will still be in phase with its original twin `output2`. The tracking of a phase can also be reset by the `reset_phase()` command.

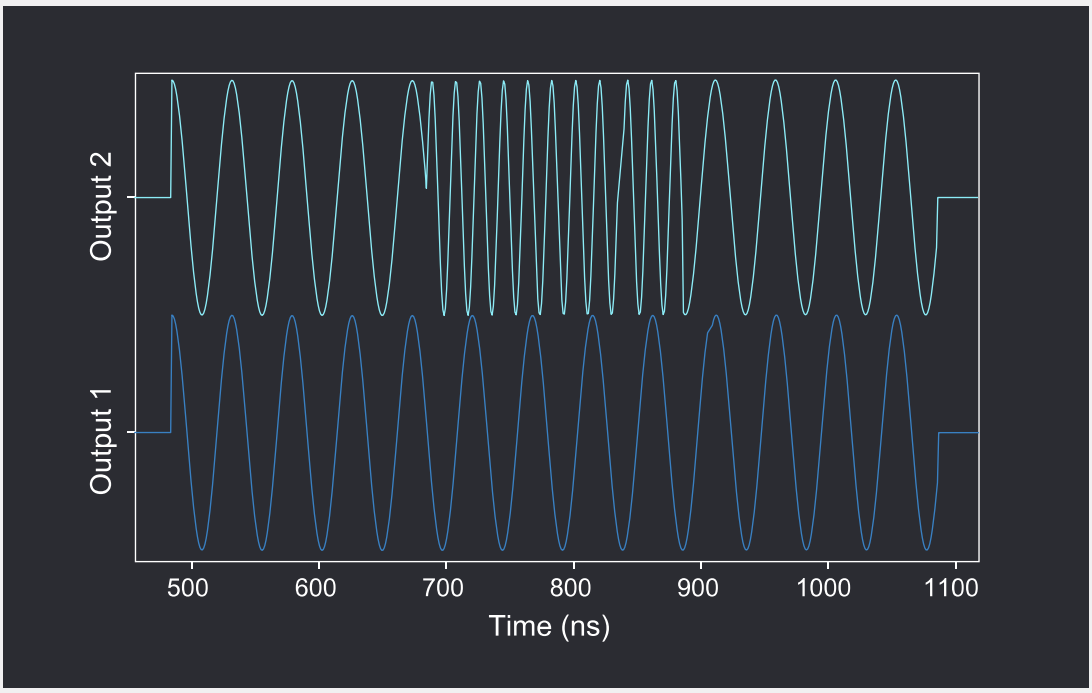



***Fig. 2*** *Outputs of an OPX+ as measured with an oscilloscope while running the frequency update (see example QUA code in text).*

OPX+ can also operate on the qubit frame, with rotations (using `frame_rotation()`) and complete reset of the time monitoring (using `reset_frame()`). Using simple commands, OPX+ offers full control over the phase. Both software and hardware are smart and intuitive, allowing you to write simple codes to run complex experiments.

**References**

**[1]** Kimble, H. Jeff. Nature 453.7198 (2008): 1023-1030.

**[2]** Bennett, C. H., et al. Physical Review A 54.5 (1996): 3824.

**[3]** Kalb, N., et al. Science 356.6341 (2017): 928-932.

**[4]** Robledo, Lucio, et al. Nature 477.7366 (2011): 574-578.

# The Quantum Orchestration Platform

**AN END TO END QUANTUM CONTROL SOLUTION TO DRIVE THE FASTEST TIME TO RESULTS, AT ANY SCALE**

## OPX+

### RUN STATE OF THE ART EXPERIMENTS WITH EASE

An architecture designed from the ground up for quantum control, the OPX+ lets you run the quantum experiments of your dreams right from the installation. With a quantum feature-rich environment, the OPX+ is built for scale and performance. Now, you can **run the most complex quantum algorithms and experiments in a fraction of the development time**.

## PULSE PROCESSING UNIT

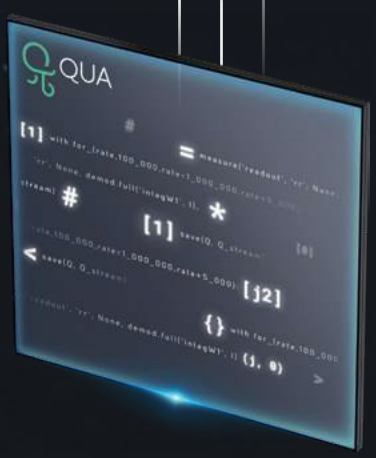
### ACHIEVE THE FASTEST TIME TO RESULTS

Within the OPX+ is the Pulse Processing Unit, QM's leading-edge quantum control technology. Progress with incomparable speed and extreme flexibility. Run even the most demanding experiments efficiently, with the fastest runtimes and the lowest latencies in the industry, including quantum protocols that require **real-time waveform generation, real-time waveform acquisition, real-time comprehensive processing, and control flow.**

