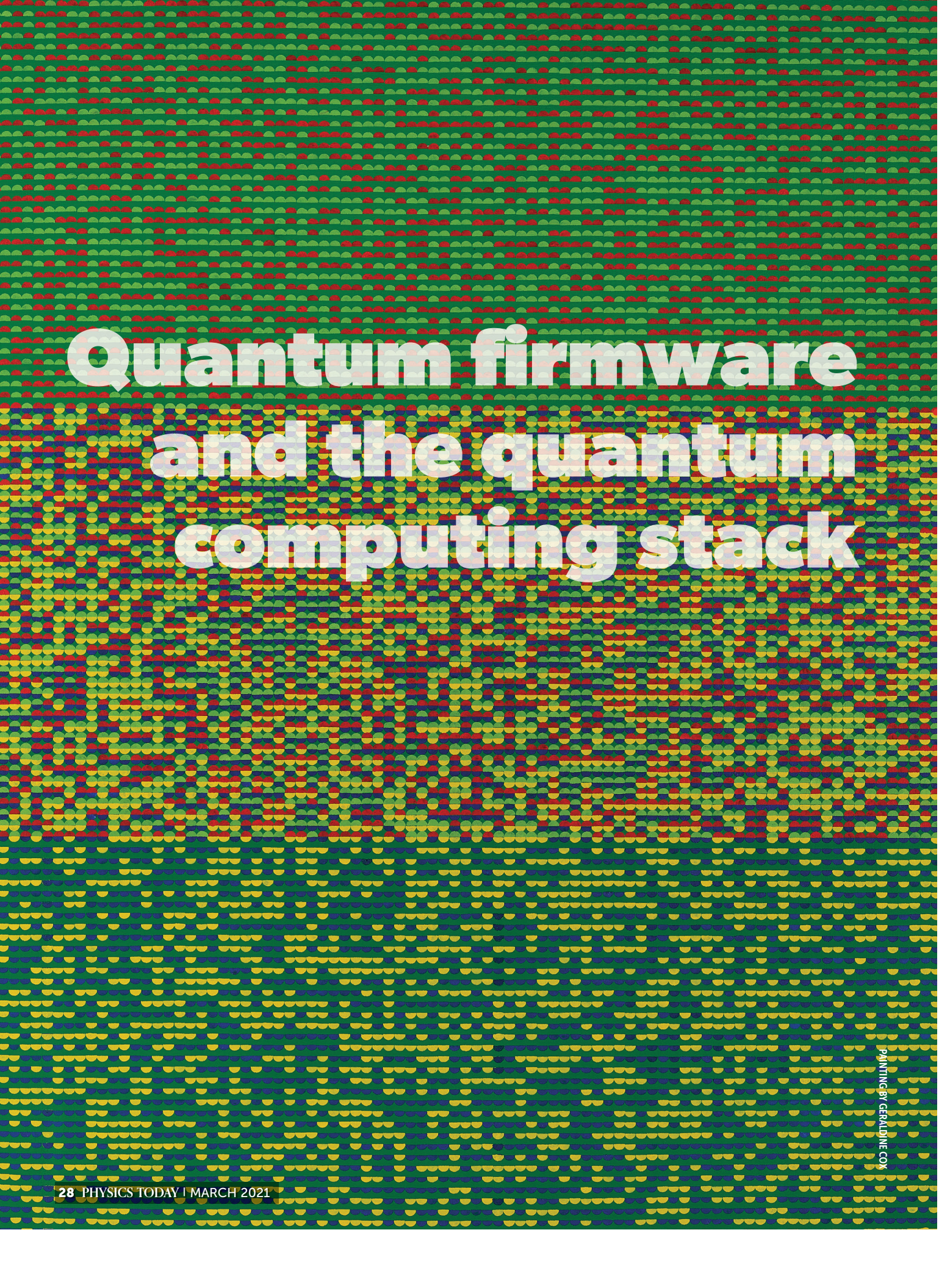
Quantum firmware and the quantum computing stack



PAINTING BY GERALDINE COX

Harrison Ball is the lead quantum research scientist at the quantum-technology company Q-CTRL in Sydney, Australia. **Michael Biercuk** is the CEO and founder of Q-CTRL. He is a professor of quantum physics and quantum technology and is a chief investigator in the Australian Research Council Centre of Excellence for Engineered Quantum Systems at the University of Sydney. **Michael Hush** is the chief scientific officer at Q-CTRL.



Harrison Ball, Michael J. Biercuk, and Michael R. Hush

Integrated quantum-control protocols could bridge the gap between abstract algorithms and the physical manipulation of imperfect hardware.

