Thesis for the Degree of Master of Engineering

Research on Quantum Computer Simulation Acceleration by using an Emerging Memory Technology

by

Sang Hyeon Lee

Department of Artificial Intelligence

The Graduate School

Pukyong National University

February, 2024

Research on Quantum Computer Simulation Acceleration by using an Emerging Memory Technology
(차세대 메모리 기술을 활용한 양자 컴퓨터 시뮬레이션 가속화 구조에 대한 연구)

Advisor: Prof. Young Sun Han

by

Sang Hyeon Lee

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Engineering

in Department of Artificial Intelligence, The Graduate School,
Pukyong National University

February, 2024

Research on Quantum Computer Simulation Acceleration by using an Emerging Memory Technology

A dissertation
by
Sang Hyeon Lee

Approved by:

_____
Professor. Yong-Tae Kim,
(Chairman)

_____ _____
Professor. Seung-Ho Yoo Professor. Young-Sun Han
(Member) (Member)

February 16th, 2024

Index

Table Index

Figure Index

Research on Quantum Computer Simulation Acceleration by using an Emerging Memory technology

Sang-Hyeon Lee

Department of Artificial Intelligence Convergence, The Graduate School,
Pukyong National University

Abstract

Quantum computers are currently regarded as an emerging computing technology that can solve problems more quickly than classical computers. However, since constructing a general quantum computer is technically difficult, quantum computer simulation has been used instead of real quantum computers. Simulating quantum computers on classical computers is challenging because the time and resources required for the vector–matrix multiplication (VMM) increase exponentially with the number of qubits. This paper proposes a new accelerator architecture called ReQUSA that leverages resistive random access memory (ReRAM) to accelerate the quantum computer simulation. The ReQUSA employs a ReRAM crossbar array structure, which is specialized for implementing the VMM, and a realized state method for reduced VMM operation.

To the best of our knowledge, ReRAM-based accelerators for quantum computer simulator has not been previously reported. Here we describe the hardware design of the architecture and compare the performances (hardware resource, simulation time, and accuracy) of our accelerator with those of current quantum computer simulators (QuEST, QPlayer, and Qiskit). On average, our proposed architecture reduced the simulation times by factors of $\times 10^4$ and $\times 10^3 (\times 10^2)$ on average from those of QuEST and QPlayer (also Qiskit), respectively. In addition, our architecture achieved 99% accuracy in 16-bit fixed-point data representation.

1. Introduction

Recently, quantum computing technology has rapidly evolved and has attracted much attention and investment from various fields. Quantum computing promises to solve problems that are too complex and time-consuming on classical computers, such as quantum searching [10], quantum sorting [12], quantum oracle algorithms [2, 6, 8], and quantum machine learning [3, 26]. Quantum computation solves complex and massive operations quickly and efficiently using quantum mechanics. Unlike classical computers that use bits (binary digits) to perform operations, quantum computers use qubits (quantum bits) to perform quantum operations. Exploiting the superposition phenomenon of quantum mechanics, a qubit can simultaneously store $|0\rangle$ and $|1\rangle$ in a quantum computer system. Therefore, a qubit can simultaneously exist in both states until it is measured. When a qubit is measured, it collapses into one of the basis states with a certain probability. The state of a qubit can be represented by a one-dimensional column vector of unit norms $[\alpha\ \beta]$, where α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. A qubit can be manipulated through quantum gates, unitary matrices that transform the state vector of the qubit. For example, the Pauli- X (X) gate flips the state of a qubit from $|0\rangle$ to $|1\rangle$ and vice versa. The Hadamard gate places a qubit in an equal superposition state of $|0\rangle$ and $|1\rangle$. The Pauli- Z (Z) gate changes the phase of the qubit by π radians. Many other single-qubit and multi-qubit gates perform various quantum

operations. By utilizing quantum properties such as superposition and entanglement, quantum computers perform certain types of calculations that are intractable for classical computers. Moreover, algorithms designed for quantum computers can potentially solve complex problems much faster than classical computing methods. However, in today's noise intermediate-scale quantum (NISQ) era, the scalability, error correction capability, coherence, and interoperability of quantum computing remain problematic [27]. These problems are avoided by employing quantum computer simulations rather than actual quantum computers.

A quantum computer simulation applies a quantum gate to a qubit following the mathematical principles of quantum physics [16, 28]. The qubit and quantum gate can be represented as a state vector and a quantum gate matrix, respectively. Accordingly, a quantum computer simulation repeatedly multiplies the matrix by the vector. To enable vector–matrix multiplication (VMM), the quantum gate matrix should be expanded to fit the dimension of the state vector, which increases with the number of qubits. This process is widely accepted although it dramatically slows the quantum computer simulation. Among the existing quantum simulators are Qiskit, which can simulate up to 30 qubits and run quantum programs on classical computers, and QuEST [15], which performs the simulation by utilizing a multiprocessor. To accelerate quantum computer simulations, researchers have investigated more efficient computing methods such as general-purpose computing on graphics processing units (GPGPU) or multi- processing [1, 7]. However, these approaches are resource-intensive and provide no inherent improvement other than the division of the repetitive VMM operations

into multiple processors. Here we propose a novel architecture of ReQUSA based on a resistive random access memory (ReRAM) crossbar, which accelerates the quantum computer simulation by leveraging the characteristics of ReRAM. In particular, the ReRAM is specialized for VMM operations in the analog domain. To avoid dimensional expansion of the state vector and quantum gate, we adopt the reduced VMM operation method based on the realized state (RS) described in [14]. The simulation time of our proposed architecture is $\times 10^4$ shorter than that of QuEST, $\times 10^3$ and $\times 10^2$ shorter than those of QPlayer and Qiskit, respectively. The key contributions of our paper are summarized below:

• We propose a novel architecture that efficiently handles many qubits in quantum computer simulations using the ReRAM crossbar structure.
• We develop a detailed hardware structure at the circuit level for manipulating the reduced VMM operations.
• We demonstrate the superior simulation time and feasible accuracy of the simulation results generated by our proposed architecture.

The remainder of this paper is organized as follows. Section 2 describes the background of the ReRAM crossbar and quantum computer simulation. Related works are summarized in Section 3. Section 4 discusses the architecture and behavior of the proposed accelerator and Section 5 describes the hardware implementation of the accelerator. In Section 6, we evaluate the performance of the architecture in terms of simulation time, hardware resources, and accuracy. The paper conclusion with Section 7.

2. Background

This section provides the background of ReRAM, quantum computer simulations, and the RS method.

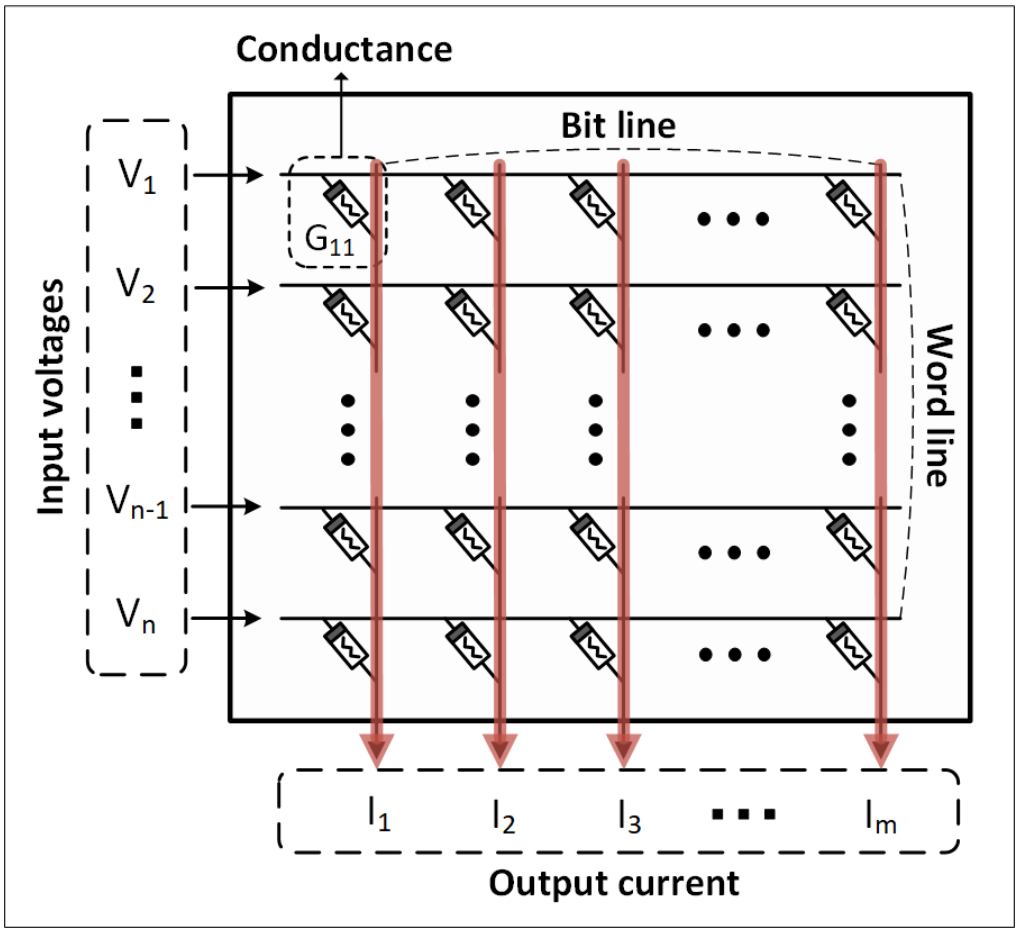


Figure 1. Basic structure of the ReRAM crossbar array. An input voltage V_i is applied equally to all bit lines along the word line. Subsequently, the read operations (red lines) are performed concurrently through the bit lines.

