

## RESEARCH ARTICLE

# A Software Platform for Programmable Linear Optical Quantum Computer

YONG KWON<sup>1</sup> AND BYUNG-SOO CHOI<sup>2</sup>, (Member, IEEE)

<sup>1</sup>Department of Physics, Pukyong National University, Busan 48513, South Korea

<sup>2</sup>Department of Scientific Computing, Pukyong National University, Busan 48513, South Korea

Corresponding author: Byung-Soo Choi (bschoi@pknu.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korean Government Ministry of Science and ICT (MSIT) under Grant 2020K1A3A1A78087782, and in part by Pukyong National University Research Fund in 2022 under Grant 202203540001.

**ABSTRACT** Optical quantum computers represent qubits as photons created by laser and implement arbitrary quantum operations through Mach-Zehnder interferometers (MZIs) which is called linear optical quantum computing (LOQC). Since the number of qubits and quantum system is getting bigger, we need a scalable and universal quantum computing platform based on LOQC. In this work, we propose a software framework which can support a universal programming with flexibility of LOQC hardware configuration. The framework consists of four parts such as decomposer, mapper, controller, and user-interface controller. As an example, we show a use case of this framework for Bell state generation on Reck et al.'s MZI network.

**INDEX TERMS** Quantum computer, software, linear optical quantum computing.

## I. INTRODUCTION


Quantum computers demand a perfect simulation of quantum phenomena due to the limitations of classical computers [1]. Although the computers surpass classical ones in solving calculation problems in a computational aspect [2] and proclaimed by physical hardware [3], they orient to the summit of calculation to quantum phenomena that can not be represented perfectly in classical computers. For instance, the Google AI team recently announced the discovery of anyonic properties in a surface code adapting to their superconducting processor [4].

However, despite the distinguished computing power and compatibility for representing physical phenomena, quantum computers do not fully work themselves so traditional computers attaching to the quantum one should be inevitable as an auxiliary purpose for controlling quantum devices [5]. From a broad point of view, the devices can be regarded as quantum hardware which is connected to the PC and controlled by software. End users make a quantum algorithm code similar to the general process of coding in classical computers. In the matter of gate-based quantum computers, their software

currently supports quantum gate-level programming and users should write their program with quantum gate sets [6]. Likewise, an intermediate-level quantum computer requires software to run various quantum algorithms and to send information for running quantum chips in their physical platform.

An optical platform, which represents another physical system for quantum information processing, has been attempted to do quantum computing. Traditionally, it has started to research photons in quantum mechanics as a field called quantum optics [7], and applying photons to quantum computing was attempted about three decades ago [8]. In 2015, an attempt to integrate them into one chip was made that contains beamsplitters and phase shifters [9]. Thus, attempts to apply quantum computing in the optical system have been continuing.

Unfortunately, a hierarchical software framework for optical quantum computing is incompletely defined. Referring to the result of optical experiments, one-qubit and two-qubit quantum gates have been implemented at the laboratory level [10], [11], [12], [13]. Also, the experiment presents verification results acquired by quantum state tomography or process tomography, which can be as a performance evaluation for quantum processors. Still, although these

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Huang .

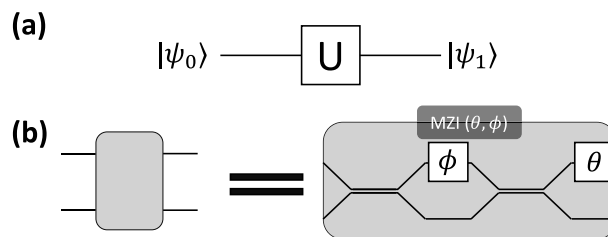
references show sufficient evidence to implement optical quantum computing, the necessary hierarchical structure from a computer perspective has not yet been established.

Since well-systematic software frameworks already exist such as QISKIT, we expected that it could be available to use them directly on photonic hardware. However, the approach might be inapplicable because physical behavior in superconducting and photonic processors shows significant differences. To clarify the argument, we express the properties of photonic qubits and quantum operations and address the reason by comparing the characteristics with a superconducting quantum computer platform. We then propose a software framework for universal quantum computer programming on arbitrary linear optical architecture.

