A Pulse-Level DSL for Real-Time Quantum Control with Hardware Compilation and Emulation

YU-HSUAN WU, Academia Sinica, Taiwan
YUE SHI, Princeton University, USA
JUNYI LIU, University of Maryland, USA
YUXIANG PENG, Purdue University, USA

Quantum computers are promising for solving classically intractable problems, but their practical utility hinges on precise, flexible, and accessible programming of quantum control systems. Despite several pulse-level languages for industrial devices, a systematic and end-to-end programming toolchain for real-time quantum control remains lacking. We introduce a domain-specific language (DSL) for pulse scheduling with native real-time control constructs (e.g., feedback, branching, and pulse updates). The DSL compiles to and executes on radio-frequency system-on-chips, enabling deterministic timing and hardware portability. To validate behaviors across the stack, we provide an emulator that co-simulates control hardware and quantum device responses. Together, the DSL, compiler, and emulator form a cohesive framework that lowers the barrier to implementing robust control, accelerates architectural design, and supports the development and testing of applications like quantum error correcting codes.

1 Introduction

Quantum computers promise advantages across domains—from integer factorization with Shor's algorithm [17] to quantum chemistry with quantum Hamiltonian simulation [7, 11]. Realizing these applications at useful scales, however, requires fault-tolerant quantum computing [19], where logical quantum gates are protected by error-correcting codes [2, 5, 8, 18]. Fault tolerance, in turn, hinges on fast, reliable real-time feedback control of quantum systems: repeatedly measuring qubits, extracting syndromes, and conditionally applying corrections to keep logical error rates below threshold.

One prominent property of quantum hardware is the continuous drifting [1, 15]: there is always a system Hamiltonian driving the evolution of the quantum state, even in the absence of any control fields. Any delay between operations therefore lands effects different from the desired. Precise feedback control must, accordingly, fix and account for timing at fine granularity, including deterministic latencies and synchronized pulse generation. These guarantees are only expressible at the pulse level, where programs specify waveforms, phases, and control flows with unambiguous real-time semantics rather than abstract gates.

Despite notable efforts [3, 12, 14, 16], pulse-level languages and toolchains remain fragmented. Many existing prototypes are designed around specific industrial quantum hardware and expose only limited, vendor-defined operations for pulse-level features, rarely real-time control. This leads to three practical issues. First, opaque control hardware hides critical implementation details, obscuring the design and reason about desired hardware

Authors' Contact Information: Yu-Hsuan Wu, Academia Sinica, Taiwan, maxyhwu@iis.sinica.edu.tw; Yue Shi, Princeton University, USA, ys2251@princeton.edu; Junyi Liu, University of Maryland, USA, junyiliu@umd.edu; Yuxiang Peng, Purdue University, USA, yxpeng@purdue.edu.

behaviors. Second, ambiguous language constructs hinder programming: programmers cannot reason rigorously about timing determinism, concurrency, or feedback latency. Third, vendor-specific software can deprecate or diverge sorely due to business purposes, derailing reproducibility and cross-hardware experimentation. Pulse-level tools for real-time quantum control—requiring an open and long-supported hardware—remain underserved.

This reality pushes us to systematically design pulse-level programming languages on open hardware [4, 6]. To this end, we introduce a domain-specific language (DSL) for pulse scheduling with native real-time control. Our DSL satisfies several key features: (1) *precise semantics* for timing and concurrency on on-chip CPU and pulse generators; (2) *first-class real-time control*, including conditional branching, control condition computing, and in-situ pulse generation; (3) *portability* across heterogeneous controllers and quantum hardware via an explicit quantum architecture and general control hardware components.

To execute programs in our DSL, we provide a compiler towards a radio-frequency system-on-chip (RFSoC) controller that integrates DAC/ADC and FPGA compute. The compiler maps high-level constructs to binaries on RISC-Q [10], an instruction set executable on Xilinx RFSoCs with extremely low latency. Deploying the executable on an RFSoC controlling a quantum hardware, we can precisely orchestrate the evolution of the quantum system with controls based on intermediate measurements. Our compilation framework is compatible with controllers like arbitrary wave generators, supported in future works.