2.1 ReRAM crossbar

ReRAM crossbar Conventional memories such as dynamic and static RAM rely on the storage of charge; therefore, they are volatile memories that lose their data when the power supply is turned off. In contrast, ReRAM is a next-generation memory device that stores data based on the characteristics of electrical resistance, which depend on the history of the previous current. Therefore, even when the power supply is removed, the most recent data can be stored continuously, achieving nonvolatile memory. When the resistance is high and the current flow is rough, the logical value is 0 and the state is called a high resistance state. Conversely, when the resistance is low and the current flows well, the logical value is 1 and the state is called a low resistance state. To utilize the characteristics of the ReRAM, a crossbar structure is generally constructed. As shown in Figure 1, the ReRAM cells in a crossbar structure are placed at each connection point of horizontal and vertical steel wires. To execute the VMM operation in a single cycle on the ReRAM crossbar, the analog conductance of the matrix coefficients are stored in memristor cells and the vector values are the input voltages to a word line. The multiplication result is then obtained as a current through a bit line. The above process is formulated as follows:

$$I_k = \sum_{i=1}^{m} (G_{ik} \times V_i) \qquad for\ all\ k, where\ 1 \le k \le m \qquad (1)$$

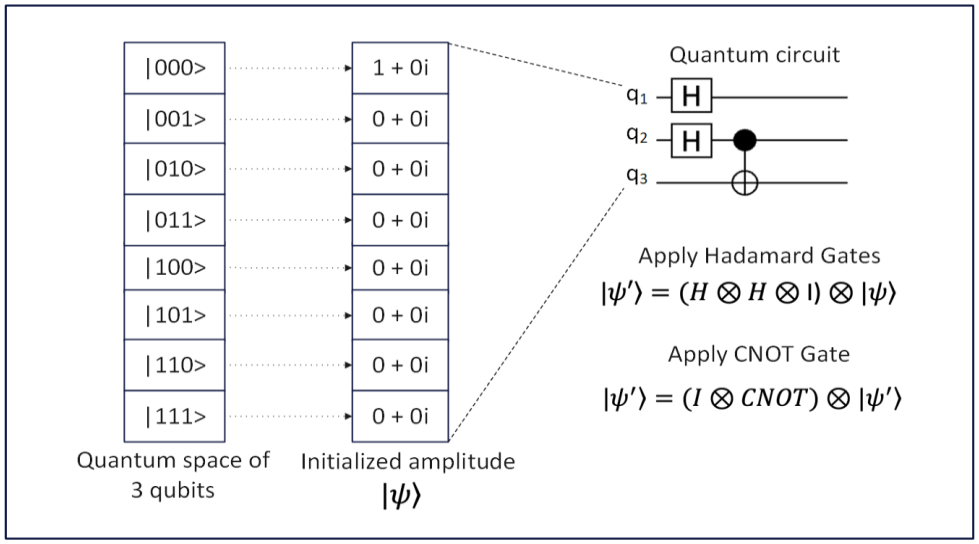
, where I is the current derived from the bit line, and G and V are the conductance of the ReRAM cell and the input voltage, respectively. ReRAM crossbar arrays offer several advantages over conventional computing architectures for VMMs [9, 18]. First, ReRAM devices can be programmed to directly perform multiply-and-accumulate operations in memory, reducing the amount of data movement between the memory and the processing elements. The lowered energy consumption and latency associated with data movement allow faster and more energy-efficient computations. Second, ReRAM crossbar arrays can perform VMM operations in a massively parallel manner, enabling high throughput and scalability. Third, ReRAM devices exhibit high device density and low power consumption, so are suitable for implementation in large-scale neuromorphic computing systems. Owing to these advantages, the crossbar array architecture also accelerates deep learning [13, 33, 20, 22].

2.2 Quantum computer simulation

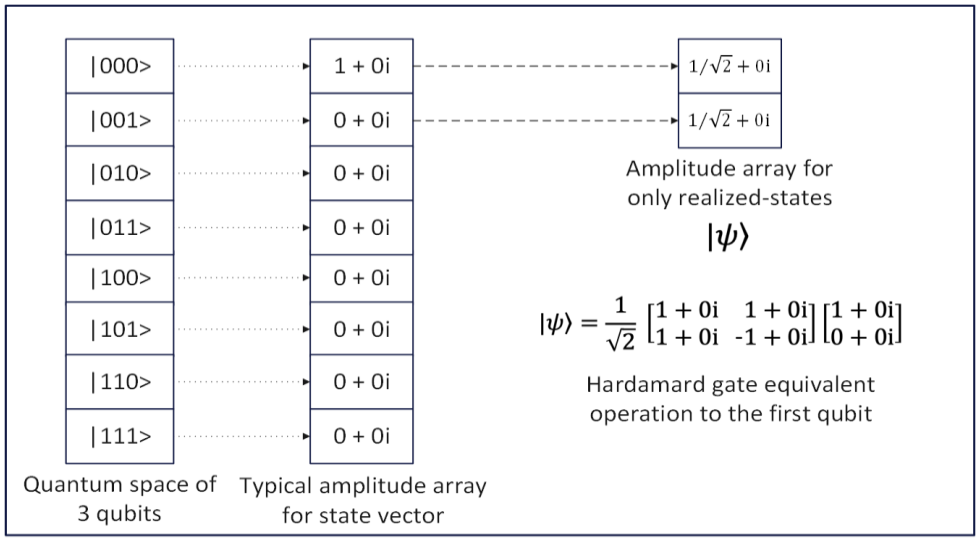
Leveraging quantum phenomena such as superposition and entanglement, quantum computers perform computations that are beyond the reach of classical computers. However, as discussed above, building and operating quantum computers is a challenging task, as quantum systems are fragile and error prone. Instead, quantum computers are simulated on classical computers, which are more reliable and accessible.

Quantum computer simulations represent quantum states and operations as vectors and matrices which can then be manipulated through classical algorithms. Such a simulation can reveal the behavior of a quantum system, run quantum algorithms, and enable the design of new quantum systems. The processes of two quantum computer simulation methods are shown in Figure 2. In the full-state quantum computer simulation method (Figure 2(a)) [32], a three-qubit system is represented as a complex vector space $|\psi\rangle$. Each element of the vector is a complex number represented as a $a + bi$. Following the quantum circuit shown in Figure 2(a), the three-qubit system is simulated by applying quantum gates. To this end, the tensor product of two Hadamard (H) gates and an Identity (I) gate is calculated to yield a $2^3 \times 2^3$ matrix of complex numbers. Then, we apply VMM to change the state of the qubits, resulting in a new quantum state vector denoted as $|\dot{\psi}\rangle$. Afterward, the quantum computer simulation is completed by performing another VMM on a matrix representing the tensor product between a Controlled-X (CNOT) gate and an I gate and the vector from the previous

calculation. This method simulates the state and behavior of a quantum computer, but the state vector and quantum gate matrix are scaled by factors of 2^n and $2^n \times 2^n$, respectively, where n denotes the number of qubits. As the number of qubits increases, the number of operations on the state vector and matrix becomes exceedingly large and the memory demands increase accordingly. Although several attempts to minimize the memory usage have been published [14, 37, 4, 16, 11, 24], an effective solution remains elusive.



(a)



(b)

Figure 2. Two different quantum computer simulation methods showing the structure and operation of a three-qubit system: (a) Full-state quantum computer simulation method [15] (b) Realized-state quantum computer simulation method [14].

2.3 Realized state quantum computer simulation

To minimize the computational burden and reduce the memory requirements of large state vectors and quantum gate matrices, Jin et al. proposed a realized-state method that represents the qubit states [14]. RSs represent only those qubits with non-zero amplitudes. For example, Figure 2(b) illustrates a three-qubit system with a quantum space of eight states. Whereas a full-state quantum computer simulation must represent each state with complex numbers, the RS simulation initializes only the first state $|000\rangle$ of the quantum system, which requires a single complex number in memory. To perform the H-gate operation and state change, multiplication is performed only on the realized pairs ($|000\rangle$ and $|001\rangle$ in this case). This process is equivalent to applying the H gate to the first qubit. The vector then contains two RSs for the simulation. The vector size in this method is determined by the realized-state changes in the quantum state obtained through operations, while the matrix size remains constant at $2^2 \times 2^2$. In contrast, the full-state simulation scales the matrix as $2^n \times 2^n$.

3. Related works

Quantum computing is a rapidly evolving field that promises to expand the limits of classical computers. However, fault tolerance and quantum supremacy have not been attained in the current NISQ era. Quantum supremacy defines the ability of a quantum computer to perform a task that cannot be managed by a classical computer within a reasonable time. In the NISQ era, quantum computers are sensitive to their environment (noise) and prone to quantum decoherence, which occurs via interactions between the qubits and the surroundings. Therefore, quantum computer simulation methods are necessary tools for developing and testing quantum algorithms within the limitations of quantum hardware.

However, simulating quantum computers is resource-intensive because the size of the vector representing the quantum state scales exponentially with the number of qubits. For example, a 50-qubit quantum computer requires a 250 sized vector or approximately 1015 elements. Such a vector requires approximately 8 petabytes of RAM storage. To alleviate the memory demands, recent studies have designed resource-efficient quantum computer simulation methods such as optimized single-node simulation frameworks [36, 14], simulators for distributed systems [15], and simulators with data compression techniques. Some of the existing frameworks run on a single classical computer, providing quantum programming support for both quantum computer simulations and real quantum computer executions. One such

framework is Qiskit [36], an open-source software development kit compatible with IBM's quantum computers and simulators. Qiskit allows users to write quantum programs in Python and execute them on either a local quantum computer simulator or a remote real quantum device.

However, Qiskit supports a maximum of only 30 qubits for simulation on a local backend. Real quantum devices with higher qubit counts remain inaccessible due to waiting time and cost for use. To simulate more qubits on a single node, QPlayer [14] applies an optimized simulation method based on the superposed qubit ratio (SQR) of the quantum circuit, which defines the ratio of qubits in the superposition state to the total number of qubits. The QPlayer can simulate 30, 40, 50, and 60 qubits with SQRs of 100%, 80%, 60%, and 40%, respectively. However, these methods are limited by the memory and processing power of a single-node classical computer.

Distributed systems increase the qubit-count simulation capacity by utilizing large resources across multiple nodes. One such distributed system is QuEST [15], a hardware-agnostic simulator of universal circuit-based quantum computers that can run on either CPUs or GPUs. QuEST claims to simulate more than 50 qubits depending on the quantum circuit and noise model. Another example is the full-state simulator with data compression [37], which reduces the memory requirements of stored quantum state vectors. The full-state simulator with data compression claims to simulate up to 61 qubits on classical supercomputers. Clearly, quantum computers beyond 61 qubits cannot be simulated on classical supercomputers.

4. ReQUSA: Proposed architecture

In this section, we propose the methodology and ReQUSA's hardware architecture to accelerate the quantum computer simulation by using the ReRAM crossbar.

