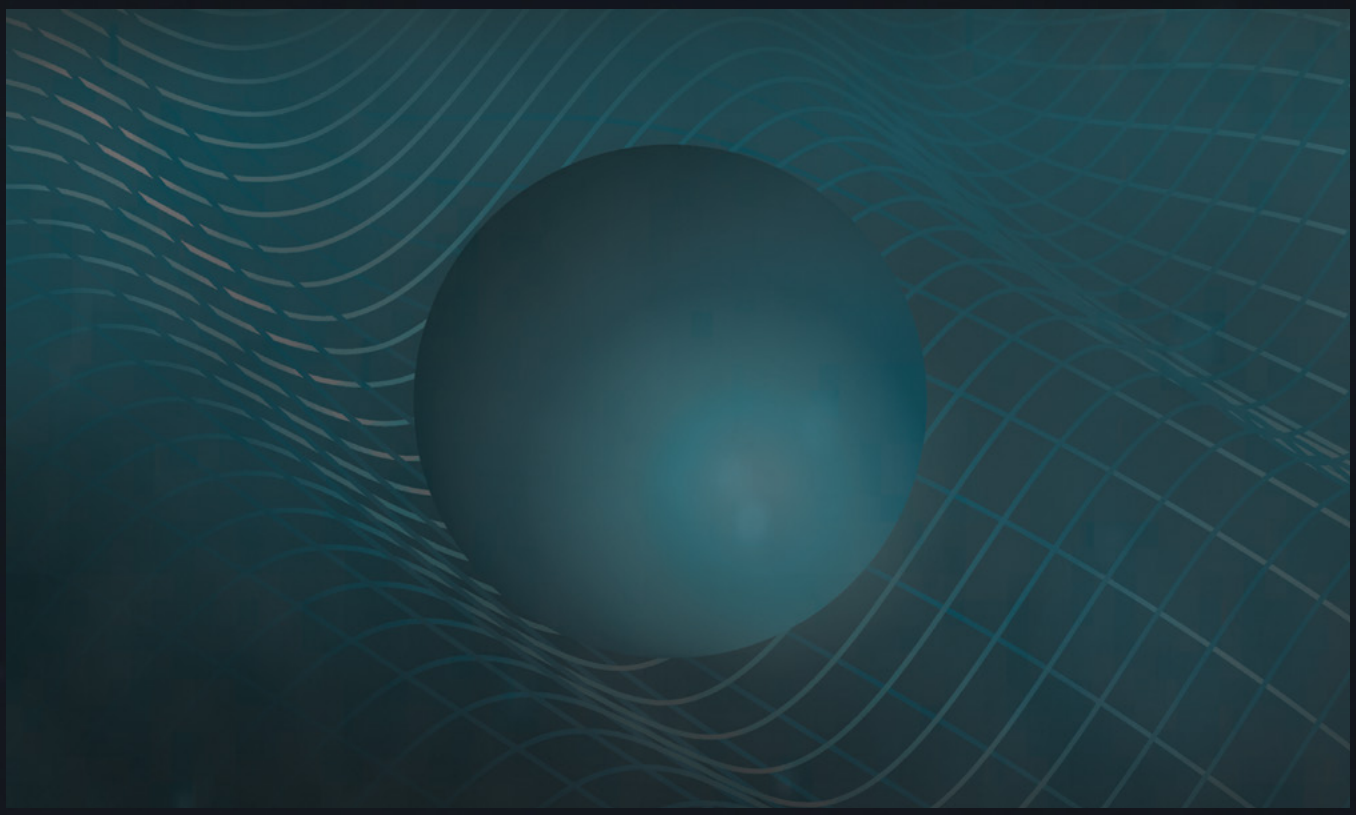
QUANTUM MACHINES

USE CASE

Quantum Orchestration for Quantum Sensing

With the Quantum Orchestration Platform, complex and tediously long experiments become first-year exercises, run with the lowest latencies and real-time decision making. Have a look at some real-world use cases with NV center sensors.



QUANTUM SENSING WITH NV CENTERS

Quantum systems have emerged as powerful probes to measure key physical quantities in recent years. Quantum sensors capitalize on the central weakness of quantum systems, their sensitivity to disturbances in the environment [1].

After many notable examples such as atomic clocks and SQUID magnetometers, the field promises to provide fundamental new opportunities, with immediate potential for practical applications. Among all solid-state spins and other platforms, the nitrogen-vacancy (NV) center in diamond has emerged as a powerful sensor for magnetic fields and nanoscale imaging due to its small size, stability, and room temperature operation.