The paper is structured as follows: Section II introduces the software platform for superconducting quantum computers reviewing the implementation of qubits and operators. We then investigate quantum computing way in an optical system to find the requirements of building up the software. We also present related software frameworks for quantum computing in photonics and review functionalities and limitations. In Sec. III, we describe linear optical quantum computing (LOQC) hardware properties extending to the brief explanations in the previous section. With this background, we clarify the reason an optical quantum chip requires its optimized software in Sec. IV. We present and explain the essential three modules, decomposer, mapper, and controller, and support our statement by comparing operation ways of superconducting quantum computing (SCQC) and LOQC. After that, we describe the overall structure of the proposed software framework and address specifically the operation of each module in Sec. V. It presents the operation of software modules, the necessary module types for optical computers, the purpose of each module, and the interconnections between them. In the remaining sections, we report the reversibility in LOQC and the analysis of required resources of time and quantity in Sec. VI, and we describe the software execution with a simple example, the Bell state algorithm for the sake of understanding the suggested platform in Sec. VII. Lastly, we conclude our work and present prospects in Sec. VIII.

## II. RELATED WORK

Superconducting-based quantum computers are one of the rapidly developed platforms in the quantum computers domain. Qubits are implemented by applying the energy level difference between two quantum states stemming from the Hamiltonian of their physical system [14], [15]. The operation of a quantum gate is manifested by applying a pulsed signal to the qubits, especially the two-qubit gate is additionally represented by the cross-resonance effect [16], [17]. With these physical properties, IBM has developed an open-source platform called QISKIT that specializes in superconducting-based quantum computers. This framework is a tightly organized system for quantum computing. It makes any user-defined quantum algorithms automatically executable. Even they recently made it possible to manipulate



**FIGURE 1.** (a) A quantum circuit representation for arbitrary unitary operation  $U$ . A qubit prepared on the state  $|\psi_0\rangle$  evolves to state  $|\psi_1\rangle$  from the operation  $U$ . (b) A single MZI unit. MZI composed two lines and one plate colored in grey, the former representing waveguides, and the latter including two beamsplitters and two phase shifters. Beamsplitters contacting both waveguides and phase shifters located with white boxes labeled  $\theta$  and  $\phi$ .

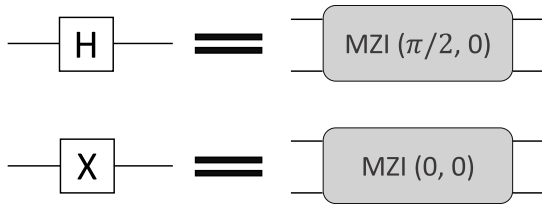
quantum chips on pulse level for quantum operations by QISKIT Pulse [18].

In contrast to the superconducting system, optical computing has significantly different characteristics to operate its hardware for information processing. One of the noteworthy properties of the system is that photons from a laser apparatus dynamically move along the optical path, and optical detectors located at the end of the path collect photons for measurement. In other words, the qubits can only travel a finite number of paths, and the length of the path limits the number of quantum operations. However, a variety of quantum operations can be easily implemented by adjusting the angles of phase shifters, which are the linear optical elements. This hardware properties are explained in more detail in the next section.