4.1 Overall architecture

Figure 3 shows the overall architecture of the proposed accelerator and the input/output (I/O) data flow of the quantum computer simulation. To execute the quantum computer simulation, the provided quantum circuit is first compiled as a quantum gate information table, the data format of our architecture. Each quantum gate has a specific order, gate name, and qubit index (control or target), which can be integrated into the table. The order defines where the quantum gate is placed in the sequence of quantum gates applied to the state vectors of the qubits. After the compilation process, the gate information is assigned to our ReQUSA accelerator.

The ReQUSA consists of a global controller, an I/O interface, multiple quantum processing units (QPUs), and routers. Viewed from the top, the routers are connected to each QPU, allowing easy transfer of the data throughout the ReQUSA. A two-dimensional (2D) mesh Network-on-Chip (NoC) structure improves the data transmission speed and latency between the routers [5]. The 2D mesh network is a regular grid-like

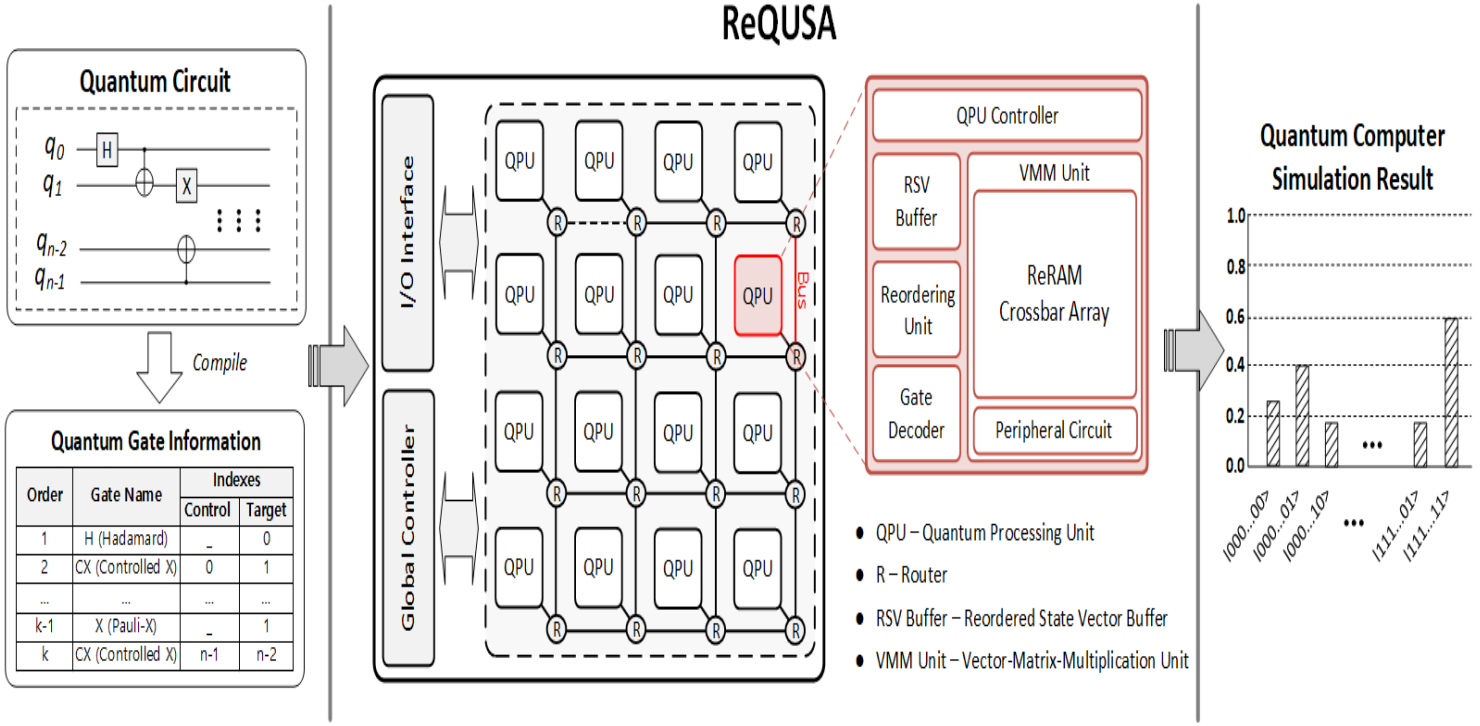
structure that connects each node to its four neighboring nodes, forming a 2D array. The data are transferred through the network bus by routing packets, which take the shortest route from the source node to the destination node. This arrangement reduces the latency and data transfer time between each QPU.

The global controller manages all situations occurring in the proposed architecture, such as the I/O data flow and the quantum computer simulation. On the ReQUSA, a qubit passing through a quantum circuit must sequentially pass through the quantum gates entangled in the circuit in a predetermined order to obtain the correct simulation result. Hence, the global controller sets the path of the routers and delivers an RS and the quantum gate information to a QPU at a predetermined location. We ensured that at this time, the quantum gates in adjacent QPUs are placed in the order in which the gates are applied.

The router determines the data path along which the packet can successfully reach the desired destination node. A router conventionally has five I/O ports: four I/O ports in the 2D mesh network that connect with other routers and one I/O port for the processing element (PE) [29], which is the QPU in our architecture. We applied the router proposed in [17] because it can be operated at the same clock frequency as other peripheral circuits and can support high bandwidth and packet delivery inside the network. The clock frequency will be further discussed in subsection 6.2.1.

The I/O interface manages the input and output data flows of the ReQUSA. In this case, the input denotes the quantum gate information, and the output is a result of the quantum computer simulation.

Figure 3. Overall architecture of ReQUSA, which consists of several states. The input is received as a quantum circuit in QASM format. Subsequently, the QASM is compiled to extract relevant information such as the operation details and qubit count. This information is fed to ReQUSA, in which multiple QPUs are connected through routers to facilitate their data transfer. Finally, the ReQUSA outputs a vector representing the quantum state and its amplitudes.



4.2 Quantum processing unit

A QPU performs the VMM operation using a quantum gate matrix and the realized state vector (RSV). The details of the QPU structure are shown in Figure 4. All operations are managed by the QPU controller, and the data flow occurred inside the QPU. Depending on the operation type, the controller activates different units such as the gate decoder, reordering unit, and VMM unit.

The gate decoder then decodes the quantum gate information to obtain the configuration of the quantum gate (gate ID, control index, and target index). We can determine the gate type and the qubit to which the gate is applied based on the presence or absence of the control and target qubit indices. For example, CNOT(0,1) is classified as a control gate because its index is 0, indicating a control qubit. H(0) is a single gate because it has no value for the control qubit index. Then, the gate decoder generates the quantum gate matrix data according to the gate ID. For this purpose, the gate ID and matrix data of each gate are stored in a look-up table (LUT).

Table 1 shows how the gates are stored in the LUT. Currently, the proposed architecture supports 12 types of gates, including the most basic single gates I, H, and X, and control gates such as CNOT and CZ. The number of gates may seem insufficient, but in a quantum computer, we can construct a universal quantum computer with a set of universal quantum gates, including CNOT and H gates. Note that since all matrices with 2×2 elements in the lower right corner of itself can be employed

for the proposed architecture, they can be added to the LUT.

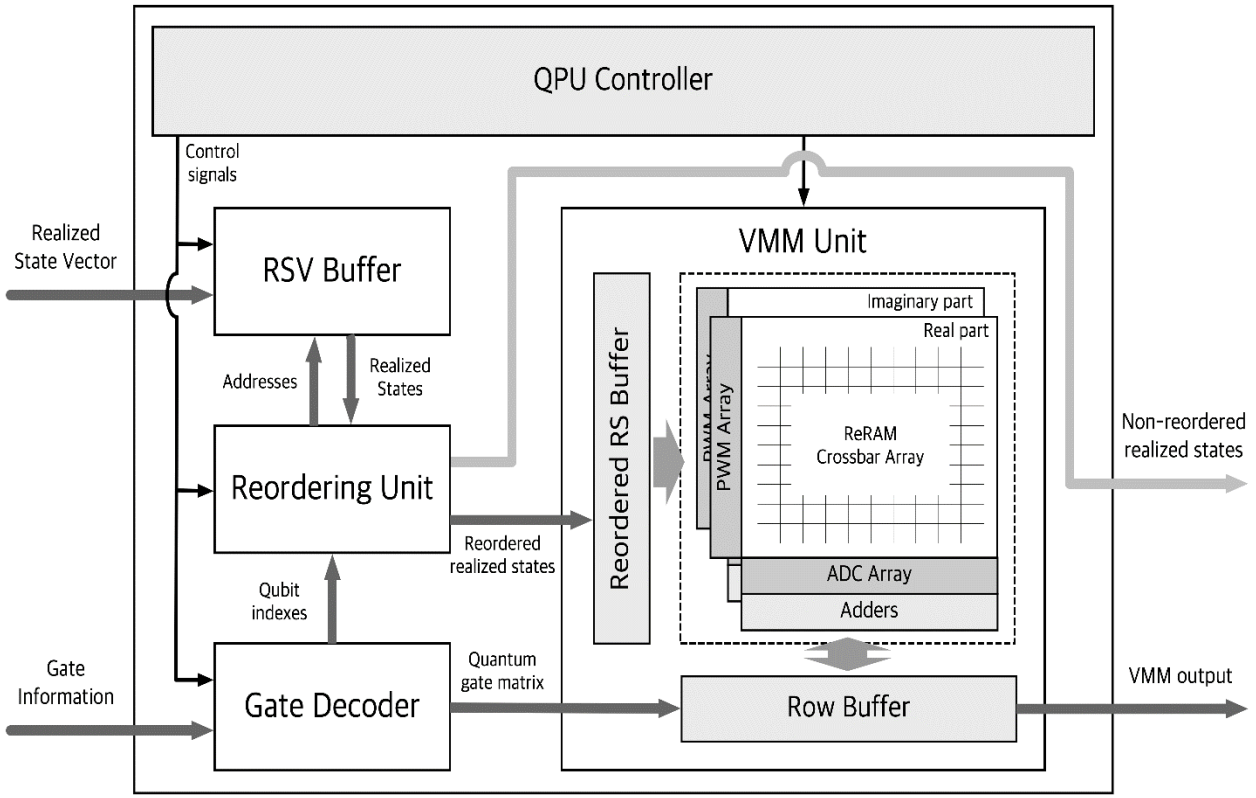
The reordering unit (ROU) executes the reordering algorithm. The ROU receives the RS from the RSV buffer and generates a pair of RSs to be transferred into the reordered RSV buffer. To realize the reordering algorithm, the RSV buffer provides two memory structures: Index Addressable Memory (IAM) and Content Addressable Memory (CAM), similar to previous work [25, 31]. Therefore, the ROU can access the RSV buffer with the RS address and the value of the RS state.