Quantum computers have rapidly advanced from laboratory curiosities to full-fledged systems operating with dozens of interacting information carriers called qubits. In 2019, researchers at Google became the first to demonstrate quantum supremacy[1]—a quantum computer capable of calculations that are impossible for conventional devices—by using just over 50 qubits.

Researchers are now on the threshold of being able to deploy quantum computers to solve a host of critical problems ranging from pharmaceutical drug discovery and industrial chemistry to codebreaking and information security. (For more on quantum cryptography, see the article by Marcos Curty, Koji Azuma, and Hoi-Kwong Lo on page 36 of this issue.) Because of ongoing developments in computational heuristics and approximate quantum algorithms, quantum computers may well be able to solve commercially relevant problems with some computational benefit, reaching what's known as quantum advantage, within the next decade.

Realizing useful computations using quantum systems requires scientists to recognize that performance is limited predominantly by hardware imperfections and failures rather than just system size. Susceptibility to noise and error remains the Achilles' heel of quantum computers and ultimately limits the algorithms they can run. Researchers are working to improve their devices' performance through passive means like circuit design, but they're also pursuing active measures; mitigating hardware errors through quantum error correction (QEC) has driven research for decades. The complexity and resource intensity of QEC—the set of algorithmic protocols necessary to ensure errors are identified and corrected—has motivated consideration of complementary techniques that enable augmented performance without that computational overhead.

Quantum firmware is a generalized designation for a set of protocols that connect quantum hardware with higher, more abstract levels in the quantum computing stack (see figure 1). More specifically, quantum firmware stipulates how physical hardware should be manipulated to improve stability and reduce various error processes—in essence, "virtualizing" the underlying imperfect hardware. Higher abstraction layers in the quantum computing stack then interact with qubits whose performance is different than that of the qubits in the bare hardware.[2] (For more on quantum computing architectures, see the article by Anne Matsuura, Sonika Johri,

and Justin Hogaboam, PHYSICS TODAY, March 2019, page 40.)

The choice of the term "firmware" reflects the fact that although the routines are usually software defined, they reside just above the physical layer in the stack and are effectively invisible to higher layers of abstraction. That approach to low-level control resembles other forms of firmware in computer engineering, such as DRAM (dynamic random-access memory) refresh protocols that stabilize classical storage hardware against degradation caused by charge leakage. Such protocols are responsible for scheduling, defining relevant control and measurement operations, executing logic for actuation, and the like. A user employing DRAM has little awareness of its presence or activity except in the small effects its execution has on, say, memory access latency.

So that's the "what." But what about the "how?"

The underlying technology

Underpinning quantum firmware's functionality is quantum control,[3] a discipline that addresses the question, How can systems that obey the laws of quantum mechanics be efficiently manipulated to create desired behaviors? Ultimately, quantum control is concerned with how the classical world interacts with quantum devices. It guides researchers in gaining information about system dynamics through measurements and enables useful performance in computing, sensing, and metrology. (For more on quantum control see the article by Ian Walmsley and Herschel Rabitz, PHYSICS TODAY, August 2003, page 43.)

The field of quantum control largely owes its existence to decades of research in nuclear magnetic resonance and electron paramagnetic resonance, in which semiclassical magnetizations formed from nuclear or electronic spins are manipulated by pulses of resonant RF or microwave radiation. In those disciplines, hardware imperfections limited the ability to spectroscopically characterize molecules. Then, in 1950, Erwin Hahn demonstrated that a dynamic-control protocol now known as the Hahn echo could mitigate the impacts of magnetic field inhomogeneities on spectroscopic resolution. His discovery led to the development of average Hamiltonian theory, which is used to analyze the temporal evolution of spin systems, and of dynamical decoupling, a technique for canceling the impacts of unwanted spin interactions in molecules.[4]

Beginning in the 1980s, a parallel research discipline emerged that sought to adapt the concepts and numeric tools from control engineering to the strictures of quantum-coherent devices. That included both the treatment of linear systems, such as quantum harmonic oscillators,[5] and the development of numeric techniques for using imperfect hardware to effectively manipulate spin systems.[6] More recently, quantum optimal-control methods have been extended to more general Hilbert spaces and Hamiltonians[7] and have become powerful tools for optimizing quantum experimental system performance.[8]

Much like NMR, quantum computing hardware—whether trapped ions, neutral atoms, superconducting circuits, or another technology—generally relies on precisely engineered light–matter interactions to enact quantum logic. (For more on qubit technology, see the article by Lieven Vandersypen and Mark Eriksson, PHYSICS TODAY, August 2019, page 38.) Those operations constitute the native machine language; a timed Gaussian pulse of microwaves on resonance with a superconducting qubit can act as an X operation, the quantum equivalent of a NOT gate on a single qubit, whereas another pulse may implement a controlled-NOT operation on a pair of qubits, similar to a classical exclusive OR. An appropriately constructed temporal and spatial composition of such electromagnetic signals makes up a quantum algorithm.