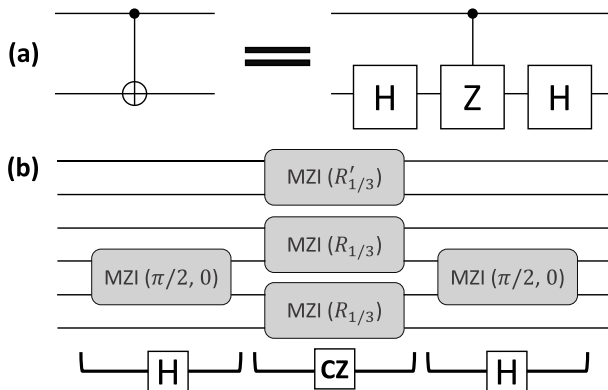
Likewise, the physical properties are significantly different, software platform for LOQC has to be configured to reflect photonic characteristics. Several studies can be found on the implementation of optical computing. The reference [19] presents a transpilation pipeline to translate from QASM to a graph representation related to LOQC. Another reference [20] is open-source software that supports the simulation of photonic quantum circuits. From the point of their features, even though they are grouped into the LOQC system, they are somewhat different in the manipulation of the actual hardware. In other words, our topic centers on the ability to reconfigure photonic quantum chips to perform various quantum computations. The first case is therefore irrelevant, while the second case merely helps to illustrate a linear photonic circuit. The software platform we present supports the feature, and the structure is presented in Sec. V.

## III. LINEAR OPTICAL QUANTUM COMPUTING HARDWARE

LOQC operates a universal quantum computer that qubits are considered photons as information carriers, and quantum operators are implemented to the manipulations of linear optical elements for processing quantum information [8], [9], [21], [22]. The linear elements comprise beamsplitters and phase shifters for processing quantum information. While beamsplitters and phase shifters are important elements of LOQC, they alone are unable to present a universal



**FIGURE 2.** Implementation of one-qubit gates in LOQC. A circuit representation for each gate equals MZI with a specific phase value. (a) Hadamard gate (b) Pauli-X gate.



**FIGURE 3.** Implementation of two-qubit gates in LOQC. (a) Gate decomposition of *CNOT* with the combination of *Hadamard* and *Controlled Z* (b) *CNOT* gate implementation with MZIs. Left and right MZI along three stacked MZIs equals to *Hadamard*, and the center part refers to the *CZ* gate.

quantum computing. To achieve the computing scheme, Mach-Zehnder interferometer (MZI), composed of linear elements, is necessary as demonstrated in Fig. 1. By setting MZI as a base unit, we define a qubit and logical quantum gates by adjusting phase values. Consequently, we can perform complex quantum computations by building a network of MZIs.

A qubit definition in LOQC is distinct from a quantum circuit representation. Referring to Fig. 1(a), a quantum state  $|\psi_0\rangle$  evolves to  $|\psi_1\rangle$  from any unitary gate  $U$ . It is a known fact that qubits pass through a single quantum wire. In LOQC, the qubit is implemented to the light traveling the two pairs of solid lines called “waveguides”, which is located on both sides of the grey plate. A photon passes through the separate two waveguides on the state

$$|\psi_0\rangle = (w_0\hat{a}_{w_0}^\dagger + w_1\hat{a}_{w_1}^\dagger) |\Omega\rangle \quad (1)$$

where  $\hat{a}_{w_0}^\dagger, \hat{a}_{w_1}^\dagger$  are creation operators,  $w_0, w_1$  are a probability amplitude of each waveguide which satisfies  $|w_0|^2 + |w_1|^2 = 1$ , and  $|\Omega\rangle$  denotes a vacuum state. Thus, a qubit is defined as light traveling through a pair of waveguides in the state of  $|\psi_0\rangle$ .

Quantum gates on the linear optical system are implemented with a MZI containing linear optical elements. These elements are characterized as linear and they can construct any arbitrary one-qubit unitary operation [8], [22], [23]. Concerning Fig. 1(a), each split beam penetrates the grey plate. While the beam passes the plate, a unitary operator

$U$  is implemented by adjusting the values  $\theta$  and  $\phi$  of the phase shifter, respectively. The unitary operation matrix for the structure in Fig. 1 is

$$\begin{aligned} U &= P_\phi B P_\theta B \\ &= \frac{1}{2} \begin{pmatrix} e^{i\phi} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \begin{pmatrix} e^{i\theta} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \\ &= ie^{i\theta/2} \begin{pmatrix} e^{i\phi} \sin(\theta/2) & e^{i\phi} \cos(\theta/2) \\ \cos(\theta/2) & -\sin(\theta/2) \end{pmatrix} \end{aligned} \quad (2)$$

where  $P_\phi, P_\theta$  are upper part of phase shifters and  $B$  is a beamsplitter. The matrices of optical elements are

$$\begin{aligned} P_\phi &= \begin{pmatrix} e^{i\phi} & 0 \\ 0 & 1 \end{pmatrix} \\ P_\theta &= \frac{1}{2} \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \\ B &= \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \end{aligned} \quad (3)$$