INDIVIDUAL ADDRESSING OF NUCLEAR SPINS

As the field progresses and complexity increases, the control system requirements become more demanding, and the need for real-time feedback arises. As a concrete example, we consider the experiment performed by Abobeih et al. [2]. In this work, the authors mapped the location of 27 individually addressed ^{13}C atoms in a diamond lattice, using an NMR-like technique, with a single nearby NV center as the quantum sensor.

In experiments of such complexity, achieving individual addressing of each nuclear spin is a very demanding challenge. The authors used a variation of a dynamical decoupling (DD) sequence, named DDRF [3], that involves the introduction of an RF pulse, in resonance with the individual nuclear spin, interleaved in between DD pulses applied to the electron spin. Such an arduous sequence was undoubtedly an electronics and programming feat.

ATOM ARRANGEMENT RESULTS

The author's approach has several nuanced requirements. Running the sequence with the OPX+ instead, allows highlighting some of its unique capabilities. For example, to have constructive phase accumulation on the nuclear spin, it is vital to maintain precise control on the RF pulse relative phase. The OPX+ generates its pulses with automatic hardware-level tracking of the phase accumulation of each output frequency, taking care of the problem of relative phase without the need for user intervention. By default, all pulses are performed in the rotating frame of the qubit. The phase is preserved when transitioning back and forth between frequencies on the same output channel.

Furthermore, the control over the phase allows for defining both single-qubit rotations and two-qubit conditional gates on nuclear spin. The addition of a global phase allows for control over the gate's rotation axis. With an OPX+, this is as simple as writing the command *frame_rotation*(ϕ) with real-time parametric waveform generation. Essentially, this means that ϕ can be a variable living on the processor. Thus it is possible to adaptively update it during the sequence, based on live inputs from the measurement.

Fig. 1 shows the simulated DDRF sequence for unconditional (Fig. 1a-b) and conditional (Fig. 1c-d) rotations. We can achieve the latter by adding a phase π for every other pulse. As mentioned by the authors, the DDRF sequence allows for parallel control of several nuclear spins, as the nuclear

spin frequency does not restrict the interpulse delay of the electron DD. Even though the authors do not explicitly demonstrate it in their work, this important addition is readily done with the OPX+, which offers easy-to-use multiplexing capabilities. Fig. 2 shows an example of multiplexing of two frequencies, as output of two different channels (Fig. 2a-b) or summed and sent through the same output channel (Fig. 2c-d).

The possibilities do not stop at just two frequencies either. The OPX+ has 18 pulser cores capable of working in parallel, allowing the control of up to 16 nuclear spins simultaneously (using 2 for the electron spin). The OPX+ allows such control to be performed in parallel and with unmatched ease, thanks to our python-based programming language QUA.