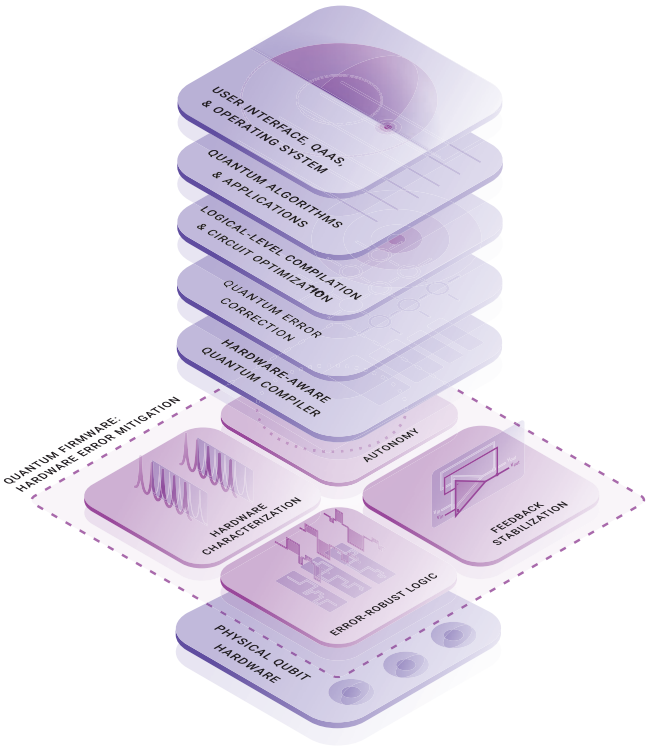


FIGURE 1. THE STACK in a fault-tolerant quantum computer is made of layers that correspond to levels of software abstraction. At the top sits Quantum As A Service (QAAS), which represents functions a user might interact with through, for instance, a cloud service. Below that are quantum algorithms and applications that are coded using developer tools that permit high-level abstraction. The algorithms and applications are compiled on the third level to enact circuits on encoded blocks. In fault-tolerant computing, that enaction is performed on logical qubits encoded using quantum error correction (QEC), although realizing the QEC code and other associated tasks occupies a dedicated layer. Physical connectivity between devices and compensation for any stray couplings are accounted for in a hardware-aware compiler. The quantum firmware layer, which is responsible for minimizing hardware error, resides between that layer and the physical hardware. It handles all tasks necessary for hardware calibration, tune-up, characterization, stabilization, and automation. (Image courtesy of Q-CTRL.)

The physical correspondence between spin-½ systems and qubits builds a natural bridge for transferring quantum-control techniques into quantum information in order to improve algorithmic success despite hardware imperfections and ambient decoherence. One of the clearest efforts to explicitly incorporate quantum control into quantum computers was articulated[2] by N. Cody Jones and coworkers at Stanford University in 2012. They introduced a so-called virtual layer that sat between quantum hardware and higher-level algorithmic abstractions in the quantum computing stack and leveraged

NMR-inspired composite pulsing.[4] Their foundational work inspired the quantum firmware layer described here. Researchers now have greater clarity about both the utility of quantum control and the structure of higher-level software abstractions with which the quantum firmware layer interacts.

Technical aims

Contemporary quantum firmware is charged with implementing the following functionality:

- ▶Error-robust quantum logic operations that are supported by measurement-free open-loop control.
- ▶Measurement-based closed-loop feedback stabilization at the hardware level.
- ▶Microscopic hardware characterization for calibration, noise identification, and Hamiltonian parameter estimation.
- ▶Machine learning–inspired approaches to realize autonomy for the above tasks in large systems.

Open-loop control refers to feedback-free actuation akin to a timed irrigation system that maintains a healthy lawn without information on soil moisture or rainfall. It's resource efficient and has proved to be remarkably effective in stabilizing quantum devices, both during free evolution and during nontrivial logic operations.[9] When open-loop error suppression is used in quantum computers, the instructions for quantum hardware manipulation are redefined such that they execute the same mathematical transformation, but in a way that is robust against error-inducing noise, such as fluctuations in ambient magnetic fields. The suppression is typically realized by temporally modulating the incident control fields that manipulate the physical devices (see the box on page 32), and the modulation patterns may be derived from Hamiltonian models or even machine-learning techniques. Thus the control solutions defined by quantum firmware constitute an effective error-robust machine language for manipulating quantum hardware.

