XQsim: Modeling Cross-Technology Control Processors for 10+K Qubit Quantum Computers

Ilkwon Byun
Seoul National University
Seoul, Republic of Korea
ik.byun@snu.ac.kr

Junpyo Kim
Seoul National University
Seoul, Republic of Korea
junpyo.kim@snu.ac.kr

Dongmoon Min
Seoul National University
Seoul, Republic of Korea
dongmoon.min@snu.ac.kr

Ikki Nagaoka
Nagoya University
Nagoya, Japan
nagaoka@super.nuee.nagoya-u.ac.jp

Kosuke Fukumitsu
Kyushu University
Fukuoka, Japan
kosuke.fukumitsu@cpc.ait.kyushu-u.ac.jp

Iori Ishikawa
Kyushu University
Fukuoka, Japan
iori.ishikawa@cpc.ait.kyushu-u.ac.jp

Teruo Tanimoto
Kyushu University
Fukuoka, Japan
tteruo@kyudai.jp

Masamitsu Tanaka
Nagoya University
Nagoya, Japan
masami_t@nagoya-u.jp

Koji Inoue
Kyushu University
Fukuoka, Japan
inoue@ait.kyushu-u.ac.jp

Jangwoo Kim*
Seoul National University
Seoul, Republic of Korea
jangwoo@snu.ac.kr

ABSTRACT

10+K qubit quantum computer is essential to achieve a true sense of quantum supremacy. With the recent effort towards the large-scale quantum computer, architects have revealed various scalability issues including the constraints in a quantum control processor, which should be holistically analyzed to design a future scalable control processor. However, it has been impossible to identify and resolve the processor's scalability bottleneck due to the absence of a reliable tool to explore an extensive design space including microarchitecture, device technology, and operating temperature.

In this paper, we present *XQsim*, an open-source cross-technology quantum control processor simulator. XQsim can accurately analyze the target control processors' scalability bottlenecks for various device technology and operating temperature candidates. To achieve the goal, we first fully implement a convincing control processor microarchitecture for the Fault-tolerant Quantum Computer (FTQC) systems. Next, on top of the microarchitecture, we develop an architecture-level control processor simulator (XQsim) and thoroughly validate it with post-layout analysis, timing-accurate RTL simulation, and noisy quantum simulation. Lastly, driven by XQsim, we provide the future directions to design a 10+K qubit quantum control processor with several design guidelines and architecture optimizations. Our case study shows that the final control processor architecture can successfully support ~59K qubits with our operating temperature and technology choices.

*Corresponding author.

CCS CONCEPTS

• **Computer systems organization → Quantum computing**; • **Hardware → Emerging tools and methodologies**.

KEYWORDS

Quantum computing, Single flux quantum (SFQ), Cryogenic computing, Modeling, Simulation

ACM Reference Format:
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 *The 49th Annual International Symposium on Computer Architecture (ISCA '22), June 18–22, 2022, New York, NY, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3470496.3527417

1 INTRODUCTION

Quantum computing is a new computing paradigm, which is expected to efficiently solve many important classically-intractable problems (i.e., *quantum supremacy*) [52, 66]. To achieve a true sense of quantum supremacy, it is necessary to build a 10+K qubit quantum computer because major practical applications require a large number of physical qubits. For example, quantum chemistry algorithms (e.g., Fermi-Hubbard simulation) require 100,000 physical qubits to perform fault-tolerant quantum operations over a sufficient number of logical qubits [47]. Therefore, architects have done their best to increase the number of qubits in quantum computers.

As the number of qubits has been successfully increased, it now becomes more important to scale a qubit control system. For example, the number of qubits in commercial quantum computers has rapidly grown from five in 2017 (IBM Canary [27]) to 127 in 2021 (IBM Eagle [28]), and we believe that this trend will continue thanks to the huge potential of quantum computing. Following this trend, architects should put more effort into designing a scalable control system, which can successfully support 10+K qubits.

However, architects now suffer from two critical challenges for designing a large-scale control system: limited scalability of a quantum-classical interface (QC interface) and a quantum control processor. First, in recent Noisy Intermediate Scale Quantum (NISQ) systems, an interface between classical hardware and qubits (i.e., QC interface) is considered the major scalability bottleneck. For example, today's superconducting quantum computers utilize per-qubit coaxial cables to send microwave pulses from room-temperature QC interface to qubits located in a dilution refrigerator (i.e., ~10mK). Unfortunately, as the number of qubits increases, this approach significantly suffers from space limitations and huge thermal loads on the millikelvin stage [39]. As a result, researchers in various areas are actively working on mitigating the challenges in QC interface scaling (e.g., scalable interconnect [40, 60], low-power cryogenic QC interface [32, 34]).

On the other hand, in future fault-tolerant quantum computer (FTQC) systems, digital processing hardware for the fault-tolerant support (i.e., quantum control processor) newly presents several scaling challenges orthogonal to those of the QC interface. First, the increasing demand for quantum error correction (QEC) limits the control processor's scalability. For example, with the increasing qubit scale, an FTQC system can fail due to the increasing instruction bandwidth requirement and error decoding latency [64, 65]. In addition, for the leading solid-state qubit technologies (i.e., superconducting qubit, spin qubit), the limited cooling capacity of dilution refrigerators (e.g., ~1.5W at 4K [39]) also constrains the control processor's scalability. Specifically, a large-scale control processor can dissipate significant heat to the 4K stage due to the 300K-to-4K data transfer through high-bandwidth digital cables or hardware units running at 4K [21, 64].

In this work, we target the scaling challenges in the quantum control processor which have not been actively explored yet. Note that as quantum supremacy is only achievable in the FTQC systems, architects should develop a scalable quantum control processor by carefully considering all the aforementioned constraints together.

However, architects have been impossible to design scalable quantum control processors due to the absence of a modeling tool. Without the reliable tool, architects cannot provide clear answers even for the basic design decisions, such as (1) how should we design their microarchitecture, (2) what temperature domain should we utilize, (3) what device technology should be used, and these questions are still under debate in both academia and industry. Therefore, to achieve a true sense of quantum supremacy, architects are now in dire need of a simulation tool to (1) evaluate the scalability of various control processor architectures and (2) provide guidelines for designing scalable control processors.

In this paper, we develop a simulation framework for designing a scalable quantum control processor (*XQsim*) and shed a light on the way toward 10+K qubit quantum computers with extensive design space exploration and optimizations. To achieve the goal, we first fully implement a convincing quantum control processor microarchitecture for the FTQC systems. As we target the quantum computer with the state-of-the-art QEC design (i.e., *surface code with lattice surgery* [25]), we design the detailed microarchitecture of the necessary hardware units and ISA to support them. We validate their functional correctness with RTL simulation.

Next, to evaluate the control processor for various design spaces, we develop *XQsim*, a cross-technology quantum control processor simulator, which supports various target temperatures (i.e., 300K, 4K), technologies (i.e., CMOS, RSFQ, ERSFQ) and microarchitecture designs. XQsim mainly consists of two submodules: XQ-estimator and XQ-simulator. XQ-estimator derives frequency, power, and area of the quantum control processor for the target temperature, technology, and microarchitecture design. With the XQ-estimator's output information, XQ-simulator identifies the scalability bottleneck and sustainable qubit scale of the target processor. We fully validate XQsim with post-layout analysis, timing -accurate RTL simulation, and noisy quantum simulation.

Finally, driven by the design-space exploration with our framework, we propose two design guidelines and four microarchitectural optimizations, and derive the 59,000 qubit-scale control processor with the proposed solutions. Starting from the fault-tolerant control processor with the current technology to the future control system, we identify the scalability bottlenecks and the effective solutions to resolve them. First, we identify that the current fault-tolerant control processor will suffer from the slow error decoding, and our first optimization extends its scalability more than seven times. Next, we reveal that the near-future control processor should resolve both 300K-4K data transfer and 4K device power bottlenecks, and our first design guideline and two microarchitecture optimizations additionally increase the scalability by 2.7~5.9 times. Lastly, as we observe that the future control processor should achieve both the fast error decoding and low 4K power consumption, we provide our second design guideline with the fourth optimization and further improve the scalability by 6.0 times. As a result, we successfully present the scalable quantum control processor architecture which manages up to 59,000 qubits.

Along with the 10+K physical qubits and scalable QC interface in the future, our simulation tool and scalable quantum processor design will highly contribute to achieving quantum supremacy. In summary, this paper makes the following contributions:

- **Fault-tolerant quantum control processor modeling**: To the best of our knowledge, this is the first work to model and evaluate the scalability of quantum control processors for FTQC systems.
- **Guidelines for scalable control processor design**: This is also the first study to explore and provide possible ways to improve the control processor's scalability over 10+K qubits.
- **XQsim tool release**: We release our XQsim tool to the community. As XQsim is the first open-source modeling framework for fault-tolerant quantum control processors, it can contribute to initiating follow-up FTQC research.

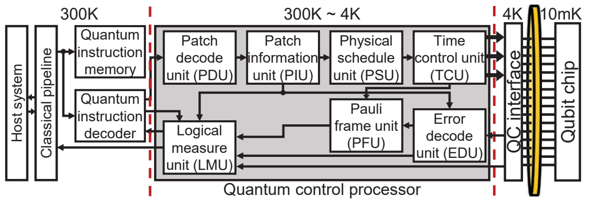

Figure 1: Fault-tolerant quantum computer system overview

2 BACKGROUND AND MOTIVATION

2.1 Fault-tolerant quantum computer system

Fig. 1 shows the overview of our target fault-tolerant quantum computer (FTQC) system for solid-state qubits (e.g., superconducting qubit, spin qubit), which consists of a qubit chip, a quantum-classical interface (QC interface), and a quantum control processor [16, 17]. First, the qubit chip is placed at the 10mK stage of a dilution refrigerator to minimize environmental errors [37, 38]. Next, the QC interface, which directly controls and measures the qubit's state, is assumed to locate at the 4K stage by following the recent research efforts to scale the interface [3, 48, 49, 70]. Finally, the quantum control processor is a core classical digital hardware, which performs all the complex operations to support fault-tolerant quantum computing (i.e., quantum error correction, fault-tolerant logical operations) [11, 23, 58]. For this purpose, the quantum control processor consists of various hardware units as shown in Fig. 1.

2.2 Quantum control processor's support for fault-tolerant quantum computing

In the FTQC systems, the quantum control processor should support two essential operations: (1) quantum error correction (QEC) and (2) fault-tolerant logical operation. In this section, we briefly introduce these operations and the required hardware supports to realize them. For the QEC protocol, we target *rotated surface code with patch-based lattice surgery* due to its low space-time overhead [14, 45].

2.2.1 Quantum error correction with the surface code. Surface code is widely considered as the prominent QEC protocol [15, 22, 31]. Fig. 2(a) shows an example surface-code patch representing a logical qubit, which is built with a square lattice of physical qubits. The surface-code patch consists of two types of physical qubits: data qubit (solid circle; D) containing state information and ancilla qubit (open circle; X or Z) used for extracting the error information. The example patch's code distance, notated as d, is three, determined by the target logical error rate. The qubit patch with larger d reduces logical error at the expense of using more physical qubits. In many classically-intractable quantum algorithms, d around 15 is required when the 0.1% physical error rate is assumed [2, 19, 20].

Error syndrome measurement. Surface code iteratively runs a quantum circuit called error syndrome measurement (ESM) to identify the data qubits' error types and locations [15, 35, 68]. ESM entangles two types of ancilla qubits (i.e., black-background X-ancilla and white-background Z-ancilla) to their adjacent data qubits by applying the controlled-Z gates in the special order (Fig. 2(b),(c)). At the end of an ESM cycle, the X-ancilla (or Z-ancilla) qubit measurement provides the information to detect Z errors (or X errors) that occurred in the adjacent data qubits. For every Z error (or X error) in the adjacent data qubits, the X-ancilla (or Z-ancilla) qubit's measurement value is flipped and such flipped measurement is called non-trivial Z (or X) error syndrome. As software-based ESM generation can suffer from a huge number of physical instructions [16, 64], the fault-tolerant control processor should automatically generate the ESM operations in the *physical schedule unit (PSU)*.

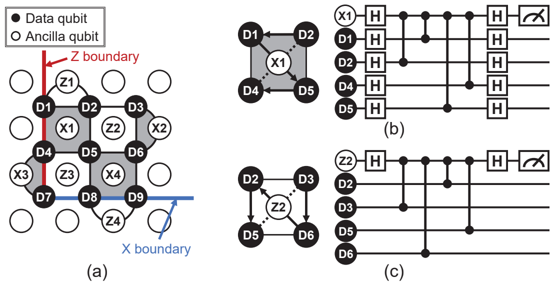

Figure 2: (a) Surface-code patch (d=3; number of data qubits on the edge) and ESM cycle for (b) X- and (c) Z-ancilla qubit

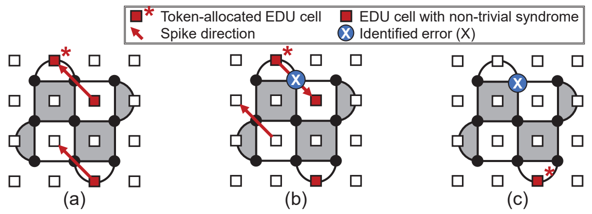

Figure 3: Error decoding example with the baseline EDU [69]

Error decoding. As a result of d-round ESM, the d-consecutive error syndromes are generated and forwarded to the *error decode unit (EDU)*. EDU, which consists of an array of per-ancilla-qubit EDU cells, runs an error decoding algorithm to find the pairs of the nearest non-trivial error syndromes. Fig. 3 shows an example based on the state-of-the-art EDU [69] where every ancilla qubit has its own EDU cell (i.e., rectangle at the ancilla qubit's position).