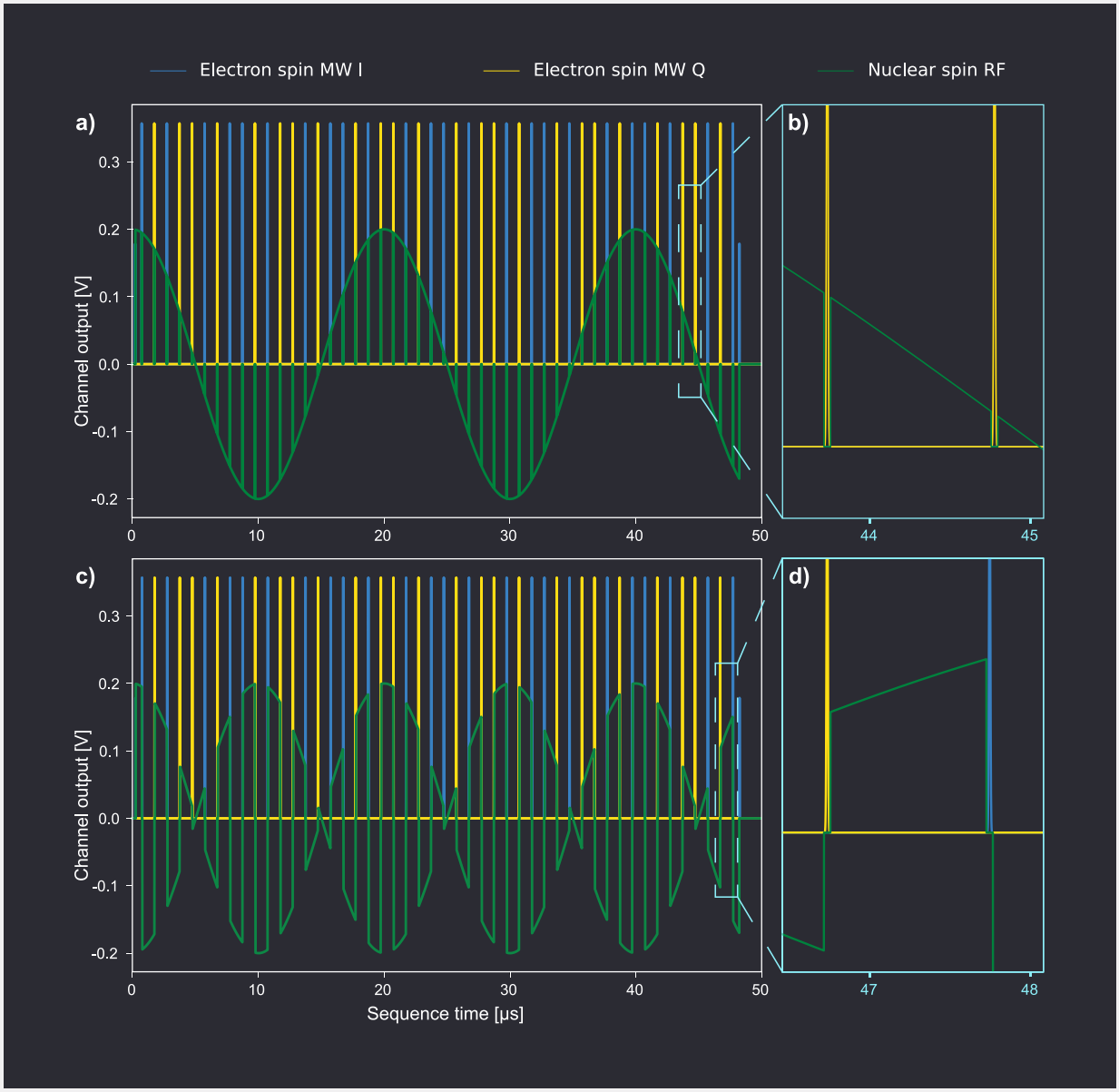


Fig. 1 *a-b) Unconditional and* ***c-d)*** *conditional rotations on the nuclear spin, using the DDRF sequence.* ***b)*** *and* ***d)*** *shows a zoomed-in view of* ***a)*** *and* ***b)****, respectively, to highlight the pulse synchronization. The blue (yellow) represents the output for the I (Q) channels for the electron spin. As no intermediate frequency is defined, these can be considered the envelope of the* π_x *and* π_y *pulses, respectively. The green represents the output for the nuclear spin RF, where the frequency was chosen arbitrarily.*