In closed-loop feedback control, actuation is determined by measurements of the system. Its use is constrained by the destructive nature of projective measurement in quantum mechanics. Several strategies may nevertheless be employed for hardware-level feedback-based stabilization; they all are designed to gain sufficient information about the underlying system without destroying encoded information needed in a computation. In fact, QEC—the gold-standard approach for large-scale quantum computers—is a form of closed-loop feedback that employs indirect measurement through ancilla qubits. The direct integration of hardware-level feedback stabilization remains an ongoing area of exploration with some exciting results.[10]

Hardware characterization, known as system identification in the control-theoretic literature, has benefited from a large body of experimental and theoretical developments.[11] The underlying techniques complement external benchmarking routines that quantify the hardware's overall performance by focusing on the determination of actionable microscopic information for system optimization and tune-up. Noise spectroscopy, which is widely used as a complementary capability to noise suppression, provides information

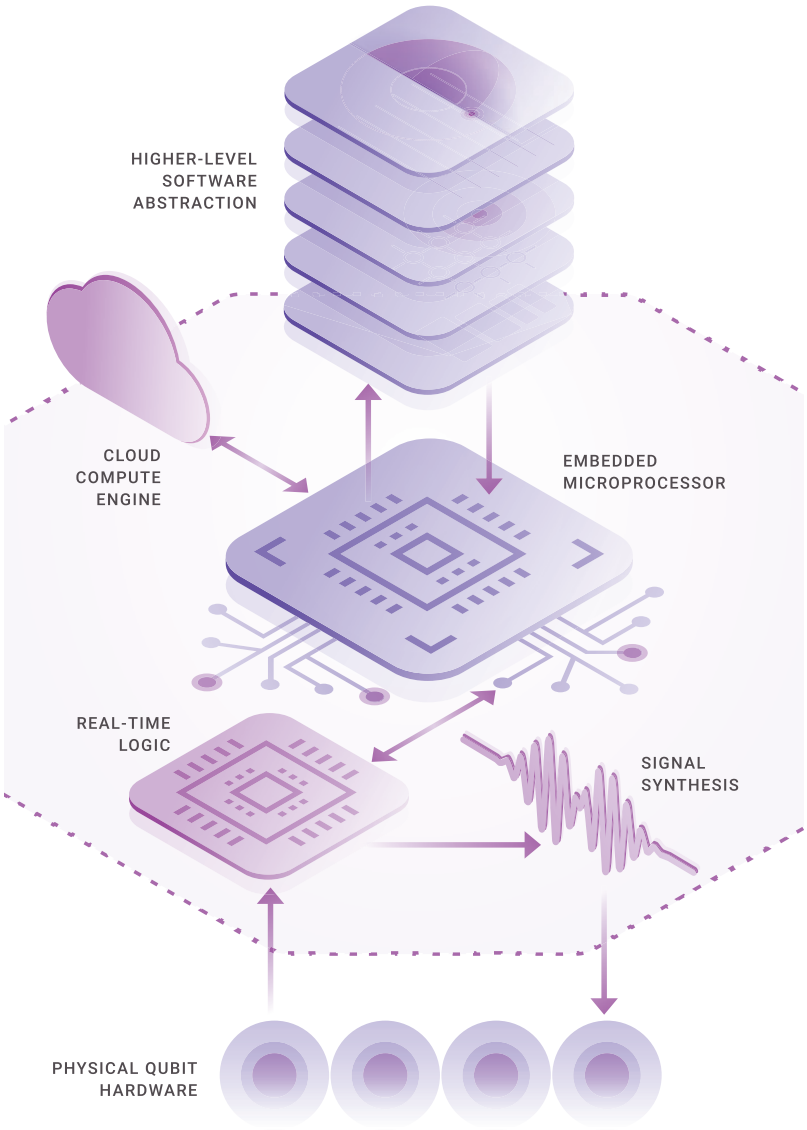


FIGURE 2. QUANTUM FIRMWARE is an abstract layer of the computing stack whose actions are orchestrated by an embedded microprocessor. The microprocessor accesses cloud-computing resources for computationally intense tasks such as open-loop-control optimization and virtualizes the hardware for its interaction with higher layers of the software stack. In the conception shown here, the microprocessor sends commands to programmable logic devices, such as field-programmable gate arrays. Those devices are responsible for processing measurement results in real time for physical-layer feedback stabilization, measurement-based decision making, and other tasks. They also provide instructions to other hardware elements such as direct digital synthesizers and arbitrary waveform generators. Arrows indicate communication pathways between elements. (Image courtesy of Q-CTRL.)

