Distributed-HISQ: A Distributed Quantum Control Architecture

Yilun Zhao[2,3,*] Kangding Zhao[1,*] Peng Zhou[4] Dingdong Liu[1] Tingyu Luo[5] Yuzhen Zheng[1]
Peng Luo[1] Shun Hu[1] Jin Lin[6,7] Cheng Guo[6] Yinhe Han[2] Ying Wang[2] Mingtang Deng[1]
Junjie Wu[1] X. Fu[1,†]

[1]College of Computer Science and Technology, National University of Defense Technology, Changsha, China
[2]Research Center for Intelligent Computing Systems, State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[3]University of Chinese Academy of Sciences, Beijing, China
[4]China Greatwall Technology Group Co., Ltd., Shenzhen, China
[5]East China Normal University, Shanghai, China
[6]Hefei National Laboratory, Hefei, China
[7]University of Science and Technology of China, Hefei, China

Abstract

The design of a scalable Quantum Control Architecture (QCA) faces two primary challenges. First, the continuous growth in qubit counts has rendered distributed QCA inevitable, yet the non-deterministic latencies inherent in feedback loops demand cycle-accurate synchronization across multiple controllers. Existing synchronization strategies — whether lock-step or demand-driven — introduce significant performance penalties. Second, existing quantum instruction set architectures are polarized, being either too abstract or too granular. This lack of a unifying design necessitates recurrent hardware customization for each new control requirement, which limits the system's reconfigurability and impedes the path toward a scalable and unified digital microarchitecture.

Addressing these challenges, we propose Distributed-HISQ, featuring: (i) HISQ, A universal instruction set that redefines quantum control with a hardware-agnostic design. By decoupling from quantum operation semantics, HISQ provides a unified language for control sequences, enabling a single microarchitecture to support various control methods and enhancing system reconfigurability. (ii) BISP, a booking-based synchronization protocol that can potentially achieve zero-cycle synchronization overhead. The feasibility and adaptability of Distributed-HISQ are validated through its implementation on a commercial quantum control system targeting superconducting qubits. We performed a comprehensive evaluation using a customized quantum software stack. Our results show that BISP effectively synchronizes multiple control boards, leading to a 22.8% reduction in average program execution time and a ~ 5× reduction in infidelity when compared to an existing lock-step synchronization scheme.

Keywords

Quantum Control Architecture, Quantum Instruction Set Architecture, Distributed Quantum Control

ACM Reference Format:
Yilun Zhao[2,3,*] Kangding Zhao[1,*] Peng Zhou[4] Dingdong Liu[1] Tingyu Luo[5] Yuzhen Zheng[1] Peng Luo[1] Shun Hu[1] Jin Lin[6,7] Cheng Guo[6] Yinhe Han[2] Ying Wang[2] Mingtang Deng[1] Junjie Wu[1] X. Fu[1,†] . 2025. Distributed-HISQ: A Distributed Quantum Control Architecture. In *58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25), October 18–22, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3725843.3756048

1 Introduction

Memory, computation, and control are the three fundamental components of a programmable computer. While qubits function as both memory and computational units for quantum information processing, an external dedicated control system is an indispensable component of a solid-state quantum computer. This system serves to bridge the quantum computational core with the classical world accessible to end users by transmitting, receiving, and processing classical electromagnetic signals.

Both realizing quantum advantage using noisy intermediate-scale quantum systems [27, 33] and achieving Fault-Tolerant Quantum Computing (FTQC) [1] necessitate a larger number of qubits, with thousands or even millions of qubits expected [16, 21]. This fact challenges the design of Quantum Control Architectures (QCA) with simultaneously supporting **programmability**[1], **feedback**[2], and **scalability**[3].

QCAs can be roughly classified into centralized architecture or distributed architecture. *Centralized quantum control architectures* [11, 12, 45, 46, 54, 56] feature a single binary executable with possible waveform configurations as input, whose digital part is usually implemented in a single FPGA chip. Two factors significantly constrain the scalability of these centralized architectures.

*Equal contribution.
†email: xiangfu@quanta.org.cn

MICRO '25, Seoul, Republic of Korea

ACM ISBN 979-8-4007-1573-0/2025/10
https://doi.org/10.1145/3725843.3756048

[1]Programmability refers to the capability of flexibly defining the sequence of quantum operations applied on target qubits.
[2]Feedback refers to using the measurement result of one or more qubits to determine the following operations applied on the same or other qubits.
[3]Scalability refers to the property that the quantum control system can satisfy the control requirements of a larger quantum system by duplicating basic modules with almost linear resource cost or lower.

Internally, the instruction issue rate associated with a single instruction cache or memory cannot afford the broader quantum operation stream required by the increasing number of qubits. Externally, the limited number of pins available on a single FPGA chip sets an upper bound on the amount of analog channels controlling qubits. To support the ever growing number of qubits, QCA inevitably adopts a distributed form.

Distributed Quantum Control Architectures (DQCA) can offer improved scalability by partitioning a quantum program into multiple binaries executed across several control units. Hence, it draws a lot of research attention and inspires various off-the-shelf products. For example, QubiC 1.0 [46] utilizes a single FPGA board with 4 analog inputs and 4 outputs, which can only control no more than 4 superconducting qubits. To increase control capacity, QubiC 2.0 [10, 47] adopts multiple RFSoC FPGAs, resulting in a DQCA. The same rationale can be observed in other academic studies [17, 20, 32, 49, 52], as well as off-the-shelf products that adopt board-level and/or chassis-level distribution [22, 34, 38].

By duplicating boards or chassis, these architectures offer improved scalability with higher instruction issue rates and more channels. Nevertheless, these architectures either fail to simultaneously support scalability, programmability, and feedback [17, 20, 32, 49], or they largely overlook two critical requirements, namely (1) the challenge of reconciling cycle-level synchronization across controllers with the concurrent execution of multiple (potentially heterogeneous) binaries [10, 51, 52], and (2) the engineering effort required to adapt these architectures to diverse control requirements. Before future technologies based on cryogenic CMOS or single-flux quantum devices get mature [4, 24, 26, 36], these problems remain open challenges.

1.1 Synchronization Challenge

Synchronization is essential for implementing multi-qubit operations (typically two-qubit gates). Such operations require target qubits simultaneously go through particular state evolution, which in turn necessitates the concurrent application of control signals. In a DQCA, control signals for the same operation may originate from different controllers. Hence, it is crucial for these controllers to commit corresponding instructions at precisely the same time.

This synchronization requirement in quantum system is by nature different from that in classical computer systems. In classical systems, synchronization primarily serves to guarantee expected data dependency across threads or processes. The critical factor is often the relative order of operations, and waiting a few clock cycles rarely invalidates the fundamental logic. In stark contrast, a timing error of even a few nanoseconds can lead to the failure of a quantum gate [12]. This corruption can further give rise to meaningless results. Therefore, synchronization in DQCA demands instructions to be committed at almost the same wall-clock time, and can be as tight as at tens of picoseconds level. This is not merely a preference for performance or a safeguard for logical order; it is a fundamental physical necessity rooted in the quantum mechanical evolution of the qubits. Fortunately, this unprecedented synchronization requirement can be reduced to **cycle-level instruction commitment synchronization**[4] in engineering practice, because it is feasible to synchronize the phase of different clocks at high precision via a clock distribution network, based on, e.g., phase-locked loops.

While lock-step synchronization—where controllers are synchronized at every clock cycle—is a straightforward approach, it introduces significant inefficiencies. In this scheme, if a controller executes instructions without knowledge of the other controllers' state, non-deterministic feedback operations can make it difficult to commit collaborative multi-qubit instructions that require precise timing. Enforcing lock-step synchronization incurs substantial execution overhead. FFor instance, the approach used by the IBM system [51] distributes the entire program flow to all controllers, with operations for other controllers replaced by *wait/idle/delay* instructions. This design forces every feedback operation to incur data transmission across all controllers, leading to unnecessary communication and latency that scales with the number of feedback operations. Furthermore, as analyzed in QuAPE [54], this solution makes it difficult to execute simultaneous feedback operations on different qubits, which can significantly harm execution fidelity.

Another solution is to allow each controller to execute instructions at its own pace and synchronize them only when required. This solution significantly reduces the amount of generated instructions, thereby increasing instruction execution efficiency. Moreover, it may increase the flexibility of programming by allowing each controller to have its own control flow [10]. To realize such an on-demand synchronization scheme, an existing solution inserts a *sync* instruction *immediately* before the instruction to be synchronized, as adopted by Qubic 2.0 [10]. In this scheme, an unavoidable latency is introduced for the sync signal bouncing back. In the context of quantum computing's pursuit of ultimate hardware performance, it is also highly desirable to eradicate this latency.

To summarize, it remains an open challenge to design a QCA that has an efficient and scalable synchronization scheme.

1.2 Adaptability Requirement

In contemporary quantum systems, the physical realization of qubits, control methodologies, and specific implementations of pulse generation and acquisition exhibit significant diversity and are undergoing rapid evolution. For example, superconducting qubits differ structurally: some utilize fixed-frequency transmons [42], while others incorporate tunable couplers to adjust gate speeds and enhance performance [1].

For reasons of time and money, it is of practical meaning for the digital part of the quantum control microarchitecture to support various gate implementation strategies to control various quantum chips with various analog implementation. However, existing Quantum Instruction Set Architecture (QISA) designs are either too high-level or too low-level. Some QISAs directly adopt quantum operations as instructions, which makes their microarchitectures difficult to support other quantum chip structures or operation implementation strategies [11, 54]. Instructions of other QISAs are tightly bound to specific control electronics, so that when adapting these architectures to another control electronic system, it

[4] In this paper, we also use an equivalent term "cycle-level instruction synchronization", "commitment" is omitted for simplicity.

inevitably leads to a clumsy and redundant microarchitecture implementation [12, 20, 38, 45–47]. Therefore, it remains an open challenge to design an adaptable QISA.

1.3 Contributions

In this paper, we introduce DISTRIBUTED-HISQ, an architectural solution to systematically address the design requirements of DQCA with the following key contributions:

(1) HISQ, a **H**ardware **I**nstruction **S**et for **Q**uantum computing. We identify a new abstraction layer for adaptable QISA design, which is not only implementable on existing hardware but also expressive enough for potential applications.
(2) BISP, a **B**ooking-based precise **I**nstruction **S**ynchronization **P**rotocol. By introducing a two-condition synchronization method and advancing the sync instruction when possible, BISP can be implemented with low hardware cost. This approach enables both neighbor-level and region-level synchronization with potentially zero-cycle overhead. When zero cycle is not possible, the overhead is still no more than that of existing synchronization schemes.
(3) We implement DISTRIBUTED-HISQ in a customizable commercial quantum control system. The adaptability of HISQ is validated by using the same HISQ core to control both the arbitrary-waveform-generator board and data-acquisition board. BISP has also been validated by multiple synchronization experiments using two boards with different instruction streams.
(4) We have designed and implemented a full quantum control software stack, which can describe and compile quantum algorithms and experiments into HISQ instructions. The feasibility of DISTRIBUTED-HISQ has been verified by various experiments on superconducting qubits with this control software stack.