Because control hardware and quantum devices are expensive and scarce, we complete the suite with an emulator that co-simulates the controller and a configurable quantum device model. This allows rapid prototyping of control programs, debugging and validation of programs, and testing of real-time behaviors (e.g., latencies and race conditions) before consuming lab time. The executable compiled from DSL programs can run on both the emulator and RFSoC hardware, enabling a smooth path from design to deployment.

In summary, this work contributes: (1) a pulse-level DSL for real-time quantum control; (2) a compiler onto RFSoCs; (3) an emulator co-simulating the control and quantum hardware.

2 Language and Compiler

The core idea of the language design is based on the need for targeting pulse-level programming for real-time quantum control systems with dynamic scheduling capability, and thus resulting a combination of classical controls with essential pulse-level operations. Figure 1 shows the syntax of the language, while Figure 3a demonstrates what the program looks like in the two examples presented in Section 4. We would also like to point out that the delay operation is omitted here, since we take it as a syntactic sugar of the play operation with a 0 amplitude pulse.

Operational Semantics. We define the semantics of the DSL based on the decoupling of the CPU time (denoted as t^c) and the Scheduler time (denoted as t^q), as the execution time of classical operations should not affect the increment on the pulse scheduler clock, while the reverse way does. This explains the differences in time increment when viewing the semantics. We follow the "as soon as possible" (ASAP) scheduling scheme between operations, and two predicates, no-conflict-Q_{sin} for play operation, and no-conflict-Q_{par} for

Identifiers.

$$p \in \text{PVar} \quad s \in \text{SVar} \quad ch \in \text{CVar} \quad d \in \text{TimeVar}$$
$$r \in \text{RealVar} \quad b \in \text{BoolVar}$$

Types.

$$T ::= () \mid \text{Pulse} \mid \text{Shape} \mid \text{Channel} \mid \text{Time} \mid \text{Real} \mid \text{Bool}$$

Expressions.

$$\begin{aligned} E ::= \;& \text{skip} \\ \mid\; & r := e_a \quad \text{(assignment)} \\ \mid\; & E_1 \,;\, E_2 \quad \text{(sequential composition)} \\ \mid\; & \text{if } c \text{ then } E_1 \text{ else } E_2 \quad \text{(conditional)} \\ \mid\; & \text{while } c \text{ do } E \quad \text{(loop)} \\ \mid\; & \text{define } p := \text{pulse}(s, r_a, d, r_p, r_f) \\ & \quad \text{(shape, amplitude, duration, phase, frequency)} \\ \mid\; & \text{run } Q \quad \text{(run pulse operations)} \end{aligned}$$

Arithmetic Expressions.

$$e_a ::= n \mid r \mid e_a \oplus e_a \mid -e_a \mid (e_a)$$
$$n \in \mathbb{R} \quad \oplus ::= + \mid - \mid \times \mid /$$

Conditions.

$$c ::= \text{false} \mid \text{true} \mid b \mid e_a \bowtie e_a \mid \neg c \mid c \wedge c \mid c \vee c$$
$$\bowtie ::= < \mid > \mid = \mid \neq$$

Pulse Operations.

$$\begin{aligned} Q ::= \;& \text{skip}_{\text{Q}} \\ \mid\; & Q_1 \,;\, Q_2 \quad \text{(sequential pulse operations)} \\ \mid\; & \text{play } ch\ p \quad \text{(play pulse)} \\ \mid\; & \text{measure } ch\ p \to b \quad \text{(measurement)} \\ \mid\; & \text{par}_{\text{Q}}\{Q_1 \parallel \cdots \parallel Q_n\} \quad \text{(parallel pulse operations)} \end{aligned}$$

Fig. 1. Syntax of the DSL.

parallel operation, are used to guarantee the validity of the schedule. The schedule *dynamically* updates during runtime and stored in the program state, and the pulse environment records defined pulses for reuse. Figure 2 shows the remaining details.