First, EDU loads error syndromes to the EDU cells and allocates a token to a cell with the non-trivial syndrome. Next, all other EDU cells with the non-trivial syndrome send the spikes toward the token-allocated cell (Fig. 3(a)). When the token-allocated cell receives a spike, it reflects and reversely propagates the spike to the original sender. With this step, EDU forms an X (or Z) error chain and identifies that X (or Z) errors occurred in the data qubits on the chain (Fig. 3(b)). Finally, if the original sender receives the reflected spike, EDU removes the matched syndromes and allocates a token to the next cell with non-trivial syndromes (Fig. 3(c)). EDU iterates the above process until no non-trivial error syndrome remains.

Error correction. With the identified errors, the *Pauli frame unit (PFU)* and the *logical measure unit (LMU)* perform error correction in the virtual manner [12, 33, 58]. Specifically, error-correcting operations are not physically applied to data qubits but they are considered when we use the data-qubit measurements. For this purpose, PFU accumulates the identified errors in its internal memory and keeps updating the error records. When LMU interprets a logical qubit measurement, the error correction is realized by flipping the data qubits' measurement values based on their error records (e.g., flip Z-basis measurement with an X error record).

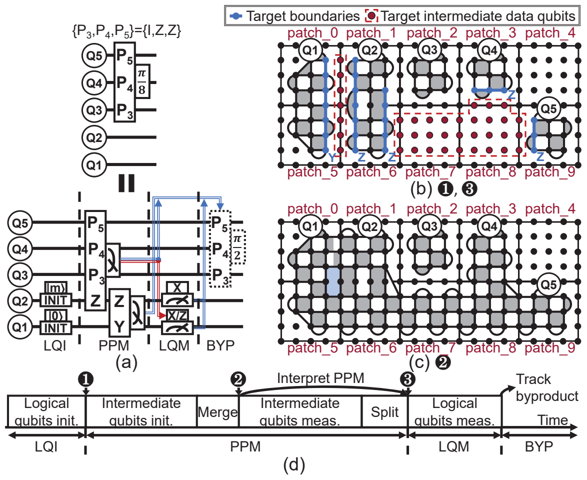
Target boundaries
Target intermediate data qubits
{P3,P4,P5}={I,Z,Z}
patch_0 patch_1 patch_2 patch_3 patch_4
patch_5 patch_6 patch_7 patch_8 patch_9
(b) ❶, ❸
(c) ❷
LQI PPM LQM BYP
(a)
Interpret PPM
Track byproduct
Logical qubits init.
Intermediate qubits init.
Merge
Intermediate qubits meas.
Split
Logical qubits meas.
Time
(d)

Figure 4: PPR$_{\frac{\pi}{8}}$ example; (a) Logical equivalent quantum circuit for PPR$_{\frac{\pi}{8}}$, (b) Qubit lattice before Merge (❶) and after Split (❸), (c) Qubit lattice right after the Merge (❷), and (d) Timeline of the logical operations for PPR$_{\frac{\pi}{8}}$

2.2.2 Fault-tolerant logical operation with the lattice surgery. For fault-tolerant quantum computing, we need a set of logical operations to manipulate the encoded logical qubit's state [56]. In the context of *patch-based lattice surgery*, we focus on $\frac{\pi}{8}$ Pauli product rotation (PPR$_{\frac{\pi}{8}}$) because arbitrary quantum algorithms can be expressed as a sequence of PPR$_{\frac{\pi}{8}}$ [45, 46]. PPR$_{\frac{\pi}{8}}$ is a quantum gate mathematically expressed as PPR$_{\frac{\pi}{8}}(P) = exp(-iP\frac{\pi}{8})$, where P is a Pauli product operator applied to the multiple target qubits (e.g., $X_n \bigotimes Z_m \bigotimes Y_l$ or $X_n Z_m Y_l$ applied to Q_n, Q_m, and Q_l).

For the natural execution of PPR$_{\frac{\pi}{8}}$ with the lattice surgery, PPR$_{\frac{\pi}{8}}$ is decomposed into the equivalent quantum circuit (Fig. 4(a)) which consists of four logical operations: (1) logical qubit initialization, (2) Pauli product measurement, (3) logical qubit measurement, and (4) byproduct tracking. With the example of PPR$_{\frac{\pi}{8}}(I_3 \bigotimes Z_4 \bigotimes Z_5)$, we provide a detailed description for each operation as follows.

Logical qubit initialization (LQI). For PPR$_{\frac{\pi}{8}}$, two additional logical qubits need to be initialized to $|0\rangle$ (Q1) and $|m\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\frac{\pi}{4}}|1\rangle)$ (Q2; magic state). Fig. 4(b) shows an example surface-code lattice after the initialization (❶).

Pauli product measurement (PPM). With the prepared logical qubits, the Pauli product measurement (PPM) is performed through dynamic lattice surgery. The dynamic lattice surgery is the process to "merge" and "split" the surface-code patches by following the target Pauli product. Specifically, the merge (or split) operation is realized through an ESM by including (or excluding) the physical qubits which locate between the target logical-qubit boundaries.

Fig. 4 shows the PPR$_{\frac{\pi}{8}}(I_3 Z_4 Z_5)$ example which performs two PPMs in parallel (i.e., $Z_2 I_3 Z_4 Z_5$, $Y_1 Z_2$ in Fig. 4(a)). These PPMs are implemented in four steps. First, all intermediate data qubits in the target area (dashed regions in Fig. 4(b)) are initialized to $|+\rangle$ (*Intermediate-qubit initialization*). Next, the target logical qubits are "merged" by applying the ESM to all physical qubits in the dashed target area. Note that each logical qubit should be merged through the boundary, whose type is the same as its target Pauli operator (i.e., Z-boundary for Q2, Q4, and Q5, Y-boundary for Q1). As a result, the lattice changes as shown in Fig. 4(c) (*Merge*). Third, the target intermediate data qubits are measured in X-basis (*Intermediate-qubit measurement*). Finally, the target qubits are "split" (❸) and the lattice returns to the original shape (Fig. 4(b)) (*Split*).

Logical qubit measurement (LQM). After finishing the PPM, two additional logical qubits (i.e., Q1, Q2) should be measured. For the X (or Z)-basis logical-qubit measurement, we first measure all the data qubits and then multiply the results of qubits on the same-type boundary (*Logical qubit measurement*). As the basis of the Q2 measurement depends on the previous PPM (i.e., for $Z_2 \bigotimes I_3 \bigotimes Z_4 \bigotimes Z_5$), the PPM result should be interpreted before the LQM (red line in Fig. 4(a)). We can interpret PPM by appropriately multiplying the intermediate error syndromes obtained from the previous Merge operation (*Interpret PPM*). Note that LMU supports both LQM and Interpret PPM operations.

Byproduct tracking (BYP). Lastly, we should handle a byproduct PPR$_{\frac{\pi}{2}}$, which can be generated depending on the results of preceding PPMs and LQMs (blue line in Fig. 4(a)). Similar to the error correction explained in Section 2.2.1, the PPR$_{\frac{\pi}{2}}$ is not directly applied to the logical qubits but is tracked and handled by LMU.

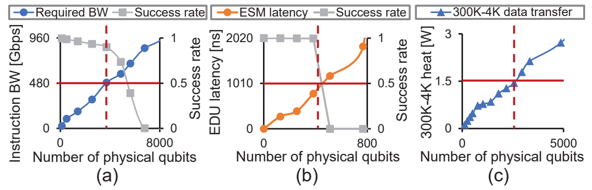

Figure 5: Scalability constraints; (a) Instruction bandwidth, (b) Error decoding latency, and (c) 300K-4K data transfer

2.3 Scalability constraints in fault-tolerant quantum control processors

With the recent efforts toward a large-scale quantum computer, architects have revealed that a quantum control processor suffers from various scalability constraints due to its FTQC support. In this section, we clarify four major constraints by analyzing the scalability of fault-tolerant control processors built with the current 300K CMOS technology (Fig. 13(a)). While following the methodology in Section 5.2, we use d=7 and run 100 random PPR$_{\frac{\pi}{8}}$ as our toy workload. To derive the application-level success rate, we adopt the same approach as [45]. Note that the success rate is the probability of executing a quantum algorithm without any logical errors. In Fig. 5, red lines indicate the points of violation for each constraint.

2.3.1 Instruction bandwidth requirement. First, the increasing instruction bandwidth requirement can limit the control processor's scalability [64]. As ESM should be applied to most physical qubits in parallel, the larger-scale qubit chip requires higher bandwidth. If the processor cannot meet the requirement due to the limited bandwidth (e.g., 480Gbps in Fig. 5(a)), it incurs a huge decoherence error. As a result, the success rate of the target quantum application exponentially decreases with the increasing qubit scale (Fig. 5(a)).

Table 1: Logical-qubit level QISA overview with the 64-bit format

Opcode [63:60]	Meas_flag [59:54]	Mreg_dst [53:41]	LQ_addr_offset [40:32]	Target [31:0]	Description
LQI			Logical qubit address offset	LQ_list (Logical qubit list)	Logical qubit initialization
MERGE_INFO			Logical qubit address offset	Pauli_list (Target Pauli product)	Patch information update for the Merge
SPLIT_INFO					Patch information update for the Split
INIT_INTMD					Intermediate data qubit initialization
MEAS_INTMD					Intermediate data qubit measurement
RUN_ESM					*d*-round ESM execution
PPM_INTERPRET	Logical measure flag	Logical measure register destination	Logical qubit address offset	Pauli_list (Target Pauli product)	PPM result interpretation
LQM_X/Z/FM	Logical measure flag	Logical measure register destination	Logical qubit address offset	LQ_list (Logical qubit list)	Single logical qubit measurement

2.3.2 Error decoding latency. Second, the increasing error decoding latency limits the scalability [23, 65]. As EDU should identify error chains one by one, the decoding latency increases with the number of physical qubits. However, when the error decoding becomes slower than the ESM execution (e.g., 1,010ns in Fig. 5(b)), the control processor should execute a huge number of extra ESM cycles and thus fails to run the target algorithm (Fig. 5(b)).

2.3.3 300K-4K data transfer. Third, if the control processor operates at 300K while QC interface is placed at 4K (Fig. 1), the increasing 300K-4K data transfer can limit the scalability [3, 57, 70]. Specifically, the cross-temperature data transfer requires the coaxial cables between them, which dissipates larger heat to 4K as the number of cables linearly increases with the qubit scale. However, due to the limited thermal budget at 4K (e.g., 1.5W [39]), the control processor running at 300K cannot be scaled over a certain point (Fig. 5(c)).

2.3.4 4K device power consumption. To mitigate the aforementioned scalability constraints, previous works suggested operating the control processor at 4K [64]. However, this approach even can degrade the scalability because complex and various hardware units in the control processor can dissipate significant power at the thermal-budget-limited 4K stage.

2.4 Research challenges and goal

As the control processor suffers from various scalability constraints, architects are in dire need of convincing design guidelines to effectively circumvent the bottlenecks. However, to the best of our knowledge, it has been impossible to explore such guidelines due to the absence of a reliable modeling tool, which accurately analyzes the target processor's scalability. To develop the scalability analysis tool, architects should resolve the following challenges.

2.4.1 Absence of the full-microarchitecture implementation. For the scalability analysis, architects need a detailed and convincing microarchitecture to estimate the control processor's performance, power, and area. Even though researchers have studied the FTQC architecture from various perspectives [10, 17, 23, 64], they lack either the detailed microarchitecture or the hardware-unit coverage.

2.4.2 Various device technologies and temperatures. To design a scalable control processor, industry and research community recently explore various target temperatures (i.e., 300K, 4K) and device technologies (i.e., RSFQ, ERSFQ, CMOS) [24, 43, 63]. However, there has been no clear answer for the best temperature and technology choices due to the complex scalability constraints. To provide the design guidelines, the scalability modeling tool should support all the various candidate temperatures and device technologies.

In this paper, we resolve the challenges as follows. We first fully implement a convincing microarchitecture for the fault-tolerant control processor (Section 3). Next, we develop a validated cross-technology control processor simulator, XQsim (Section 4). Finally, driven by our tool, we provide two guidelines and four optimizations for designing a 10+K qubit-scale control processor (Section 5).

3 FAULT-TOLERANT QUANTUM CONTROL PROCESSOR MICROARCHITECTURE

3.1 Logical-qubit level quantum ISA

We design our quantum instruction-set architecture (QISA) at the logical-qubit level by considering the QISA's scalability. Even though the QISA also can be designed at the physical-qubit level, the physical-qubit level QISA can suffer from huge qubit addressing overhead in a large-scale quantum computer.

Table 1 shows the overview of our 64-bit QISA with the detailed instruction fields. First, our QISA adopts Single-Operation-Multiple-Qubit execution at the logical-qubit level. Our ISA has a 32-bit Target field, two bits per logical qubit to express its target Pauli operator (i.e., I, X, Y, Z in Pauli_list) or indicate whether the qubit is a target or not (i.e., LQ_list). Therefore, we can apply the target logical operation to 16 consecutive logical qubits indexed with the 9-bit LQ_addr_offset. That is, our QISA can support a large-scale system which consists of up to 8,192 logical qubits.

With the instructions in Table 1, our QISA supports all the logical operations required for PPR$_{\frac{\pi}{8}}$. For example, LQI, PPM_INTERPRET, and LQM instructions correspond to the logical qubit initialization, interpret PPM, and logical qubit measurement in Section 2.2.2. Among them, PPM_INTERPRET and LQM utilize 6-bit measure flag (Meas_flag) and 13-bit measurement register destination (Mreg_dst) fields to perform and store the logical measurements. The Meas_flag field includes various control bits for determining the basis of LQM_FM and tracking the byproduct PPR$_{\frac{\pi}{2}}$ (blue line in Fig. 4(a)). Note that the LQM_FM instruction corresponds to the feedback measurement which requires a preceding PPM_INTERPRET's result for its basis choice (e.g., LQM on Q1 with the red line in Fig. 4(a)).