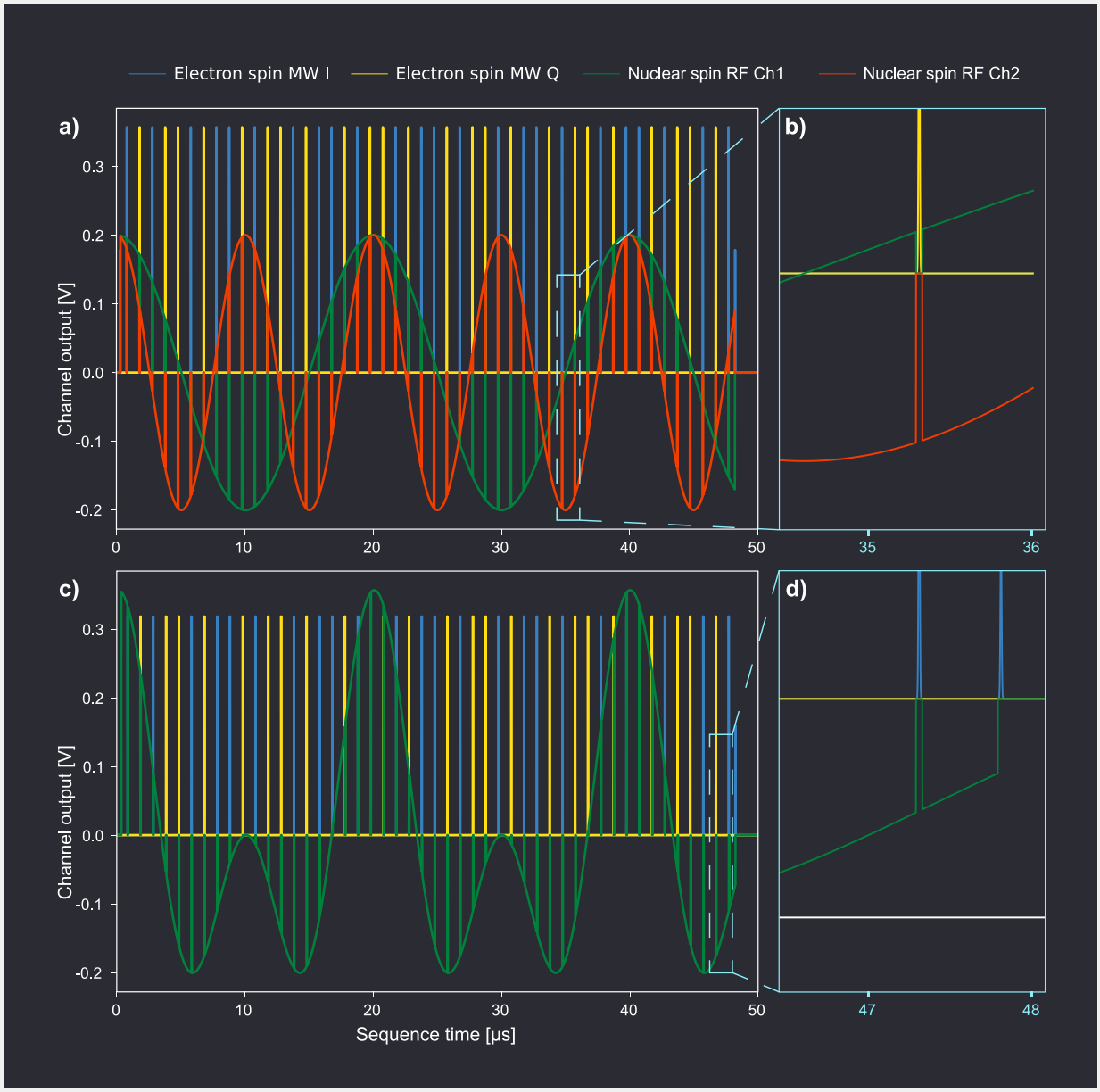


Fig. 2 a-b) *DDRF sequence for unconditional rotation of nuclear spins, with two RF frequencies as output* ***a-b)*** *in parallel to two different channels (green and red) and* ***c-d)*** *multiplexed to the same output channel.* ***b)*** *and* ***d)*** *shows a zoomed-in view of a) and b), respectively, to highlight the pulse synchronization. The blue (yellow) represents the output for the I (Q) channels for the electron spin.*

CONTROLLING NUCLEAR SPIN WITH QUA

The ability to make macros that simplify the code is one of the advantages of QUA. It allows you to troubleshoot and re-use compartmentalized code that makes writing new sequences exceptionally fast. As an example, let's write the above DDRF sequence.

First, we define the macro, `xy8_block()`, which creates the 8 MW pulses, to the electron spin, of a single XY8 block. The `play("pi", "spin_qubit")` tells the OPX+ to output a pulse named pi, both predefined in a setup specific configuration file. The `wait()` command is used for the interpulse delay, while we use `frame_rotation()` and `reset_frame()` to control the phase of the pulses. Notice that as the pulser tracks the phase of the rotating frame, the phase defined in the `frame_rotation()` command is simply the phase of the Y pulse in the rotating frame, i.e., $\pi/2$.

```
0         def xy8_block():
1             # A single XY8 block, ends at x frame.
2             play("pi", "spin_qubit")  # 1 X
3             wait(tt, "spin_qubit")
4
5             frame_rotation(np.pi / 2, "spin_qubit")
6             play("pi", "spin_qubit")  # 2 Y
7             wait(tt, "spin_qubit")
8
9             reset_frame("spin_qubit")
10            play("pi", "spin_qubit")  # 3 X
11            wait(tt, "spin_qubit")
12
13            frame_rotation(np.pi / 2, "spin_qubit")
14            play("pi", "spin_qubit")  # 4 Y
15            wait(tt, "spin_qubit")
16
17            play("pi", "spin_qubit")  # 5 Y
18            wait(tt, "spin_qubit")
19
20            reset_frame("spin_qubit")
21            play("pi", "spin_qubit")  # 6 X
22            wait(tt, "spin_qubit")
23
24            frame_rotation(np.pi / 2, "spin_qubit")
25            play("pi", "spin_qubit")  # 7 Y
26            wait(tt, "spin_qubit")
27
28            reset_frame("spin_qubit")
29            play("pi", "spin_qubit")  # 8 X
```