Compiler. To execute programs written in our DSL, we implement a compilation framework that targets an RFSoC–based control architecture. Our compiler is based on RISC-Q, which integrates the memory-map and RISC-V compiler, along with utilizing only the RV32I and RV32M set. It translates high-level DSL programs into lightweight RISC-Q binaries, producing executables on Xilinx RFSoCs with extremely low latency.

3 Emulator

To show the usability of our language, we provide a real-time emulator by connecting a quantum control simulator and a quantum processor simulator using real-time communication to run compiled binaries. Specifically, we utilize the architecture described in [10] as the target of the quantum control simulator, since it depicts a minimal but ideal framework for simulating low-latency quantum control system-on-chips and for displaying our language prototype, and run the simulation using Verilator [20]. The simulated RISC-V CPU is set to run at 500 MHz, with DACs running 16x faster at 8 GHz, and ADCs running at a 4x faster 2 GHz frequency. We provide a DAC drive channel, a DAC measurement channel, and an ADC channel per qubit.

We adopt QuTiP-QIP [9] package for simulating quantum systems. Multiple backend quantum hardware models are available through QuTiP-QIP, and we leverage the linear spin chain model as an example: Given σ_i^x, σ_i^y, and σ_i^z as the single-qubit control Hamiltonian, $\sigma_i^x \sigma_{i+1}^x + \sigma_i^y \sigma_{i+1}^y$ as the exchange Hamiltonian for interaction, $\Omega_i^x, \Omega_i^y, \Omega_i^z$, and g_i as the control coefficients, and N as the number of qubits, the one-dimensional, open-ended chain layout gives us the control Hamiltonian as $H = \sum_{i=0}^{N-1} \Omega_i^x(t)\sigma_i^x + \Omega_i^y(t)\sigma_i^y + \Omega_i^z(t)\sigma_i^z + \sum_{i=0}^{N-2} g_i(t)(\sigma_i^x \sigma_{i+1}^x + \sigma_i^y \sigma_{i+1}^y)$, and by definition, we have $\Omega_i^x = E_i \cos\phi_i$ and $\Omega_i^y = E_i \sin\phi_i$, where E_i and ϕ_i are the envelope and the phase of the control pulse.

The connection between the quantum control simulator and the quantum processor simulator is via multi-thread FIFOs targeting DACs/ADCs. We also implement a Wait-Until-Fetch scheme, as the calculation time per batch on the processor side is incomparable

Configurations. Program evaluation is defined as:

$$\Gamma \vdash \langle e, \sigma, t^q, t^c \rangle \to \Gamma' \vdash \langle e', \sigma', t^{q\prime}, t^{c\prime} \rangle$$

where:

- Γ, Γ' — program context
- e, e' — program expression
- $\sigma = (\gamma, \delta, \pi), \sigma' = (\gamma', \delta', \pi')$ — program state:
  - γ, γ' — classical store, mapping:

$$\text{RealVar} \cup \text{BoolVar} \cup \text{TimeVar} \to \text{Real} \cup \text{Bool} \cup \text{Time}$$

  - δ, δ' — scheduled pulses:

$$\text{Channel} \to [(\text{Pulse}, \text{Time} \times \text{Time})]$$

  - π, π' — pulse environment:

$$\text{PVar} \to \text{Pulse}$$

- $t^q, t^{q\prime} \in \text{Time}$ — current Scheduler time
- $t^c, t^{c\prime} \in \text{Time}$ — current CPU time

Condition Evaluation.

$$\begin{array}{rcl} \text{true} & \Downarrow & \text{true} \\ \text{false} & \Downarrow & \text{false} \\ \neg c & \Downarrow & \text{not } (c \Downarrow \text{true}) \\ c_1 \wedge c_2 & \Downarrow & \text{true iff } c_1 \Downarrow \text{true and } c_2 \Downarrow \text{true} \\ c_1 \vee c_2 & \Downarrow & \text{true iff } c_1 \Downarrow \text{true or } c_2 \Downarrow \text{true} \end{array}$$

Skip. skip is a terminal form and has no transition.