For example, two single qubit operators  $H, X$  in Fig. 2 can be implemented with phase  $\theta = \pi/2, \phi = 0$  and  $\theta = 0, \phi = 0$ . Qubits encoded in two waveguides presented in (1) path through the MZI-adjusted phases and the computations are performed when each qubit exits the MZI.

Similarly, two-qubit unitary operations such as *Controlled-NOT* (*CNOT*) and *Controlled-Z* (*CZ*) can be implemented with MZI as in the case of single operations. However, other condition is required besides expressing a single operation as MZI. Two-qubit operators should form a specific network with MZIs, rather than relying on linear components in MZI for two-qubit operation [10], [24]. As in the case of a single operation, we demonstrate a *CNOT* in Fig. 3 with a typical circuit expression and a combination of MZIs.

Lastly, measurement in LOQC-based quantum information processing is accomplished through photon detectors. The detectors are located at the end of the quantum chip. From the preceding quantum operations, the different number of photons from each waveguide is collected at the detector. Thus, information results from optical measurements can be interpreted to derive computational results in terms of quantum measurements.

## IV. REQUIREMENTS FOR SOFTWARE IMPLEMENTATION

### A. SOFTWARE MODULES

The modules of a quantum computing software framework can be organized in a variety of ways, however we present them here as three components: decomposer, mapper, and controller. We describe the role of modules in the following list.

- (a) Decomposer has the ability to transform high-level quantum computing language into assembly language for customized hardware [25]. Various users are able to write a quantum code in their preferred programming style. The module then translates the code into a standard format to ensure the proper operation of quantum computers, similar to classical computers receiving a

standard format such as machine language. The module decomposes quantum circuits by assuming logical qubits with no constraints on qubit connectivity.

- (b) Mapper investigates the optimal layout of qubits and schedules for quantum gates. In other words, the module conducts a process that reformulates a decomposed algorithm suited for a given real quantum processor. When considered on the qubit level, logical qubits have to correspond to physical qubits in the quantum processor. Therefore, qubit mapping on the real quantum device must be done without losing the functionality of the logical quantum circuit.
- (c) Controller converts the mapping data into signals to be performed to an actual quantum processor. In addition, this module operates usually the last operation in the software platform, it henceforth generates the final data to be sent to the processor.

## B. DIFFERENCES WITH SUPERCONDUCTING QUANTUM COMPUTER PLATFORM

With the concept of software modules, we introduce the modules for LOQC by comparing with SCQC. Figure 4 illustrates the physical representations of each quantum computing platform described by the inputs and outputs of the modules. We express the differences between SCQC and LOQC with a few distinct points in each module. Through this discussion, we justify the necessity of a customized software platform for LOQC due to the differences in each physical system. Without loss of generality, we assume the Bell state generation circuit as an example input to describe each module.

### 1) DECOMPOSER

- (a) In the first perspective on decomposer, we discuss in the sight of the structure of a quantum processor. SCQC decomposer assumes a fully connected graph of a superconducting processor, when the module sets the quantum gates to be applied in each qubit. Since the decomposer does not consider the constraint of gate applications on logical level, the module simply assumes the number of logical qubits required by the imported circuit and configures the quantum gates according to the position of the specified qubits. On the other hand, LOQC decomposer arranges the qubit position and quantum gates matching the structure of a standard chip. The qubits are located on fixed points by waveguides in a photonic processor, and MZI plates actually implement quantum gates, not the qubits themselves.
- (b) In the second perspective, we observe a set of quantum gates on two quantum computers. SCQC decomposer converts large unitary circuits into primitive gates of their physical system from quantum algorithms as primary ability [26]. It is effective to represent the circuit with a minimal set of quantum gates, which depends on the type of SCQC processor. In other words, the module in SCQC can change quantum gates that are applicable in the software, such as *Pauli-X*, *Y*, *Z*,