This paper is organized as follows. After Section 2 details the challenges of DQCA with corresponding insights, Section 3 - 5 introduces the single-node architecture, the synchronization protocol, and the distributed network of DISTRIBUTED-HISQ, respectively. Verification and evaluation are performed in Section 6. After some discussion in Section 7, Section 8 concludes.

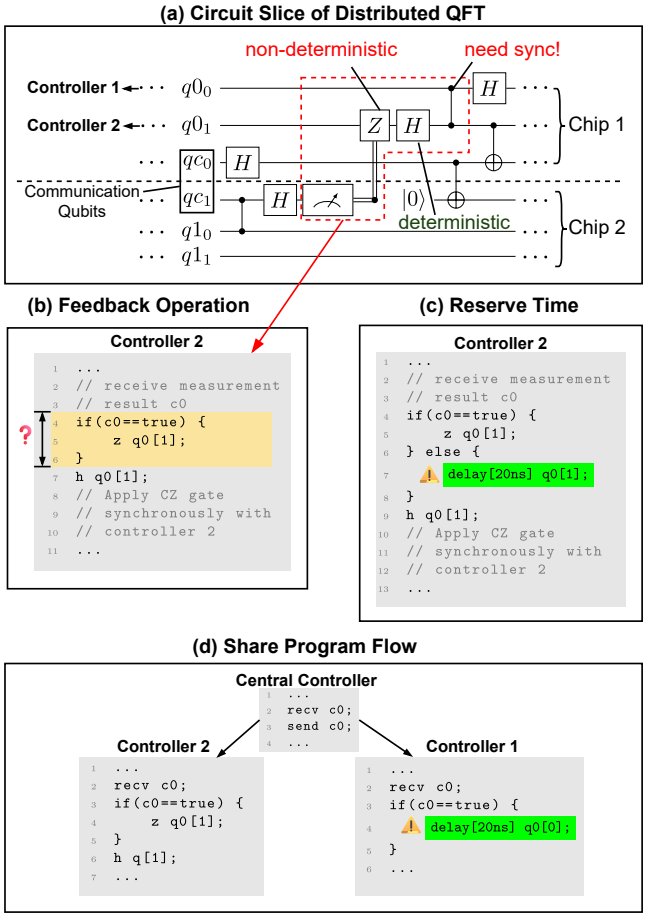


Figure 1: Motivational example of the synchronization challenge. (a) Example circuit slice derived from compiling QFT algorithm running on two quantum chips [6, 23]. (b) OpenQASM [5] code snippets of the highlighted part in the circuit diagram. (c) Inserting a *delay* instruction into the false branch. (d) Distribute the control flow to other controllers.

2 Challenges and Insights

In this section, we concretize challenges in synchronization among controllers and adaptability of a hardware instruction set, and elaborate on how the challenges drive the design of DISTRIBUTED-HISQ.

2.1 Synchronization

2.1.1 Origination of Synchronization Challenge. The synchronization challenge of DQCA originates from the non-deterministic nature of *dynamic quantum circuits*, which serve as critical components in many application scenarios. For example, distributed quantum computing [6, 35] leverages quantum teleportation to connect multiple quantum chips to increase system scale, long-range entanglements [3] can be created by dynamic circuits with lower circuit depth, and logical T-gate in surface code [9] needs to be implemented via logical state teleportation.

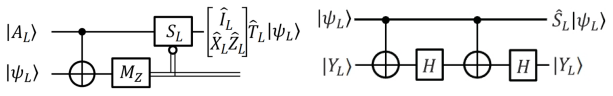
Consider a system with multiple quantum chips connected via inter-chip quantum communication channels, with each qubit controlled by a distinct controller. As illustrated in Figure 1(a), a QFT algorithm running on this system relies on real-time feedback to implement cross-chip two-qubit gates [6, 23, 44] [highlighted in dashed box, with corresponding pseudo-code in Figure 1(b)]. To achieve the minimal execution time, every gate should be executed as early as possible. Since the execution of the Z gate depends on a previous measurement result, the earliest time to execute the H gate and following CZ gate becomes unpredictable at compile time. However, the implementation of CZ requires synchronization between these two controllers, which is challenging due to the unpredictable timing behaviors.

To solve this problem, we can either keep all the controllers synchronized at all time, or allow each controller to operate independently and synchronize with others only when necessary.

2.1.2 Lock-Step Synchronization. Based on whether there exist information exchange about control flow across controllers, two different methods can be used to achieve lock-step synchronization. Unfortunately, both suffer from high execution overhead.

The first idea is to **"reserve" the time for the non-deterministic operation**, as seen in Figure 1(c). No matter the Z operation is performed or not, the 20 ns time slot should be consumed. The immediate drawback of this method is the introduction of "dead time" when the conditional operation is not performed. Such unnecessary delay accumulates linearly as the number of feedback operations or the duration of conditional operation increases. The longer execution time can degrade precious quantum fidelity. More importantly, this solution cannot support repeat-until-success circuits with non-deterministic number of feedback loops [37].

The second idea is to **"share" the program flow information**, a strategy implemented in the IBM quantum control system [51] [Figure 1(c)]. When a measurement is performed, the outcome is broadcasted to all controllers so every controller proceed in lock-step down the same branch. This solution is inherently limited by enforcing the same program flow on all controllers. Firstly, this introduces unnecessary control flow and *delay* instructions that may exaggerate the quantum operation issue rate problem limiting the scalability. Secondly, it undermines programming flexibility as other controllers can only stay idle when some are performing feedback. For example, it is difficult to realize simultaneous feedback [54]. When the conditioned sub-circuit is long, temporally stacked feedbacks can accumulate much longer execution time than the parallelized version. For example, some implementation scheme of the logical T gate [Figure 2(a)] relies on the conditional logical-S gate [Figure 2(b)], which in turn is a sub-circuit with multiple logical operations that can take a substantial execution time. [9]. If the feedback in logical T gates can only be executed sequentially, the execution time of the entire program will grow significantly. Ultimately, both issues can incur execution time overhead than expected that dampens program fidelity.



(a) Logical T gate. **(b) Logical S gate.**

Figure 2: (a) One implementation of logical T gate relies on the conditional logical S gate. (b) Logical S gate is a sub-circuit with multiple logical operations that take a long execution time.

2.1.3 Insights for Efficient Synchronization Protocol. Given the substantial overhead of lock-step synchronization, an alternative is to synchronize controllers in an as-needed manner. This approach usually inserts a *sync* instruction immediately before the instructions that requires cycle-level synchronization [10]. Nonetheless, it will still introduce unavoidable latency for sync signals bouncing back.

In order to achieve the utmost synchronization efficiency, we draw inspiration from a common scenario in daily life: an individual organizing a meeting seeks to start it at the earliest possible time. Each participant, upon determining their earliest available time, sends a message specifying this time to the organizer. After collecting responses from all participants, the organizer identifies the latest of these times—representing the earliest feasible start for the meeting—and notifies all participants accordingly. Provided the notification reaches participants before the designated start time, the meeting can proceed at the earliest time.

Since quantum operations usually have a fixed duration [28], a controller can potentially predict in advance the exact time at which it will reach a *synchronization point*[5], similar to a participant determining their earliest available time for a meeting. Additionally, a common router can serve as the meeting organizer. It is possible for us to design a concise synchronization protocol based on the following insight:

Insight #1

A controller can "book" a synchronization point thanks to the deterministic nature of quantum operations.

2.1.4 Insights for Common Synchronization Scenarios. In the meeting example, the condition for the meeting to start at the earliest time is that participants receive the meeting time notification before its scheduled start. For humans, the communication overhead is negligible. However, a quantum controller must spend several cycles to communicate with other controllers. Hence, to achieve the similar effect as meeting start, the communication latency must be suppressed as much as possible.

To meet this requirement, we identify two key characteristics of quantum applications and quantum devices which we can take advantage of:

(1) Synchronization within a qubit region is a common scenario. Quantum programs are often executed with multiple repetitions. Before each repetition, it is usually necessary to perform a global synchronization among the involved controllers. Considering that a quantum program is often mapped to a group of connected qubits (qubit region), the region-level synchronization becomes a common scenario.
(2) Synchronization between controllers for neighboring qubits is a common scenario, since two-qubit gates are common cases and they can only be executed between physically adjacent qubits. To minimize communication latency in these scenarios, the ideal situation is that these controllers are directly connected as neighbors.

Insight #2

Synchronization among controllers for a qubit region, and between controllers for neighboring qubits, are two common scenarios.

[5]The time point at which a controller finishes all operations before executing the operations that need be be synchronized.

These insights naturally give rise to a hybrid topology design with a simple routing mechanism to reduce communication latency, which will be detailed in Section 5.

2.2 Adaptability

We can get a better understanding on the adaptability challenge of a QISA by digging into how a quantum operation should be implemented, or how the control electronics should behave. Indeed, the hardware behavior of a quantum instruction can be affected by multiple factors.

- Different **qubit implementation technologies** require different control signals for the same quantum gate. For example, to implement a CZ gate, the corresponding control signals might be square pulses for superconducting qubits, while a set of modulated lasers for Rydberg atoms [39].
- For the same qubit implementation technology, like superconducting qubits, different **quantum chip structures** require different control signals. Take the CZ gate as an example. Capacitor-coupled transmon qubits may require two square flux pulses and other possible auxiliary flux pulses to tune the frequencies of the target and adjacent qubits, respectively [7, 41]. While tunable-coupler-coupled qubits can achieve the same effect using only three pulses [48].
- For the same quantum chip structure, different **operation implementation strategies** require the use of different control signals [55]. For the same coupler-coupled qubits as mentioned above, it is also possible to use all microwaves instead of square pulses to implement a CZ gate [31].
- Finally, even for the same control strategy of the same gate on the same qubits, different **implementations of the analog part** in the quantum control system can lead to different behaviors of the digital part. Taking the X gate as an example, some systems may require the microarchitecture to trigger a pair of intermediate-frequency outputs from Digital-to-Analog Converters (DAC), which are later IQ-modulated with a radio-frequency carrier wave. On the contrary, direct-microwave-synthesis-based systems require to set the frequency and phase of the numerically controlled oscillator (NCO) and then directly trigger radio-frequency output from the DAC with a given envelope [25] .

Since quantum chip fabrication, gate implementation strategies, and quantum control systems often advance at asynchronous pace, it is of ample economic and engineering significance for the same digital part of the quantum control microarchitecture to support various gate implementation strategies to control various quantum chips with various analog implementation.

However, it is difficult to adapt existing QISAs to various quantum chip structures and gate implementation strategies. Some QISAs [11, 54] directly adopt quantum operations as instructions, which are relatively too high-level for hardware implementation. As a result, the corresponding microarchitecture can support the execution of these instructions targeting no more than one quantum chip structure without breaking the instruction definition. For example, eQASM allows using two-qubit gates. However, its instantiated microarchitecture cannot support other quantum chips than the targeted seven-qubit chip. In contrast, some other QISAs tightly bind the instruction semantics to the output channel behaviors [20, 38, 45–47]. The microarchitecture of these systems can hardly be adapted to other kinds of output channels or hardware behavior without sacrificing existing or adding new instructions or bit-fields with corresponding microarchitecture implementation. This fact usually leads to a clumsy microarchitecture implementation. For example, many bit fields in QubiC [47] will be sacrificed if the definitions of instructions are altered to generate a marker signal.