Assignment.

$$\frac{e_a \Downarrow v}{\Gamma \vdash \langle r := e_a, (\gamma, \delta, \pi), t^q, t^c \rangle \to \langle \text{skip}, (\gamma[r \mapsto v], \delta, \pi), t^q, t^{c\prime} \rangle}$$

Sequential Composition.

$$\frac{\langle e_1, \sigma, t^q, t^c \rangle \to \langle e_1', \sigma', t^{q\prime}, t^{c\prime} \rangle}{\Gamma \vdash \langle e_1; e_2, \sigma, t^q, t^c \rangle \to \langle e_1'; e_2, \sigma', t^{q\prime}, t^{c\prime} \rangle}$$

$$\frac{}{\Gamma \vdash \langle \text{skip}; e_2, \sigma, t^q, t^c \rangle \to \langle e_2, \sigma, t^q, t^c \rangle}$$

If-Else.

$$\frac{\gamma \vdash c \Downarrow \text{true}}{\Gamma \vdash \langle \text{if } c \text{ then } e_1 \text{ else } e_2, \sigma, t^q, t^c \rangle \to \langle e_1, \sigma, t^q, t^{c\prime} \rangle}$$

$$\frac{\gamma \vdash c \Downarrow \text{false}}{\Gamma \vdash \langle \text{if } c \text{ then } e_1 \text{ else } e_2, \sigma, t^q, t^c \rangle \to \langle e_2, \sigma, t^q, t^{c\prime} \rangle}$$

While.

$$\frac{}{\Gamma \vdash \langle \text{while } c \text{ do } e, \sigma, t^q, t^c \rangle \to \langle \text{if } c \text{ then } (e; \text{while } c \text{ do } e) \text{ else skip}, \sigma, t^q, t^{c\prime} \rangle}$$

Define Pulse.

$$\frac{}{\Gamma \vdash \langle \text{define } p = \text{pulse}(s, a, d, \phi, \omega), (\gamma, \delta, \pi), t^q, t^c \rangle \to \langle \text{skip}, (\gamma, \delta, \pi[p \mapsto \text{pulse}(s, a, d, \phi, \omega)]), t^q, t^{c\prime} \rangle}$$

Run Pulse Operations.

$$\frac{}{\Gamma \vdash \langle \text{run } Q, \sigma, t^q, t^c \rangle \to \langle \text{skip}, \sigma', t^{q\prime}, t^{c\prime} \rangle}$$

Skip for Pulse Operations. skip_Q is a terminal form and has no transition.

Sequential Pulse Operations.

$$\frac{\langle Q_1, \sigma, t^q, t^c \rangle \to \langle Q_1', \sigma', t^{q\prime}, t^{c\prime} \rangle}{\Gamma \vdash \langle Q_1; Q_2, \sigma, t^q, t^c \rangle \to \langle Q_1'; Q_2, \sigma', t^{q\prime}, t^{c\prime} \rangle}$$

$$\frac{}{\Gamma \vdash \langle \text{skip}_Q; Q_2, \sigma, t^q, t^c \rangle \to \langle Q_2, \sigma, t^q, t^c \rangle}$$

Play Pulse.