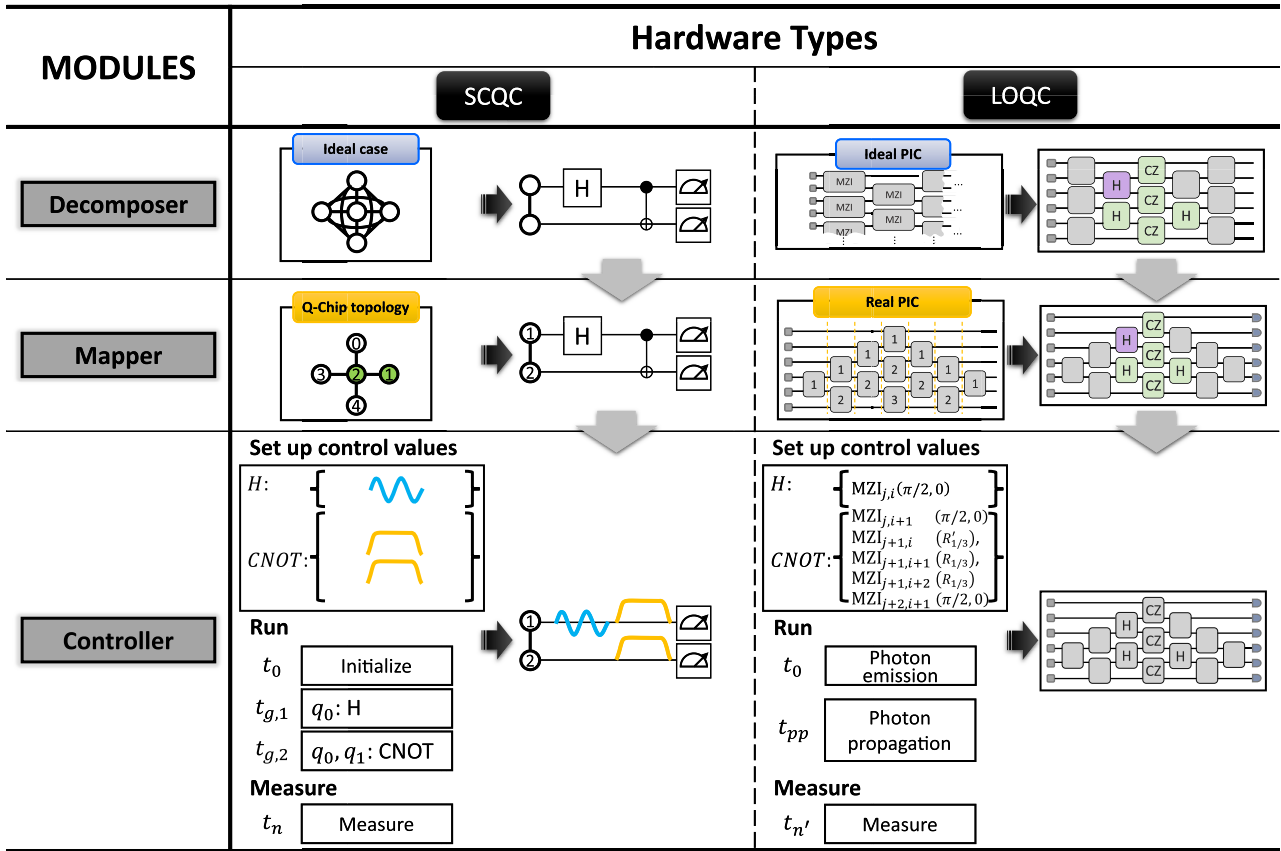
*R<sub>x</sub>*, *R<sub>y</sub>*, *R<sub>z</sub>*, *H*, *I*, *S*, *T*, *CNOT* and *CZ*, for example to *H*, *S*, *T*, *CNOT*. In contrast to SCQC, LOQC decomposer has no restriction to the types of quantum gates, since the MZIs influence the implementation of gates in photonic processor. As we mentioned in Sec. III, MZI expressed in (2) can represent different types of quantum operators, which implies that it does not need to be decomposed into primitive gates. However, a two-qubit gate implementation in a photonic processor makes the issue that the decomposer depends more on physical hardware such as *CNOT* referring to Fig. 3(b).