In addition, our QISA includes the patch information update instructions (i.e., MERGE_INFO, SPLIT_INFO) to support the Merge and Split operations. The MERGE_INFO (or SPLIT_INFO) instruction is used to appropriately change the target patches' information before the merging (or splitting) ESM. Therefore, Merge (or Split) is realized by executing d-round ESM cycles with the RUN_ESM instruction right after the MERGE_INFO (or SPLIT_INFO).

In our implementation, other FTQC system supports (e.g., error decoding, error correction, BYP) are not supported at the ISA level but automatically performed at the hardware level (e.g., EDU, LMU).

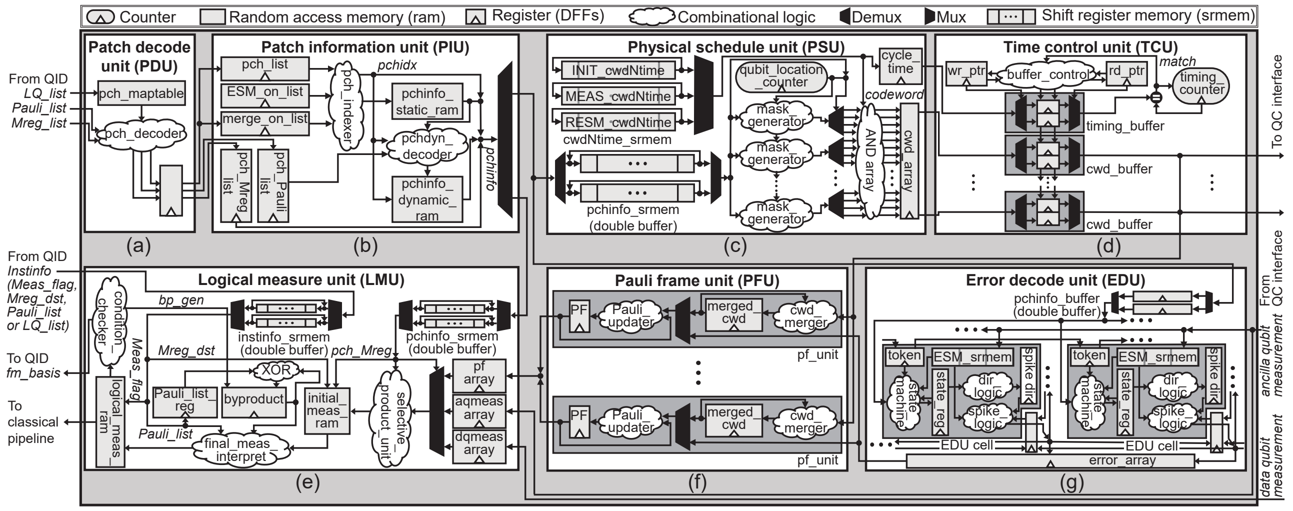

Figure 6: Full microarchitecture implementation of the fault-tolerant quantum control processor

Table 2: Patch information example

pchinfo_static			
pch_idx	pch_type	Z_boundary	X_boundary
patch_3	mapped, $\|0\rangle$	Bottom	Left
patch_8	intermediate	None	None
patch_9	mapped, $\|+\rangle$	Left	Bottom

(a) Static patch information

pchinfo_dynamic						
pch_idx	ESM_left	ESM_top	ESM_right	ESM_bottom	ESM_on	merge_on
patch_3	Z→ Z	X→ X	Z→ Z	X→ Z&X	T→ T	F→ T
patch_8	None→ Z&X	None→ Z&X	None→ Z&X	None→ Z	F→ T	F→ T
patch_9	X→ Z&X	Z→ Z	X→ X	Z→ Z	T→ T	F→ T

(b) Dynamic patch information

3.2 Full microarchitecture implementation

Fig. 6 shows the full microarchitecture implementation of our quantum-control processor. We introduce the detailed microarchitecture of each hardware unit as follows.

3.2.1 Quantum instruction decoder (QID). QID decomposes instructions into several fields (e.g., Meas_flag, Mreg_dst, Pauli_list, LQ_list) and forwards them to PDU and LMU. To efficiently perform this operation, QID accumulates several instructions which can be executed in the same time step (e.g., LQM or PPM_INTERPRET with different Mreg_dst) and forwards them at once (e.g., Mreg_dst values are accumulated into Mreg_list). In addition, QID translates LQM_FM into LQM_X or LQM_Z by taking the feedback measurement basis obtained in LMU (condition_checker). In Fig. 6, we omit QID due to its simple microarchitecture and functionality.

3.2.2 Patch decode unit (PDU). PDU (Fig. 6(a)) translates the logical-qubit level target lists (i.e., LQ_list, Pauli_list, Mreg_list) into patch-level lists (i.e., pch_list, pch_Pauli_list, pch_Mreg_list). For this purpose, PDU keeps logical-qubit to patch-location mappings in the pch_maptable. With the mappings, pch_decoder generates the patch-level lists and forwards them to PIU.

3.2.3 Patch information unit (PIU). For the lattice surgery, PIU dynamically updates surface-code patches' information and forwards the information to other hardware units (i.e., PSU, EDU, LMU). We first briefly introduce the patch information types and then explain how PIU updates and forwards the information.

Patch information. PIU manages two types of patch information: static and dynamic information. We explain each patch information type with Table 2, which shows the simplified example for patch_3, patch_8, and patch_9 in Fig. 4.

(1) Static patch information. The static patch information (pchinfo_static) mainly consists of pch_type, Z_boundary, and X_boundary. The pch_type indicates the logical-qubit mapping and the mapped logical qubit's initialization type. For example, 'mapped, $|0\rangle$' in patch_3 indicates that a logical qubit is mapped and initialized with $|0\rangle$. Note that Z or X_boundary is the location of each boundary, which is used to identify the target physical qubits for Merge, Split, or LQM operations.

(2) Dynamic patch information. The dynamic patch information (pchinfo_dynamic) mainly consists of ESM type for patch boundaries (i.e., ESM_left~bottom), ESM_on, and merge_on. The ESM_left~bottom provide the information to find exact target physical qubits for the ESM. For example, ESM_left=Z indicates that we should apply the ESM only to the Z-ancilla among the ancilla qubits on the left patch boundary. Meanwhile, the ESM_on and merge_on indicate whether the patch participates in the ESM and the Merge or not, respectively. The arrow in the table represents the change of dynamic patch information from Fig. 4(b) to Fig. 4(c).

Patch information update. With the MERGE_INFO and SPLIT_INFO instructions, PIU updates the target patches' dynamic information, for one patch per cycle. For example, for the MERGE_INFO, PIU updates the information based on the input pch_list and pch_Pauli_list. First, with the pch_list, PIU sets the patch-level list of merge_on (merge_on_list) and pch_indexer generates the target patch's address one by one (pch_indexer). Next, based on the adjacent patches' Pauli operators (written in pch_Pauli_list), pchdyn_decoder generates new dynamic information for the merged patch. In this step, the pchdyn_decoder also takes the target patch's static information as its input, which is obtained from the pchinfo_static_ram. Finally, the generated dynamic information is written to the pchinfo_dynamic_ram for the same pchidx. For every clock cycle, PIU changes the pchidx and iterates the above steps until every target patch's information is updated. Note that ESM_on_list is also updated during this process.

Patch information forwarding. For other instructions, PIU forwards the target patches' information (pchinfo) to PSU, EDU, or LMU. For this purpose, the pch_indexer iteratively generates the pchidx based on the target instruction type (e.g., ESM_on_list for RUN_ESM, merge_on_list for PPM_INTERPRET). Next, PIU reads both static and dynamic information with the pchidx (pchinfo) and forwards them to the target unit (e.g., PSU for RUN_ESM, LMU for PPM_INTERPRET). PIU repeats the above process until every target patch's information is forwarded. Note that the pchidx and the corresponding pch_Mreg are also concatenated and forwarded with the pchinfo for later use in PSU and LMU.

3.2.4 Physical schedule unit (PSU). For the target logical instruction, PSU (Fig. 6(c)) schedules corresponding physical-qubit level instructions (i.e., codeword) to the target physical qubits. In our context, the scheduling indicates the exact spatial allocation of target operations rather than the temporal management for them. To support dynamically changing target instruction and its target qubits, PSU utilizes two types of shift register memories (i.e., cwdNtime_srmem, pchinfo_srmem) and an array of mask_generator. Note that we adopt the shift register memories to iteratively generate the fixed-order data sequences with a low hardware cost.

First, from the cwdNtime_srmem, PSU generates the target codeword-cycle_time pair and broadcasts the codeword to the array of per-physical-qubit AND gates (AND array). The cwdNtime_srmem holds three series of codeword-cycle_time pairs which indexed by the target logical operation type (i.e., RESM_cwdNtime for RUN_ESM, INIT_cwdNtime for LQI or INIT_INTMD). To perform the target operation, PSU iteratively shifts the cwdNtime_srmem to schedule all the codeword-cycle_time pairs in the corresponding shift register.

At the same time, the pchinfo_srmem forwards target patches' information to the mask_generator array. To run the control processor without unnecessary stalls, we design pchinfo_srmem with the multiple read ports and double-buffer architecture. The double buffer interchangeably utilizes two separate memories to simultaneously perform the read/write operations. That is, the pchinfo_srmem can provide the patch information while filling an empty shift register with the next target information.

Next, with the provided patch information, the mask_generator array generates the mask bits, transfers the masks to the AND array, and determines whether to schedule the codeword for the target qubits or not. As the target qubit's location inside the patch (e.g., left~bottom boundary) is necessary for the accurate scheduling, each mask_generator also takes this information (from qubit_location_counter) as its input. In our PSU design, we can adjust the number of mask_generators with the demultiplexer, which routes the generated mask bits to the correct target qubits (pchidx in the patch information and qubit_location_counter bits are used for the routing). When the number of mask_generator is lower than the number of target physical qubits, PSU iteratively accumulates the masked codewords to the cwd_array register, while changing the qubit_location_counter. After finishing the scheduling for a codeword, PSU forwards the cwd_array and the cycle_time to TCU.

3.2.5 Time control unit (TCU). TCU (Fig. 6(d)) takes the codeword array from PSU and sends it to QC interface at the accurate timing. For this purpose, TCU utilizes two types of FIFO buffers (i.e., cwd_buffer, timing_buffer) and a counter called timing_counter.

First, when the FIFO buffers are not full, TCU pushes the incoming codeword array to the per-physical-qubit cwd_buffers. At the same time, the corresponding cycle_time is also pushed to the global timing_buffer whose head tracks the preceding codeword's execution time. For each time step, all cwd_buffers take the same codeword because PSU serves only one logical operation at once. Therefore, in our TCU, all FIFO buffers are globally controlled by the buffer_control and share the same wr_ptr and rd_ptr registers.

Next, with the timing_counter, TCU pops the consecutive codeword arrays without idle time. For example, when the preceding codeword array is sent to the QC interface, TCU starts to increase the timing_counter's value. If the timing_counter's value reaches the cycle_time in the timing_buffer's head (i.e., execution time of the preceding codeword), all cwd_buffers pop their codewords and send them to the QC interface. At the same time, the timing_counter's value is reset to zero and repeats the above process for the subsequent codeword arrays.

3.2.6 Error decode unit (EDU). EDU identifies the types and locations of errors that occurred during the ESM. Fig. 6(g) shows our baseline microarchitecture following the state-of-the-art EDU [69]. As explained in Section 2.2.1, the EDU baseline consists of the array of per-ancilla-qubit EDU cells.

First, when EDU receives the ESM results from QC interface, EDU takes them into the ESM_srmem and starts the error decoding by giving a token to the first EDU cell. Next, the token is continuously forwarded to the adjacent EDU cell (i.e., Round-robin token setup) until an EDU cell with the non-trivial syndrome (i.e., value '1' in ESM_srmem) receives the token. With the token allocation, the global match signal is broadcast to all EDU cells and their states are determined (state_machine) based on the existence of token and non-trivial syndrome (e.g.,'spike-source' state for the cell with non-trivial syndromes but not with the token. Based on the state (state_reg) and location, dir_logic manages each EDU cell's spike-forwarding direction (spike_dir). Finally, the spike_logic forwards (or generates) the spikes following the spike_dir. With this microarchitecture, EDU repeatedly finds error chains (Section 2.2.1) and accumulates the identified errors to the error_array register.

To accurately support the lattice surgery, we modify and extend the previous EDU microarchitecture [69]. In the lattice surgery, ESM operations are applied to the ancilla qubits or not based on their dynamic patch information Section 3.2.3. Therefore, we add the patch-information buffer (pchinfo_buffer) and modify the state_machine to take the dynamic patch information as its input. Note that the previous design [69] cannot support the lattice surgery due to the lack of dynamic ESM consideration.

3.2.7 Pauli frame unit (PFU). PFU (Fig. 6(f)) tracks the errors with the Pauli frame (PF) registers and provides them to LMU for the virtual error correction (Section 2.2.1). For this purpose, PFU consists of an array of per-data-qubit pf_unit, which continuously updates the PF with the identified errors (error_array) and the codewords from EDU and TCU, respectively.

First, when PFU receives the error_array from EDU, Pauli_updater updates each data qubit's PF based on the input error and the current PF value. Next, PFU again updates the PF based on the merged_cwd register's value. If other physical operations are applied to a data qubit during the error decoding step, the effective error type can be different from the decoded error type (e.g., X error followed by H gate becomes Z error). To handle this situation, cwd_merger accumulates incoming codewords to merged_cwd and pushes it to the Pauli_updater after the PF is updated with the decoded error.