The second macro to complete the electron spin part of the XY8-N sequence is the `xy8_n()` macro, allowing us to create the full sequence:

```
0         def xy8_n(n):
1             i = declare(int)
2             wait(t, "spin_qubit")
3             xy8_block()
4             with for_(i, 0, i < n - 1, i + 1):
5                 wait(tt, "spin_qubit") # tt = 2*t
6                 xy8_block()
7             wait(t, "spin_qubit")
```

The `for()` loop is written in QUA, and this means the processor doesn't get a code that is replicated n times, but a command to repeat in real-time the code written within the loop. Thus, there is no limit on how long the XY8 can be memory-wise, and long sequences won't cause a long overhead time in waveform uploads. QUA allows all control flow operations to run on the pulse processing unit (PPU). Similar to the XY8 macros, we can then define the operations necessary for the RF pulses that control the nuclear spin, first a single RF block or `rf_block()`, then `rf_n()`.

Using these macros the DDRF sequence is straightforward. For example, if we want to apply an unconditional rotation on `nuclear_spin1`,

```
0         play("pi2", "spin_qubit")
1         xy8_n(N)
2         rf_n(N, 'nuclear_spin1', False)
3         play("pi2", "spin_qubit")
```

Then, adding a second conditional rotation on `nuclear_spin2` is a single line of code.

```
0         play("pi2", "spin_qubit")
1         xy8_n(N)
2         rf_n(N, 'nuclear_spin1', False)
3         rf_n(N, 'nuclear_spin2', True)
4         play("pi2", "spin_qubit")
```

The macros are executed on different threads, and so they will run in parallel, saving time and reducing complexity. Fig. 3 shows the result of such a pulse sequence.

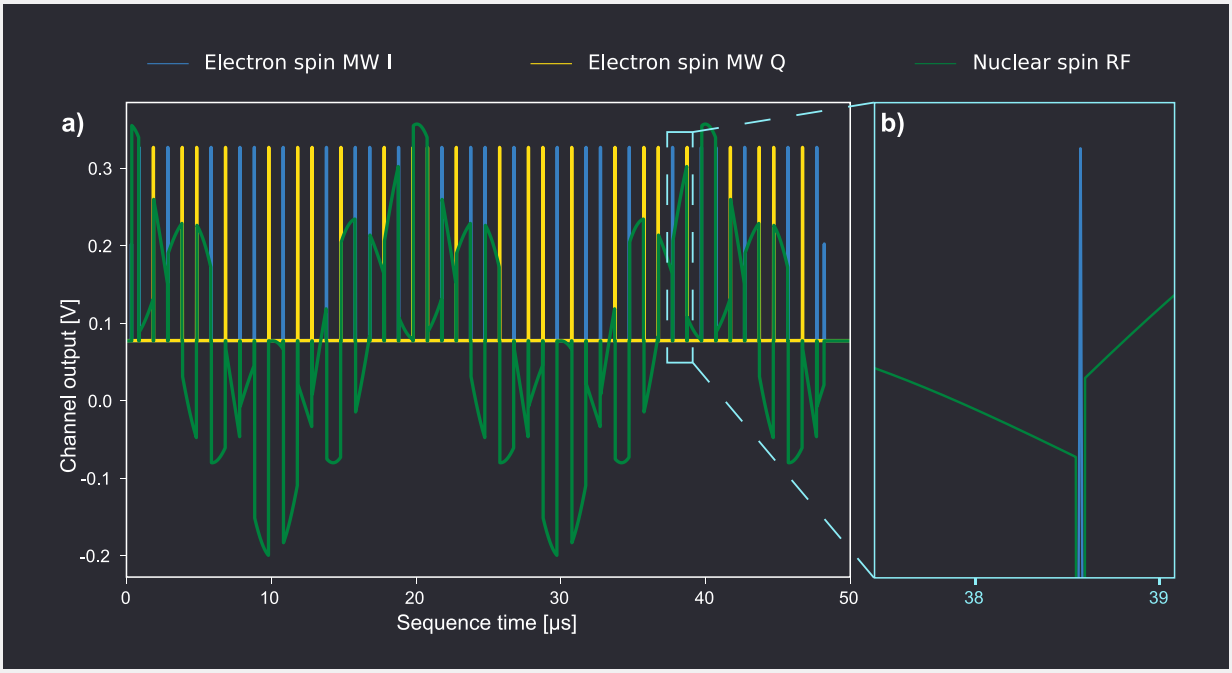


Fig. 3 a) *DDRF sequence with two RF frequencies multiplexed on the same output channel (green), simultaneously applying an unconditional rotation on the first nuclear spin and a conditional rotation on the second nuclear spin.* ***b)*** *shows a zoomed-in view of a) to highlight the pulse synchronization. The blue (yellow) represents the output for the I (Q) channels for the electron spin.*