Based on the above analysis, it is easy to comprehend that the key for designing an adaptable QISA is to figure out an abstraction layer, which is **not only implementable** on existing hardware, **but also expressive enough** for potential applications. As quantum controllers are responsible for sending required control signals to expected qubits with correct timing, we can extract the following insight for the instruction set design by decoupling quantum instructions from the operation semantics (top) as well as the concrete electronics behavior (bottom):

Insight #3

At the instruction level, quantum control can be abstracted into: sending particular codewords, to particular ports, at particular time-points.

By decoupling quantum semantics from hardware instructions, it significantly simplifies hardware implementation and allows this instruction to hook various underlying electronics implementations. On the other hand, the quantum semantics can be taken care of another higher-level software instruction set, which can focus on the expressiveness of quantum computation and portability across various hardwares.

3 Design Overview

Being a DQCA, Distributed-HISQ comprises multiple control nodes coordinated by routers [Figure 3(d)]. A single node [Figure 3(c)] integrates digital and analog components. Its digital part, the HISQ core, abstracts quantum operations as "sending particular codewords to particular ports at particular time-points". This abstraction decouples the instruction set from quantum semantics, allowing adaptation to diverse analog implementations. To achieve efficient synchronization, the microarchitecture [Figure 3(a)] builds on top of queue-based event timing control mechanism. This mechanism can hide classical pipeline non-determinism and issue quantum instructions at precise timing. By incorporating the meeting appointment insight, we design a highly efficient booking-based synchronization protocol (BISP) among control nodes [Figure 3(e)], which can allow multiple controllers to re-sync their instruction execution at cycle-level precision after non-deterministic program subroutines. This protocol is implemented by the newly introduced synchronization unit with modified timing control unit [Figure 3(a)]. To enable efficient synchronization and feedback between remote controllers, Distributed-HISQ employs a hybrid topology [Figure 3(d)], which will be detailed in Section 5. Figure 3(b) shows two types of boards implemented based on HISQ, which form the

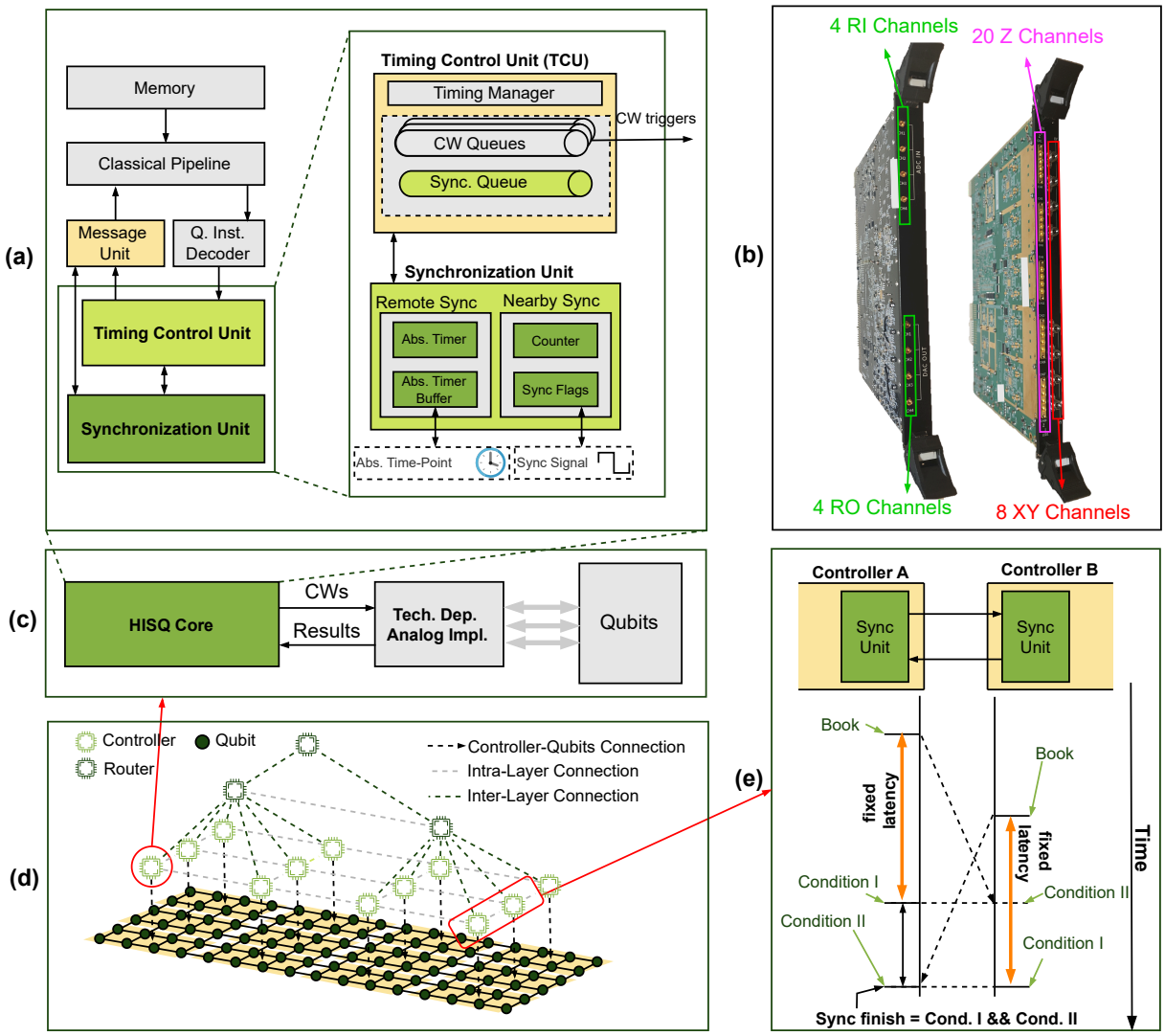


Figure 3: Overview of Distributed-HISQ. (a) Microarchitecture; (b) AWG board and readout board based on HISQ; (c) Single-node architecture; (d) Multi-node architecture;(e) Synchronization scheme.

individual nodes of Distributed-HISQ. The detailed hardware implementation and configuration are described in Section 6.

3.1 Instruction Set Architecture

The core of designing a hardware-implementable instruction set is to provide the capability to describe operations of quantum controllers, including:

(1) Real-time classical register update and program flow control;
(2) Triggering particular actions at specific locations, including quantum operations and other behaviors like quantum error decoding;
(3) Timing control of controllers, including synchronization;
(4) Classical communication across controllers to support, e.g., feedback control.

Hence, HISQ is designed to support the above four parts.

3.1.1 Real-Time Classical Computation. To reduce the burden in hardware and software implementation, HISQ is designed to be an extension to the RISC-V 32I instruction set. To avoid disrupting timing behavior, we currently disable instructions/functionalities related to interrupts and memory fence.

3.1.2 Triggering Operations. As detailed in Section 2.2, triggering operations can be abstracted as "sending particular codewords, to particular ports, at particular time-points". To support flexibly specifying time-points, HISQ employs the timing control mechanism as proposed by QuMA [11, 12]. Hence, both the immediate and register version of `wait` instruction are included.

A set of codeword (`cw`) instructions are used to describe "sending particular codewords, to particular ports". The syntax of these instructions is:

```
cw.x.x <port>, <codeword>
```

Where, `x` could be either `i` or `r`, indicating `<port>` and `<codeword>` specified by an immediate or a general-purpose register (GPR). For example, `cw.i.r 3, r3` means sending the codeword specified by the GPR `r3` to the port No. 3. The meaning of a codeword depends on the compiler and hardware configurations, thus it can vary in concrete implementations. For example, a codeword can correspond to triggering a Gaussian pulse, setting the frequency of Numerically Controlled Oscillator (NCO), or any hardware action(s) that can be abstracted into a digital number. A port may direct to the channels for I/Q, flux, readin/readout, or any other components that might need be controlled. This port-based abstraction makes HISQ independent of the concrete implementations of operation/qubits while preserving sufficient expressiveness for controlling the hardware.

3.1.3 Synchronization. To support as-needed synchronization, HISQ includes a synchronization instruction with following syntax:

```
sync <tgt>
```

The `sync` instruction can only function in combination with other `sync` instructions, with each `sync` instruction running on one controller. The effect of a group of `sync` instructions is to synchronize the clocks of controllers involved.

The field `<tgt>` is an immediate value, which can designates the address of (i) a controller or (ii) a router. In the former case, `<tgt>` must refer to a nearest-neighbor controller, and there must be another `sync` instruction running on the neighbor controller with its `<tgt>` field referring to this controller. The result of both instructions is to synchronize this controller and this neighbor controller. In the latter case, `<tgt>` must refer to a ancestor router of this controller, and the effect is to synchronize with a subset of controllers managed by the same ancestor router. For the sake of brevity, we will describe the synchronization mechanism in detail in Section 4.

3.1.4 Classical Communication. Classical information like measurement results should be transmitted across controllers to support real-time feedback and quantum error syndrome decoding. To this end, HISQ includes `send/recv` instructions, which are executed by a Message Unit (MsgU).

3.2 Single-Node Microarchitecture

The microarchitecture supporting HISQ is designed based on the QuMA microarchitecture [11, 12]. It differs from QuMA by slight modification [Figure 3(a)] in the timing control unit (TCU) and the introduction of synchronization unit (SyncU) and the message unit (MsgU).

SyncU is used to support the `sync` instruction and detailed in Section 4. Since sending and receiving messages across chips is well studied in previous network-related research, how MsgU supports `send/recv` is omitted in this paper for brevity. Here, we mainly discuss the the basic working principle of TCU with required modification to support synchronization.

To achieve precise timing control, TCU employs the queue-based timing control mechanism [12]. It contains a set of event queues with each corresponding to a port, where instructions can be enqueued at non-precise timing while issued at designated precise timing. The overall effect of TCU is to issue corresponding events at expected precise time-points. We refer interested readers to Ref. [11, 12] for more details about queue-based timing control.

In the original queue-based timing control, there is no mechanism for TCU to wait for an external signal, and it can only support precise timing control in which all waiting durations are encoded into a fixed instruction sequence and calculated at runtime. While synchronization is performed, one controller needs to wait for a non-deterministic duration depending on another controller, which cannot be supported by the original TCU. Addressing this problem, TCU for HISQ is equipped with multiple ports receiving external triggers, that can be used to pause and resume the timer in TCU. In this way, TCU can allow events with non-deterministic timing while preserving precise timing control between non-deterministic-timing events.

4 Synchronization Scheme

In a DQCA, feedback operations create dynamic timelines, making controller synchronization a key challenge (cf. Section 2.1). To achieve efficient synchronization, we introduce the BISP protocol. We first demonstrate the simplicity of BISP through its single-node hardware behavior. Then, we use examples to show the potential of BISP to achieve zero-cycle sync overhead in both nearby and remote synchronization scenarios.