3.2.8 Logical measure unit (LMU). LMU (Fig. 6(e)) derives the logical-qubit level measurements based on the physical qubit measurements (i.e., LQM, PPM_INTERPRET). At the same time, LMU (1) performs virtual error correction, (2) tracks byproduct PPR$_{\frac{\pi}{2}}$ (BYP), and (3) manages the conditions for the feedback measurement and the byproduct generation (red line and blue line in Fig. 4(a)).

Interpretation with virtual error correction. For the target instruction (LQM or PPM_INTERPRET), LMU derives the logical-level measurements (i.e., interpretation) while performing the virtual error correction. For this purpose, LMU takes the Pauli frame values (PF) as its input as well as the ancilla/data qubit measurements (pf/aqmeas/dqmeas_array). Next, in a patch-by-patch manner, LMU identifies the target qubits (e.g., data qubits on the X-boundary for LQM_X) based on the patch information (from pch-info_srmem) and calculates the product of their measurements (selective_product_unit). At the same time, the selective_product_unit performs the error correction by flipping the product of measurements based on the number of X-error records in the corresponding PFs. The calculated patch-wise interpretation is accumulated to its destination (indexed by pch_Mreg) in initial_meas_ram.

Byproduct tracking (BYP). LMU not only considers the byproduct PPR$_{\frac{\pi}{2}}$'s impact on the interpretation but also continuously accumulates the generated byproduct. LMU maintains the byproduct register to accumulate the Pauli products of PPR$_{\frac{\pi}{2}}$. By comparing this with the target instruction's Pauli_list (from instinfo_srmem), LMU flips the value in initial_meas_ram and derives the final interpretation (final_meas_interpret). After finishing all the four logical measurements in PPR$_{\frac{\pi}{8}}$, LMU checks bp_gen signal, takes XOR for the target PPR$_{\frac{\pi}{8}}$'s Pauli product (Pauli_list_reg) and the byproduct, and updates the byproduct register.

Condition management. Based on the final interpretation of the logical measurements in PPR$_{\frac{\pi}{8}}$, LMU identifies the measurement basis of LQM_FM (fm_basis) and the generation of byproduct PPR$_{\frac{\pi}{2}}$ (bp_gen). For these signals, LMU stores the target instruction's final interpretation to logical_meas_ram with its condition_flag bits (included in Meas_flag). Then, the condition checker takes the final results and Meas_flag as its inputs and generates fm_basis and bp_gen bits. The fm_basis and bp_gen are forwarded to QID and the LMU's byproduct register, respectively.

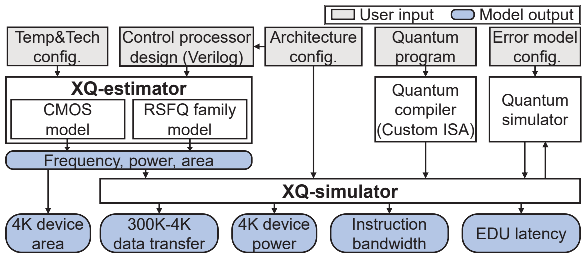

Figure 7: Our simulation framework (XQsim) overview

4 XQSIM: QUANTUM CONTROL PROCESSOR SIMULATION FRAMEWORK

To evaluate the scalability of various control processor architectures, we need a reliable simulation framework which can explore an extensive design space including microarchitecture, operating temperature, and device technology. Therefore, on top of our microarchitecture, we build XQsim, a cross-technology control processor simulation framework. Fig. 7 shows the overview of XQsim with two simulation engines: XQ-estimator and XQ-simulator. First, XQ-estimator takes the target control processor design (i.e., Verilog), temperature, and technology as inputs and derives the frequency, power, and area of the target control processor. Next, based on the obtained frequency, power, and area information, XQ-simulator reports the achievable qubit-scale and scalability bottleneck. In this section, we explain each engine in detail.

4.1 XQ-estimator

To cover various device technologies and temperatures, we model the control processors built with RSFQ logic families (i.e., RSFQ, ERSFQ) and CMOS devices operating at 300K or 4K. We first briefly introduce each temperature-device candidate and then XQ-estimator's modeling with thorough validation.

4.1.1 Temperature and device technology candidates. XQ-estimator supports four temperature and device candidates as follows.

300K CMOS. CMOS device running at 300K is mature and standard device technology. As today's quantum control processor is built with 300K CMOS, we cover 300K CMOS in our modeling.

4K CMOS. As a promising way to improve the system's scalability, researchers are actively trying to utilize cryogenic CMOS devices operating at 4K [3, 53, 70]. Therefore, our model supports 4K CMOS technology by following this recent trend.

4K RSFQ. Along with the 4K CMOS technology, superconductor Single-Flux-Quantum (SFQ) logic family also has been recognized as a highly promising solution for maximizing an in-fridge (i.e., 4K) control processor's scalability [32, 48, 64]. As Rapid SFQ (RSFQ) [44] is the most practical and mature SFQ technology today, we include 4K RSFQ technology in our model.

4K ERSFQ. Energy-efficient RSFQ (ERSFQ) [36] is an advanced technology over the RSFQ with zero static power consumption. As the most critical limitation of RSFQ is its huge static power, our 4K ERSFQ model can explore the future potential of the RSFQ family.

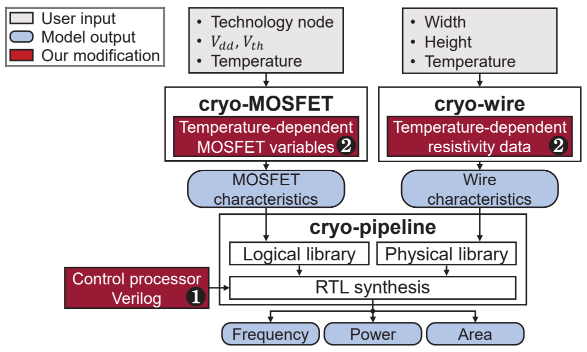

Figure 8: 300K & 4K CMOS modeling overview

4.1.2 300K & 4K CMOS modeling. We model 300K and 4K CMOS quantum control processors on top of the state-of-the-art cryogenic CMOS-modeling framework, CC-Model [7].

CC-Model overview. Fig. 8 shows the overview of CC-Model which consists of MOSFET (cryo-MOSFET), wire (cryo-wire), and processor model (cryo-pipeline). First, cryo-MOSFET is a validated cryogenic MOSFET model which takes the model card and the target V_{dd} and V_{th} as its inputs, and then derives the MOSFET characteristics (i.e., I_{on}, I_{leak}) at the target temperature as outputs. Second, cryo-wire takes a metal-layer specification (e.g., width, height) as an input and generates the wire resistivity at cryogenic temperatures as outputs. Third, for the given processor design, cryo-pipeline can predict its frequency, power, and area at low temperatures. For the purpose, cryo-pipeline synthesizes a target Verilog design with the commercial synthesis tool (i.e., Design Compiler Topographical Mode [62]) while using the logical and physical libraries generated by cryo-MOSFET and cryo-wire, respectively.

Modifications on CC-Model. As CC-Model only covers CPU designs running at 300K-to-77K range, we extend CC-Model to support our target designs. With our two modifications, CC-Model can predict the frequency, power, and area of CMOS-based quantum control processors operating at 300K and 4K.

(1) Circuit design extension. To extend CC-Model's coverage to control processors, we modify CC-Model to take a control-processor Verilog as its target design (❶). The modification enables us to model the target quantum control processor.

(2) Target-temperature extension. We extend CC-Model's temperature coverage by adjusting cryo-MOSFET's three fabrication-related and temperature-dependent MOSFET variables (i.e., carrier mobility [4], saturation velocity [30], and threshold voltage [5]) based on recent cryogenic MOSFET measurement at 4K. In addition, we also extend cryo-wire's resistivity model coverage to 4K range, by adopting previous measurement data from Intel [55] (❷). Note that we thoroughly follow the same modeling methodology of previous works [7, 41] while using the real-measurement data.

4.1.3 RSFQ & ERSFQ modeling. We model 4K RSFQ and ERSFQ-based control processor with the methodology shown in Fig. 9. Our RSFQ family model first generates the RSFQ-specific gate-level netlist and then derives frequency, power, and area based on the netlist. Note that we adopt the same modeling approach for both RSFQ and ERSFQ technologies because the Josephson junction (i.e., JJ) biasing scheme is the only difference between them.

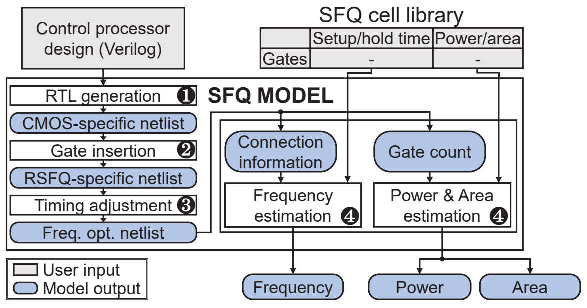

Figure 9: RSFQ & ERSFQ modeling overview

We provide a detailed description for each step as follows.

❶ **Gate-level RTL generation.** As the RSFQ family has a similar logic gate set with CMOS (e.g., OR, AND), we can generate the target design's RSFQ gate netlist starting from the CMOS-based netlist. Thus, we first make a CMOS-specific netlist by using the input processor design and existing synthesis tool (i.e., Yosys [71]).

❷ **SFQ-specific gate insertion.** To convert the CMOS-specific netlist to the RSFQ-specific netlist, we insert D flip-flops (DFF) and splitter trees into the generated gate netlist. First, due to the gate-level pipelined nature of RSFQ circuits, we should balance the datapath depth of all input data signals for every RSFQ gate. For this purpose, we first figure out the depth of each logic gate by running the Depth-First-Search (DFS) algorithm and then appropriately insert DFF gates to adjust the pipeline depth.

Next, due to the limited fanout of RSFQ gates, an RSFQ circuit needs splitter trees to deliver the clock and gate-output signals to multiple gates. Therefore, we analyze the required fanout of the clock tree for each pipeline depth and then insert the splitter trees built with fanout-2 splitters. In addition, we insert splitter trees for every logic gate in a similar manner.

As a result of the gate insertions, we generate the functionally correct gate-level netlist of the target RSFQ circuits.

❸ **SFQ-specific timing adjustment.** With the functionally-correct RSFQ-specific gate netlist, we optimize target circuit's frequency by appending RSFQ wire components (i.e., passive transmission line (PTL), buffer). To maximize an RSFQ circuit's frequency, we should minimize the timing difference between the input clock and data signals for each gate (i.e., δt in Eq. (1)) while satisfying various timing constraints (e.g., setup time, hold time, input-to-input time). Therefore, we appropriately add the wire components into the clock and data lines and thus finally obtain a frequency-optimized netlist for the target RSFQ circuit.

❹ **Frequency, power, and area estimation.** We estimate the frequency, power, and area of the RSFQ design by analyzing its frequency-optimized netlist. For the ERSFQ modeling, we use the RSFQ cell library because they can share the same logical gate set. Specifically, based on the RSFQ library, we assume the same timing parameters and area, zero static power, and twice higher dynamic energy for the ERSFQ gates [29].

$$\begin{aligned} f_{max,circuit} &= 1/\mathrm{Max}(CCT_{min,gate}) \\ CCT_{min,gate} &= SetupTime + \mathrm{Max}(HoldTime, \delta t) \end{aligned} \tag{1}$$

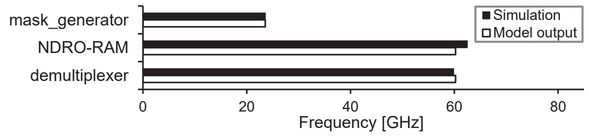

Figure 10: XQ-estimator validation with MITLL library

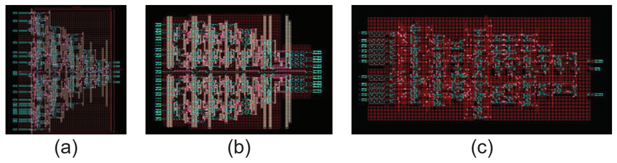

Figure 11: Layout of (a) EDU_cell_spike_logic, (b) EDU_cell_dir_logic, and (c) pf_unit designed with AIST process library

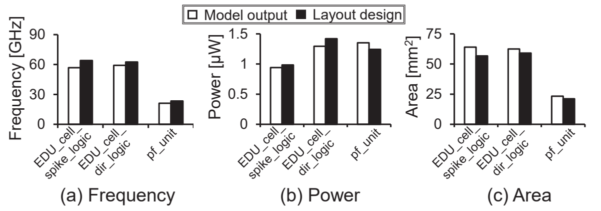

Figure 12: XQ-estimator validation with AIST layout designs

(1) Maximum-frequency estimation. To find the maximum frequency of the entire circuit ($f_{max,circuit}$), we first derive the minimum clock cycle time of every gate ($CCT_{min,gate}$) and then take the reciprocal of the maximum $CCT_{min,gate}$ value as shown in Eq. (1) [29], where $SetupTime$, $HoldTime$, and δt are the setup time, hold time, and the timing difference between the clock and data signals for each gate, respectively. As we already minimize δt at ❸, we can easily derive the maximized clock frequency of the entire circuit.

(2) Power and area estimation. For power and area estimation, we multiply the gate counts of each gate (from gate-level netlist) with the gate-specific power and area overhead (from gate library), and then sum up the multiplied results.

4.1.4 *Validation: XQ-estimator.*

We validate XQ-estimator by comparing the RSFQ model's estimation with the results from timing-accurate simulation and post-layout analysis. For the validation, we use the RSFQ gate libraries from two major fabrication processes (i.e., MITLL [59] and AIST [72]).

Validation with MITLL library. With the MITLL library, we validate our model's frequency prediction by comparing it with the timing-accurate RTL simulation. For the simulation, we generate gate-level RTL of the target RSFQ circuit by using the frequency-optimized netlist (❸ in Fig. 9) and hardware description language (HDL) model in the library. As the HDL model includes accurate timing parameters for every gate, we can measure the maximum frequency by repeating simulations while reducing the clock period.