It's easy to embed such a sequence within a QUA loop, updating the frequency of the RF pulse dynamically. If we then also include a measurement command with the capability of time tagging photons received by a photo-diode, then we obtain a full DDRF sequence within a spectroscopy scan, running dynamically on the PPU:

```
0         with for_(freq, f_min, freq <= f_max, freq + df):  # Implicit Align
1             update_frequency("nuclear_spin1", freq)
2             play("pi2", "spin_qubit")
3             xy8_n(repsN)
4             rf_n(repsN, 'nuclear_spin1', False)
5             play("pi2", "spin_qubit")
6             measure("readout", "green_laser", None, time_tagging.analog(times, 300,
7                     counts_ref))
```

QUANTUM NON-DEMOLITION EXPERIMENTS

Dynamical decoupling (DD) protocols are the basis for many experimental protocols for NV center sensing applications, both procedural and state of the art. For example, correlating the phase accumulation between different DD sequences (instead of averaging them) allows sensing of long coherence signals [4]. This approach is not limited by the quantum sensor coherence time but rather by the classical clock used to control the experiment and thus allows for sensing with arbitrary frequency resolution [5]. Such a result can be improved by performing a quantum non-demolition (QND) measurement [6], using an ancilla nuclear spin, increasing the number of photons collected for each readout while increasing the minimal possible sampling rate of the sequence (See more about single-shot readout on our NV centers use-cases page).

ADAPTIVE REAL-TIME PROTOCOLS WITH THE OPX+

The OPX+ is capable of real-time decision making, improving the QND protocol by shortening its time while maintaining high fidelity. To see how the protocol in [5] would work on an OPX+, let's examine a typical QND histogram, where enough repetitions were made such that the population distribution for each state is relatively well separated. As we show in Fig. 4a, in this scenario, we can define a single threshold between the two distributions and this is enough for declaring a state with high fidelity.

If not enough repetitions were made so the two states are not well separated (Fig. 4b), a single threshold between the two distributions results in relatively low fidelity. To increase the state fidelity in such case, we can define two separate thresholds: if the number of photons is larger than threshold 2, we declare state $|\uparrow\rangle$; if the number of photons is lower than threshold 1, we declare state $|\downarrow\rangle$; if the result is between the thresholds, we declare an inconclusive measurement.

This translates to some of the measurements being discarded in post-processing, resulting in a trade-off between fidelity and total measurement time. This tradeoff is even less desirable when you look at the correlations between measurements, as inconclusive measurements are holes in your results vector.