OPTIMIZED GATES FOR SINGLE-QUBIT LOGIC OPERATIONS

Existing medium-scale superconducting quantum computers provide an ideal platform for studies of quantum control because they allow for cloud access to advanced hardware. Using one such platform—a cloud-based IBM quantum computer—the research team at Q-CTRL, a quantum computing startup with facilities in Sydney, Australia, and Los Angeles, California, has explored the efficacy of quantum-control optimization in real systems. They employed specialized analog-layer programming that permits direct control of physical signals.

As an example, we demonstrate how to make an effective machine language that defines quantum logic operations that are resilient against the typical sources of hardware instability. In the illustration here, we show different techniques to produce gates that perform a Pauli X "spin-flip" operation,

$$\sigma_x = |0\rangle\langle 1| + |1\rangle\langle 0| = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

which is the quantum mechanical analog of a classical NOT gate. In each of the Bloch spheres shown, the quantum state—represented by the locus of a unit-length vector on the sphere's surface—follows a path from the sphere's north pole to its south pole. However, the qubit subject to a default implementation (panel a) takes a substantially different path from those of the two qubits subject to error-robust pulses (panels b and c).

The Pauli X operation is implemented using a pulse of microwave radiation that enacts a control Hamiltonian

$$H_c(t) = \tfrac{1}{2}\,(\Omega(t)e^{i\varphi(t)}\hat{a}).$$

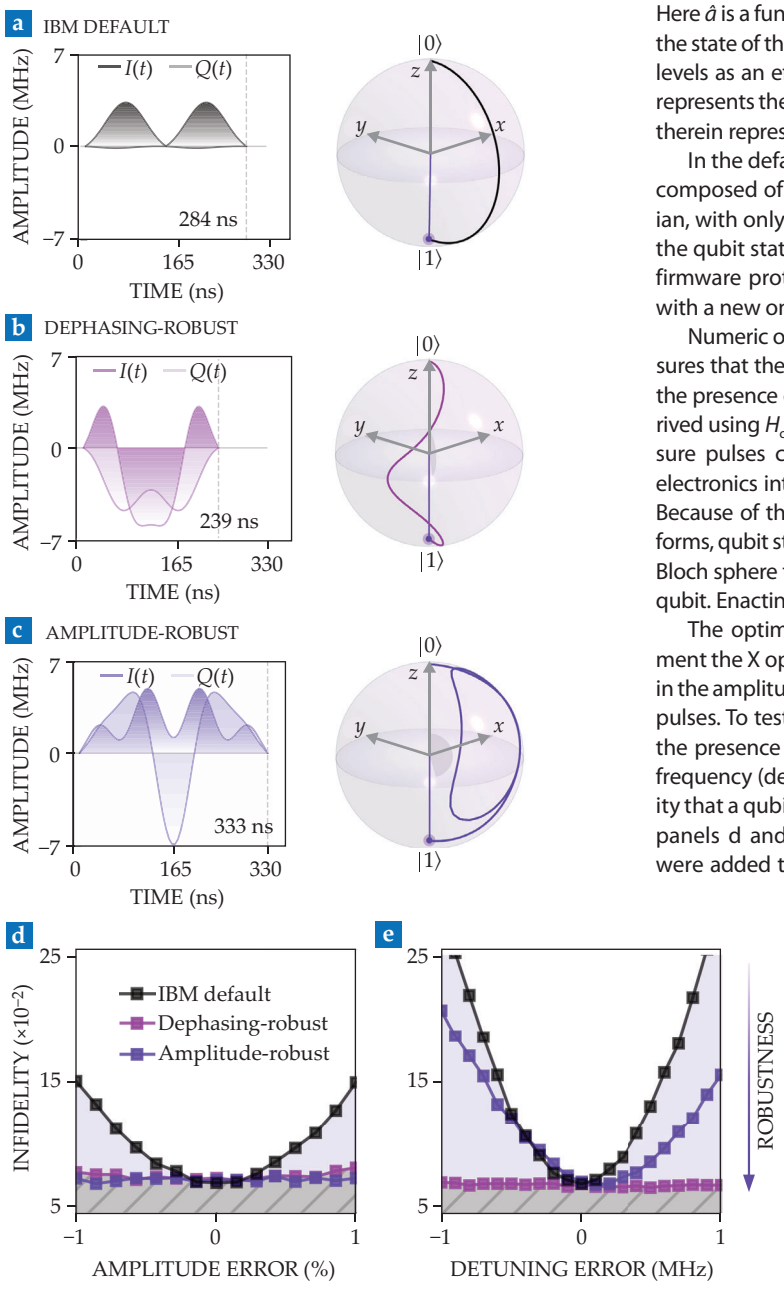
Here $\hat{a}$ is a function of a and $a^\dagger$, the lowering and raising operators for the state of the superconducting qubit. (We treat only the two lowest levels as an effective qubit.) The coupling term $\Omega(t)e^{i\varphi(t)} \equiv I(t) + iQ(t)$ represents the control-pulse waveform, and the functions $I(t)$ and $Q(t)$ therein represent user-adjustable controls.