4.1 Node Actions to Achieve Synchronization

The hardware implementation of the BISP protocol is very simple. Taking the nearby synchronization between two adjacent controllers as an example, its single-node hardware behavior is as follows (Figure 4):

Synchronization begins when the TCU sends a sync event to the SyncU at the "booking" time B_0. Upon receiving this event, the SyncU transmits a 1-bit signal to the target controller and starts an N-cycle countdown. Here, N is set to match the transmission delay between the hardware ports of the specific pair of neighboring controllers. Since this delay is fixed and can be calibrated once the

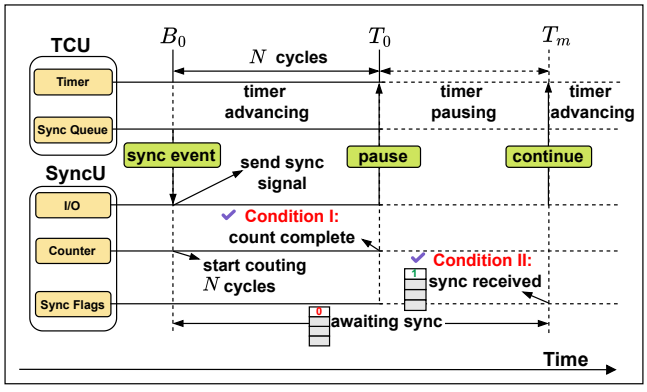


Figure 4: Single-node hardware behavior of BISP. The sync signal is exchanged with a neighbor controller. Upon receiving the signal, the corresponding sync flag (represented by stacked boxes for each neighbor) is set and cleared after being read.

hardware connections are established, N can be pre-configured in hardware for each connection.

Synchronization is achieved when both conditions are met:

- **Condition I:** The N-cycle count completes.
- **Condition II:** The sync signal from the target controller is received.

Note that when the N-cycle count completes, the SyncU may stall the TCU if the sync signal has not yet been received. In this case, the TCU pauses executing quantum operations until the sync signal arrives; otherwise, if the sync signal is received before the count completes, the TCU proceeds without interruption.

In our FPGA implementation, SyncU consumes only 13 LUTs. **Such a lightweight scheme not only ensures cycle-level instruction synchronization, but also can realize zero-cycle overhead in some scenarios.**

4.2 How Synchronization is Achieved

The effectiveness of BISP in nearby synchronization scenario can be observed from Figure 5(a).

Firstly, we can observe that controllers C_0 and C_1 execute the synchronous task at the same time-point.

- At the booking time-point B_0, C_0 sends a sync signal to C_1 and simultaneously starts counting for the signal transmission latency L_0. At $T_0 = B_0 + L_0$, C_0 has met **Condition I**. However, C_0 does not receive the sync signal from C_1 until T_1, and thus only meets **Condition II** at T_1. Therefore, it executes the synchronous task at T_1.
- C_1 sends its sync signal at B_1, which arrives at T_1, thereby satisfying **Condition I** at T_1. Meanwhile, the sync signal arrives at C_1 at T_0, satisfying **Condition II** at T_0. Since $T_0 < T_1$, C_1 satisfies both conditions and begins executing the synchronous task at T_1.

Thus, both C_0 and C_1 execute the synchronous task at T_1. In this diagram, C_0 sends the sync signal before C_1. If we swap C_0 and C_1 so that C_1 sends the signal first, both controllers still begin executing the synchronous task at the same time. This demonstrates that BISP ensures adjacent controllers start the synchronous task at the same time-point, thereby achieving cycle-level instruction synchronization.

Secondly, the *synchronization overhead* [6] is potentially "zero-cycle". For both C_0 and C_1, assume there are several *deterministic tasks* (light-yellow blocks) between their last *non-deterministic task* [7] (dark green) and the synchronous task (light green). The time-points that C_0 and C_1 finish these deterministic tasks are T_0 and T_1, respectively. Hence, the earliest time that C_0 and C_1 can co-execute the synchronous task is given by $\max(T_0, T_1) = T_1$. Consider both C_0 and C_1 execute the synchronous task at exactly T_1, we can consider that BISP achieves zero-cycle overhead in this case.

The core idea of BISP lies in the "booking" mechanism. As long as there are deterministic tasks with sufficient duration to cover communication latency, we can book a synchronization point in advance. **This allows us to insert a sync instruction ahead of the synchronization point (Figure 6), rather than placing it immediately before the synchronization point as done in Qubic [10].** Consequently, BISP minimizes the synchronization overhead (cf. Insight 1). This approach can also be easily extended to more complex remote synchronization scenarios.

[6] In this paper, synchronization overhead refers to the time interval between the last entity reaching the synchronization point and the first entity beginning the synchronous task.

[7] A non-deterministic task refers to a task with unpredictable duration, e.g., a feedback operation. Similarly, a deterministic task has a fixed duration, e.g., a quantum gate.

4.3 Extending to Remote Synchronization

Figure 5 exemplifies a remote synchronization scenario, where controllers C_0, C_1, and C_2 synchronize with each other through a router R.

These controllers, whether adjacent or not, can still achieve synchronization with zero-cycle overhead. As in the nearby synchronization case, we assume each controller has deterministic tasks preceding the synchronous task. Each controller sends its earliest possible start time (T_0, T_1, or T_2) to R before completing its deterministic tasks. Once R receives all synchronization requests, it determines and broadcasts the earliest common start time, $T_m = \max(T_0, T_1, T_2)$, to all controllers. Consequently, all controllers begin executing the synchronous task at T_m, achieving cycle-level synchronization at the earliest possible time with zero-cycle overhead.

After sending $T_0/T_1/T_2$, each controller C_i achieves synchronization when both of the following conditions are met:

- **Condition I**: The current absolute time reaches T_i.
- **Condition II**: The earliest sync point T_m has been received.

At this point, the SyncU checks whether the current time [indicated by the Abs. Timer shown in Figure 3(a)] has reached T_m [stored in the Abs. Timer Buffer shown in Figure 3(a)], pausing the TCU's timer if necessary and resuming it precisely at T_m.

4.4 Condition for Zero-Cycle Overhead

In real-world scenarios, there may not always be a sufficient number of deterministic tasks available prior to a synchronization task to mask the communication latency. In such scenarios, BISP may still introduce synchronization overhead.

In the example shown in Figure 7, all controllers can accomplish their tasks before synchronization at T_2, which forms the theoretical earliest synchronization time. However, the duration of deterministic tasks of C_2 is $D_2 = T_2 - B_2 < L_2$, which is insufficient to hide the sync communication latency L_2 between C_2 and R. Thus all controllers involved cannot synchronize until T_2', resulting in a synchronization overhead of $L_2 - D_2$.

In this case, the actual earliest possible start time for C_0, C_1, C_2 is $\max(T_0', T_1', T_2') = T_2'$, which exceeds the theoretical earliest start time T_2. As a result, achieving zero-cycle overhead is impossible. More generally, we can conclude that zero-cycle overhead can be realized if and only if the actual earliest start time is the same with the theoretical earliest start time, formally expressed as $\max(\{B_i + L_i\}) = \max(\{T_i\})$.

5 Distributed Architecture

As a distributed QCA, Distributed-HISQ requires a network topology tailored for quantum applications to minimize communication overhead that could impair architectural scalability. In this section, we first describe the topology design of Distributed-HISQ,

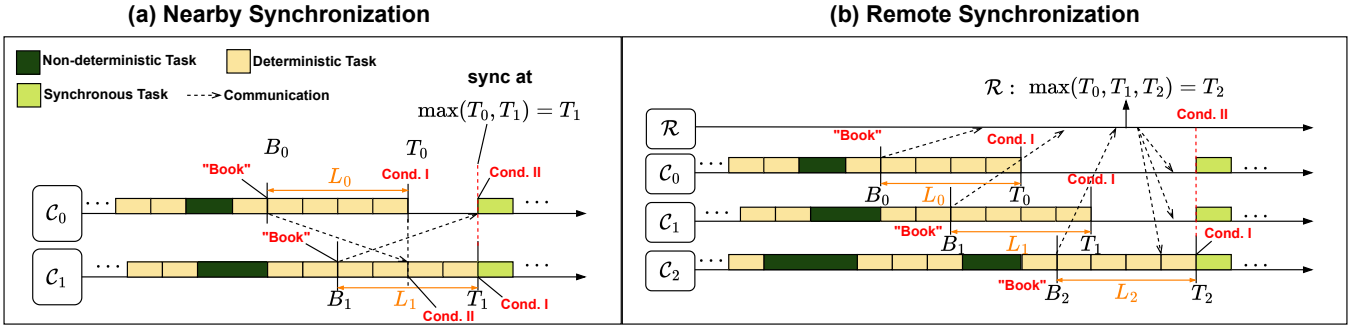


Figure 5: Example timing diagrams of nearby (a) and remote (b) synchronization using BISP.

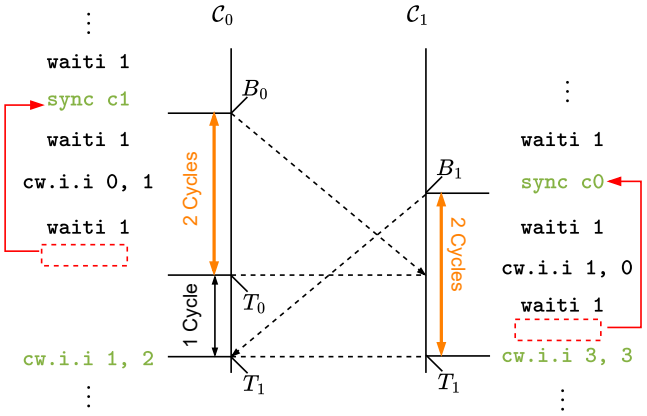


Figure 6: Example instructions for nearby synchronization.

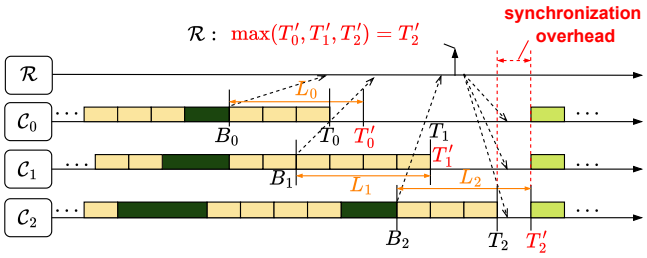


Figure 7: Example timing diagram of remote synchronization with non-zero overhead.

followed by an illustration of the router design, emphasizing its routing mechanism.

5.1 Design of Topology

We design a hybrid topology in DISTRIBUTED-HISQ, which consists of a tree-like inter-layer topology and a mesh-like intra-layer topology as shown in Figure 3(d). The controllers at the bottom layer are coordinated by higher-level routers.

The tree-like inter-layer topology is adopted to minimize network edges while also reducing communication hops at region-level. For a connected graph with N nodes, the minimum number of edges is $N-1$, forming a tree since each controller has finite connections.

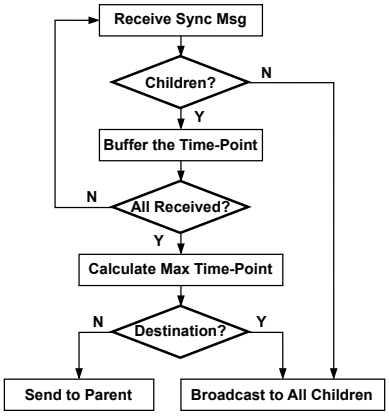


Figure 8: Router actions of region-level synchronization.