$$\text{no-conflict-Q}_{\text{sin}}(\delta, ch, t_1, t_2) \triangleq \forall (t_s, t_e) \in \delta(ch) \text{ s.t. } [t_1, t_2) \cap [t_s, t_e) = \emptyset$$

$$\frac{\pi(p) = \text{pulse}(s, a, d, \phi, \omega) \quad \text{no-conflict-Q}_{\text{sin}}(\delta, ch, t^q, t^{q\prime})}{\Gamma \vdash \langle \text{play}(ch, p), (\gamma, \delta, \pi), t^q, t^c \rangle \to \langle \text{skip}_Q, (\gamma, \delta', \pi), t^{q\prime}, t^{c\prime} \rangle}$$

where: $t^{q\prime} = t^q + d, \quad \delta' = \delta \cup \{(ch, p, t^q, t^{q\prime})\}$

Parallel Pulse Operations.

$$\text{no-conflict-Q}_{\text{par}}(\delta_i, \delta_j) \triangleq \forall\, ch \in \text{dom}(\delta_i) \cap \text{dom}(\delta_j), \forall\, (t_s, t_e) \in \delta_i(ch),\ (t_s', t_e') \in \delta_j(ch) \text{ s.t. } [t_s, t_e) \cap [t_s', t_e') = \emptyset$$

$$\frac{\forall i,\ \langle Q_i, \sigma, t^q, t^c \rangle \to^* \langle \text{skip}_Q, (\gamma_i, \delta_i, \pi_i), t_i^q, t_i^c \rangle \quad t_{\max}^q = \max_i t_i^q \quad \forall i \neq j,\ \text{dom}(\delta_i) \cap \text{dom}(\delta_j) = \emptyset \ \vee\ \text{no-conflict-Q}_{\text{par}}(\delta_i, \delta_j)}{\Gamma \vdash \langle \text{par}_Q\{Q_1 \parallel \cdots \parallel Q_n\}, \sigma, t^q, t^c \rangle \to \langle \text{skip}_Q, (\bigcup_i \gamma_i, \bigcup_i \delta_i, \bigcup_i \pi_i),\ t_{\max}^q, t^{c\prime} \rangle}$$

Measurement.

$$\frac{\pi(p_{\text{measure}}) = \text{pulse}(s, a, d, \phi, \omega) \quad \text{meas}[i] \Downarrow b}{\langle \text{play}(ch, p_{\text{measure}}); x := \text{meas}[i], \sigma, t^q, t^c \rangle \to^* \langle \text{skip}, (\gamma[x \mapsto b], \delta', \pi), \max(t^{q\prime}, t^{c\prime}), t^{c\prime} \rangle}$$

where: $t^{q\prime} = t^q + d, \quad \delta' = \delta \cup \{(ch, p_{\text{measure}}, t^q, t^{q\prime})\}, \quad \text{meas}[i]$ = decoded measurement result of channel i

Fig. 2. Operational semantics of the DSL.

to the cycle time on the control side, the hardware clock will be held and wait for returning pulses to be fetched by ADCs before incrementing the clock, to ensure concurrency.

4 Examples

We run two experiments, Rabi oscillation and quantum tomography, to display two examples of our DSL, compile and execute them on the emulator.

Rabi oscillation. In quantum hardware, two energy levels are chosen as the qubit states. Coherent control between these states—essential for quantum operations—is achieved by

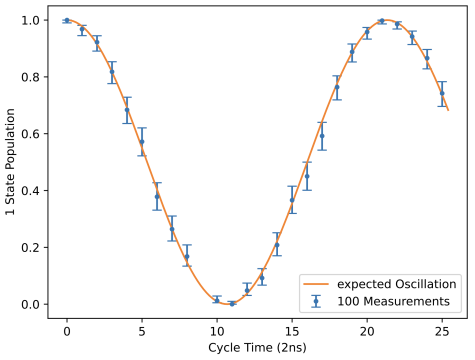
applying an oscillatory drive resonant with their energy splitting [13], a process known as Rabi oscillation. In our experiment, we generate this drive (corresponding to the σ_x term in the Hamiltonian in Section 3) using cosine pulses from high-speed DACs and measure the response through DACs and ADCs on the same chip. By varying the drive duration via our DSL, we observe coherent state population oscillations on the emulator, consistent with the expected Rabi dynamics. Figure 3b shows the results, and the corresponding DSL program is in Figure 3a.