Fig. 10 shows the validation results for various circuits used in PSU and TCU (i.e., mask_generator (50,782 JJs), NDRO-RAM (3,003 JJs), demultiplexer (3,368 JJs)). We intentionally target these circuits as they will be built with the RSFQ family in our final control processor architecture. The figure shows that XQ-estimator accurately predicts the frequency with the 3.7% of maximum error.

Table 3: XQ-simulator validation specification and result

Benchmark specification					
Benchmark	PPR$_{\frac{\pi}{8}}(Z_3Z_4Z_5)$	PPR$_{\frac{\pi}{8}}(Y_3X_4Z_5X_6)$	PPR$_{\frac{\pi}{8}}(Y_3Y_4Z_5Z_6)$	QFT	QAOA
# logical qubits	3	4	4	2	4
# patches	15	25	25	15	25
Code distance (d)	3	3	3	5	5
# physical qubits	480	800	800	1080	1800
Validation result					
dTV (Qiskit v.s. XQsim)	0.0351	0.0533	0.0455	0.013	0.0479

Validation with AIST library. With the AIST process library, we validate XQ-estimator's accuracy by comparing it with the post-layout analysis. Fig. 11 shows the circuit layouts designed with the AIST library for our validation (EDU_cell_spike_logic (1,381 JJs), EDU_cell_dir_logic (1,915 JJs) and pf_unit (2,376 JJs)). We select EDU_cell_spike_logic and EDU_cell_dir_logic because they also will be located inside RSFQ-based EDU in our final architecture. Fig. 12 shows the validation result which clearly reveals that our RSFQ model accurately estimates the frequency, power, and area with the maximum errors of 12.8%, 8.9%, and 6.3%, respectively.

4.2 XQ-simulator

4.2.1 *Cycle-accurate architecture simulation.*

XQ-simulator takes a quantum binary compiled with our custom ISA, architecture configuration (e.g., code distance, temperature mappings for each hardware unit), and XQ-estimator's output as its input, runs a cycle-accurate simulation, and reports the four scalability metrics.

First, XQ-simulator tracks all the inter-unit data transfers and calculates the 300K-4K data transfer rate based on the temperature mapping information. Second, by tracking the activated cycles for each hardware unit, our simulator derives its power consumption at 4K. Third, XQ-simulator obtains the instruction bandwidth requirement by tracking the timing buffer and the codeword buffers in TCU. Fourth, XQ-simulator derives a realistic error decoding latency by forwarding TCU's output codewords to a quantum simulator (e.g., Stim [18]), performing noisy simulations with the input error model, and running an error decoding algorithm with the simulated error syndromes.

4.2.2 *Validation: XQ-simulator.*

We validate the XQ-simulator's functional correctness by comparing the results of our physical-qubit-level simulation with Qiskit's logical-qubit-level simulation [1]. First, for the physical-level simulation, we forward XQ-simulator's output codewords to Stim [18] and run noisy simulation with the Pauli quantum error model [68]. On the other hand, for the logical-level simulation, we run the same workloads with Qiskit without any errors. We run 2048-shot simulations for three PPR circuits (i.e., PPR$_{\frac{\pi}{8}}(Z_3Z_4Z_5)$, PPR$_{\frac{\pi}{8}}(Y_3X_4Z_5X_6)$, PPR$_{\frac{\pi}{8}}(Y_3Y_4Z_5Z_6)$) and two representative benchmarks (i.e., Quantum Fourier Transform (QFT), Quantum Approximate Optimization Algorithm (QAOA)[13]).

Table 3 shows the validation results with the total variation distance (dTV) between the two simulations. As shown in Table 3, XQ-simulator reports highly accurate results where the maximum dTV value is only 0.0533 for PPR$_{\frac{\pi}{8}}(Y_3X_4Z_5X_6)$. Note that our validation result is the first to show the functional correctness of the fully-implemented fault-tolerant quantum control processor with the large-scale quantum simulations (480~1800 physical qubits).

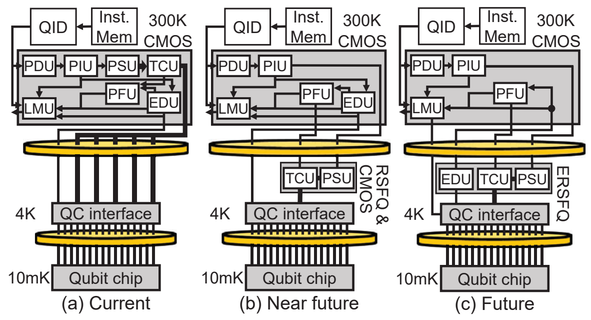

Figure 13: Target quantum computer systems; (a) Current system, (b) Near-future system, and (c) Future system

Table 4: Scalability analysis setup

Error decoder parameters		
Physical error rate	Code distance	Baseline error decoder
0.10% [20]	15 [20]	QECOOL [69]
Physical quantum gate latency		
Single-qubit gate	Two-qubit gate	Measurement
14ns [9]	26ns [9]	600ns [9]
Refrigeration and wiring		
4K power budget	4K area budget	300K-4K cable
1.5W [39]	620cm^2 [6, 39]	Digital coaxial cable (10Gbps, 31mW [21])
Clock frequency of quantum control processors		
300K CMOS	4K CMOS	RSFQ & ERSFQ
1.5GHz	1.5GHz	21.0GHz

5 FUTURE DIRECTION FOR DESIGNING SCALABLE CONTROL PROCESSOR

In this section, driven by XQsim, we provide the future directions for designing a 10+K qubit-scale quantum control processor. For the purpose, we analyze various control processors' scalability bottlenecks, provide effective design guidelines and optimizations to resolve them, and show the improved scalability. Note that our analysis covers the scalability constraints of control processors only, and thus our qubit-scale results can change if we also consider other scaling challenges (which will be discussed in Section 6.1).

5.1 Summary of our target control systems

We first introduce our control systems to be studied. Fig. 13 summarizes our quantum control processors evolving in chronological order: (1) current system with 300K CMOS, (2) near-future system with RSFQ and 4K CMOS, and (3) future system with ERSFQ.

Current system with 300K CMOS. In the recent NISQ system, control processors are designed with 300K CMOS technology. Therefore, we start our analysis from the fault-tolerant control processor built with 300K CMOS. Even though there is no currently available fault-tolerant control processor, we name this system "current system" because it is based on the current 300K CMOS.

Near-future system with RSFQ and 4K CMOS. As the 300K-4K data transfer becomes the bottleneck in current systems, our control processor is evolved to utilize RSFQ and 4K CMOS technologies at 4K domain. We name this system "near-future system" as these technologies are currently available and mature enough.

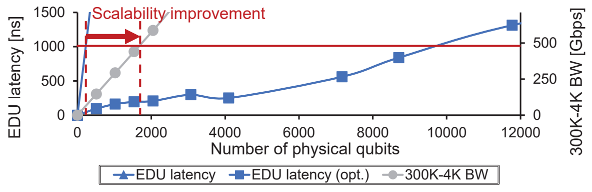

Figure 14: Scalability of current system with 300K CMOS

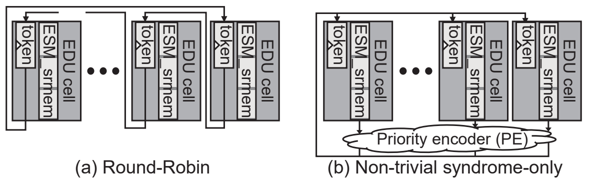

Figure 15: EDU token setup schemes; (a) Round-Robin and (b) Non-trivial syndrome-only setup

Future system with ERSFQ. As the 4K power consumption or error decoding latency becomes the bottleneck in the near-future system, our quantum control processor is evolved to adopt ultra-fast and low-power ERSFQ technology. However, as the ERSFQ technology is still immature, we name this system "future system".

5.2 Scalability analysis setup

For our scalability analysis, we run random PPR$_{\frac{\pi}{8}}$ while increasing the number of logical qubits. To set the constraint for each scalability metric, we adopt the widely-used values from the recent sources as specified in Table 4. For the technology libraries of our model, we intentionally use the open-source FreePDK 45nm [61] and MITLL libraries as we target to release our tool as an open-source. We also specify the clock frequency of our control processors in Table 4.

5.3 Current system with 300K CMOS

5.3.1 Scalability bottleneck. We first analyze the scalability bottleneck of the current system with 300K CMOS technology. Our analysis shows that 300K CMOS system's scalability is highly limited due to the long error decoding latency. Fig. 14 shows the error decoding latency and the 300K-4K data transfer with their scalability constraints (i.e., red horizontal line). Even though the 300K CMOS control processor has a potential to support 1,700 qubits, it cannot support even 250 qubits due to the slow error decoding. Therefore, we should accelerate the error decoding to maximize the 300K CMOS control processor's scalability.

5.3.2 Optimization #1 - EDU acceleration. To resolve the bottleneck, we propose *non-trivial syndrome-only token setup*. Our key insight is that the Round-Robin (RR) token setup of the state-of-the-art EDU baseline [69] wastes a huge amount of cycles because it consumes one cycle for every qubit to shift the token and check the existence of non-trivial error syndrome (Fig. 15(a)).

Based on the insight, we accelerate the error decoding by skipping all the wasteful token checkups. We modify the baseline EDU by adding a priority encoder to directly allocate a token to the next EDU cell with a non-trivial syndrome (Fig. 15(b)). This implementation saves a thousand of cycles per non-trivial syndrome.

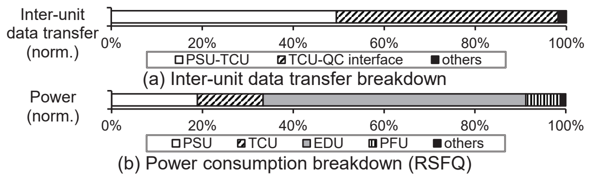

Figure 16: Unit-level breakdown of (a) inter-unit data transfer and (b) power consumption (RSFQ)

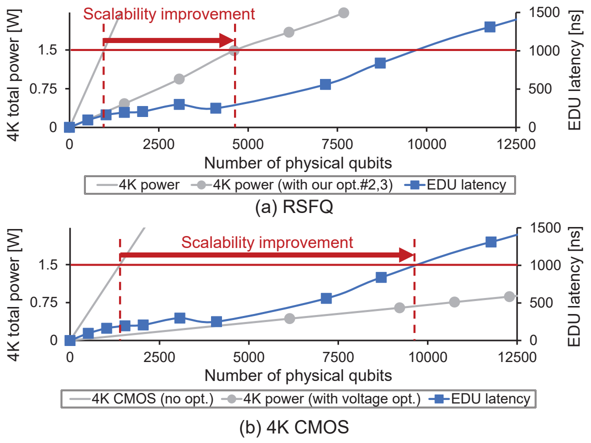

Figure 17: Scalability of near-future system with (a) RSFQ and (b) 4K CMOS

5.3.3 Improved scalability. With the EDU speed-up, the scaling limit from the error decoding latency greatly increases to 9,800 qubits (Fig. 14). However, the 300K-4K data transfer, which was hidden before, now limits the 300K system's scalability to 1,700 qubits. Therefore, we should minimize the 300K-4K data transfer to further increase the scalability. As running a control processor at 4K is the most effective way to minimize the data transfer, we decide to utilize the 4K domain in the near-future system.

5.4 Near-future system with RSFQ & 4K CMOS

We now explore the direction to maximize the control processor's scalability by using RSFQ and 4K CMOS technologies, as they are the major and mature technologies for computing at 4K.

5.4.1 Guideline #1 - Move only PSU and TCU to 4K domain. Through an in-depth analysis, we propose to move only PSU and TCU to the 4K stage as our first guideline. Fig. 16 clarifies our guideline with the unit-level breakdown of inter-unit data transfer and power consumption (for RSFQ technology). First, PSU and TCU dominate the inter-unit data transfer (98.1% in Fig. 16(a)), and thus operating PSU and TCU at 4K is enough to resolve the 300K-4K data transfer bottleneck. Second, other hardware units (e.g., EDU, PFU) consume much more power (65.4%) than PSU and TCU (33.4%), which can significantly limit the scalability due to the excessive 4K power consumption. Therefore, we design the near-future control processor architecture by placing PSU and TCU at 4K while locating other units at 300K (Fig. 13(b)).

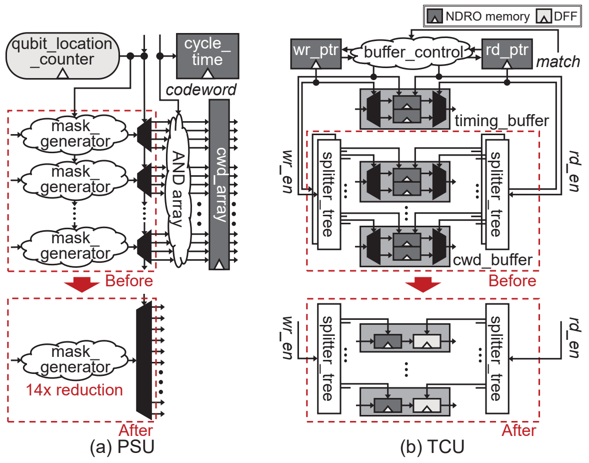

Figure 18: Microarchitecture optimizations for the near-future system with RSFQ; (a) PSU and (b) TCU

5.4.2 Scalability bottleneck. We analyze the scalability bottleneck for both RSFQ and 4K CMOS technologies. With our analysis, we observe that just moving PSU and TCU to the 4K stage cannot improve the scalability due to the huge 4K power consumption. Even though our near-future architecture resolves the previous 300K-4K data transfer bottleneck, Fig. 17 shows that 4K power consumption significantly limits the scalability of both RSFQ and 4K-CMOS system to 970 and 1,400 qubits, respectively (even lower than the current system's limit, 1,700 qubits). Therefore, we should minimize the 4K power consumption. As the possible directions, we explore microarchitecture optimizations for the RSFQ-based system and device-level optimization for the 4K CMOS-based system.