The VMM unit is the main QPU unit for the VMM operation. It consists of a row buffer, the reordered RS buffer, the ReRAM crossbar array, and peripheral circuits such as a pulse width modulator (PWM) array and analog-to-digital converter (ADC) array. The row buffer receives the quantum gate matrix data and the reordered RSV buffer temporarily stores the RSV generated by the ROU. This process improves the efficiency of multiplication by storing all RSVs according to the characteristics of the ReRAM crossbar array. The VMM operation is then completed in one cycle rather than by multiplying with the RSV generated at each time.

Table 1. An example of 2 x 2 size quantum gates in the LUT

Gate	Definition	Matrix
I	Identity gate	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
H	Hadamard gate	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
X	Pauli X gate	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
CNOT	Controlled X gate	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

Figure 4. Proposed quantum processing unit (QPU) architecture consisting of a realized state vector buffer (RSV Buffer), reordering unit, gate decoder, and vector– matrix multiplication (VMM) units. It executes a reduced VMM operation after receiving RSV and gate information.



4.3 Our proposed acceleration methodology

As explained in subsection 2.2, the VMM operation is the main part of the quantum computer simulation, and the 2×2 size quantum gate matrix largely reduces the memory consumption. However, this method does not significantly reduce the simulation time because all qubit state vectors must be repeatedly multiplied by the 2×2 matrix. We thus adopt a more efficient simulation technique combined with the RS developed in [14]. By applying quantum gates only to the RSs of qubits, our algorithm drastically reduces the number of VMM operations from that of the previous simulation technique. Algorithm 1 describes the process that lowers the number of VMM operations. The algorithm starts to find a pair of RSs based on the qubit state index and the target qubit index. For example, an X(1) gate is applied only to the qubit index 1, the target qubit of this gate. To create a pair of RSs, the algorithm first selects one index from the quantum register which manages the qubit index, amplitudes, and left-shifts the selected index as much as the target qubit, until it reaches the target index to generate the other index of the pair. After finding the RS pairs, the algorithm reads the amplitude of each pair. If the index does not exist in the quantum register, the amplitude is set to zero. Consequently, the amplitude of each RS is multiplied by the matrix to obtain a new amplitude. The control gate is handled by the same process as the single qubit with one difference: determining whether the control index of the qubit is indicated by the control qubit index of the gate. This algorithm is slightly modified to suit

the hardware structure for handling in the accelerator. Lines 9 and 22 of Algorithm 1 are modified from the original algorithm [14] by flipping the bit of the qubit pointed to by the index rather than by left-shifting the selected index. This change is possible because the qubit state is stored in binary format.

Algorithm 1: Reordering Algorithm

Input:
RS: realized state,
T_Index: target qubit index of gate,
C_Index: control qubit index of gate,
address: address to access RSV Buffer,
Output:
R_RS: reordered realized states,
N_RS: non-reordered realized states,

```
1  procedure ROU(RS, T_Index, C_Index)
2      j = 0, k = 0 // index for ReorderedRSV
3      if C_Index is not exist then
4          for address ← 0 to RS.length do
5              // Extract the RS by address
6              RS = RSV[address]
7              // qubit state in |q_n...q_1 q_0⟩
8              pairState = (RS.state) ∧ (1 ≪ T_Index)
9              // Find the pair_RS
10             pair_RS = RSV.get(pairState)
11             R_RS[j++] = RS
12             R_RS[j++] = pair_RS
13         end for
14         return R_RS
15     else
16         for address ← 0 to RSV.length do
17             RS = RSV[address]
18             isControlled = (RS.state) & (1 ≪ C_Index)
19             if isControlled then
20                 pairState = (RS.state) ∧ (1 ≪ T_Index)
21                 pair_RS = RSV.get(pairState)
22                 R_RS[j++] = RS
23                 R_RS[j++] = pair_RS
24             else
25                 N_RS[k++] = RS
26         end for
27         return R_RS, N_RS
28 end procedure
```

4.4 Realization of quantum computer simulation in QPU

To realize the proposed acceleration method inside the QPU, we divide the simulation process into two subprocess: write and read.

The write operation is the process of writing a quantum gate matrix to a ReRAM crossbar array in the VMM unit. When the QPU receives the write operation signal from the QPU controller, the gate decoder is first activated. The gate decoder decodes the quantum gate information to obtain the gate configuration (qubit indexes and gate ID), which is transmitted to the ROU. The gate decoder interprets the gate ID and produces the corresponding quantum gate matrix. The quantum gate matrix is passed to the PWM array inside the VMM unit. The matrix elements are stored as conductance in the cells of the ReRAM crossbar. Owing to the characteristics of the ReRAM crossbar array, the multiplication results are summed and printed out along one column. To prohibit interruption of the VMM result by other multiplication results, the same matrix elements are diagonally placed on the ReRAM crossbar array.

The read operation performs the VMM operation by transferring the RSV to the ReRAM crossbar array. As shown in Figure 4, the RS from the RSV buffer is passed into the ROU, which executes the reordering algorithm based on the control and target index as explained in subsection 4.3. The reordering process iterates until pairs are found for all RSVs stored in the RSV buffer. Once the reordering is completed, the ROU sends the reordered RSV to the reordered RS buffer. Figure 5

shows the example of a detailed process inside the QPU after the reordering process when each RS has eight-bit precision. The original eight-bit VMM operation becomes complicated because the 16-bit result cannot be handled by the eight-bit ADC. To solve this problem, we split the eight-bit into upper and lower four-bit components [29]. The upper four-bit contain the four highest digits, including the most significant bit (MSB). Therefore, the elements of one quantum gate matrix are placed in two rows and eight columns across two ReRAM crossbar arrays (e.g. γ_{1H} and γ_{1L} in the blue dashed box of Figure 5). In this situation, two reordered RSs must be picked and multiplied by the quantum gate matrix. In Figure 5, the real parts of RSs, i.e., α_1 and α_2, are chosen and stored in the reordered RS buffer. The α_1 and α_2 are then divided into upper and lower parts α_{1H}, α_{1L} and α_{2H}, α_{2L}, respectively. These data are modulated by the PWM, which converted them to pulse-width equivalents, and assigned the converted pulse to the word line. The applied pulse changes the state of the cell passing through the ReRAM cell of the crossbar storing the data of the element in the quantum gate matrix. By reading the current through the columns, we can derive the sum of current which is the same as a partial product of the VMM result, such as $\alpha_{1L}\gamma_{1H} + \alpha_{2L}\gamma_{2H}$. The currents are converted into an eight-bit digital representation through the eight-bit ADC and it takes a shift operation suitable for each bit order. For example, $\alpha_{1H}\gamma_{1H} + \alpha_{2H}\gamma_{2H}$ is preceded by an eight-bit shift operation. Finally, the 16-bit Adder sums the shifted data to obtain one element of the VMM matrix, namely, $\alpha_1\gamma_1 + \alpha_2\gamma_2$. The other RSs are processed similarly to obtain the remaining matrix elements.

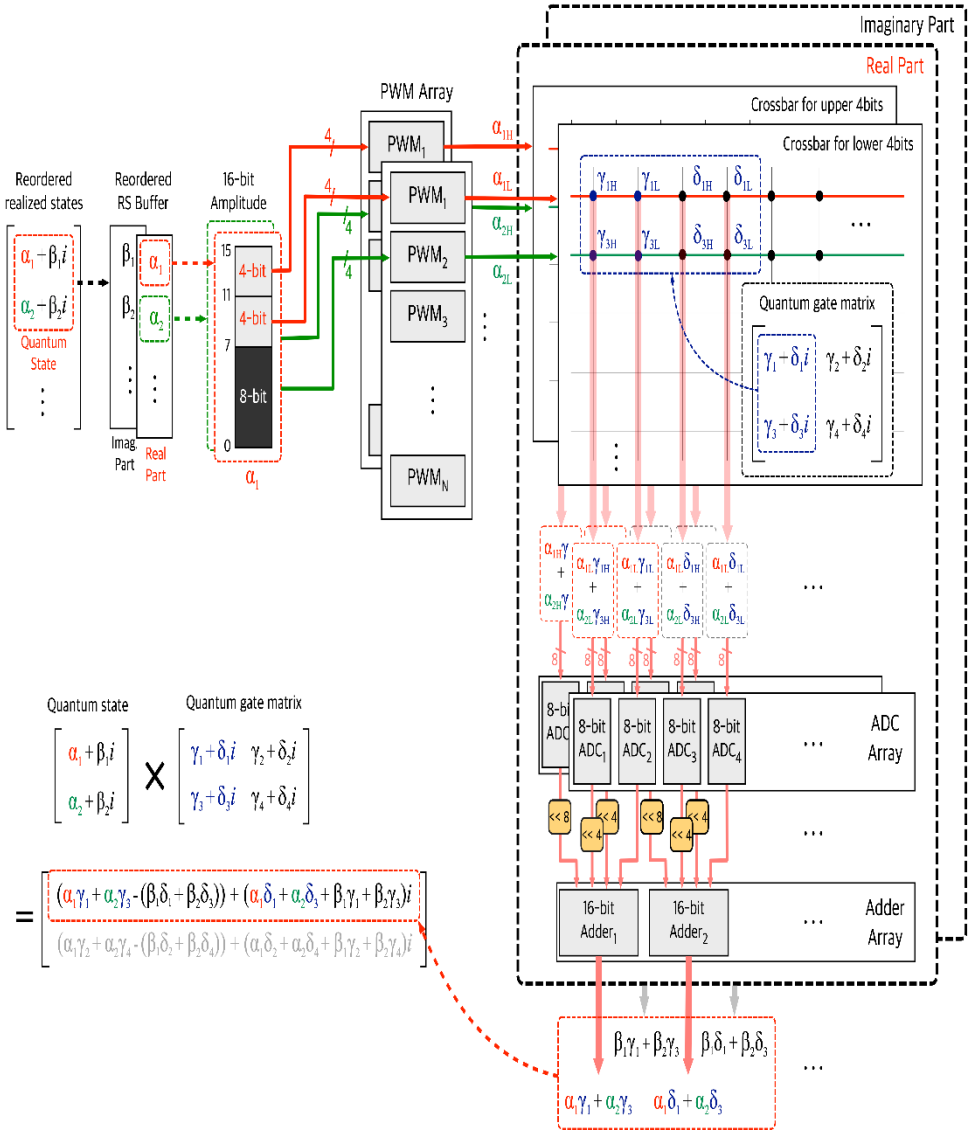


Figure 5. Example of a detailed process inside the QPU when each amplitude of RS uses eight-bit precision. The reordered RS is separated by four-bit and injected into the pulse width modulator (PWM), where it is multiplied by the matrix coefficient stored as a period of amplitude in the cell. After passing the analog-to-digital (ADC) and Adder arrays, the multiplication result (a current) is converted into digital form.