Quantum tomography. Measuring only the population of a quantum state does not provide complete information about the state. For instance, the states $|0\rangle + |1\rangle$ and $|0\rangle - |1\rangle$ yield identical population statistics despite having opposite relative phases. Moreover, decoherence can transform a pure state into a mixed state—indistinguishable from a pure state by population measurements alone. To fully characterize an unknown state, quantum state tomography is employed, in which the state is repeatedly prepared and measured in a complete set of orthogonal bases [21].

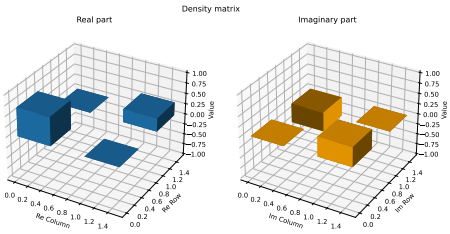
In our experiment, we generate a qubit state by applying a random Rabi-drive duration, then perform quantum state tomography pulse sequence via our DSL. Basis changes are implemented by applying single-qubit σ_x and σ_y rotations using the same DACs that drive the Rabi oscillations. Figure 3c shows the reconstructed state obtained from tomography.

```
#include "pulsedsl.h"

int main() {
  Init(4);

  PulseParams p_m = {
    .shape = CONSTANT,
    .amplitude = 0x7ff0,
    .duration = 50,
    .phase = PHASE_PI(0.5),
    .frequency = FREQ_GHZ(0.1) };

  for (int i = 1; i <= 20; i++) {
    PulseParams p_dx = {
      .shape = CONSTANT,
      .amplitude = 0x7ff0,
      .duration = i,
      .phase = PHASE_PI(0.0),
      .frequency = FREQ_GHZ(0.5) };

    Play(p_dx, 1);
    Measure(p_m, 1);
  }
  return 0;
}
```

(a) DSL program used for the Rabi oscillations



(b) Rabi oscillations



(c) Tomography at cycle time 4

Fig. 3. Programs in our language and emulation results.

References

[1] Mohammed AbuGhanem. 2025. IBM Quantum Computers: Evolution, Performance, and Future Directions. *The Journal of Supercomputing* 81 (2025), 687. doi:10.1007/s11227-025-07047-7

[2] Google Quantum AI and Collaborators. 2025. Quantum Error Correction Below the Surface Code Threshold. *Nature* 638 (2025), 920–926. doi:10.1038/s41586-024-08449-y

[3] Thomas Alexander, Naoki Kanazawa, Daniel J Egger, Lauren Capelluto, Christopher J Wood, Ali Javadi-Abhari, and David C McKay. 2020. Qiskit pulse: programming quantum computers through the cloud with pulses. *Quantum Science and Technology* 5, 4 (Aug. 2020), 044006. doi:10.1088/2058-9565/aba404

[4] Susan M. Clark, Daniel Lobser, Melissa C. Revelle, Christopher G. Yale, David Bossert, Ashlyn D. Burch, Matthew N. Chow, Craig W. Hogle, Megan Ivory, Jessica Pehr, Bradley Salzbrenner, Daniel Stick, William Sweatt, Joshua M. Wilson, Edward Winrow, and Peter Maunz. 2021. Engineering the Quantum Scientific Computing Open User Testbed. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–32. doi:10.1109/TQE.2021.3096480

[5] Eric Dennis, Alexei Y. Kitaev, Andrew Landahl, and John Preskill. 2002. Topological quantum memory. *J. Math. Phys.* 43, 9 (2002), 4452–4505. doi:10.1063/1.1499754

[6] Stavros Efthymiou, Sergi Ramos-Calderer, Carlos Bravo-Prieto, Adrián Pérez-Salinas, Diego García-Martín, Artur Garcia-Saez, José Ignacio Latorre, and Stefano Carrazza. 2021. Qibo: a framework for quantum simulation with hardware acceleration. *Quantum Science and Technology* 7, 1 (dec 2021), 015018. doi:10.1088/2058-9565/ac39f5

[7] I. M. Georgescu, S. Ashhab, and Franco Nori. 2014. Quantum simulation. *Rev. Mod. Phys.* 86 (Mar 2014), 153–185. Issue 1. doi:10.1103/RevModPhys.86.153

[8] Daniel Gottesman. 1997. *Stabilizer Codes and Quantum Error Correction.* Ph.D. thesis. California Institute of Technology, Pasadena, CA.