In the default implementation of the X operation (panel a), $I(t)$ is composed of two sequential pulses that are approximately Gaussian, with only a small component in $Q(t)$. Those pulses largely drive the qubit state along a meridian of the Bloch sphere. In a quantum firmware protocol, the simple physical definition for X is replaced with a new one that parameterizes the gate in terms of $I(t)$ and $Q(t)$.

Numeric optimization is used to minimize a cost function that ensures that the quantum logic gate is implemented correctly, even in the presence of noise. The controls applied in panels b and c are derived using $H_c(t)$ and take into account smoothing functions that ensure pulses can be faithfully transmitted from room-temperature electronics into a dilution refrigerator where the qubits are housed. Because of the error reduction incorporated into the control waveforms, qubit states subject to optimized controls take paths along the Bloch sphere that are more complex than those taken by the default qubit. Enacting those complex paths often requires a longer pulse.

The optimized controls described here are designed to implement the X operation in a manner that is robust against either errors in the amplitude $\Omega(t)$ or in the driving frequency that implements the pulses. To test the control's performance, it is repeatedly applied in the presence of either quasistatic pulse-amplitude errors or pulse-frequency (detuning) errors, and the gate's infidelity—the probability that a qubit state evolves to the wrong target—is measured (see panels d and e). Even when large amplitude or detuning errors were added to the applied pulse, each optimized solution shows a flat response, which is a signature of robustness. The control designed to be robust against dephasing (panel b) is indeed flat in the presence of dephasing errors (panel e). Likewise, the amplitude-robust control response (panel c) is flat despite amplitude errors (panel d).

The shaded areas in panels d and e indicate the overall improvement achieved through the choice of appropriate optimized pulses. Additional controls designed to exhibit robustness to both error processes simultaneously have also been demonstrated. (For a full explanation of the experiment described here, see reference 9.)



used in the design of open-loop controls. To implement it, a qubit can serve as a sensor that when subjected to appropriately designed time-domain control, probes noise at different frequencies.

Hardware imperfections can be identified through various techniques generally classified as Hamiltonian parameter estimation. A simple implementation might allow for the determination of the coupling rates between physical devices and control fields and reveal transmission losses experienced by signals en route to the qubits. More complex routines are commonly employed to determine the frequencies of unwanted energy levels in a device or to characterize unknown couplings between qubits. The information obtained from such protocols informs open-loop error-suppressing controls and dynamic models for feedback-based stabilization.

Artificial intelligence–enabled autonomy, the final class of techniques in quantum firmware operation, represents one of the most exciting exploratory areas of research in quantum information. Scaled-up systems will require high-efficiency routines that can tune up, calibrate, optimize, and characterize the underlying hardware with minimal user intervention. An interdisciplinary effort integrating machine learning, robotic control, and data inference is showing how adaptive measurement routines may be deployed to reduce the number of destructive measurements required[1,12] and to enable rapid autonomous system bring-up and operation.

Integration strategies

Any practical implementation of quantum control must be tailored to the needs of a hardware system; each scheme will require a particular subset of the functions described above. But the control techniques have much in common, and they are increasingly being used in state-of-the-art experimental and commercial-grade quantum computers. (See PHYSICS TODAY, November 2020, page 22, for more about the commercialization of quantum computers.) Researchers have thus been motivated to organize the relevant functionalities into an identifiable abstraction layer. Creating that organizational structure, however, is distinct from determining how quantum control should be integrated into real systems.

One approach involves integrating the control functions into their own encapsulated layer in the quantum computing stack. In that conception, quantum firmware is responsible for defining and executing all actions that bridge the gap between high-level abstractions, such as compilation or application programming, and the many low-level quantum-control routines customized for particular hardware systems. Firmware can be embedded into appropriate computational hardware to virtualize the underlying quantum-coherent hardware. That is, the firmware changes the behavior and performance of the hardware such that high-level abstraction layers have no visibility into the "bare" performance of the underlying hardware.