5.4.3 Optimizations #2 & #3 - Low-power RSFQ PSU & TCU design. To reduce the 4K power consumption, we present microarchitecture optimizations for the RSFQ-based PSU and TCU. First, we reduce the number of mask generators in PSU by sharing them for more physical qubits. The key insight is that if the PSU becomes much faster with the RSFQ technology, a single mask generator can produce the masks for much more physical qubits while satisfying the bandwidth requirement. Fig. 18(a) shows the microarchitectural changes with our idea where a single mask generator is shared by 14 times more physical qubits with the comparable demultiplexer overhead. With the 14 times faster operation, our optimization reduces the PSU's power consumption by 5.5 times.

Second, we replace the TCU's FIFO buffers with simpler buffers which consist of a single NDRO-based buffer and its following DFFs. Through our microarchitecture implementation, we identify that RSFQ-based TCU suffers from the huge multiplexer and demultiplexer overhead even with the two-entry FIFO buffers. At the same time, we also observe that only one buffer entry is enough for the exact timing control in the FTQC system. Based on the observations, we maintain one NDRO-based buffer entry per physical qubit and remove all the demultiplexers and multiplexers. In addition, we control the codeword transfer timing by directly using the TCU's timing-match signal as the subsequent DFFs' clock signal (Fig. 18(b)). As a result, the TCU's power consumption is reduced by 4.0 times with our buffer simplification.

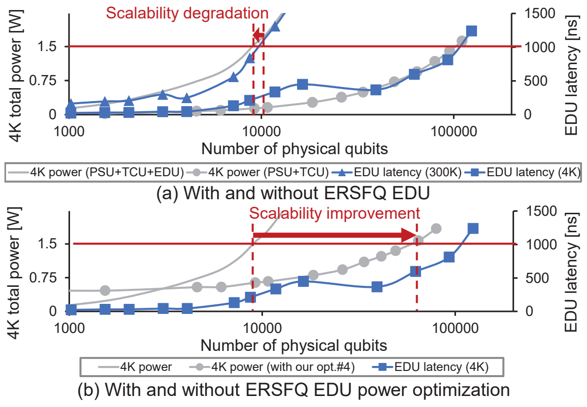

Figure 19: Scalability of future system; With and without (a) ERSFQ EDU and (b) EDU power optimization

5.4.4 Power-oriented voltage scaling for 4K CMOS. With the 4K CMOS technology, we show the potential power reduction through the voltage scaling enabled at 4K. As shown in the previous studies [7, 42], the major benefit of cryogenic CMOS computing is the nearly eliminated leakage current and correspondingly enabled threshold voltage (V_{th}) scaling. The reduced V_{th} enables further voltage scaling to maximize the performance within the same power budget (i.e., performance-oriented) or to minimize the power without the performance loss (i.e., power-oriented). For our near-future system, we adopt the power-oriented voltage scaling because its scalability bottleneck is in the 4K power consumption. As a result of the aggressive voltage scaling, the total power consumption of 4K CMOS-based PSU and TCU is reduced by 15.3 times.

5.4.5 Improved scalability. Thanks to the reduced 4K power consumption with our optimizations, the scaling limit greatly increases to 4,600 (Fig. 17(a)) and 9,800 qubits (Fig. 17(b)) in the RSFQ and 4K CMOS-based systems, respectively. However, as the achieved scalability is still lower than our target (i.e., 10+K qubits), we should further reduce the 4K power consumption or EDU latency. In this context, we decide to utilize the ERSFQ technology for our future system because the ultra-fast and low-power ERSFQ is highly promising to resolve the near future system's bottlenecks.

5.5 Future system with ERSFQ

Finally, we further improve the control processor's scalability by assuming the ERSFQ technology becomes quite mature. We start our analysis by applying ERSFQ to the near-future RSFQ-based system derived in Section 5.4.

5.5.1 Guideline #2 - Move EDU to 4K domain. Interestingly, our analysis reveals that the naive ERSFQ adoption cannot increase the scalability due to the slow error decoding again. Fig. 19(a) shows that the ERSFQ technology greatly extends the scaling limit to 102,000 qubits from the 4K-power perspective. However, due to the long error decoding latency, the future system's scalability is limited to 9,800 qubits, which is similar to that of the near-future system. Therefore, our next design direction is to improve the error decoding speed by additionally designing EDU with ERSFQ, which is 14 times faster than 300K CMOS counterpart.

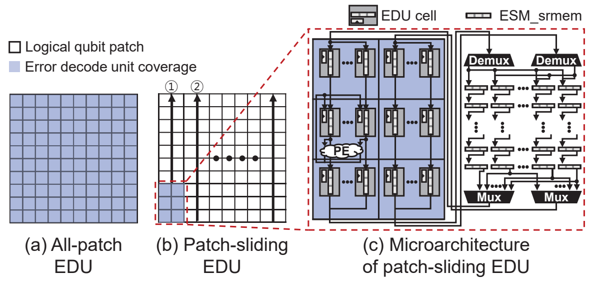

Figure 20: Error decoding process; (a) With per-qubit EDU cell and (b) With power-optimized patch-sliding scheme

5.5.2 Scalability bottleneck. Even with the ERSFQ-based EDU, the future system's scalability cannot be improved because it dissipates huge dynamic power at 4K. In Fig. 19(a) with ERSFQ EDU, the error decoding's scaling limit reaches 105,000 qubits but EDU's 4K power significantly degrades the 4K power's scaling limit down to 8,100 qubits. Therefore, to achieve meaningful scalability improvement in the future system, we should minimize the EDU's power consumption while maintaining its error decoding speed.

5.5.3 Optimization #4 - EDU power reduction. To minimize the EDU's power consumption, we propose *patch-sliding error decoding*. In our baseline EDU, every ancilla qubit has its own dedicated EDU cell (Fig. 20(a)) and thus EDU's power consumption linearly increases with the qubit scaling. However, our key insight is that we do not need such a per-qubit EDU cell because the non-trivial error syndromes can be paired within the code distance. In other words, error syndrome matching only for the adjacent patches shows totally same result as the baseline EDU.

Based on our insight, we reduce the EDU's power consumption by using a constant number of EDU cells and sharing them in a patch-sliding manner. The patch-sliding EDU first loads the error syndromes corresponding to the six surface-code patches and performs error syndrome matching by allocating the token only to the EDU cells in the left-center EDU patch (Fig. 20(c)). After finishing the matching for the loaded syndromes, the EDU slides the window (i.e., syndromes in ESM_srmem) by one patch row following the direction in Fig. 20(b) (① and then ②). At the same time, EDU takes the next two-patch syndromes with the support of the global ESM_srmems, multiplexers, and demultiplexers (Fig. 20(c)). With this scheme, we can significantly reduce the EDU's power consumption by 18.8 times while maintaining the same performance.

5.5.4 Improved scalability. With our ERSFQ-based EDU design and power optimization, we resolve both 4K-power and error-decoding bottlenecks and thus significantly extend the scaling limit to 59,000 qubits (Fig. 19(b)). The final design's ERSFQ-based unit area is within the area constraint (i.e., <620cm^2 in Table 4) even though the μm-scale technology is assumed (i.e., >5000μm^2/gate). With various SFQ technology-scaling techniques [8, 67] considered, the 4K chip area will not limit the future system's scalability. This future system study clearly shows that even the advanced device technology cannot solely improve the quantum computer's scalability without the in-depth analysis and corresponding architectural solutions.

6 DISCUSSION

6.1 Scalability challenges in QC interface

Quantum computer architects should model and design both quantum control processor and QC interface. Among the two issues, this work targeted the former challenge which has not been actively explored yet. However, as the challenges in QC interface scaling are also important ongoing research topics, we discuss the major issues and recent efforts on the QC interface side.

6.1.1 Scaling of analog microwave cable. First of all, the increasing number of analog microwave cables significantly limits the scalability of recent QC interfaces. In today's superconducting quantum computers, the room-temperature QC interface controls qubits by generating and sending microwave pulses through per-qubit coaxial cables (e.g., one drive/flux line for each qubit).

However, this approach imposes a huge thermal load on the millikelvin stage as the number of 300K-to-10mK cables increases with the qubit scale. For example, in the recent report, a state-of-the-art dilution refrigerator cannot support even more than 150 qubits when it uses commercial coaxial cables [39]. To mitigate this challenge, researchers are actively working on developing scalable interconnect technologies, which have a low thermal conductivity as well as a small cross-sectional area (e.g., superconducting cable [60], photonic link [40]). In another direction, architects are also exploring effective ways to reduce the number of required microwave cables (e.g., multiplexing a single cable for multiple qubits [54]).

6.1.2 Power consumption of QC interface. The huge power consumption of the cryogenic QC interface is also a major scaling challenge. For the superconducting (or spin) qubits, the QC interface consists of various complex CMOS-based analog circuits (e.g., DAC, ADC) to generate high-fidelity control/readout microwaves. Nowadays, architects are trying to operate this CMOS-based QC interface at the 4K stage (i.e., CryoCMOS QC interface) with the same motivation as Section 6.1.1. As a result, several prototype chips recently have been demonstrated by leading industries [3, 53, 70].

However, the CryoCMOS QC interface's scalability is still limited due to the huge power dissipation at 4K. For example, a recent fully-integrated CryoCMOS prototype chip [53] consumes around 25mW/qubit in its analog circuits, and thus it cannot be scaled to even more than 40 qubits with the 1W budget assumed. To address this issue, researchers are continuously lowering the CryoCMOS QC interface's power consumption [34] while also exploring an SFQ-based pure digital QC interface [26, 32].

6.2 Evolution of technology parameters

Our analysis is based on the current-technology parameters, and thus the detailed scalability number can vary with the technology evolution. For example, the lower-error physical qubit, higher cooling capacity of the refrigerators, and lower-power optical wire can alleviate the control processor's scalability constraints. Although we do not provide a case study for these scenarios, XQsim is useful for exploring any type of future system. Specifically, architects can analyze future systems with the aforementioned technologies by changing XQsim's input parameters (e.g., physical error rate, 4K power budget). Therefore, our tool release will help architects to explore various un-studied future system's scalability.

7 RELATED WORK

Fault-tolerant control processor. Fu et al.[16, 17] introduced the overview of the architectural unit and ISA required for the fault-tolerant quantum control processor. Tannu et al. [64], Holmes et al. [23], Riesebos et al. [58], Das et al. [10], and Ueno et al. [69] discussed the microarchitecture of the specific units. However, there has been no full microarchitecture implementation and thus no holistic scalability analysis.

SFQ and cryogenic CMOS modeling. Lee et al. [41], Byun et al. [7], and Min et al. [50, 51] developed a 77K cryogenic CMOS modeling tool and performed design space exploration for various computer devices including DRAM, cache, and processor. Zokaee et al. [73] proposed a 4K CMOS modeling tool for the DRAM. Meanwhile, Ishida et al. [29] developed an SFQ-based NPU modeling tool. However, there has been no model supporting the comparison for different cryogenic device technologies.

To the best of our knowledge, our work is the first study to develop a cross-technology modeling tool for fault-tolerant quantum control processors, and holistically analyze the scalability based on the full microarchitecture implementation.

8 CONCLUSION

In this paper, we presented XQsim, an open-source cross-technology quantum control processor simulator to accurately analyze the processors' scalability bottleneck for various device technologies and operating temperatures. For the purpose, we fully implemented the convincing control processor microarchitecture for the FTQC systems. Next, on top of the microarchitecture, we developed an architecture-level simulation framework, XQsim, and thoroughly validated the tool with post-layout analysis, timing-accurate RTL simulation, and noisy quantum simulation. By using XQsim, we provided the future directions to design a 10+K qubit quantum control processor with several design guidelines and architecture optimizations. We release XQsim as the first open-source modeling framework for fault-tolerant quantum control processors and it can contribute to initiating follow-up FTQC research.

ACKNOWLEDGMENTS

This work was supported in part by the Samsung Electronics, National Research Foundation of Korea (NRF) funded by the Korean Government under Grants NRF-2019R1A5A1027055, NRF-2021R1A2C3014131, NRF-2021M3F3A2A02037893, and NRF-2019-Global Ph.D. Fellowship Program; in part by the Institute of Information & Communications Technology Planning & Evaluation funded by the Korean Government under Grant 1711080972, 2021000853; in part by the Creative Pioneering Researchers Program through Seoul National University; and in part by the Automation and Systems Research Institute (ASRI) and Inter-university Semiconductor Research Center at Seoul National University. We also appreciate the support from JST MIRAI (JPMJMI18E1), JST PRESTO (JPMJPR2015), and JSPS KAKENHI (JP19H01105). The circuit was designed with the support by VDEC of the University of Tokyo in collaboration with Cadence Design Systems, Inc.