5. Hardware implementation

This section describes the circuit-level implementation of the ReRAM crossbar array and the peripheral circuit.

5.1 ReRAM crossbar array

To implement a memristor crossbar, we require ReRAM technology utilizing HfO2-based one-transistor one-resistor (1T1R) cells. The top and bottom electrodes of the ReRAM cell are connected to the bit line and transistor drain, respectively, and the source line is attached to the transistor source. A prior study [30] reported a ReRAM test chip with a 1T1R crossbar array of area less than $12F^2$(F is the lithography feature size) fabricated via an 18-nm CMOS process. This chip allows the construction of small cell arrays and efficient architectural designs. The precision of the ReRAM cell is a critical factor in the memristor crossbar structure, as it influences the accuracy of the VMM outcome and determines cell conductance. Here we adopt the analog ReRAM technique, which can store multiple bits, rather than digital ReRAM which requires a large number of qubits. Although the analog ReRAM can store n-bit weights in one ReRAM cell, it is usually vulnerable to noise which lowers its accuracy. However, the authors of [21] demonstrated reliable inference accuracy on the Modified National Institute of Standards and Technology database (MNIST) by using two-, four-, and eight-bit

weighted analog ReRAM with eight-bit precision. We similarly evaluated the accuracy of our simulation results by adjusting the precision of the weights (see subsection 6.2.3). To handle negatively valued data, we also adopted the crossbar structure proposed in [34].

5.2 Pulse width modulator

A pulse width modulator (PWM) conveys information by periodically varying the width of the pulse signal. To this end, it varies the duty cycle of the signal, defined as the ratio of the pulse width to the period of the signal. The PWM array is an electronic circuit that generates a sequence of pulse signals with varying duty cycles. The modulated signal retains the frequency and amplitude of the original signal but has a varying duty cycle. The average amplitude of the modulated signal is proportional to the duty cycle, denoting that a higher duty cycle corresponds to a higher amplitude. The conductance of the ReRAM cell can be set according to the pulse width of the modulated input voltage. Moreover, as the write and read operations of the ReRAM cell can be distinguished by pulse amplitude, the PWM arrangement can be shared among the load and calculation operations. When compared with pulse amplitude modulation (PAM), PWM is more suited for applications where precise control of power is required, while PAM is more suited for applications where accurate transmission of analog information is required.

5.3 Analog to digital converter

The ReRAM crossbar array outputs a current value through each of its columns. The currents must be converted to digital format through an ADC. The ADC array is a collection of cooperating ADC circuits that simultaneously convert multiple analog signals to digital signals. The ADC array can thereby process large amounts of analog data quickly and efficiently. The ADC array samples the analog signals at a specific rate and converts each sample to a digital value representing the amplitude of the analog signal at the time of sampling. An ADC array is characterized by the resolution, speed, and accuracy of each ADC circuit in the array. The resolution of an ADC circuit refers to the number of bits representing the digital value of each sample. In a high-resolution circuit, the digital value can represent a larger range of analog amplitudes than in low-resolution circuits, allowing a more accurate conversion of the analog signal into a digital signal.

The resolution similarly affects the accuracy of the VMM operation performed by the ReRAM crossbar array. Therefore, we here set the ADC resolution to eight-bit (see Figure 6). As the amplitude of a qubit ranges from -1 to 1, at least one bit must be assigned to the sign of the amplitude. Referring to the paper [35] and considering that the area and power requirements of the ADC placement consume a large portion of the overall circuit, we applied an eight-bit Flash ADC rather than a successive approximation register (SAR) ADC. The Flash ADC also provides reliable accuracy and optimal hardware, which are more

important than resolution in our present study.

The Flash ADC is sometimes called the parallel ADC because it compares the input signal to a set of reference voltages using a parallel comparison technique. According to [35], the SAR is slightly better than the Flash at resolutions below five-bit. Above five-bit, the Flash covers a larger area and demands more power than the SAR but is more suitable for our architecture because its high-speed sampling rates can accelerate the simulation.

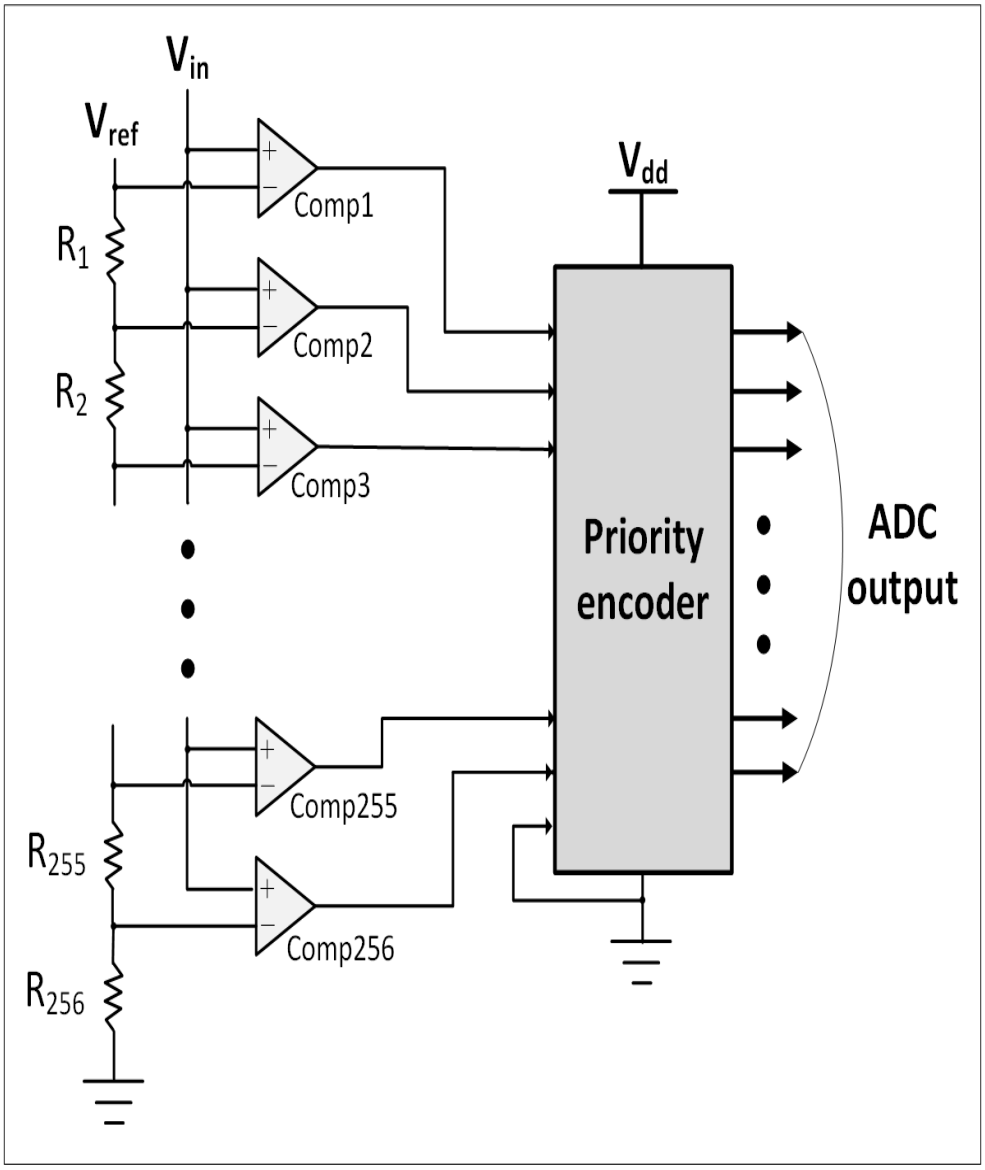


Figure 6. Eight-bit analog-to-digital converter circuit. It consists of 256 resistors and comparators and one priority encoder.

6. Evaluation

This section describes the experimental environment in which we compared the performance (simulation time, hardware resources, and accuracy) of our simulator with those of the Qiskit [36], QuEST [15], and QPlayer [14] simulators.

6.1 Experimental setup

To evaluate the hardware performance, we first constructed an accelerator with 12 QPUs having eight banks of a ReRAM crossbar array to manipulate complex numbers and improve the accuracy of the simulation result. The number of QPUs equals the number of built-in gate types in the LUT because each QPU serves only one type of gate during processing. The size of the ReRAM crossbar array depends on the number of deployed quantum gates. Each ReRAM cell stores eight-bit to satisfy the precision requirements as explained in subsection 5.3. The ReRAM crossbar array was accessed through the Destiny V2 tool [23] based on NVSim, a device and circuit simulator designed for modeling emerging non-volatile memory (NVM) technologies. The ReRAM was configured to simulate the multi-level cell (MLC) configured in [30] (see Table 2). The MLC ReRAM has a 1T1R structure and is manufactured using CMOS technology. One transistor and one ReRAM cell are used for cell selection and data storage, respectively. Also, we refer to the paper [39, 38, 29] for researching the configuration of the

high-precision ReRAM cell. Furthermore, our gate decoder and ROU were designed using Verilog Hardware Description Language (HDL) and synthesized using the 40-nm technology node in the Synopsys design compiler. To verify both the runtime and accuracy of simulations in our proposed architecture, we selected a quantum assembly language (QASM) as the benchmark [19] and exploited the CrossSim simulator, a crossbar simulator that mimics resistive memory in neuromorphic computing. Table 3 summarizes the benchmark quantum circuits used in this experiment. Depending on their number of qubits, the quantum circuits can be separated into three scales (small, medium, and large). Small-, medium-, and large-scale quantum circuits are composed of 2–5 qubits, 6–15 qubits, and 15 or more qubits, respectively. Table 3 presents the properties and numbers of qubits, quantum gates, and CNOT gates in each quantum circuit. Finally, the RSV column denotes the maximum length of the RSV derived from each benchmark, which is crucial for the reordering process in our simulator. All benchmarks were evaluated in a server with Dual Intel Xeon(R) Silver 4214R processor@2.40 GHz (24 cores, 48 threads) and 256 GB of DDR4 main memory.