The network's maximum communication latency depends on the topology graph's diameter, which in a tree is $2 \times h$, where h is the tree height. Hence, a balanced tree with minimal height is adopted to reduce latency.

Additionally, Insight 3 indicates that the intra-layer topology should mirror the qubit device topology, naturally resulting in a mesh-like structure.

5.2 Routing Mechanism

Remote synchronization at region-level and nearby synchronization, are two common scenarios (Section 2.1.4). While nearby synchronization involves only direct communication between neighboring controllers, region-level remote synchronization requires routers with efficient routing mechanisms to reduce communication latency.

The router incorporates a simple routing mechanism that leverages the nature of tree topology (Figure 8).

(1) Upon receiving a message, it buffers the message if it is from a child; otherwise, it broadcasts the message to all children.
(2) After receiving messages from all children, it computes the maximum time-point (Section 4.3).
(3) If the message is destined for itself, it broadcasts the maximum time-point to all children; otherwise, it sends the time-point to its parent.

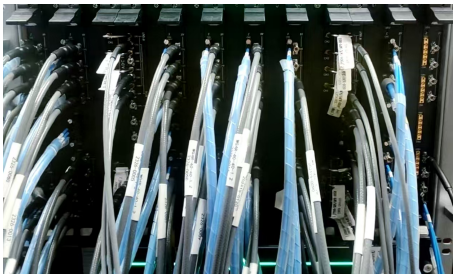


Figure 9: Hardware implementation of Distributed-HISQ.

Table 1: Overview of FPGA resource consumption of HISQ on the control and readout board.

Type	#LUTs	#Block RAM (32Kb per block)	#FF
Control Board	4,155	75	6,392
Readout Board	2,435	45	3,192
Event Queue (38bit x 1024)	86	1.5	160

6 Evaluation

6.1 Hardware Implementation

We implement Distributed-HISQ in a commercial distributed quantum control system (named DQCtrl, Figure 9). The control boards and readout boards, as shown Figure 3(b), form the leaf nodes of Distributed-HISQ. The control board has eight XY channels for x/y rotations and 20 Z channels for flux control. The readout board comprises four pairs of input and output channels. Each board is controlled by a single HISQ core. All control boards and readout boards are connected using a back-plane, through which, the readout board can connect to each control board, each with two Low-Voltage-Differential-Signal (LVDS) channels and a dedicated channel for global trigger distribution. Due to the limited connectivity, we are not able to realize region-level synchronization.

The same microarchitecture as shown in Figure 3(a) is deployed on both the control and readout board, with the only difference between them being the number of codeword queues, which matches the amount of channels on each board. The same HISQ instruction set is used to control all the XY channels and Z channels on the AWG boards, as well as measurement excitation and data acquisition on the readout boards. Nevertheless, the same codeword can produce entirely different behaviors on different boards. For example, the instruction `cw.i.i 1, 1` on the AWG applies an X gate, while on the readout board it triggers a measurement result discrimination. Such kind of implementation is a simple verification of the adaptability of HISQ.

The HISQ implementation is characterized by its high efficiency in FPGA resource utilization, as detailed in Table 1. A full 28-channel control board requires only 4155 LUTs, 6392 FFs, and 2.46 Mb of Block RAM, while an 8-channel readout board uses 2435 LUTs, 3192 FFs, and 1.47 Mb of Block RAM. For precise timing control, the TCU operates at 250 MHz, enabling a 4 ns resolution grid that improves upon the classic 200 MHz pipeline based on the PicoRV32I core [43].

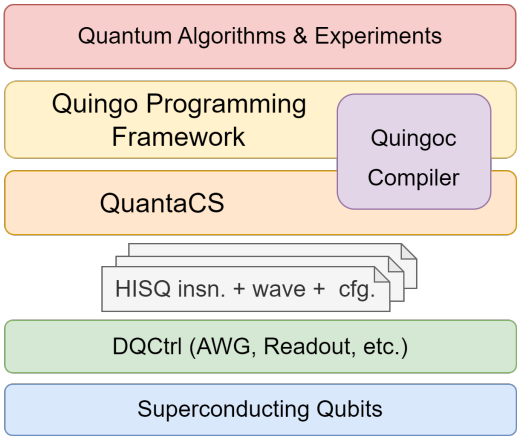


Figure 10: Quantum software stack for Distributed-HISQ.

In the context of modern FPGAs, which offer vast logic and memory resources, the hardware cost of HISQ is minimal. Its compact and efficient design ensures high scalability and straightforward portability to other systems without concerns about logic complexity becoming a bottleneck.

6.2 Qubit-Level Verification

To validate Distributed-HISQ on physical qubits, we developed a full quantum software stack (Fig. 10) comprising the Quingo programming framework [13], the MLIR-based Quingoc compiler, QuantaCS control software, and the SISQ instruction set. The Quingo framework integrates Python for data analysis with its native quantum language, which supports both high-level algorithms and low-level calibrations. Quingoc compiles Quingo programs into circuit-layer SISQ, which is then lowered to a hardware-agnostic, pulse-level representation. Finally, a new compiler backend for DQCtrl partitions the pulse-level program, generating HISQ binaries, waveform tables, and configuration files for the control hardware.

The target device is a superconducting quantum chip with 66 qubits and 110 couplers. Qubit operating frequencies range from 3.953 GHz to 4.757 GHz, with readout frequencies ranging from 6.220 GHz to 6.560 GHz. 66 XY channels, 176 Z channels, and 11 readout channels with each capacitively coupling 6 qubits, are used to control and measure these qubits.

We conducted a series of calibration experiments on multiple superconducting qubits to demonstrate the capabilities of Distributed-HISQ. Figure 11 presents four selected experiments, each designed to characterize a fundamental property of the control signals. Figure 11(a) shows a self-verification experiment for the readout board. A measurement excitation pulse with a linearly increasing **phase** is emitted, and the response is collected, IQ-demodulated, and integrated. This process yielded a characteristic circular pattern in the IQ plane. The observed deviation from an ideal circle arises from small but non-negligible interference from adjacent qubits coupled to the same feedline. Figure 11(b) depicts a spectroscopy experiment to determine the qubit's operating

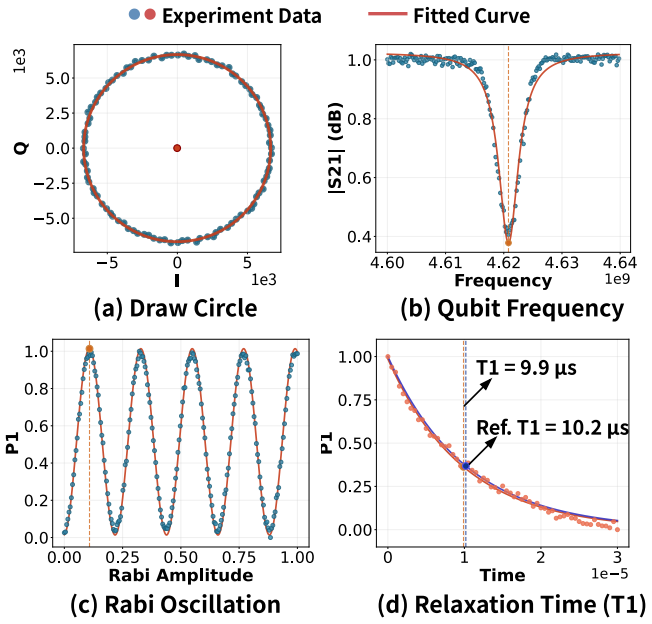


Figure 11: Four calibration experiments performed on a superconducting qubit, which show the capability of Distributed-HISQ controlling signal (a) phase, (b) frequency, (c) amplitude, (d) timing and (a,c,d) pulse envelop.

frequency. An x-rotation pulse was applied at varying **frequencies**, followed by a measurement of the qubit state, identifying the qubit resonance at 4.62 GHz. Figure 11(c) displays a Rabi oscillation experiment, performed to find the optimal pulse **amplitude** for implementing a high-fidelity X gate. Figure 11(d) presents a measurement of the qubit relaxation time (T_1), which characterizes the decay of the qubit's excited state population over **time**. This measurement yielded a relaxation time of 9.9 µs. For comparison, we also measured the working frequency and relaxation time of the same qubit using identical hardware but with an alternative and more mature firmware and software stack, obtaining values of 4.64 GHz and 10.2 µs, respectively. The minor discrepancies are well within the expected range attributable to the natural temporal fluctuations of the qubit's state. Collectively, the fact that all experiments generated data conforming to their expected theoretical patterns demonstrates that Distributed-HISQ can produce high-fidelity qubit control signals with precise, real-time manipulation of phase, frequency, amplitude, and timing.

6.3 Electronics-Level Verification

We designed two programs (Figure 12) that respectively run on a control board and a readout board to verify the feasibility of BISP. Both boards repeatedly execute sync instructions. Except for the sync instruction, the readout board contains only deterministic tasks, while the control board contains a non-deterministic task – the `waitr $1` instruction. The varied timing of control board makes its progress unpredictable to the readout board. As such, we emulated the as-needed synchronization scenario in DQCA.

```
# Control board
addi $2,$0,120
addi $1,$0,0
waiti 1
cw.i.i 21,2
addi $1,$1,40
cw.i.i 20,2
waitr $1 non-deterministic
sync 2
waiti 8
cw.i.i 7,1
waiti 50
bne $1,$2,-28
jal $0,-44
```

outer loop

inner loop

```
# Readout board
waiti 2
sync 1
waiti 6
waiti 57
cw.i.i 5,1
jal $0,-20
```

Figure 12: HISQ instructions running on the control and readout boards. Note that since the two boards have different triggering delays, we added a 57-cycle delay (the `waiti 57`) before the synchronous operation of readout board to mitigate this difference.

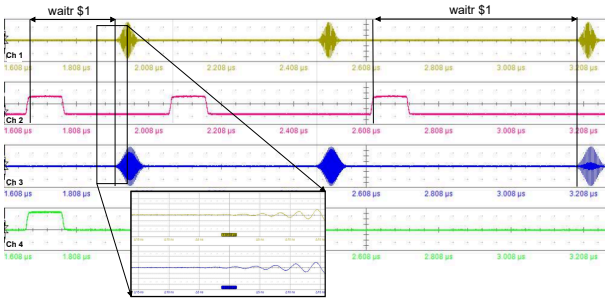


Figure 13: Waveforms illustrating synchronization between nearby control board and readout board.

The result of the instruction execution is shown in the waveform diagram (Figure 13). Channels 1 and 2 of the oscilloscope reveal that the start time of the control board's sync increases by 120 ns in each inner loop iteration, reflecting the increment of the register value $1. The instructions requiring synchronization are highlighted in yellow and blue (Figure 12) on both the control and readout boards, corresponding to the yellow and blue pulses in Figure 13. As shown, regardless of changes in $1, the yellow and blue instructions are always executed synchronously at the cycle level.