REFERENCES

[1] MD SAJID ANIS, Abby-Mitchell, Héctor Abraham, AduOffei, Rochisha Agarwal, Gabriele Agliardi, Merav Aharoni, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Sashwat Anagolum, Anthony-Gandon, Eli Arbel, Abraham Asfaw, Anish Athalye, Artur Avkhadiev, Carlos Azaustre, PRATHAMESH BHOLE, Abhik Banerjee, Santanu Banerjee, Will Bang, Aman Bansal, Panagiotis Barkoutsos, Ashish Barnawal, George Barron, George S. Barron, Luciano Bello, Yael Ben-Haim, M. Chandler Bennett, Daniel Bevenius, Dhruv Bhatnagar, Arjun Bhobe, Paolo Bianchini, Lev S. Bishop, Carsten Blank, Sorin Bolos, Soham Bopardikar, Samuel Bosch, Sebastian Brandhofer, Brandon, Sergey Bravyi, Nick Bronn, Bryce-Fuller, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adam Carriker, Ivan Carvalho, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Franck Chevallier, Kartik Chinda, Rathish Cholarajan, Jerry M. Chow, Spencer Churchill, CisterMoke, Christian Claus, Christian Clauss, Caleb Clothier, Romilly Cocking, Ryan Cocuzzo, Jordan Connor, Filipe Correa, Zachary Crockett, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Navaneeth D, Sean Dague, Tareq El Dandachi, Animesh N Dangwal, Jonathan Daniel, Marcus Daniels, Matthieu Dartiailh, Abdón Rodríguez Davila, Faisal Debouni, Anton Dekusar, Amol Deshmukh, Mohit Deshpande, Delton Ding, Jun Doi, Eli M. Dow, Eric Drechsler, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Grant Eberle, Amir Ebrahimi, Pieter Eendebak, Daniel Egger, ElePT, Emilio, Alberto Espiricueta, Mark Everitt, Davide Facoetti, Farida, Paco Martín Fernández, Samuele Ferracin, Davide Ferrari, Axel Hernández Ferrera, Romain Fouilland, Albert Frisch, Andreas Fuhrer, Bryce Fuller, MELVIN GEORGE, Julien Gacon, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Tanya Garg, Shelly Garion, James R. Garrison, Tim Gates, Leron Gil, Austin Gilliam, Aditya Giridharan, Juan Gomez-Mosquera, Gonzalo, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, Dani Guijo, John A. Gunnels, Harshit Gupta, Naman Gupta, Jakob M. Günther, Mikael Haglund, Isabel Haide, Ikko Hamamura, Omar Costa Hamido, Frank Harkins, Kevin Hartman, Areeq Hasan, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Junye Huang, Rolf Huisman, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Ishwor, Raban Iten, Toshinari Itoko, Alexander Ivrii, Ali Javadi, Ali Javadi-Abhari, Wahaj Javed, Qian Jianhua, Madhav Jivrajani, Kiran Johns, Scott Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, John Judge, Tal Kachmann, Akshay Kale, Naoki Kanazawa, Jessica Kane, Kang-Bae, Annanay Kapila, Anton Karazeev, Paul Kassebaum, Josh Kelso, Scott Kelso, Vismai Khanderao, Spencer King, Yuri Kobayashi, Kovi11Day, Arseny Kovyrshin, Rajiv Krishnakumar, Vivek Krishnan, Kevin Krsulich, Prasad Kumkar, Gawel Kus, Ryan LaRose, Enrique Lacal, Raphaël Lambert, Haggai Landa, John Lapeyre, Joe Latone, Scott Lawrence, Christina Lee, Gushu Li, Jake Lishman, Dennis Liu, Peng Liu, Lolcroc, Abhishek K M, Liam Madden, Yunho Maeng, Saurav Maheshkar, Kahan Majmudar, Aleksei Malyshev, Mohamed El Mandouh, Joshua Manela, Manjula, Jakub Marecek, Manoel Marques, Kunal Marwaha, Dmitri Maslov, Paweł Maszota, Dolph Mathews, Atsushi Matsuo, Farai Mazhandu, Doug McClure, Maureen McElaney, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Dekel Meirom, Corey Mendell, Thomas Metcalfe, Martin Mevissen, Andrew Meyer, Antonio Mezzacapo, Rohit Midha, Daniel Miller, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Alejandro Montanez, Gabriel Monteiro, Michael Duane Mooring, Renier Morales, Niall Moran, David Morcuende, Seif Mostafa, Mario Motta, Romain Moyard, Prakash Murali, Jan Müggenburg, Tristan NEMOZ, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Aziz Ngoueya, Johan Nicander, Nick-Singstock, Pradeep Niroula, Hassi Norlen, NuoWenLei, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Tamiya Onodera, Raul Otaolea, Steven Oud, Dan Padilha, Hanhee Paik, Soham Pal, Yuchen Pang, Ashish Panigrahi, Vincent R. Pascuzzi, Simone Perriello, Eric Peterson, Anna Phan, Kuba Pilch, Francesco Piro, Marco Pistoia, Christophe Piveteau, Julia Plewa, Pierre Pocreau, Alejandro Pozas-Kerstjens, Rafał Pracht, Milos Prokop, Viktor Prutyanov, Sumit Puri, Daniel Puzzuoli, Jesús Pérez, Quant02, Quintiii, Rafey Iqbal Rahman, Arun Raja, Roshan Rajeev, Isha Rajput, Nipun Ramagiri, Anirudh Rao, Rudy Raymond, Oliver Reardon-Smith, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Matt Riedemann, Rietesh, Drew Risinger, Marcello La Rocca, Diego M. Rodríguez, RohithKarur, Ben Rosand, Max Rossmannek, Mingi Ryu, Tharrmashastha SAPV, Nahum Rosa Cruz Sa, Arijit Saha, Abdullah Ash-Saki, Sankalp Sanand, Martin Sandberg, Hirmay Sandesara, Ritvik Sapra, Hayk Sargsyan, Aniruddha Sarkar, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Mark Schulterbrandt, Joachim Schwarm, James Seaward, Sergi, Ismael Faro Sertage, Kanav Setia, Freya Shah, Nathan Shammah, Rohan Sharma, Yunong Shi, Jonathan Shoemaker, Adenilton Silva, Andrea Simonetto, Deeksha Singh, Divyanshu Singh, Parmeet Singh, Phattharaporn Singkanipa, Yukio Siraichi, Siri, Jesús Sistos, Iskandar Sitdikov, Seyon Sivarajah, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, Igor Sokolov, Vicente P. Soloviev, SooluThomas, Starfish, Dominik Steenken, Matt Stypulkoski, Adrien Suau, Shaojun Sun, Kevin J. Sung, Makoto Suwama, Oskar Słowik, Hitomi Takahashi, Tanvesh Takawale, Ivano Tavernelli, Charles Taylor, Pete Taylour, Soolu Thomas, Kevin Tian, Mathieu Tillet, Maddy Tod, Miroslav Tomasik, Caroline Tornow, Enrique de la Torre, Juan Luis Sánchez Toural, Kenso Trabing, Matthew Treinish, Dimitar Trenev, TrishaPe, Felix Truger, Georgios Tsilimigkounakis, Davindra Tulsi, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Adish Vartak, Almudena Carrera Vazquez, Prajjwal Vijaywargiya, Victor Villar, Bhargav Vishnu, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Johannes Weidenfeller, Rafal Wieczorek, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, WinterSoldier, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Steve Wood, James Wootton, Matt Wright, Lucy Xing, Jintao YU, Bo Yang, Unchun Yang, Jimmy Yao, Daniyar Yeralin, Ryota Yonekura, David Yonge-Mallo, Ryuhei Yoshida, Richard Young, Jessie Yu, Lebin Yu, Christopher Zachow, Laura Zdanski, Helena Zhang, Iulia Zidaru, Christa Zoufal, aeddins ibm, alexzhang13, b63, bartek bartlomiej, bcamorrison, brandhsn, charmerDark, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgios ts, gruu, hhorii, hykavitha, itoko, jeppevinkel, jessica angel7, jezerjojo14, jliu45, jscott2, klinvill, krutik2966, ma5x, michelle4654, msuwama, nico lgrs, ntgiwsvp, ordmoj, sagar pahwa, pritamsinha2304, ryancocuzzo, saktar unr, saswati qiskit, septembrr, sethmerkel, sg495, shaashwat, smturro2, sternparky, strickroman, tigerjack, tsura crisaldo, vadebayo49, welien, willhbang, wmurphy collabstar, yang.luh, and Mantas Čepulkovskis. 2021. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2573505

[2] Ryan Babbush, Craig Gidney, Dominic W Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. 2018. Encoding electronic spectra in quantum circuits with linear T complexity. *Physical Review X* 8, 4 (2018), 041015.

[3] Joseph C. Bardin, Evan Jeffrey, Erik Lucero, Trent Huang, Sayan Das, Daniel Thomas Sank, Ofer Naaman, Anthony Edward Megrant, Rami Barends, Ted White, Marissa Giustina, Kevin J. Satzinger, Kunal Arya, Pedram Roushan, Benjamin Chiaro, Julian Kelly, Zijun Chen, Brian Burkett, Yu Chen, Andrew Dunsworth, Austin Fowler, Brooks Foxen, Craig Gidney, Rob Graff, Paul Klimov, Josh Mutus, Matthew J. McEwen, Matthew Neeley, Charles J. Neill, Chris Quintana, Amit Vainsencher, Hartmut Neven, and John Martinis. 2019. Design and Characterization of a 28-nm Bulk-CMOS Cryogenic Quantum Controller Dissipating Less Than 2 mW at 3 K. *IEEE Journal of Solid-State Circuits* 54, 11 (2019), 3043–3060. https://doi.org/10.1109/JSSC.2019.2937234

[4] Arnout Beckers, Farzan Jazaeri, and Christian Enz. 2018. Characterization and modeling of 28-nm bulk CMOS technology down to 4.2 K. *IEEE Journal of the Electron Devices Society* 6 (2018), 1007–1018.

[5] Arnout Beckers, Farzan Jazaeri, Alexander Grill, Subramanian Narasimhamoorthy, Bertrand Parvais, and Christian Enz. 2020. Physical Model of Low-Temperature to Cryogenic Threshold Voltage in MOSFETs. *IEEE Journal of the Electron Devices Society* 8 (2020), 780–788.

[6] Bluefors. 2022. XLD Dilution Refrigerator System. https://bluefors.com/products/xld-dilution-refrigerator [Online Accessed, 23-April-2022].

[7] Ilkwon Byun, Dongmoon Min, Gyu-hyeon Lee, Seongmin Na, and Jangwoo Kim. 2020. CryoCore: A fast and dense processor architecture for cryogenic computing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 335–348.

[8] Manuel A Castellanos-Beltran, David I Olaya, Adam J Sirois, Paul D Dresselhaus, Samuel P Benz, and Peter F Hopkins. 2019. Stacked Josephson junctions as inductors for single flux quantum circuits. *IEEE Transactions on Applied Superconductivity* 29, 5 (2019), 1–5.

[9] Zijun Chen, Kevin J. Satzinger, Juan Atalaya, Alexander N. Korotkov, Andrew Dunsworth, Daniel Sank, Chris Quintana, Matt McEwen, Rami Barends, Paul V. Klimov, Sabrina Hong, Cody Jones, Andre Petukhov, Dvir Kafri, Sean Demura, Brian Burkett, Craig Gidney, Austin G. Fowler, Harald Putterman, Igor Aleiner, Frank Arute, Kunal Arya, Ryan Babbush, Joseph C. Bardin, Andreas Bengtsson, Alexandre Bourassa, Michael Broughton, Bob B. Buckley, David A. Buell, Nicholas Bushnell, Benjamin Chiaro, Roberto Collins, William Courtney, Alan R. Derk, Daniel Eppens, Catherine Erickson, Edward Farhi, Brooks Foxen, Marissa Giustina, Jonathan A. Gross, Matthew P. Harrigan, Sean D. Harrington, Jeremy Hilton, Alan Ho, Trent Huang, William J. Huggins, L. B. Ioffe, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Kostyantyn Kechedzhi, Seon Kim, Fedor Kostritsa, David Landhuis, Pavel Laptev, Erik Lucero, Orion Martin, Jarrod R. McClean, Trevor McCourt, Xiao Mi, Kevin C. Miao, Masoud Mohseni, Wojciech Mruczkiewicz, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Michael Newman, Murphy Yuezhen Niu, Thomas E. O'Brien, Alex Opremcak, Eric Ostby, Bálint Pató, Nicholas Redd, Pedram Roushan, Nicholas C. Rubin, Vladimir Shvarts, Doug Strain, Marco Szalay, Matthew D. Trevithick, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, Sergio Boixo, Vadim Smelyanskiy, Yu Chen, Anthony Megrant, and Julian Kelly. 2021. Exponential suppression of bit or phase flip errors with repetitive error correction. arXiv:2102.06132 [quant-ph]