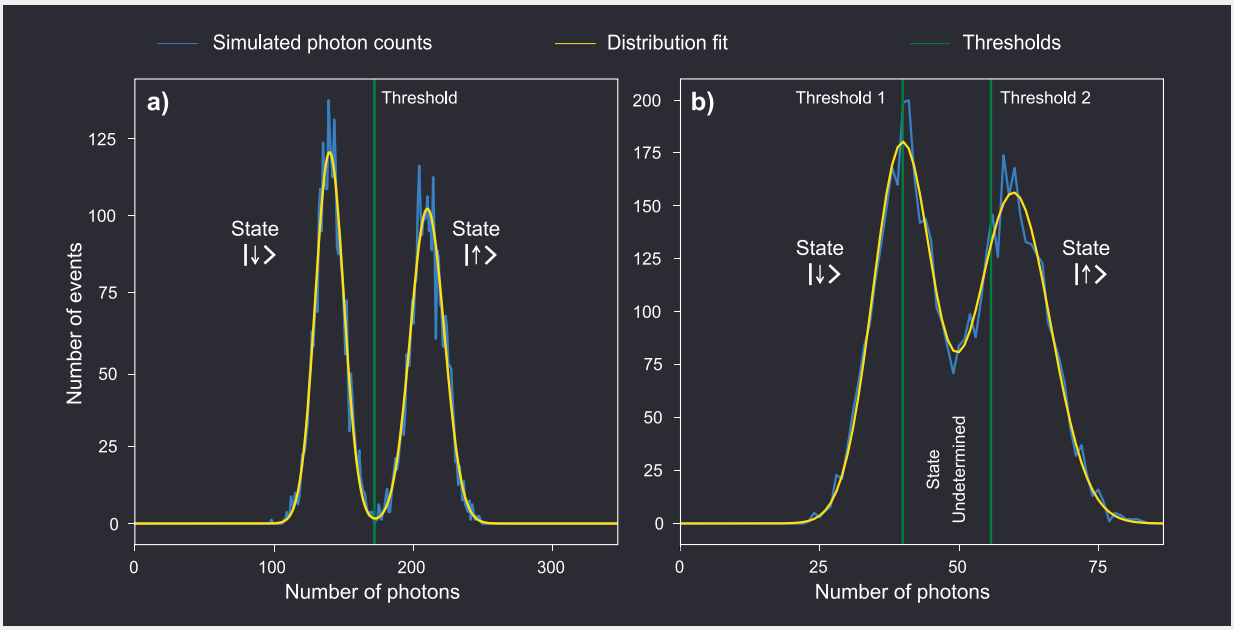


Fig. 4 *Simulated photon counting histograms.* ***a)*** *Measurement repeated enough times to clearly separate the photon distribution function of the two nuclear states. High fidelity readout can be readily achieved using a single threshold.* ***b)*** *Measurement repeated not enough times, showing overlapping distributions adding uncertainty in the state discrimination. High fidelity readout can still be achieved using two thresholds, at the price of adding a third undetermined state, due to which many events will be discarded.*

This is where real-time processing and decision-making can be beneficial. It is possible to define dynamic thresholds in the OPX+ dependent on the number of repetitions, preserving the desired readout fidelity while reducing the average readout length. The OPX+ can autonomously try to declare a state after a minimal number of repetitions, and if the result is inconclusive, it will just continue with the QND measurement until it is possible to declare a state with the desired preconfigured fidelity. We can quickly write such an example protocol in our simple python-based programming language QUA:

```
assign(counts_total, 0)
with while_((j < M) & (achieved_fidelity == False)):
    with for_(i, 0, i < N, i + 1):
        conditional_pi()
        measure("readout", "NV", None, time_tagging.analog(times, 300, counts))
        assign(counts_total, counts_total + counts)
    threshold = calculate_thresholds(j)
    assign(j, j + 1)
    with if_(counts_total > threshold[0]):
        assign(measured_state, 0)
        assign(achieved_fidelity, True)
    with if_(counts_total < threshold[1]):
        assign(measured_state, 1)
        assign(achieved_fidelity, True)
```

Where `conditional_pi()` and `calculate_thresholds()` are predefined macro functions. One applies a conditional π pulse on the electron spin based on the nuclear spin state, while the other calculates, in real-time, the thresholds needed for the desired fidelity (based on the number of repetitions represented by the variable `j`). N is the minimal number of repetitions necessary for a distinguishable difference in the expectation value for the number of photons, and M is the maximum number of repetitions we allow, based on the desired sampling rate of the experiment. This simple code allows you to perform an optimized and adaptive version of the QND protocol with an OPX+, right out of the box.

STREAM PROCESSING WITH THE OPX+

As for sequences such as the one in the work of Boss et. al. [5], or other measurements of systems with long coherence times (even up to minutes), large amounts of data are produced. The OPX+ not only allows for incredible real-time decision making and parameter updates, but also offers an intermediate processing step via the attached Linux server. While the pulse processing unit (PPU) can make decisions live during the experiment step-by-step with the lowest possible latency, the attached server can perform demanding calculations, like correlations or FFT, still within the time-scale of the experiment. This further empowers the OPX+ decision making capabilities and allows for complex and dynamic experimental protocols, while sending to the user's PC only the results deemed necessary.

For example, when little is known beforehand about the signal to be measured, one could greatly benefit from a sequence that smoothly transitions from fast sampling rate and low accuracy to a more precise narrow-range approach, focusing on the signal's frequency. This adaptive change, based on e.g. FFT calculations, can be performed in the measurement's time-scale with the OPX+ dynamic calculations and parameter updates.

References

[1] Degen, C. L., et. al. Reviews of modern physics 89.3 (2017): 035002.

[2] Abobeih, M. H., et al. Nature 576.7787 (2019): 411-415.

[3] Bradley, Conor E., et al. Physical Review X 9.3 (2019): 031045.

[4] Schmitt, Simon, et al. Science 356.6340 (2017): 832-837.

[5] Boss, Jens M., et al. Science 356.6340 (2017): 837-840.

[6] Neumann, Philipp, et al. Science 329.5991 (2010): 542-544.