The implementation of a dedicated quantum firmware layer brings several potential benefits. First, the development of efficient high-level programming frameworks such as Cirq, Quil, and Qiskit has led to an explosion of capability at the application and algorithmic level. Building a framework to standardize quantum-control integration may also encourage the quantum-control and machine-learning communities to develop more diverse technical solutions for efficient hardware manipulation.

A dedicated firmware layer could autonomously orchestrate quantum-control tasks that span different classical computational hardware. Those processes could exploit local processing to support automated scheduling and unsupervised stabilization, distributed computing infrastructure to execute computationally intensive optimization tasks, and low-latency programmable logic to conduct real-time processing. For example, a local microcontroller can, on a schedule, initiate a cloud-based numeric optimization of a multiqubit gate (see figure 2). That solution can be used to seed a hardware-executed tune-up of the final control waveform, which is then written to embedded memory. Slaved to the microcontroller is a field-programmable gate array that both directs signal-synthesis hardware to output the waveform used to manipulate the qubits and also processes measurement results from the qubits. (A logically distinct "embedded operating system" always remains that defines and enables the functionality of the classical electronics in use.)

For the near term, researchers are exploring how the distinctions between layers in the emerging stack could be blurred to deliver maximum performance. For example, it's possible to pursue a hardware–firmware co-design strategy to directly integrate certain critical tasks into the classical electronics[13] while others remain in the experimental software.

Opportunities may also arise to fundamentally rethink the organization of quantum computer software stacks based on the functionality provided through exploitation of quantum control. The potential value of such approaches is evident in hardware-aware compilation, in which optimal control is used to efficiently produce high-fidelity hardware-optimized logical blocks. Quantum algorithms may then be compiled into a library of numerically optimized "analog" control sequences that would replace a smaller but more general set of universal gates.[14]

System-level impacts

Regardless of how quantum firmware is realized, recent experiments have made clear that the quantum-control functionality encompassed therein could affect or even reshape higher abstraction layers. That's because the virtualization produced by quantum firmware fundamentally transforms the behavior of the underlying hardware, especially as it pertains to the characteristics of hardware errors.

Open-loop control strategies are broadly used to suppress errors in state-of-the-art quantum computer hardware; for example, in certain settings, DRAG (derivative removal by adiabatic gate) pulses—an example of open-loop control—have been shown to reduce gate errors in superconducting qubits by approximately an order of magnitude compared with conventional Gaussian pulses. More recent results demonstrate that numerically optimized gates can mitigate the effects of hardware imperfections in cloud quantum computers, thereby suppressing pulse-amplitude, off-resonance, and cross-talk errors. Those demonstrations are particularly powerful because the error processes effectively addressed by quantum firmware often generate far larger effects than one would expect from best-case-scenario benchmark routines.

In both research-grade systems and publicly available cloud

systems, best- and worst-case qubit error rates across a device often differ by more than an order of magnitude. Those errors can arise from fabrication variances and coupling inhomogeneities between qubits and the ambient electromagnetic environment. Quantum firmware homogenizes hardware performance in space and time. Optimized quantum-control operations implemented in real systems have brought error rates for all qubits close to the best-case performance; likewise, drift-robust controls can extend typical calibration windows on cloud and laboratory hardware from 6–12 hours to more than five days.[9]

Why do those improvements matter? To start, current algorithmic compilers can improve performance by trading an increase in compiled-circuit complexity for the ability to avoid poorly performing devices. But in large-scale systems with substantial performance variation, that compilation process can become quite complicated, and shuttling information around the worst-performing devices may require many more gates and time steps. By homogenizing device performance in space and time, quantum firmware can simplify higher-level compilation protocols,[15] thereby reducing the complexity and duration of the implemented algorithm.

Quantum control will also have a long-term impact on the performance of QEC. Both the hardware-level feedback stabilization and open-loop control found in firmware exploit the fact that noise processes often vary slowly in space and time; those methods provide little benefit for truly stochastic errors. On the other hand, QEC formulations generally assume statistically independent error models. Thus quantum firmware works in concert with QEC to correct for a broad range of error types and effectively preconditions the properties of the residual errors to be compatible with QEC.[16] But more than that, the way in which quantum firmware closes the gap between the best and worst performing qubits and reduces statistical correlations in the residual errors[17] actually reduces QEC's resource intensity. It's a win–win combination.

The future of quantum firmware

Quantum computing is complex, so algorithm designers and end users need a framework through which they can efficiently exploit quantum computers without having detailed technical knowledge of the underlying hardware. They expect high-performance quantum hardware to be stable and provide consistent outputs irrespective of small changes in an algorithm's structure. Quantum firmware enables those capabilities.