Table 2. Quantum circuits used as benchmarks in the evaluation

Scale	QASM	Description	Qub	Gat	CN	RSV
Small	lpn_n5	Learning parity with noise	5	11	2	2
	deutsch_n2	Deutsch algorithm with 2 qubits for f(x)=x	2	5	1	4
	qec_en_n5	Quantum repetition on code encoder	5	25	10	8
	teleportation_	Quantum teleportation	3	8	2	8
	toffoli_n3	Toffoli gate	3	18	6	2
Medium	bb84_n8	A quantum key distribution circuit	8	27	0	32
	seca_n11	Shor,s error correction algorithm for teleportation	11	216	84	16
	simon_n6	Simon,s algorithm	6	44	14	16
	ising_n10	Ising model simulation via QC	10	480	90	1,024
	bv_n14	Bernstein Vazirani Algorithm	14	41	13	16,38
Large	cat_state_n22	Cat state	22	10	99	2
	ghz_state_n23	GHZ state preparation and assessment	23	23	22	2
	multiplier_n25	Quantum multiplier	25	372	750	4

Table 3. Parameters of the ReRAM simulation

Appearance	MLC ReRAM
Cell Structure	1T1R (HfO2)
Cell Area	20
Bit per Cell	4
Num of Banks	8
Sensing Scheme	PSRC Current
Optimization Target	Write Latency

6.2 Performance analysis

6.2.1 Simulation time

To evaluate the VMM operation time of the proposed accelerator, we define the equation for read operation based on the clock frequency for one QPU as follows:

$$T_{read} = T_{load} + T_{vmm} + T_{output} \quad (2)$$

T_{load} is the time of loading the RSV from the RSV buffer into the reordered RS buffer in preparation for the VMM operation. Since the reordering time of one RS is the period of one cycle, T_{load} consumes 2N cycles, where N is the number of RSs. The time T_{vmm} of one VMM is 1 because multiplication consumes only one cycle owing to the structure of the crossbar. The number of output operations is the number of inputted RSs and the time T_{output} of generating the output is the time of 2N cycles. Therefore, the total simulation time T_{sim} is then given as:

$$T_{sim} = T_{write} + \sum_{i=1}^{k} (T_{read} + T_{routing}) \quad (3)$$

In Equation 3, T_{sim} is the sum of T_{write}, the time of writing the quantum gate matrix into the QPUs, and the sums of T_{read} and $T_{routing}$, the times of reading and transferring the result from the i^{th}–QPU to the $(i+1)^{th}$–

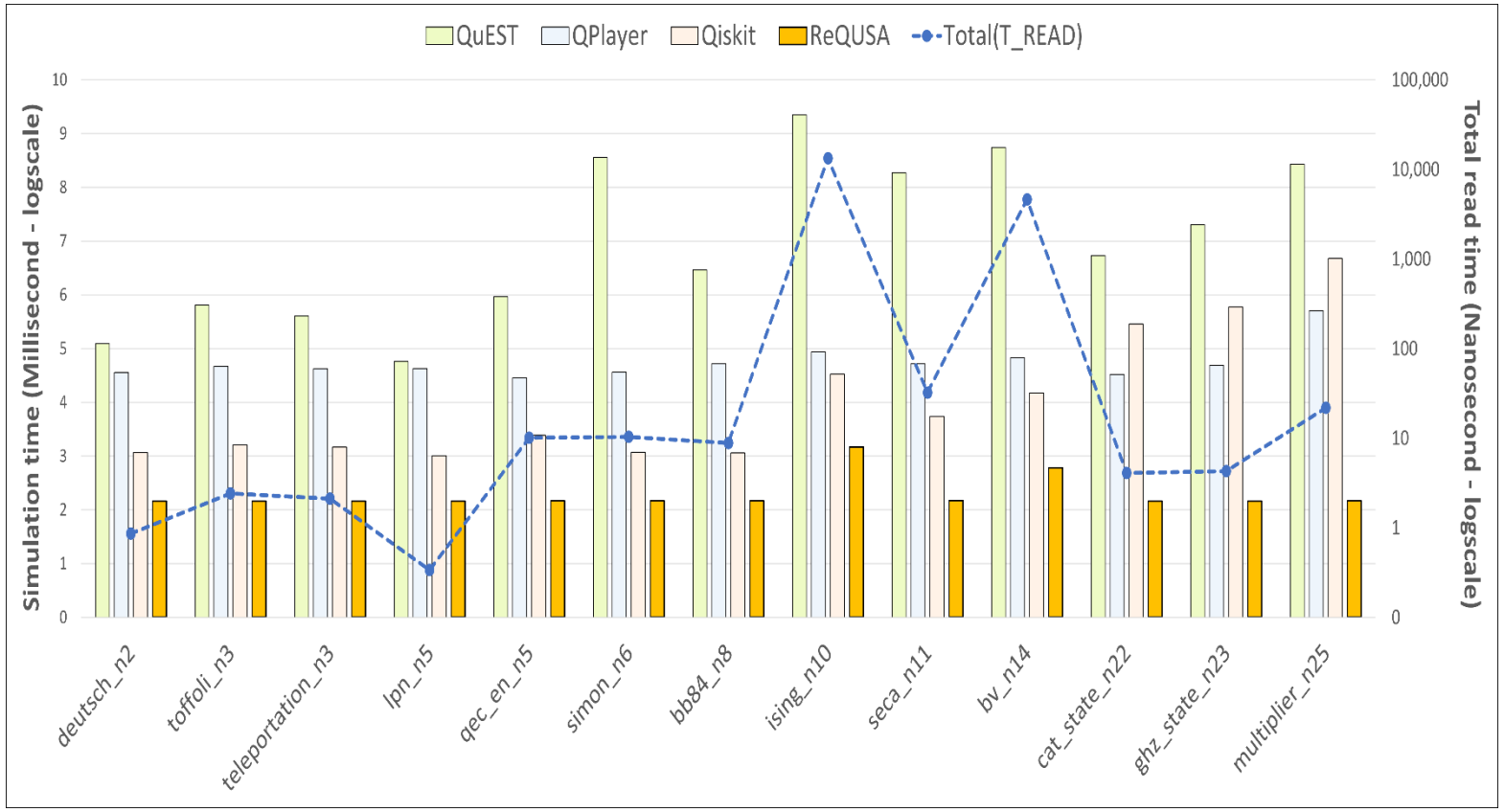
QPU, respectively, where k denotes the number of quantum gates. To estimate $T_{routiing}$, we must first examine the flit size and packet length of the router. A packet defines a complete unit of data delivered through the network, whereas a flit is the smallest unit of transmissible data, which (as a piece of the packet) can be transmitted more efficiently across the network than a packet. The router in our architecture produces a flit size of 64-bit and a packet length of 16- flit [17]. Therefore, one packet can transfer 1,024-bit of data (16 − flits × 64 − bit). The proposed router can also be operated at a clock frequency of 500 MHz in the NoC structure, meaning that all circuits inside the QPU and ReQUSA can be synchronized with a clock cycle time of 2.0ns, i.e., T_{clock_cycle}. Therefore, $T_{routing}$ is calculated as follows:

$$T_{routing} = \lceil L_{rs}/(S_{packet}/S_{rs}) \rceil \times T_{clock\_cycle} \quad (4)$$

where S_{packet} and S_{rs} denote the data size of one packet and one RS, respectively. T_{clock_cycle} is the clock cycle time, and L_{rs} is the length of the output data of the current QPU. One RS contains 32-bit of data, 16-bit each for the real and imaginary parts of the complex number. The division of S_{packet} by S_{rs} gives the maximum number of RSs that can be contained in one packet. Moreover, after dividing L_{rs} by the obtained maximum number, we can determine the number of packets to be transmitted. Finally, multiplying this result by T_{clock_cycle} gives the time required for routing. Figure 7 compares the quantum computer simulation time of ReQUSA based on Equation 3 with the simulation times of other simulation tools (QuEST, QPlayer, and Qiskit) using the

benchmark circuits listed in Table 2. The times are presented on the log10 scale. At this time, in order to unify the environment of all simulators, we only applied OpenMP to QuEST without GPU acceleration (GPGPU) technology. On all benchmarks, the simulation time of ReQUSA far exceeded those of the other simulators. On average, the runtime of ReQUSA was $\times 10^4$ faster than that of QuEST, which exhibited the longest simulation time on all benchmarks, and $\times 10^3$ and $\times 10^2$ faster than those of QPlayer and Qiskit, respectively. The simulation times of QPlayer and ReQUSA, which use RS techniques, do not gradually increase with an increasing number of qubits because a quantum circuit with a large number of qubits cannot assure a large number of entanglements as same as the RS, the main factor of the VMM operation. Thus, 'cat_state_n22' with more qubits but less entanglement runs faster than the 'ising_n10' circuit, which consists of only 10 qubits but has many entanglements. More precisely, when analyzing the T_{sim} of ReQUSA, we found that even if the variability of T_{sim} is small in overall time across all benchmarks, there is significant variability in T_{read} depending on the number of RSs like in the case of 'ising_n10'. This phenomenon may seem like the RS method does not affirm accelerating simulation in a large-scale quantum system, however, it is clear that the ReQUSA shows much better performance than other simulators.

Figure 7. Comparison of simulation times of QuEST, QPlayer, Qiskit, and our proposed accelerator ReQUSA. The blue dashed line, Total ($\boldsymbol{T_{read}}$), indicates the total time for the read operation in the quantum computer simulation.