[10] Poulami Das, Christopher A Pattison, Srilatha Manne, Douglas Carmean, Krysta Svore, Moinuddin Qureshi, and Nicolas Delfosse. 2020. A scalable decoder micro-architecture for fault-tolerant quantum computing. *arXiv preprint arXiv:2001.06598* (2020).
[11] Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic Chong. 2018. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 828–840.
[12] David P DiVincenzo and Panos Aliferis. 2007. Effective fault-tolerant quantum computation with slow measurements. *Physical review letters* 98, 2 (2007), 020501.
[13] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).
[14] Austin G Fowler and Craig Gidney. 2018. Low overhead quantum computation using lattice surgery. *arXiv preprint arXiv:1808.06709* (2018).
[15] 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.
[16] Xiang Fu, Lingling Lao, Koen Bertels, and Carmen G Almudever. 2019. A control microarchitecture for fault-tolerant quantum computing. *Microprocessors and Microsystems* 70 (2019), 21–30.
[17] Xiang Fu, Leon Riesebos, Lingling Lao, Carmen G Almudever, Fabio Sebastiano, Richard Versluis, Edoardo Charbon, and Koen Bertels. 2016. A heterogeneous quantum computer architecture. In *Proceedings of the ACM International Conference on Computing Frontiers*. 323–330.
[18] Craig Gidney. 2021. Stim: a fast stabilizer circuit simulator. *Quantum* 5 (July 2021), 497. https://doi.org/10.22331/q-2021-07-06-497
[19] Craig Gidney and Martin Ekerå. 2021. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* 5 (2021), 433.
[20] Craig Gidney and Austin G Fowler. 2019. Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation. *Quantum* 3 (2019), 135.
[21] Yoshihito Hashimoto, Shinichi Yorozu, Toshiyuki Miyazaki, Yoshio Kameda, Hideo Suzuki, and Nobuyuki Yoshikawa. 2007. Implementation and experimental evaluation of a cryocooled system prototype for high-throughput SFQ digital applications. *IEEE transactions on applied superconductivity* 17, 2 (2007), 546–551.
[22] Adam Holmes, Yongshan Ding, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T Chong. 2019. Resource optimized quantum architectures for surface code implementations of magic-state distillation. *Microprocessors and Microsystems* 67 (2019), 56–70.
[23] Adam Holmes, Mohammad Reza Jokar, Ghasem Pasandi, Yongshan Ding, Massoud Pedram, and Frederic T Chong. 2020. NISQ+: Boosting quantum computing power by approximating quantum error correction. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 556–569.
[24] J. M. Hornibrook, J. I. Colless, I. D. Conway Lamb, S. J. Pauka, H. Lu, A. C. Gossard, J. D. Watson, G. C. Gardner, S. Fallahi, M. J. Manfra, and D. J. Reilly. 2015. Cryogenic Control Architecture for Large-Scale Quantum Computing. *Phys. Rev. Applied* 3 (Feb 2015), 024010. Issue 2. https://doi.org/10.1103/PhysRevApplied.3.024010
[25] Clare 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.
[26] Caleb Howington, Alex Opremcak, Robert McDermott, Alex Kirichenko, Oleg A. Mukhanov, and Britton L. T. Plourde. 2019. Interfacing Superconducting Qubits With Cryogenic Logic: Readout. *IEEE Transactions on Applied Superconductivity* 29, 5 (2019), 1–5. https://doi.org/10.1109/TASC.2019.2908884
[27] IBM. 2020. IBM's roadmap for scaling quantum technology. https://research.ibm.com/blog/ibm-quantum-roadmap [Online Accessed, 23-April-2022].
[28] IBM. 2021. IBM Quantum breaks the 100-qubit processor barrier. https://research.ibm.com/blog/127-qubit-quantum-processor-eagle [Online Accessed, 23-April-2022].
[29] Koki Ishida, Ilkwon Byun, Ikki Nagaoka, Kosuke Fukumitsu, Masamitsu Tanaka, Satoshi Kawakami, Teruo Tanimoto, Takatsugu Ono, Jangwoo Kim, and Koji Inoue. 2020. SuperNPU: An extremely fast neural processing unit using superconducting logic devices. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 58–72.
[30] Canali Jacoboni, C Canali, G Ottaviani, and A Alberigi Quaranta. 1977. A review of some charge transport properties of silicon. *Solid-State Electronics* 20, 2 (1977), 77–89.
[31] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R Brown, Margaret Martonosi, and Frederic T Chong. 2017. Optimized surface code communication in superconducting quantum computers. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 692–705.
[32] 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. *arXiv preprint arXiv:2202.01407* (2022).
[33] N Cody Jones, Rodney Van Meter, Austin G Fowler, Peter L McMahon, Jungsang Kim, Thaddeus D Ladd, and Yoshihisa Yamamoto. 2012. Layered architecture for quantum computing. *Physical Review X* 2, 3 (2012), 031007.
[34] Kiseo Kang, Donggyu Minn, Seunghun Bae, Jaeho Lee, Seongun Bae, Gichang Jung, Seokhyeong Kang, Moonjoo Lee, Ho-Jin Song, and Jae-Yoon Sim. 2022. A Cryo-CMOS Controller IC With Fully Integrated Frequency Generators for Superconducting Qubits. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 362–364.
[35] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, I.-C. Hoi, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and John M. Martinis. 2015. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature* 519, 7541 (01 Mar 2015), 66–69. https://doi.org/10.1038/nature14270
[36] DE Kirichenko, Saad Sarwana, and AF Kirichenko. 2011. Zero static power dissipation biasing of RSFQ circuits. *IEEE Transactions on Applied Superconductivity* 21, 3 (2011), 776–779.
[37] Morten Kjaergaard, Mollie E Schwartz, Jochen Braumüller, Philip Krantz, Joel I-J Wang, Simon Gustavsson, and William D Oliver. 2020. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics* 11 (2020), 369–395.
[38] Jens Koch, M Yu Terri, Jay Gambetta, Andrew A Houck, David I Schuster, Johannes Majer, Alexandre Blais, Michel H Devoret, Steven M Girvin, and Robert J Schoelkopf. 2007. Charge-insensitive qubit design derived from the Cooper pair box. *Physical Review A* 76, 4 (2007), 042319.
[39] Sebastian Krinner, Simon Storz, Philipp Kurpiers, Paul Magnard, Johannes Heinsoo, Raphael Keller, Janis Luetolf, Christopher Eichler, and Andreas Wallraff. 2019. Engineering cryogenic setups for 100-qubit scale superconducting circuit systems. *EPJ Quantum Technology* 6, 1 (2019), 2.
[40] Florent Lecocq, Franklyn Quinlan, Katarina Cicak, Jose Aumentado, SA Diddams, and JD Teufel. 2021. Control and readout of a superconducting qubit using a photonic link. *Nature* 591, 7851 (2021), 575–579.
[41] Gyu-hyeon Lee, Dongmoon Min, Ilkwon Byun, and Jangwoo Kim. 2019. Cryogenic computer architecture modeling with memory-side case studies. In *Proceedings of the 46th International Symposium on Computer Architecture*. 774–787.
[42] Gyu-Hyeon Lee, Seongmin Na, Ilkwon Byun, Dongmoon Min, and Jangwoo Kim. 2021. CryoGuard: A Near Refresh-Free Robust DRAM Design for Cryogenic Computing. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 637–650.
[43] E. Leonard, M. A. Beck, J. Nelson, B.G. Christensen, T. Thorbeck, C. Howington, A. Opremcak, I.V. Pechenezhskiy, K. Dodge, N.P. Dupuis, M.D. Hutchings, J. Ku, F. Schlenker, J. Suttle, C. Wilen, S. Zhu, M.G. Vavilov, B.L.T. Plourde, and R. McDermott. 2019. Digital Coherent Control of a Superconducting Qubit. *Phys. Rev. Applied* 11 (Jan 2019), 014009. Issue 1. https://doi.org/10.1103/PhysRevApplied.11.014009
[44] Konstantin K Likharev and Vasilii K Semenov. 1991. RSFQ logic/memory family: A new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Transactions on Applied Superconductivity* 1, 1 (1991), 3–28.
[45] Daniel Litinski. 2019. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum* 3 (2019), 128.
[46] Daniel Litinski. 2019. Magic state distillation: Not as costly as you think. *Quantum* 3 (2019), 205.
[47] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. 2020. Quantum computational chemistry. *Reviews of Modern Physics* 92, 1 (2020), 015003.
[48] R McDermott, MG Vavilov, BLT Plourde, FK Wilhelm, PJ Liebermann, OA Mukhanov, and TA Ohki. 2018. Quantum–classical interface based on single flux quantum digital logic. *Quantum science and technology* 3, 2 (2018), 024004.
[49] Mohammadreza Mehrpoo, Bishnu Patra, Jiang Gong, JPG van Dijk, H Homulle, G Kiene, A Vladimirescu, F Sebastiano, E Charbon, and M Babaie. 2019. Benefits and challenges of designing cryogenic CMOS RF circuits for quantum computers. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[50] Dongmoon Min, Ilkwon Byun, Gyu-Hyeon Lee, Seongmin Na, and Jangwoo Kim. 2020. Cryocache: A fast, large, and cost-effective cache architecture for cryogenic computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 449–464.
[51] Dongmoon Min, Yujin Chung, Ilkwon Byun, Junpyo Kim, and Jangwoo Kim. 2022. CryoWire: wire-driven microarchitecture designs for cryogenic computing. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 903–917.
[52] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. 2019. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1015–1029.
[53] Jong-Seok Park, Sushil Subramanian, Lester Lampert, Todor Mladenov, Ilya Klotchkov, Dileep J. Kurian, Esdras Juarez-Hernandez, Brando Perez-Esparza,

Sirisha Rani Kale, K. T. Asma Beevi, Shavindra Premaratne, Thomas Watson, Satoshi Suzuki, Mustafijur Rahman, Jaykant B. Timbadiya, Saksham Soni, and Stefano Pellerano. 2021. 13.1 A Fully Integrated Cryo-CMOS SoC for Qubit Control in Quantum Computers Capable of State Manipulation, Readout and High-Speed Gate Pulsing of Spin Qubits in Intel 22nm FFL FinFET Technology. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 64. 208–210. https://doi.org/10.1109/ISSCC42613.2021.9365762

[54] S. J. Pauka, K. Das, R. Kalra, A. Moini, Y. Yang, M. Trainer, A. Bousquet, C. Cantaloube, N. Dick, G. C. Gardner, M. J. Manfra, and D. J. Reilly. 2021. A cryogenic CMOS chip for generating control signals for multiple qubits. *Nature Electronics* 4, 1 (01 Jan 2021), 64–70. https://doi.org/10.1038/s41928-020-00528-y

[55] JJ Plombon, Ebrahim Andideh, Valery M Dubin, and Jose Maiz. 2006. Influence of phonon, geometry, impurity, and grain size on copper line resistivity. *Applied physics letters* 89, 11 (2006), 113124.

[56] John Preskill. 1998. Fault-tolerant quantum computation. In *Introduction to quantum computation and information*. World Scientific, 213–269.

[57] Moinuddin Qureshi and Swamit Tannu. 2021. Quantum Computing and the Design of the Ultimate Accelerator. *IEEE Micro* 41, 5 (2021), 8–14.

[58] Leon Riesebos, Xiang Fu, Savvas Varsamopoulos, Carmen G Almudever, and Koen Bertels. 2017. Pauli frames for quantum computer architectures. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.

[59] Ashish Shukla, Benjamin Chonigman, Anubhav Sahu, Dmitri Kirichenko, Amol Inamdar, and Deepnarayan Gupta. 2019. Investigation of passive transmission lines for the MIT-LL SFQ5ee process. *IEEE Transactions on Applied Superconductivity* 29, 5 (2019), 1–7.

[60] Jennifer Pearl Smith, Benjamin A. Mazin, Alex B. Walter, Miguel Daal, J.I. Bailey, Clinton Bockstiegel, Nicholas Zobrist, Noah Swimmer, Sarah Steiger, and Neelay Fruitwala. 2021. Flexible Coaxial Ribbon Cable for High-Density Superconducting Microwave Device Arrays. *IEEE Transactions on Applied Superconductivity* 31, 1 (2021), 1–5. https://doi.org/10.1109/TASC.2020.3008591

[61] James E. Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W. Rhett Davis, Paul D. Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, and Ravi Jenkal. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. 173–174. https://doi.org/10.1109/MSE.2007.44

[62] Synopsys. 2022. Synopsys DC Ultra. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html [Online Accessed, 23-April-2022].

[63] Swamit S Tannu, Douglas M Carmean, and Moinuddin K Qureshi. 2017. Cryogenic-DRAM based memory system for scalable quantum computers: a feasibility study. In *Proceedings of the International Symposium on Memory Systems*. 189–195.

[64] Swamit S Tannu, Zachary A Myers, Prashant J Nair, Douglas M Carmean, and Moinuddin K Qureshi. 2017. Taming the instruction bandwidth of quantum computers via hardware-managed error correction. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 679–691.

[65] Barbara M Terhal. 2015. Quantum error correction for quantum memories. *Reviews of Modern Physics* 87, 2 (2015), 307.

[66] Barbara M Terhal. 2018. Quantum supremacy, here we come. *Nature Physics* 14, 6 (2018), 530–531.

[67] Sergey K Tolpygo, Vladimir Bolkhovsky, Terence J Weir, Alex Wynn, Daniel E Oates, Leonard M Johnson, and Mark A Gouker. 2016. Advanced fabrication processes for superconducting very large-scale integrated circuits. *IEEE Transactions on Applied Superconductivity* 26, 3 (2016), 1–10.

[68] Yu Tomita and Krysta M Svore. 2014. Low-distance surface codes under realistic quantum noise. *Physical Review A* 90, 6 (2014), 062320.

[69] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. 2021. QECOOL: On-Line Quantum Error Correction with a Superconducting Decoder for Surface Code. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 451–456. https://doi.org/10.1109/DAC18074.2021.9586326

[70] Jeroen Petrus Gerardus Van Dijk, Bishnu Patra, Sushil Subramanian, Xiao Xue, Nodar Samkharadze, Andrea Corna, Charles Jeon, Farhana Sheikh, Esdras Juarez-Hernandez, Brando Perez Esparza, Huzaifa Rampurawala, Brent R. Carlton, Surej Ravikumar, Carlos Nieva, Sungwon Kim, Hyung-Jin Lee, Amir Sammak, Giordano Scappucci, Menno Veldhorst, Lieven M. K. Vandersypen, Edoardo Charbon, Stefano Pellerano, Masoud Babaie, and Fabio Sebastiano. 2020. A Scalable Cryo-CMOS Controller for the Wideband Frequency-Multiplexed Control of Spin Qubits and Transmons. *IEEE Journal of Solid-State Circuits* 55, 11 (2020), 2930–2946. https://doi.org/10.1109/JSSC.2020.3024678

[71] Clifford Wolf. 2016. Yosys open synthesis suite.

[72] Yuki Yamanashi, Toshiki Kainuma, Nobuyuki Yoshikawa, Irina Kataeva, Hiroyuki Akaike, Akira Fujimaki, Masamitsu Tanaka, Naofumi Takagi, Shuichi Nagasawa, and Mutsuo Hidaka. 2010. 100 GHz demonstrations based on the single-flux-quantum cell library for the 10 kA/cm^2 Nb multi-layer process. *IEICE transactions on electronics* 93, 4 (2010), 440–444.

[73] Farzaneh Zokaee and Lei Jiang. 2021. SMART: A Heterogeneous Scratchpad Memory Architecture for Superconductor SFQ-based Systolic CNN Accelerators. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 912–924.