Quantum-control demonstrations have confirmed improvements of about a factor of 10 in the performance of quantum logic operations relative to naive gate implementations.[9] Similarly, dynamic memory stabilization has extended qubit lifetimes to time scales measured in minutes. In those settings, the performance gains have been limited by either incoherent processes or the capabilities of classical electronics, but both are showing steady gains with time and specialization for the quantum computing market. We therefore expect that control systems and device performance will improve in parallel with quantum firmware protocols.

The effect of using quantum-control technologies such as error-robust open-loop control on algorithmic performance can be quantified with benchmarks. One such benchmark is quantum volume, a metric that accounts for architectural features, including hardware connectivity, and device-level parameters, such as the one- and two-qubit error rates across the device.[18] Honeywell has claimed a quantum volume of 128 with just a handful of qubits compared with approximately 64 from IBM's larger systems; the results demonstrate that hardware performance is the primary bottleneck.

Improving both one- and two-qubit error rates by more than a factor of 10, as has been demonstrated experimentally, would have a massive impact on system-level performance. Those impacts would be largest in devices with weak connectivity, where the spatial rearrangement of qubit data requires many multiqubit swap operations. Device sizes are increasing rapidly—Google and IBM have each released a road map to 1000-qubit systems—and quantum control provides a means to ensure system utility at the algorithmic level tracks with system size.

Ultimately, we believe that building and operating large-scale quantum computers is effectively impossible without integrating advanced quantum-control techniques into a quantum firmware abstraction layer. Autonomous vehicles, walking robots, and advanced avionics systems have all demonstrated the importance of dynamic control and automation. Similarly, in quantum computing, advanced control-theoretic strategies were instrumental in the calibration and tune-up of devices used to achieve quantum supremacy. Many techniques from the fields of machine learning and robotic control are likely to improve performance and increase autonomy, thereby allowing future quantum developers to confidently abstract away the details of a computer's underlying hardware.

REFERENCES

1. F. Arute et al., *Nature* **574**, 505 (2019).
2. N. Cody Jones et al., *Phys. Rev. X* **2**, 031007 (2012).
3. H. M. Wiseman, G. J. Milburn, *Quantum Measurement and Control*, Cambridge U. Press (2011).
4. U. Haeberlen, J. S. Waugh, *Phys. Rev.* **175**, 453 (1968).
5. T. J. Tarn, G. Huang, J. W. Clark, *Math. Modell.* **1**, 109 (1980); J. W. Clark, D. G. Lucarelli, T.-J. Tarn, *Int. J. Mod. Phys. B* **17**, 5397 (2003); D. Dong, I. R. Petersen, *IET Control Theory Appl.* **4**, 2651 (2010); L. Bouten, R. Van Handel, M. R. James, *SIAM J. Control Optim.* **46**, 2199 (2007); H. I. Nurdin, M. R. James, I. R. Petersen, *Automatica* **45**, 1837 (2009).
6. N. Khaneja et al., *J. Magn. Reson.* **172**, 296 (2005).
7. S. Chaudhury et al., *Phys. Rev. Lett.* **99**, 163002 (2007); A. Smith et al., *Phys. Rev. Lett.* **111**, 170502 (2013); R. W. Heeres et al., *Nat. Commun.* **8**, 94 (2017).
8. A. D. Tranter et al., *Nat. Commun.* **9**, 4360 (2018); P. B. Wigley et al., *Sci. Rep.* **6**, 25890 (2016); B. M. Henson et al., *Proc. Natl. Acad. Sci. USA* **115**, 13216 (2018).
9. A. R. R. Carvalho et al., https://arxiv.org/abs/2010.08057.
10. N. Ofek et al., *Nature* **536**, 441 (2016).
11. H. Ball et al., https://arxiv.org/abs/2001.04060.
12. R. S. Gupta et al., *NPJ Quantum Inf.* **6**, 53 (2020).
13. H. Ball et al., *Phys. Rev. Appl.* **6**, 064009 (2016); S. J. Pauka et al., https://arxiv.org/abs/1912.01299; V. Negnevitsky et al., *Nature* **563**, 527 (2018).
14. Y. Shi et al., in *ASPLOS '19: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Association for Computing Machinery (2019), p. 1031.
15. S. Nishio et al., https://arxiv.org/abs/1903.10963; P. Murali et al., https://arxiv.org/abs/1901.11054.
16. J. P. Barnes et al., *Phys. Rev. A* **95**, 062338 (2017).
17. C. L. Edmunds et al., *Phys. Rev. Res.* **2**, 013156 (2020).
18. A. W. Cross et al., *Phys. Rev. A.* **100**, 032328 (2019).

PT