6.2.2 Hardware resource

As all QPUs have the same circuit configuration, measuring a single QPU is sufficient for checking the hardware resources. The peripheral circuit includes all components of the QPU except the ReRAM crossbar array (ADC and PWM arrays). Table 4 shows the hardware resources (area, energy, and leakage power) of the QPUs. The performance was evaluated at 16-bit fixed precision while the crossbar array size was increased from 4×32 to 128×1024. Doubling the size of the crossbar increased the total area at the same rate. More specifically, doubling the crossbar- array size increased the area of the peripheral circuit. It also increased the area of the crossbar arrangement, but the increase was less than 1% of the total area under each condition. Meanwhile, the energy was at least doubled with increasing crossbar-array size and later increased by approximately fourfold. The crossbar arrangement consumed more than 90% of the total energy. The leakage power behaved similarly to the energy consumption. The most important point is that at any array size, the clock cycle time could be 2.0ns. Thus, regardless of the area of the peripheral circuit and crossbar, the hardware can operate at the same clock frequency. Expanding the circuit size did not affect the performance of the accelerator.

Table 4. QPU hardware resource result when the crossbar size is doubled

Crossbar size	Area (mm^2)			Energy (nJ)			Leakage power (mW)		
	Array	Peri	Total	Array	Peri	Total	Array	Peri	Total
4x32	0.004	1.74E+01	17.400	0.031	2.64E+01	26.400	1.200	4.73E-04	1.200
8x64	0.018	3.48E+01	34.800	0.058	5.28E+01	52.800	0.936	8.99E-04	0.937
16x128	0.029	6.95E+01	69.600	0.078	1.06E+02	106.000	4.134	1.89E-03	4.140
32x256	0.102	1.39E+02	139.000	0.290	2.11E+02	211.000	7.695	4.21E-03	7.700
64x512	0.462	2.78E+02	279.000	4.771	4.22E+02	427.000	18.610	8.96E-03	18.600
128x1024	0.719	5.56E+02	557.000	1.789	8.44E+02	846.000	88.000	1.91E-02	87.900

6.2.3 Simulation accuracy

To evaluate the accuracy of our proposed method, we utilized the VMM method presented in Figure 5. The bit accuracy ranged from four- to 16-bit, reserving one bit for the sign of the quantum state and the remaining bits for its fractional part. As the quantum state values ranged from 1 to -1, we opted to disregard the integer part of the state. The quantum state value was directly converted from 1.0 to 0.999..., thereby sacrificing one bit of accuracy to include an additional fraction bit. Table 5 shows the accuracies of the ReQUSA simulations on various quantum circuits ranging from small to large scales. Since the accuracy of the simulation result is over 90% to be considered a reliable value, we marked the corresponding values in grayscale as shown in Table 5. In the table, the simulation accuracy results above 90% are highlighted in blue. The accuracy was calculated by comparing the RS value to the Qiskit value. Simulations using four- to eight-bit typically obtained a low accuracy owing to the low bit precision and consequent inaccurate floating-point representation during VMM operations. The accuracy of ReQUSA is primarily determined by the circuit's gate, with the number of qubits playing a secondary role. For instance, despite having only three qubits, the circuit 'toffoli_n3' shows a lower accuracy than the 'multiplier_n25' circuit. Because the 'toffoli_n3' contains multiple Hadamard gates and the 'multiplier_n25' does not. This demonstrates that ReQUSA's accuracy is compromised when multiplication involves numerous floating-point numbers. Another example is the fully-

superposed circuit 'bv_n14', in which an H gate is applied to each qubit. Simulations of this circuit, which has 214 RSs, yielded no results at a bit precision lower than 10 because the result approached 0 during VMM operations at very low bit precisions. On this circuit, the outcome is much more dependable at precisions of 14-bit and higher.

Table 5. Simulation accuracies of different quantum circuits with different bit precisions on ReQUSA. The accuracy is defined as the value of the vector relative to the Qiskit simulation result vector. Accuracies above 0.9 are grayed out.

Quantum circuit	Bit precision and Accuracy									
	4	5	6	7	8	9	10	11	12	13
deustch_n2	0.078	0.665	0.758	0.916	0.939	0.98	0.99	0.995	0.997	0.998
toffoli_n3	0	0	0.166	0.592	0.79	0.905	0.952	0.975	0.987	0.993
teleportation_n3	0.047	0.441	0.649	0.858	0.912	0.97	0.984	0.992	0.995	0.997
lpn_n5	0.691	0.869	0.92	0.967	0.981	0.993	0.996	0.998	0.999	0.999
simon_n6	0	0	0.015	0.505	0.743	0.879	0.943	0.97	0.985	0.991
bb84_n8	0	0	0	0.351	0.615	0.839	0.916	0.961	0.977	0.988
seca_n11	0	0	0	0	0.239	0.657	0.825	0.91	0.954	0.976
bv_n14	0	0	0	0	0	0	0.002	0.279	0.729	0.821
cat_state_n22	0.273	0.322	0.363	0.717	0.863	0.936	0.968	0.984	0.992	0.997
ghz_state_n23	0.273	0.322	0.333	0.703	0.857	0.933	0.967	0.983	0.991	0.995
multiplier_n25	0	0.175	0.575	0.784	0.891	0.945	0.972	0.986	0.993	0.996

	14	15	16
	0.999	0.999	0.999
	0.996	0.998	0.999
	0.998	0.999	0.999
	0.999	0.999	0.999
	0.995	0.998	0.998
	0.992	0.997	0.998
	0.986	0.994	0.996
	0.91	0.958	0.978
	0.997	0.999	0.999
	0.997	0.998	0.999
	0.998	0.999	0.999

7. Conclusion

This paper proposed our quantum computer simulation accelerator ReQUSA based on ReRAM for high-speed VMM operations. The proposed architecture reduces the simulation time by applying the RS method to reduced VMMs, thereby leveraging the advantages of the ReRAM crossbar array. Our designed architecture includes multiple QPUs consisting of the ReRAM crossbar array, a reordering unit for the RS method, and peripheral circuits such as the PWM, ADC, and Adder array. We further confirmed that our architecture outperforms the existing simulators. Specifically, it decreases the simulation time on average by $\times 10^4$ from that of QuEST, and by a least $\times 10^2$ and $\times 10^3$ from those of Qiskit and QPlayer, respectively. A reasonably correct result was obtained at eight-bit precision in a non-fully superposed quantum circuit and the accuracy was improved by extending the bit precision to 16-bit. Consequently, the ReRAM-based accelerator ReQUSA promises to significantly reduce the simulation time while providing a minimized hardware resource and accurate results in quantum computer simulations.

References

[1] S.-S. Sheu *et al.*, 'A 4Mb embedded SLC resistive-RAM macro with 7.2ns read-write random-access time and 160ns MLC-access capability', in *2011 IEEE International Solid-State Circuits Conference*, 2011, pp. 200–202.

[2] S. Mittal, 'A Survey of ReRAM-Based Architectures for Processing-In-Memory and Neural Networks', *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 75–114, 2019.

[3] M. J. Marinella *et al.*, 'Multiscale Co-Design Analysis of Energy, Latency, Area, and Accuracy of a ReRAM Analog Neural Training Accelerator', *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 86–101, 2018.

[4] M. Hu *et al.*, 'Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication', Austin, Texas, 2016.

[5] S. A. Ghasemi, B. Jahannia, and H. Farbeh, 'GraphA: An efficient ReRAM-based architecture to accelerate large scale graph processing', *Journal of Systems Architecture*, vol. 133, p. 102755, 2022.

[6] D. Lelmini and H.-S. P. Wong, 'In-memory computing with resistive switching devices', *Nat Electron 1*, 2019.

[7] Y. Li, X. Chen, X. Zhao, Y. Yang, and H. Liu, 'Round-trip latency prediction for memory access fairness in mesh-based many-core architectures', *IEICE Electronics Express*, vol. 11, no. 24, pp. 20141027–20141027, 2014.

[8] X.-C. Wu *et al.*, 'Full-State Quantum Circuit Simulation by Using Data Compression', in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, 2019.

[9] K.-S. Jin and G.-I. Cha, 'QPlayer: Lightweight, scalable, and fast quantum simulator', *ETRI Journal*, vol. n/a, no. n/a.

[10] J. M. Correll *et al.*, 'An 8-bit 20.7 TOPS/W Multi-Level Cell ReRAM-based Compute Engine', in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2022, pp. 264–265.

[11] P. Yao *et al.*, ‘Face classification using electronic synapses’, *Nature communications*, vol. 8, no. 1, p. 15199, 2017.

[12] J.-H. Yoon, M. Chang, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, and A. Raychowdhury, ‘29.1 a 40nm 64kb 56.67 tops/w read-disturb-tolerant compute-in-memory/digital rram macro with active-feedback-based read and in-situ write verification’, in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, 2021, vol. 64, pp. 404–406.

[13] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, ‘QASMBench: A Low-Level Quantum Benchmark Suite for NISQ Evaluation and Simulation’, *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–26, 2023.

[14] A. W. Harrow, A. Hassidim, and S. Lloyd, ‘Quantum Algorithm for Linear Systems of Equations’, *Phys. Rev. Lett.*, vol. 103, p. 150502, Oct. 2009.

[15] P. W. Shor, ‘Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer’, *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[16] L. K. Grover, ‘A Fast Quantum Mechanical Algorithm for Database Search’, in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, Philadelphia, Pennsylvania, USA, 1996, pp. 212–219.

[17] A. Shafiee *et al.*, ‘ISAAC: A Convolutional Neural Network Accelerator with in-Situ Analog Arithmetic in Crossbars’, *SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, Jun. 2016.

[18] A. Shafiee *et al.*, ‘ISAAC: A Convolutional Neural Network Accelerator with in-Situ Analog Arithmetic in Crossbars’, in *Proceedings of the 43rd International Symposium on Computer Architecture*, 2016, pp. 14–26.

[19] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston, ‘A Lightweight Fault-Tolerant Mechanism for Network-on-Chip’, in *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, 2008, pp. 13–22.

[20] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[21] B. Verbruggen, J. Craninckx, M. Kuijk, P. Wambacq, and G. Van der Plas, ‘A 2.2 mW 1.75 GS/s 5 Bit Folding Flash ADC in 90 nm Digital CMOS’,

IEEE Journal of Solid-State Circuits, vol. 44, no. 3, pp. 874–882, Mar. 2009.

[22] Q. Wang, Y. Kim, and P. Li, 'Neuromorphic Processors with Memristive Synapses: Synaptic Interface and Architectural Exploration', *ACM Journal on Emerging Technologies in Computing Systems*, vol. 12, no. 4, pp. 1–22, Jul. 2016.