The Quantum Orchestration Platform

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

OPX+

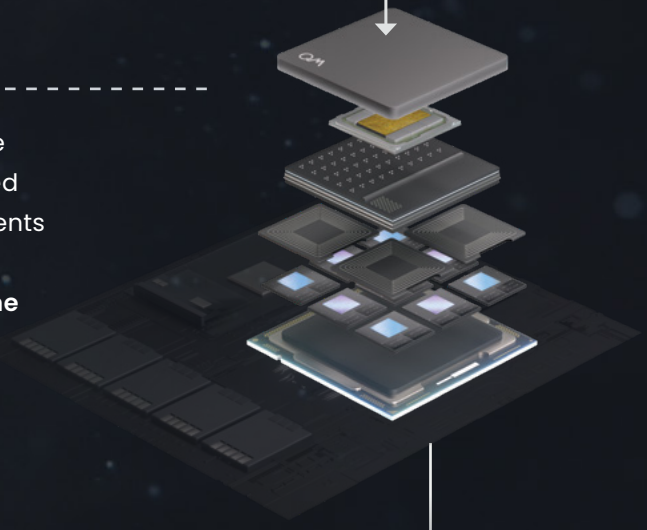
RUN STATE OF THE ART EXPERIMENTS WITH EASE

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

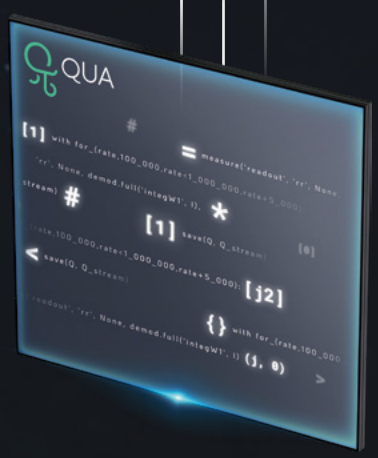
Within the OPX+ is the Pulse Processing Unit, QM's leading-edge quantum control technology. Progress with incomparable speed and extreme flexibility. Run even the most demanding experiments efficiently, with the fastest runtimes and the lowest latencies in the industry, including quantum protocols that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



QUA

CODE QUANTUM PROGRAMS SEAMLESSLY

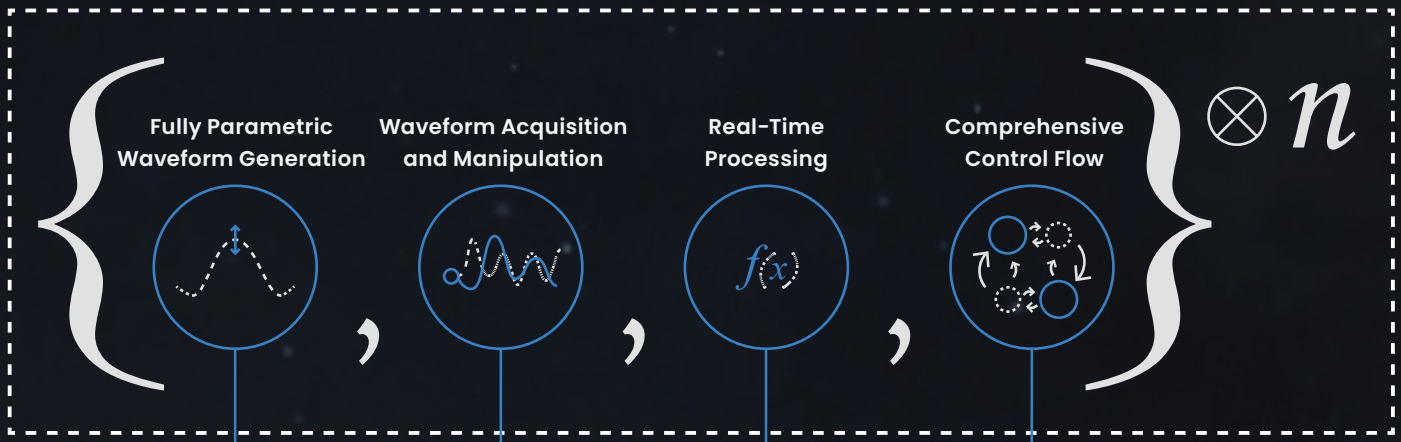
Implement the protocols of your wildest dreams as easily as writing pseudocode. Designed for quantum control, QUA is the first universal quantum pulse-level programming language. Code even the most advanced programs and run them with the best possible performance. Natively describe your most challenging 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

---