6.4 Simulation-Level Verification

6.4.1 Simulation Platform. To enable efficient evaluation, we also developed a simulator, CACTUS-Light, based on the open-source QCA simulator CACTUS [14, 53], but with the microarchitecture under investigation modeled at transaction level. CACTUS-Light adds support for the synchronization module as proposed in Section 4. It has been verified at two levels. The logical correctness is verified using multiple small-scale benchmarks whose execution produces expected quantum state or measurement results. The timing information in the simulation result is verified against the FPGA implementation using Timing Event Logging Format (TELF) [53]

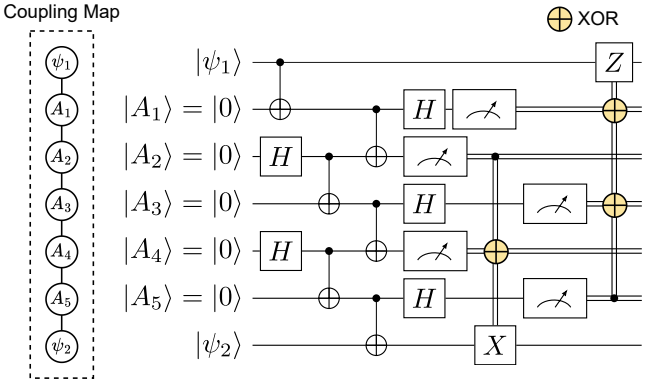


Figure 14: Long-range CNOT gate between $|\psi_1\rangle$ and $|\psi_2\rangle$ based on gate-teleportation [3]. Although SWAP gates can enable a CNOT between $|\psi_1\rangle$ and $|\psi_2\rangle$, the circuit depth increases linearly with qubit count. In contrast, this scheme maintains constant circuit depth as the number of qubits grows.

data. In our evaluation, we set 20 ns (40 ns) for single (two)-qubit gates, and 300 ns for measurements.

6.4.2 Benchmarks. We constructed two types of benchmarks.

(1) **Near-term dynamic circuits.** Qubit connectivity constraints in quantum devices have spurred extensive research into qubit mapping and routing challenges [30, 40]. By supporting arbitrary feedback operations, these constraints can, in principle, be overcome using non-unitary dynamic circuits [3]. Figure 14 illustrates a circuit diagram for implementing a long-range CNOT between two distant qubits via dynamic circuits. This approach trades spatial resources for temporal efficiency, utilizing additional ancilla qubits to eliminate cumbersome SWAP gates and achieve reduced circuit depth. Based on this approach, we have converted several static circuits from QASMBench [29] to dynamic circuits by randomly substituting CNOTs between non-adjacent qubits with long-range CNOTs. Subsequently, we convert these OpenQASM [5] programs into HISQ programs to serve as benchmarks.

(2) **Logical T gate-based QEC circuits.** With the long-term goal of achieving FTQC, various QEC protocols have been developed, with the surface code recognized as a leading candidate [8, 9]. Logical T gates are both resource-intensive and frequent in this protocol [9, 50]. Implementing a logical T gate involves a logical feedback operation (Figure 2), making it an ideal case to evaluate Distributed-HISQ's performance in QEC experiments. We construct and validate logical T gates using lattice surgery [19] with Stim [15], then convert these circuits into HISQ programs. As our focus is synchronization efficiency, we do not implement error decoding, but model its latency by inserting `wait` instructions based on existing hardware decoder data [2], assuming each router has a dedicated decoder. Given the high overhead of magic state distillation, we assume pre-prepared magic states and only simulate the logical feedback portion of the T gate.

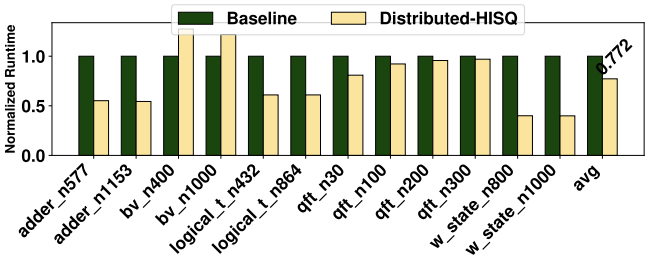


Figure 15: Runtime comparison with baseline. These benchmarks are dynamic circuits obtained by compiling static circuits with long-range CNOT gates [3].

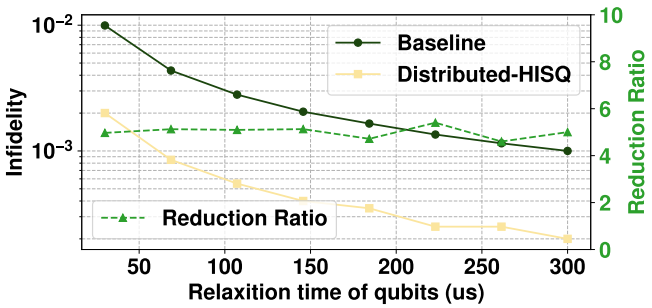


Figure 16: Fidelity comparison between Distributed-HISQ and baseline.

6.4.3 Baseline. As the experiment baseline, we implemented the lock-step synchronization scheme as proposed in [18, 51]. The control microarchitecture of each controller is identical to that of Distributed-HISQ. A central controller orchestrates the execution of all other controllers with a star topology. This approach limits the flexibility of feedback operation execution, particularly under conditions with numerous concurrent feedback operations. In contrast, our proposed scheme enables asynchronous execution of concurrent feedback operations, aligning them only when required. In simulation, we assume unlimited connectivity for the baseline and treat the communication latency of a feedback operation as constant, regardless of the number of qubits. In practice, however, this assumption is unrealistic, and thus our simulation results actually overestimate the performance of the baseline.

6.4.4 Results Analysis. Figure 15 shows the normalized end-to-end runtime of a HISQ program, demonstrating that Distributed-HISQ reduces execution time by an average of 22.8%. The reason is that there is no simultaneous feedback in this benchmark, making the advantage of Distributed-HISQ not significant. On the other hand, the communication latency of Distributed-HISQ grows as the system scale up, but we assume constant communication latency in baseline. As such, the performance of Distributed-HISQ is worse than baseline for "bv" benchmark.

6.4.5 Impact on Fidelity. As an example to showcase the effect of our reduced latency on fidelity, we consider the long-range CNOT

circuit shown in Figure 14. We compare the infidelity between Distributed-HISQ and baseline with the T1/T2 time ranging from 30 µs to 300 µs (Figure 16). It can be observed that Distributed-HISQ constantly reduces infidelity around 5×. This effect is also due to the ability to enable simultaneous feedback. In the baseline scheme, all controllers are forced to follow the same program flow, requiring the second set of measurements to occur after the conditional X gate. In contrast, Distributed-HISQ allows these measurements to be performed immediately after the H gates, thereby reducing execution time.

7 Discussion

7.1 Adaptability of HISQ

While our current single-core DQCtrl implementation — where one HISQ core controls all 28 ports on an FPGA — is adequate for many tasks, it faces a potential instruction issue rate bottleneck [11] in time-critical operations. To overcome this scalability problem, our architecture allows for a multi-core configuration on a single FPGA. By partitioning the control ports among multiple HISQ cores, we eliminate the issuance bottleneck, ensuring robust performance for computationally demanding experiments.

7.2 Scalability of Distributed-HISQ

The scalability of a quantum control architecture is determined by various factors, including instruction issue rate, memory consumption, and synchronization latency, and is ultimately limited by its most critical bottleneck. While this work does not propose a universally scalable quantum control architecture, it targets two key challenges: achieving efficient synchronization and enabling a lightweight hardware implementation for the digital control logic. Distributed-HISQ utilizes a queue-based event timing mechanism that compiles a single quantum program into independent instruction streams for multiple controllers. These streams execute in parallel and synchronize only on demand, significantly reducing the instruction load on each controller. This partitioned execution model enhances the efficiency of individual controllers, thereby improving the overall scalability of the quantum control architecture.

8 Conclusion

We have presented Distributed-HISQ, a distributed quantum control architecture addressing key challenges in scalability and adaptability for evolving quantum hardware. Its core contributions are twofold: a hardware-implementable yet expressive abstraction layer, and an efficient synchronization scheme that provides near-zero overhead and enhances compiler flexibility. We implemented and verified the architecture on a realistic superconducting qubit control system. Furthermore, simulation results demonstrate that Distributed-HISQ reduces execution overhead and improves result fidelity. By effectively decoupling instruction execution while maintaining precise control, Distributed-HISQ provides a viable pathway toward fully scalable quantum control architectures.

Acknowledgments

We thank the anonymous reviewers for their insightful feedback. We thank Dr. Lianchen Han for his support in debugging the firmware of DQCtrl. This work was supported in part by National Natural Science Foundation of China (Grant No. 62025404, 62222411), National Key Research and Development Program of China (Grant No. 2023YFB4404400).

References

[1] Rajeev Acharya, Dmitry A. Abanin, Laleh Aghababaie-Beni, Igor Aleiner, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Nikita Astrakhantsev, Juan Atalaya, Ryan Babbush, Dave Bacon, Brian Ballard, Joseph C. Bardin, Johannes Bausch, Andreas Bengtsson, Alexander Bilmes, Sam Blackwell, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, David A. Browne, Brett Buchea, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Anthony Cabrera, Juan Campero, Hung-Shen Chang, Yu Chen, Zijun Chen, Ben Chiaro, Desmond Chik, Charina Chou, Jahan Claes, Agnetta Y. Cleland, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Sayan Das, Alex Davies, Laura De Lorenzo, Dripto M. Debroy, Sean Demura, Michel Devoret, Agustin Di Paolo, Paul Donohoe, Ilya Drozdov, Andrew Dunsworth, Clint Earle, Thomas Edlich, Alec Eickbusch, Aviv Moshe Elbag, Mahmoud Elzouka, Catherine Erickson, Lara Faoro, Edward Farhi, Vinicius S. Ferreira, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, Suhas Ganjam, Gonzalo Garcia, Robert Gasca, Élie Genois, William Giang, Craig Gidney, Dar Gilboa, Raja Gosula, Alejandro Grajales Dau, Dietrich Graumann, Alex Greene, Jonathan A. Gross, Steve Habegger, John Hall, Michael C. Hamilton, Monica Hansen, Matthew P. Harrigan, Sean D. Harrington, Francisco J. H. Heras, Stephen Heslin, Paula Heu, Oscar Higgott, Gordon Hill, Jeremy Hilton, George Holland, Sabrina Hong, Hsin-Yuan Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Stephen Jordan, Chaitali Joshi, Pavol Juhas, Dvir Kafri, Hui Kang, Amir H. Karamlou, Kostyantyn Kechedzhi, Julian Kelly, Trupti Khaire, Tanuj Khattar, Mostafa Khezri, Seon Kim, Paul V. Klimov, Andrey R. Klots, Bryce Kobrin, Pushmeet Kohli, Alexander N. Korotkov, Fedor Kostritsa, Robin Kothari, Borislav Kozlovskii, John Mark Kreikebaum, Vladislav D. Kurilovich, Nathan Lacroix, David Landhuis, Tiano Lange-Dei, Brandon W. Langley, Pavel Laptev, Kim-Ming Lau, Loïck Le Guevel, Justin Ledford, Joonho Lee, Kenny Lee, Yuri D. Lensky, Shannon Leon, Brian J. Lester, Wing Yan Li, Yin Li, Alexander T. Lill, Wayne Liu, William P. Livingston, Aditya Locharla, Erik Lucero, Daniel Lundahl, Aaron Lunt, Sid Madhuk, Fionn D. Malone, Ashley Maloney, Salvatore Mandrà, James Manyika, Leigh S. Martin, Orion Martin, Steven Martin, Cameron Maxfield, Jarrod R. McClean, Matt McEwen, Seneca Meeks, Anthony Megrant, Xiao Mi, Kevin C. Miao, Amanda Mieszala, Reza Molavi, Sebastian Molina, Shirin Montazeri, Alexis Morvan, Ramis Movassagh, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Chia-Hung Ni, Murphy Yuezhen Niu, Thomas E. O'Brien, William D. Oliver, Alex Opremcak, Kristoffer Ottosson, Andre Petukhov, Alex Pizzuto, John Platt, Rebecca Potter, Orion Pritchard, Leonid P. Pryadko, Chris Quintana, Ganesh Ramachandran, Matthew J. Reagor, John Redding, David M. Rhodes, Gabrielle Roberts, Eliott Rosenberg, Emma Rosenfeld, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Andrew W. Senior, Michael J. Shearn, Aaron Shorter, Noah Shutty, Vladimir Shvarts, Shraddha Singh, Volodymyr Sivak, Jindra Skruzny, Spencer Small, Vadim Smelyanskiy, W. Clarke Smith, Rolando D. Somma, Sofia Springer, George Sterling, Doug Strain, Jordan Suchard, Aaron Szasz, Alex Sztein, Douglas Thor, Alfredo Torres, M. Mert Torunbalci, Abeer Vaishnav, Justin Vargas, Sergey Vdovichev, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Steven Waltman, Shannon X. Wang, Brayden Ware, Kate Weber, Travis Weidel, Theodore White, Kristi Wong, Bryan W. K. Woo, Cheng Xing, Z. Jamie Yao, Ping Yeh, Bicheng Ying, Juhwan Yoo, Noureldin Yosri, Grayson Young, Adam Zalcman, Yaxing Zhang, Ningfeng Zhu, and Nicholas Zobrist. 2025. Quantum Error Correction below the Surface Code Threshold. *Nature* 638, 8052 (2025), 920–926. https://doi.org/10.1038/s41586-024-08449-y