[23] I.-M. Yi, N. Miura, H. Fukuyama, and H. Nosaka, 'A 15.1-mW 6-GS/s 6-bit Single-Channel Flash ADC With Selectively Activated 8× Time-Domain Latch Interpolation', *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 455–464, Feb. 2021.

[24] A. Ankit *et al.*, 'PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference', in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 715–731.

[25] P. Chi *et al.*, 'PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory', in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27–39.

[26] J.-H. Yoon, M. Chang, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, and A. Raychowdhury, '29.1 A 40nm 64Kb 56.67TOPS/W Read-Disturb-Tolerant Compute-in-Memory/Digital RRAM Macro with Active-Feedback-Based Read and In-Situ Write Verification', in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, 2021, pp. 404–406.

[27] T. Li, N. Jing, J. Jiang, Q. Wang, Z. Mao, and Y. Chen, 'A Novel Architecture Design for Output Significance Aligned Flow with Adaptive Control in ReRAM-based Neural Network Accelerator', *ACM Transactions on Design Automation of Electronic Systems*, vol. 27, no. 6, pp. 1–22, Nov. 2022.

[28] J. M. Correll *et al.*, 'An 8-bit 20.7 TOPS/W Multi-Level Cell ReRAM-based Compute Engine', in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2022, pp. 264–265.

[29] T. H. Johnson, S. R. Clark, and D. Jaksch, 'What is a quantum simulator?', *EPJ Quantum Technology*, vol. 1, no. 1, p. 10, Dec. 2014.

[30] P. Yao *et al.*, 'Face classification using electronic synapses', *Nature*

Communications, vol. 8, no. 1, p. 15199, May 2017.

[31] S. Mittal, R. Wang, and J. Vetter, 'DESTINY: A Comprehensive Tool with 3D and Multi-Level Cell Memory Modeling Capability', *Journal of Low Power Electronics and Applications*, vol. 7, no. 3, p. 23, Sep. 2017.

[32] Y. Li *et al.*, 'A Survey of MRAM-Centric Computing: From Near Memory to In Memory', *IEEE Transactions on Emerging Topics in Computing*, pp. 1–12, 2022.

[33] W.-C. Chien *et al.*, 'Multi-level 40nm WOX resistive memory with excellent reliability', in *2011 International Electron Devices Meeting*, 2011, p. 31.5.1-31.5.4.

[34] D. S. Steiger, T. Häner, and M. Troyer, 'ProjectQ: An Open Source Software Framework for Quantum Computing', *Quantum*, vol. 2, p. 49, Jan. 2018.

[35] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, 'QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation'. arXiv, May-2022.

[36] C. Huang *et al.*, 'Rescuing ReRAM-based Neural Computing Systems from Device Variation', *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 1, pp. 1–17, Jan. 2023.

[37] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, 'QuEST and High Performance Simulation of Quantum Computers', *Scientific Reports*, vol. 9, no. 1, p. 10736, Jul. 2019.

[38] H. De Raedt *et al.*, 'Massively parallel quantum computer simulator, eleven years later', *Computer Physics Communications*, vol. 237, pp. 47–61, Apr. 2019.

[39] K. D. Raedt *et al.*, 'Massive Parallel Quantum Computer Simulator'.

[40] R. LaRose, 'Distributed Memory Techniques for Classical Simulation of Quantum Circuits'.

[41] S.-S. Sheu *et al.*, 'A 4Mb embedded SLC resistive-RAM macro with 7.2ns read-write random-access time and 160ns MLC-access capability', in *2011 IEEE International Solid-State Circuits Conference*, 2011, pp. 200–202.

[42] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston, 'A Lightweight Fault-Tolerant Mechanism for Network-on-Chip', in *Second ACM/IEEE*

International Symposium on Networks-on-Chip (nocs 2008), 2008, pp. 13–22.

[43] J. Jose, B. Nayak, K. Kumar, and M. Mutyam, 'DeBAR: Deflection Based Adaptive Router with Minimal Buffering', in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, 2013, pp. 1583–1588.

[44] M. A. Khan and A. Q. Ansari, 'Design of 8-Bit Programmable Crossbar Switch for Network-on-Chip Router', in *Trends in Network and Communications*, vol. 197, D. C. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, and D. Nagamalai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 526–535.

[45] G. Krishnan, S. K. Mandal, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao, 'Impact of On-chip Interconnect on In-memory Acceleration of Deep Neural Networks', *ACM Journal on Emerging Technologies in Computing Systems*, vol. 18, no. 2, pp. 1–22, Apr. 2022.

[46] X. Zhou, P. Hao, and D. Liu, 'PCCNoC: Packet Connected Circuit as Network on Chip for High Throughput and Low Latency SoCs', *Micromachines*, vol. 14, no. 3, p. 501, Feb. 2023.

[47] 'MoDe-X: Microarchitecture of a Layout-Aware Modular Decoupled Crossbar for On-Chip Interconnects', *IEEE Transactions on Computers*, vol. 63, no. 3, pp. 622–636, Mar. 2014.

[48] W.-C. Tsai, Y.-C. Lan, Y.-H. Hu, and S.-J. Chen, 'Networks on Chips: Structure and Design Methodologies', *Journal of Electrical and Computer Engineering*, vol. 2012, pp. 1–15, 2012.

[49] S. Onsori and F. Safaei, 'Performance Enhancement of Routers in Networks-on-Chip Using Dynamic Virtual Channels Allocation', 2014.

[50] A. M. R, A. N. Subrahmanya, and A. D'Souza, 'Performance Analysis of Mesh-based NoC's on Routing Algorithms', *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 8, no. 5, p. 3368, Oct. 2018.

[51] R. James, J. Jose, and J. K. Antony, 'Smart Port Allocation for Adaptive NoC Routers', in *2015 28th International Conference on VLSI Design*, 2015, pp. 475–480.

[52] R. Wille, R. Van Meter, and Y. Naveh, 'IBM's Qiskit Tool Chain: Working with and Developing for Real Quantum Computers', in *2019 Design,*

Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 1234–1240.

[53] M. Courbariaux, Y. Bengio, and J.-P. David, 'Training deep neural networks with low precision multiplications', *arXiv: Learning*, 2014.

[54] N. Khammassi, I. Ashraf, X. Fu, C. G. Almudever, and K. Bertels, 'QX: A high-performance quantum computer simulation platform', in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 464–469.

[55] J. Preskill, 'Quantum Computing in the NISQ era and beyond', *Quantum*, vol. 2, p. 79, Aug. 2018.

[56] National Academies of Sciences Engineering, Medicine, and Others, 'Quantum computing: progress and prospects', 2019.

[57] A. Avila, A. Maron, R. Reiser, M. Pilla, and A. Yamin, 'GPU-Aware Distributed Quantum Simulation', in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, Gyeongju, Republic of Korea, 2014, pp. 860–865.

[58] J. Doi, H. Takahashi, R. Raymond, T. Imamichi, and H. Horii, 'Quantum Computing Simulator on a Heterogenous HPC System', in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, Alghero, Italy, 2019, pp. 85–93.

[59] S. Agarwal *et al.*, 'Resistive memory device requirements for a neural algorithm accelerator', in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 929–938.

[60] H. Ji, L. Song, L. Jiang, H. Li, and Y. Chen, 'ReCom: An efficient resistive accelerator for compressed deep neural networks', in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 237–240.

[61] K. Pagiamtzis and A. Sheikholeslami, 'Content-addressable memory (CAM) circuits and architectures: a tutorial and survey', *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[62] Y. C. Shin, R. Sridhar, V. Demjanenko, P. W. Palumbo, and S. N. Srihari, 'A special-purpose content addressable memory chip for real-time image processing', *IEEE Journal of Solid-State Circuits*, vol. 27, no. 5, pp. 737–744, 1992.

[63] P. W. Shor, ‘Algorithms for quantum computation: discrete logarithms and factoring’, in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[64] S. Bravyi and D. Gosset, ‘Improved Classical Simulation of Quantum Circuits Dominated by Clifford Gates’, *Phys. Rev. Lett.*, vol. 116, p. 250501, Jun. 2016.

[65] J. Niwa, K. Matsumoto, and H. Imai, ‘General-purpose parallel simulator for quantum computing’, *Phys. Rev. A*, vol. 66, p. 062317, Dec. 2002.

[66] T. Häner and D. S. Steiger, ‘0.5 Petabyte Simulation of a 45-Qubit Quantum Circuit’, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, 2017.

[67] Hoyer, Neerbek, and Shi, ‘Quantum Complexities of Ordered Searching, Sorting, and Element Distinctness’, *Algorithmica*, vol. 34, no. 4, pp. 429–448, Nov. 2002.

[68] L. K. Grover, ‘Quantum Mechanics Helps in Searching for a Needle in a Haystack’, *Phys. Rev. Lett.*, vol. 79, pp. 325–328, Jul. 1997.

[69] E. Bernstein and U. Vazirani, ‘Quantum Complexity Theory’, *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1411–1473, 1997.

[70] D. Deutsch and R. Penrose, ‘Quantum theory, the Church–Turing principle and the universal quantum computer’, *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[71] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, ‘Quantum machine learning’, *Nature*, vol. 549, no. 7671, pp. 195–202, Sep. 2017.

[72] D. Peral García, J. Cruz-Benito, and F. José García-Peñalvo, ‘Systematic Literature Review: Quantum Machine Learning and its applications’, *arXiv e-prints*, p. arXiv:2201.04093, Jan. 2022.

[73] L. Song, X. Qian, H. Li, and Y. Chen, ‘PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning’, in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 541–552.

[74] Y. Long, X. She, and S. Mukhopadhyay, ‘Design of Reliable DNN Accelerator with Un-reliable ReRAM’, in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1769–1774.

[75] E. Farhi, J. Goldstone, and S. Gutmann, ‘A Quantum Algorithm for the Hamiltonian NAND Tree’, *arXiv [quant-ph]*. 2007.

[76] S. N. Truong and 민경식, ‘New Memristor-Based Crossbar Array Architecture with 50-% Area Reduction and 48-% Power Saving for Matrix-Vector Multiplication of Analog Neuromorphic Computing’, *JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE*, vol. 14, no. 3, pp. 356–363, 2014.