## QUA

### CODE QUANTUM PROGRAMS SEAMLESSLY

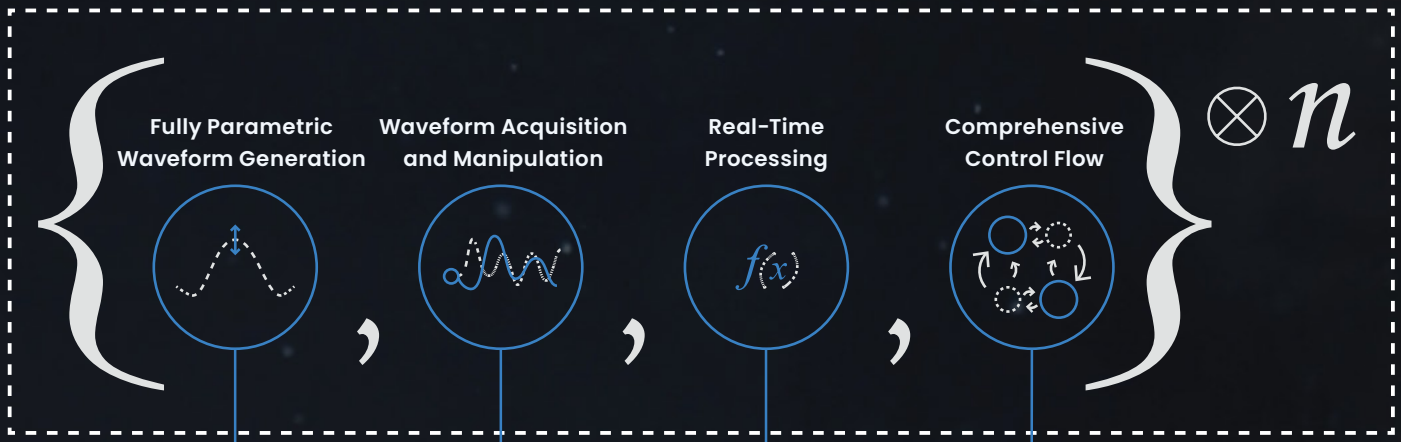
**Implement the protocols of your wildest dreams as easily as writing pseudocode.** Designed for quantum control, QUA is the first universal quantum pulse-level programming language. Code even the most advanced programs and run them with the best possible performance. Natively describe your most challenging experiments, from complex AI-based multi-qubit calibrations to multi-qubit quantum error correction.




**All of the information above is also valid for the OPX*

QUANTUM MACHINES

# YOUR PROTOCOLS LIVE IN THIS PHASE SPACE




### Real-Time Waveform Generation

- Fully parametrized: length, frequency, phase (relative & absolute), amplitude, bandwidth, chirp
- Compensations: Crosstalk matrix, FIR and IIR filters
- And many more

### Real-Time Measurements

- High fidelity analog to digital conversion
- General integration & demodulation
- Weighted Integrations accumulated integrations, sliced integrations, etc.
- Time Tagging, TTL counting

### Real-Time Processing

Turing complete:
- Basic arithmetics
- Evaluation of trigonometric functions
- Vector operations
- Casting of variable types
- And more (Turing complete)

### Real-Time Control Flow

- If/else
- For loops
- While loops
- Switch case

### Real-Time 'Quantum' Estimations

- State estimations
- Error estimations
- Bayesian estimations
- Correlation functions
- Neural nets based estimations

### Real-Time Multi-Qubit Feedback

- Qubit stabilization & tracking
- Quantum error correction
- And many more

## THE QUANTUM ORCHESTRATION PLATFORM COVERS THIS SPACE!

- Easily express quantum algorithms and experimental protocols that comprise all of the above.
- Seamlessly sync measurements, real-time calculations, and pulses of different quantum elements.
- Loop over a wide range of parameters in real-time, including intermediate frequencies, amplitudes, phases, delays, integration parameters, measurement axes, etc.
- Use if/else and switch-case statements to condition operations in real time (real time feedback).
- Define procedures (macros) to be reused in the code and access an extensive family of libraries.

QUANTUM MACHINES


## If you wish to learn more:

info@quantum-machines.co

# About Quantum Machines

Quantum Machines (QM) drives quantum breakthroughs that accelerate the path towards the new age of quantum computing. The company's Quantum Orchestration Platform (QOP) fundamentally redefines the control and operations architecture of quantum processors.

The full-stack hardware and software platform is capable of running even the most complex algorithms right out of the box, including quantum error correction, multi-qubit calibration, and more. Helping achieve the full potential of any quantum processor, the QOP allows for unprecedented advancement and speed-up of quantum technologies as well as the ability to scale into the thousands of qubits. Visit us at: www.quantum-machines.co

---