[2] Ben Barber, Kenton M. Barnes, Tomasz Bialas, Okan Buğdaycı, Earl T. Campbell, Neil I. Gillespie, Kauser Johar, Ram Rajan, Adam W. Richardson, Luka Skoric, Canberk Topal, Mark L. Turner, and Abbas B. Ziad. 2025. A Real-Time, Scalable, Fast and Resource-Efficient Decoder for a Quantum Computer. *Nature Electronics* 8, 1 (2025), 84–91. https://doi.org/10.1038/s41928-024-01319-5

[3] Elisa Bäumer, Vinay Tripathi, Derek S. Wang, Patrick Rall, Edward H. Chen, Swarnadeep Majumder, Alireza Seif, and Zlatko K. Minev. 2024. Efficient Long-Range Entanglement Using Dynamic Circuits. *PRX Quantum* 5, 3 (2024), 030339. https://doi.org/10.1103/PRXQuantum.5.030339

[4] Ilkwon Byun, Junpyo Kim, Dongmoon Min, Ikki Nagaoka, Kosuke Fukumitsu, Iori Ishikawa, Teruo Tanimoto, Masamitsu Tanaka, Koji Inoue, and Jangwoo Kim. 2022. XQsim: Modeling Cross-Technology Control Processors for 10+K Qubit Quantum Computers. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 366–382. https://doi.org/10.1145/3470496.3527417

[5] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. 2022. OpenQASM 3: A Broader and Deeper Quantum Assembly Language. *ACM Transactions on Quantum Computing* 3, 3 (2022), 12:1–12:50. https://doi.org/10.1145/3505636

[6] Stephen DiAdamo, Marco Ghibaudi, and James Cruise. 2021. Distributed Quantum Computing and Network Control for Accelerated VQE. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–21. https://doi.org/10.1109/TQE.2021.3057908

[7] L. DiCarlo, J. M. Chow, J. M. Gambetta, Lev S. Bishop, B. R. Johnson, D. I. Schuster, J. Majer, A. Blais, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf. 2009. Demonstration of Two-Qubit Algorithms with a Superconducting Quantum Processor. *Nature* 460, 7252 (2009), 240–244. https://doi.org/10.1038/nature08121

[8] Austin G. Fowler and Craig Gidney. 2019. Low Overhead Quantum Computation Using Lattice Surgery. https://doi.org/10.48550/arXiv.1808.06709 arXiv:1808.06709

[9] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. 2012. Surface Codes: Towards Practical Large-Scale Quantum Computation. *Physical Review A* 86, 3 (2012), 032324. https://doi.org/10.1103/PhysRevA.86.032324

[10] Neelay Fruitwala, Gang Huang, Yilun Xu, Abhi Rajagopala, Akel Hashim, Ravi K. Naik, Kasra Nowrouzi, David I. Santiago, and Irfan Siddiqi. 2024. Distributed Architecture for FPGA-based Superconducting Qubit Control. arXiv:2404.15260

[11] X. Fu, L. Riesebos, M. A. Rol, Jeroen van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. 2019. eQASM: An Executable Quantum Instruction Set Architecture. In *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture*. 224–237. https://doi.org/10.1109/HPCA.2019.00040

[12] X. Fu, R. N. Schouten, C. G. Almudever, L. DiCarlo, K. Bertels, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, and W. J. Vlothuizen. 2017. An Experimental Microarchitecture for a Superconducting Quantum Processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 813–825. https://doi.org/10.1145/3123939.3123952

[13] X. Fu, Jintao Yu, Xing Su, Hanru Jiang, Hua Wu, Fucheng Cheng, Xi Deng, Jinrong Zhang, Lei Jin, Yihang Yang, Le Xu, Chunchao Hu, Anqi Huang, Guangyao Huang, Xiaogang Qiang, Mingtang Deng, Ping Xu, Weixia Xu, Wanwei Liu, Yu Zhang, Yuxin Deng, Junjie Wu, and Yuan Feng. 2021. Quingo: A Programming Framework for Heterogeneous Quantum-Classical Computing with NISQ Features. *ACM Transactions on Quantum Computing* 2, 4 (2021), 19:1–19:37. https://doi.org/10.1145/3483528

[14] Xiang Fu, Mengyu Zhang, and Zhou Peng. 2019. CACTUS: A quantum control architecture simulator. https://github.com/gtaifu/CACTUS

[15] Craig Gidney. 2021. Stim: A Fast Stabilizer Circuit Simulator. *Quantum* 5 (2021), 497. https://doi.org/10.22331/q-2021-07-06-497

[16] Google. 2025. Google quantum computing roadmap. https://quantumai.google/roadmap

[17] Cheng Guo, Jin Lin, Lian-Chen Han, Na Li, Li-Hua Sun, Fu-Tian Liang, Dong-Dong Li, Yu-Huai Li, Ming Gong, Yu Xu, Sheng-Kai Liao, and Cheng-Zhi Peng. 2022. Low-Latency Readout Electronics for Dynamic Superconducting Quantum Computing. *AIP Advances* 12, 4 (2022), 045024. https://doi.org/10.1063/5.0088879

[18] Frank Haverkamp, Juergen Saalmueller, Markus Buehler, Tristan Müller, and Thilo Maurer. 2023. Central controller for a quantum system. Patent No. WO2023208815A1, Filed Apr. 24th., 2023, Issued Nov. 2th., 2023.

[19] Dominic Horsman, Austin G. Fowler, Simon Devitt, and Rodney Van Meter. 2012. Surface Code Quantum Computing by Lattice Surgery. *New Journal of Physics* 14, 12 (2012), 123011. https://doi.org/10.1088/1367-2630/14/12/123011

[20] L. Hu, Y. Ma, W. Cai, X. Mu, Y. Xu, W. Wang, Y. Wu, H. Wang, Y. P. Song, C.-L. Zou, S. M. Girvin, L.-M. Duan, and L. Sun. 2019. Quantum Error Correction and Universal Gate Set Operation on a Binomial Bosonic Logical Qubit. *Nature Physics* 15, 5 (2019), 503–508. https://doi.org/10.1038/s41567-018-0414-3

[21] IBM. 2023. Expanding the IBM Quantum roadmap to anticipate the future of quantum-centric supercomputing. https://research.ibm.com/blog/ibm-quantum-roadmap-2025

[22] Zurich Instruments. 2025. Quantum Computing Control System. https://www.zhinst.com/en/quantum-computing-systems/qccs

[23] Interlin-q. 2025. *Distributed QC for Qiskit.* https://github.com/Interlin-q/diskit/

[24] Mohammad Reza Jokar, Richard Rines, Ghasem Pasandi, Haolin Cong, Adam Holmes, Yunong Shi, Massoud Pedram, and Frederic T. Chong. 2022. DigiQ: A Scalable Digital Controller for Quantum Computers Using SFQ Logic. In *Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture*. 400–414. https://doi.org/10.1109/HPCA53966.2022.00037

[25] William D. Kalfus, Diana F. Lee, Guilhem J. Ribeill, Spencer D. Fallek, Andrew Wagner, Brian Donovan, Diego Riste, and Thomas A. Ohki. 2020. High-Fidelity Control of Superconducting Qubits Using Direct Microwave Synthesis in Higher Nyquist Zones. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–12. https://doi.org/10.1109/TQE.2020.3042895

[26] Junpyo Kim, Dongmoon Min, Jungmin Cho, Hyeonseong Jeong, Ilkwon Byun, Junhyuk Choi, Juwon Hong, and Jangwoo Kim. 2024. A Fault-Tolerant Million Qubit-Scale Distributed Quantum Computer. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, Vol. 2. 1–19. https://doi.org/10.1145/3620665.3640388

[27] Andrew D. King, Alberto Nocera, Marek M. Rams, Jacek Dziarmaga, Roeland Wiersema, William Bernoudy, Jack Raymond, Nitin Kaushal, Niclas Heinsdorf, Richard Harris, Kelly Boothby, Fabio Altomare, Mohsen Asad, Andrew J. Berkley, Martin Boschnak, Kevin Chern, Holly Christiani, Samantha Cibere, Jake Connor, Martin H. Dehn, Rahul Deshpande, Sara Ejtemaee, Pau Farre, Kelsey Hamer, Emile Hoskinson, Shuiyuan Huang, Mark W. Johnson, Samuel Kortas, Eric Ladizinsky, Trevor Lanting, Tony Lai, Ryan Li, Allison J. R. MacDonald, Gaelen Marsden, Catherine C. McGeoch, Reza Molavi, Travis Oh, Richard Neufeld, Mana Norouzpour, Joel Pasvolsky, Patrick Poitras, Gabriel Poulin-Lamarre, Thomas Prescott, Mauricio Reis, Chris Rich, Mohammad Samani, Benjamin Sheldan, Anatoly Smirnov, Edward Sterpka, Berta Trullas Clavera, Nicholas Tsai, Mark Volkmann, Alexander M. Whiticar, Jed D. Whittaker, Warren Wilkinson, Jason Yao, T. J. Yi, Anders W. Sandvik, Gonzalo Alvarez, Roger G. Melko, Juan Carrasquilla, Marcel Franz, and Mohammad H. Amin. 2025. Beyond-Classical Computation in Quantum Simulation. *Science* 388, 6743 (2025), 199–204. https://doi.org/10.1126/science.ado6285