[9] Boxi Li, Shahnawaz Ahmed, Sidhant Saraogi, Neill Lambert, Franco Nori, Alexander Pitchford, and Nathan Shammah. 2022. Pulse-level noisy quantum circuits with QuTiP. *Quantum* 6 (Jan. 2022), 630. doi:10.22331/q-2022-01-24-630

[10] Junyi Liu, Yi Lee, Haowei Deng, Connor Clayton, Gengzhi Yang, and Xiaodi Wu. 2025. RISC-Q: A Generator for Real-Time Quantum Control System-on-Chips Compatible with RISC-V. arXiv:2505.14902 [cs.AR] https://arxiv.org/abs/2505.14902

[11] Guang Hao Low and Isaac L. Chuang. 2019. Hamiltonian Simulation by Qubitization. *Quantum* 3 (July 2019), 163. doi:10.22331/q-2019-07-12-163

[12] David C. McKay, Thomas Alexander, Luciano Bello, Michael J. Biercuk, Lev Bishop, Jiayin Chen, Jerry M. Chow, Antonio D. Córcoles, Daniel Egger, Stefan Filipp, Juan Gomez, Michael Hush, Ali Javadi-Abhari, Diego Moreda, Paul Nation, Brent Paulovicks, Erick Winston, Christopher J. Wood, James Wootton, and Jay M. Gambetta. 2018. Qiskit Backend Specifications for OpenQASM and OpenPulse Experiments. arXiv:1809.03452 [quant-ph] https://arxiv.org/abs/1809.03452

[13] Y. Nakamura, Yu. A. Pashkin, and J. S. Tsai. 2001. Rabi Oscillations in a Josephson-Junction Charge Two-Level System. *Phys. Rev. Lett.* 87 (Nov 2001), 246601. Issue 24. doi:10.1103/PhysRevLett.87.246601

[14] Thien Nguyen and Alexander McCaskey. 2022. Enabling Pulse-Level Programming, Compilation, and Execution in XACC. *IEEE Trans. Comput.* 71, 3 (March 2022), 547–558. doi:10.1109/TC.2021.3057166

[15] Timothy Proctor, Melissa Revelle, Erik Nielsen, et al. 2020. Detecting and tracking drift in quantum information processors. *Nature Communications* 11 (2020), 5396. doi:10.1038/s41467-020-19074-4

[16] quil-lang. 2021. *Quil.* https://github.com/quil-lang/quil

[17] Peter W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science.* 124–134. doi:10.1109/SFCS.1994.365700

[18] Peter W. Shor. 1995. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A* 52 (Oct 1995), R2493–R2496. Issue 4. doi:10.1103/PhysRevA.52.R2493

[19] Peter W. Shor. 1996. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science.* 56–65. doi:10.1109/SFCS.1996.548464

[20] Wilson Snyder. 2025. Verilator v5.032. https://www.veripool.org/wiki/verilator.

[21] Matthias Steffen, M. Ansmann, R. McDermott, N. Katz, Radoslaw C. Bialczak, Erik Lucero, Matthew Neeley, E. M. Weig, A. N. Cleland, and John M. Martinis. 2006. State Tomography of Capacitively Shunted Phase Qubits with High Fidelity. *Phys. Rev. Lett.* 97 (Aug 2006), 050502. Issue 5. doi:10.1103/PhysRevLett.97.050502