[28] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. 2019. A Quantum Engineer's Guide to Superconducting Qubits. *Applied Physics Reviews* 6, 2 (2019), 021318. https://doi.org/10.1063/1.5089550

[29] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. 2023. QASMBench: A Low-Level Quantum Benchmark Suite for NISQ Evaluation and Simulation. *ACM Transactions on Quantum Computing* 4, 2 (2023), 10:1–10:26. https://doi.org/10.1145/3550488

[30] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*. 1001–1014. https://doi.org/10.1145/3297858.3304023

[31] Shaowei Li, Daojin Fan, Ming Gong, Yangsen Ye, Xiawei Chen, Yulin Wu, Huijie Guan, Hui Deng, Hao Rong, He-Liang Huang, Chen Zha, Kai Yan, Shaojun Guo, Haoran Qian, Haibin Zhang, Fusheng Chen, Qingling Zhu, Youwei Zhao, Shiyu Wang, Chong Ying, Sirui Cao, Jiale Yu, Futian Liang, Yu Xu, Jin Lin, Cheng Guo, Lihua Sun, Na Li, Lianchen Han, Cheng-Zhi Peng, Xiaobo Zhu, and Jian-Wei Pan. 2022. Realization of Fast All-Microwave Controlled-Z Gates with a Tunable Coupler. *Chinese Physics Letters* 39, 3 (2022), 030302. https://doi.org/10.1088/0256-307X/39/3/030302

[32] Jin Lin, Fu-Tian Liang, Yu Xu, Li-Hua Sun, Cheng Guo, Sheng-Kai Liao, and Cheng-Zhi Peng. 2019. Scalable and Customizable Arbitrary Waveform Generator for Superconducting Quantum Computing. *AIP Advances* 9, 11 (2019), 115309. https://doi.org/10.1063/1.5120299

[33] Minzhao Liu, Ruslan Shaydulin, Pradeep Niroula, Matthew DeCross, Shih-Han Hung, Wen Yu Kon, Enrique Cervero-Martín, Kaushik Chakraborty, Omar Amer, Scott Aaronson, Atithi Acharya, Yuri Alexeev, K. Jordan Berg, Shouvanik Chakrabarti, Florian J. Curchod, Joan M. Dreiling, Neal Erickson, Cameron Foltz, Michael Foss-Feig, David Hayes, Travis S. Humble, Niraj Kumar, Jeffrey Larson, Danylo Lykov, Michael Mills, Steven A. Moses, Brian Neyenhuis, Shaltiel Eloul, Peter Siegfried, James Walker, Charles Lim, and Marco Pistoia. 2025. Certified Randomness Using a Trapped-Ion Quantum Processor. *Nature* 640, 8058 (2025), 343–348. https://doi.org/10.1038/s41586-025-08737-1

[34] Quantum Machines. 2025. OPX1000: Modular High-Density Hybrid Control Platform. https://www.quantum-machines.co/products/opx1000/

[35] P. Magnard, S. Storz, P. Kurpiers, J. Schär, F. Marxer, J. Lütolf, T. Walter, J.-C. Besse, M. Gabureac, K. Reuer, A. Akin, B. Royer, A. Blais, and A. Wallraff. 2020. Microwave Quantum Link between Superconducting Circuits Housed in Spatially Separated Cryogenic Systems. *Physical Review Letters* 125, 26 (2020), 260502. https://doi.org/10.1103/PhysRevLett.125.260502

[36] Dongmoon Min, Junpyo Kim, Junhyuk Choi, Ilkwon Byun, Masamitsu Tanaka, Koji Inoue, and Jangwoo Kim. 2023. QIsim: Architecting 10+K Qubit QC Interfaces Toward Quantum Supremacy. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–16. https://doi.org/10.1145/3579371.3589036

[37] Adam Paetznick and Krysta M. Svore. 2014. Repeat-until-Success: Non-Deterministic Decomposition of Single-Qubit Unitaries. *Quantum Info. Comput.* 14, 15-16 (2014), 1277–1301. https://doi.org/10.48550/arXiv.1311.1074

[38] Colm A. Ryan, Blake R. Johnson, Diego Ristè, Brian Donovan, and Thomas A. Ohki. 2017. Hardware for Dynamic Quantum Computing. *Review of Scientific Instruments* 88, 10 (2017), 104703. https://doi.org/10.1063/1.5006525 arXiv:1704.08314

[39] Yuan Sun. 2024. Buffer-Atom-Mediated Quantum Logic Gates with off-Resonant Modulated Driving. *Science China Physics, Mechanics & Astronomy* 67, 12 (2024), 120311. https://doi.org/10.1007/s11433-024-2478-8

[40] Bochen Tan and Jason Cong. 2020. Optimal Layout Synthesis for Quantum Computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9. https://doi.org/10.1145/3400302.3415620

[41] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. DiCarlo. 2017. Scalable Quantum Circuit and Control for a Superconducting Surface Code. *Physical Review Applied* 8, 3 (2017), 034021. https://doi.org/10.1103/PhysRevApplied.8.034021

[42] Ken Xuan Wei, Isaac Lauer, Emily Pritchett, William Shanks, David C. McKay, and Ali Javadi-Abhari. 2024. Native Two-Qubit Gates in Fixed-Coupling, Fixed-Frequency Transmons Beyond Cross-Resonance Interaction. *PRX Quantum* 5, 2 (2024), 020338. https://doi.org/10.1103/PRXQuantum.5.020338

[43] Claire Xenia Wolf. 2023. PicoRV32: A Size-Optimized RISC-V CPU. https://github.com/cliffordwolf/picorv32

[44] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. 2022. AutoComm: A Framework for Enabling Efficient Communication in Distributed Quantum Programs. In *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture*. 1027–1041. https://doi.org/10.1109/MICRO56248.2022.00074

[45] Liang Xiang, Zhiwen Zong, Zhenhai Sun, Ze Zhan, Ying Fei, Zhangjingzi Dong, Chongxin Run, Zhilong Jia, Peng Duan, Jianlan Wu, Yi Yin, and Guoping Guo. 2020. Simultaneous Feedback and Feedforward Control and Its Application to Realize a Random Walk on the Bloch Sphere in an Xmon-Superconducting-Qubit System. *Physical Review Applied* 14, 1 (2020), 014099. https://doi.org/10.1103/PhysRevApplied.14.014099

[46] Yilun Xu, Gang Huang, Jan Balewski, Ravi Naik, Alexis Morvan, Bradley Mitchell, Kasra Nowrouzi, David I. Santiago, and Irfan Siddiqi. 2021. QubiC: An Open-Source FPGA-Based Control and Measurement System for Superconducting Quantum Information Processors. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–11. https://doi.org/10.1109/TQE.2021.3116540

[47] Yilun Xu, Gang Huang, Neelay Fruitwala, Abhi Rajagopala, Ravi K. Naik, Kasra Nowrouzi, David I. Santiago, and Irfan Siddiqi. 2023. QubiC 2.0: An Extensible Open-Source Qubit Control System Capable of Mid-Circuit Measurement and Feed-Forward. https://doi.org/10.48550/arXiv.2309.10333 arXiv:2309.10333

[48] Fei Yan, Philip Krantz, Youngkyu Sung, Morten Kjaergaard, Daniel L. Campbell, Terry P. Orlando, Simon Gustavsson, and William D. Oliver. 2018. Tunable Coupling Scheme for Implementing High-Fidelity Two-Qubit Gates. *Physical Review Applied* 10, 5 (2018), 054062. https://doi.org/10.1103/PhysRevApplied.10.054062

[49] Yuchen Yang, Zhongtao Shen, Xing Zhu, Ziqi Wang, Gengyan Zhang, Jingwei Zhou, Xun Jiang, Chunqing Deng, and Shubin Liu. 2022. FPGA-based Electronic System for the Control and Readout of Superconducting Quantum Processors. *Review of Scientific Instruments* 93, 7 (2022), 074701. https://doi.org/10.1063/5.0085467

[50] Keyi Yin, Xiang Fang, Travis S. Humble, Ang Li, Yunong Shi, and Yufei Ding. 2024. Surf-Deformer: Mitigating Dynamic Defects on Surface Code via Adaptive Deformation. In *Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture*. 750–764. https://doi.org/10.1109/MICRO61859.2024.00061

[51] George Zettles, Scott Willenborg, Blake R. Johnson, Andrew Wack, and Brian Allison. 2022. 26.2 Design Considerations for Superconducting Quantum Systems. In *Proceedings of the 2022 IEEE International Solid- State Circuits Conference*. 1–3. https://doi.org/10.1109/ISSCC42614.2022.9731706

[52] Fang Zhang, Xing Zhu, Rui Chao, Cupjin Huang, Linghang Kong, Guoyang Chen, Dawei Ding, Haishan Feng, Yihuai Gao, Xiaotong Ni, Liwei Qiu, Zhe Wei, Yueming Yang, Yang Zhao, Yaoyun Shi, Weifeng Zhang, Peng Zhou, and Jianxin Chen. 2024. A Classical Architecture for Digital Quantum Computers. *ACM Transactions on Quantum Computing* 5, 1 (2024), 1–24. https://doi.org/10.1145/3626199

[53] Mengyu Zhang. 2018. *QuMAsim: A Quantum Architecture Simulation and Verification Platform*. Master's thesis. Delft University of Technology.

[54] Mengyu Zhang, Lei Xie, Zhenxing Zhang, Qiaonian Yu, Guanglei Xi, Hualiang Zhang, Fuming Liu, Yarui Zheng, Yicong Zheng, and Shengyu Zhang. 2021. Exploiting Different Levels of Parallelism in the Quantum Control Microarchitecture for Superconducting Qubits. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 898–911. https://doi.org/10.1145/3466752.3480116

[55] Xinfang Zhang, Zhihao Wu, Gregory A. L. White, Zhongcheng Xiang, Shun Hu, Zhihui Peng, Yong Liu, Dongning Zheng, Xiang Fu, Anqi Huang, Dario Poletti, Kavan Modi, Junjie Wu, Mingtang Deng, and Chu Guo. 2025. Learning and Forecasting Open Quantum Dynamics with Correlated Noise. *Communications Physics* 8, 1 (2025), 29. https://doi.org/10.1038/s42005-025-01944-2

[56] Xiang Zou, Shavindra P. Premaratne, M. Adriaan Rol, Sonika Johri, Viacheslav Ostroukh, David J. Michalak, Roman Caudillo, James S. Clarke, Leonardo DiCarlo, and A. Y. Matsuura. 2020. Enhancing a Near-Term Quantum Accelerator's Instruction Set Architecture for Materials Science Applications. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–7. https://doi.org/10.1109/TQE.2020.2965810