### 2) MAPPER

- (a) We first discuss the labeling qubits for each physical system. SCQC mapper assigns the output result from the SCQC decomposer to physical qubits according to the topology of the quantum chip, and the module generates a sequence of quantum circuits taking into account the connectivity between the qubits. In contrast to an ideal quantum chip, the topology of an actual chip partially omits connectivity to some qubits. The module then finds the best layout to compose the shortest circuit depth. Similarly, LOQC mapper assigns quantum gates to appropriate MZI locations that match execution timing in an actual photonic integrated circuit (PIC). In contrast to SCQC, as we discussed in Sec. III, a qubit on LOQC is defined by a pair of waveguides, and the location is already occupied. Regarding the illustration in Fig. 4, the first and second qubits are defined by two pairs of waveguides, waveguide 2 and 3, and waveguide 4 and 5, respectively.
- (b) Second, from the perspective of gate mapping, SCQC mapper merely depends on time arranging quantum gates to selected qubits. Since physical qubits in a superconducting processor are static, the gates are directly operated to them matching the time interval in a SCQC mapping process. However, LOQC mapper does not only consider time but also spaces regarding a network of PIC. As opposed to the SCQC case, quantum gates are operated by MZIs expressed in (2), not by qubits themselves. According to the qubit position defined with a pair of waveguides, the MZI implements all quantum gates, but the MZI may be existed or omitted relying on the network of PIC. It implies that the LOQC mapper should take into account the absence of some MZIs in the PIC and find the exact MZI locations to map the decomposition result.

### 3) CONTROLLER

- (a) In controller, we focus on a control method for an actual operation of quantum gates, divided by setting up control values and running quantum experiment. In SCQC, quantum operators are implemented by affecting unique pulse signals to the physical qubits. It means that SCQC controller prepares the pulses such as *H* and *CNOT* before the processor operates. They will apply the gates to the qubits in a fixed time period



**FIGURE 4.** Comparison illustration of software modules between SCQC and LOQC. Each module reflects hardware with different physical characteristics: qchip topology for SCQC and MZI network for LOQC. For each module, a black thick arrow indicates the result illustration. The software proceeds following the light grey arrows. To make it easier to understand the difference, the Bell state generation circuit is assumed.

when the quantum computing begins. LOQC controller applies unique phase values for each gate to MZIs as a physical object. Unlike the former case, the LOQC case does not consider the application timing of quantum gates, since this process is already included in the mapping step. Therefore, the module prepares phase values for all gates and locations described in  $MZI_{j,i}$  where  $j$  th step and  $i$  th MZI to be applied.

- (b) We now follow up on the rest part of the execution. When the SCQC processor starts the quantum experiment, the initialization to the physical qubits is performed on  $t_0$ . Quantum gates then immediately perform on the allocated physical qubits, which are sequentially applied at  $t_{g,1}$  to  $t_{g,n}$  for  $n$  gates. After the time has passed, the measurement is performed on  $t_n$ . Similarly, the photonic experiment begins the photon emission from laser source immediately at  $t_0$ . Photon travels in the waveguides penetrating the MZIs in a photonic processor for photon propagation time  $t_{pp}$ , afterwards measurement results are collected from detectors located on each waveguide at  $t_{n'}$ . The comparable point is the total execution time of quantum processor. The duration of SCQC processor depends on how many quantum gates are applied in the total time  $\sum_{k=0}^{n-1} |t_{g,k+1} - t_{g,k}|$ . On the other hand, LOQC case only relies on the propagation

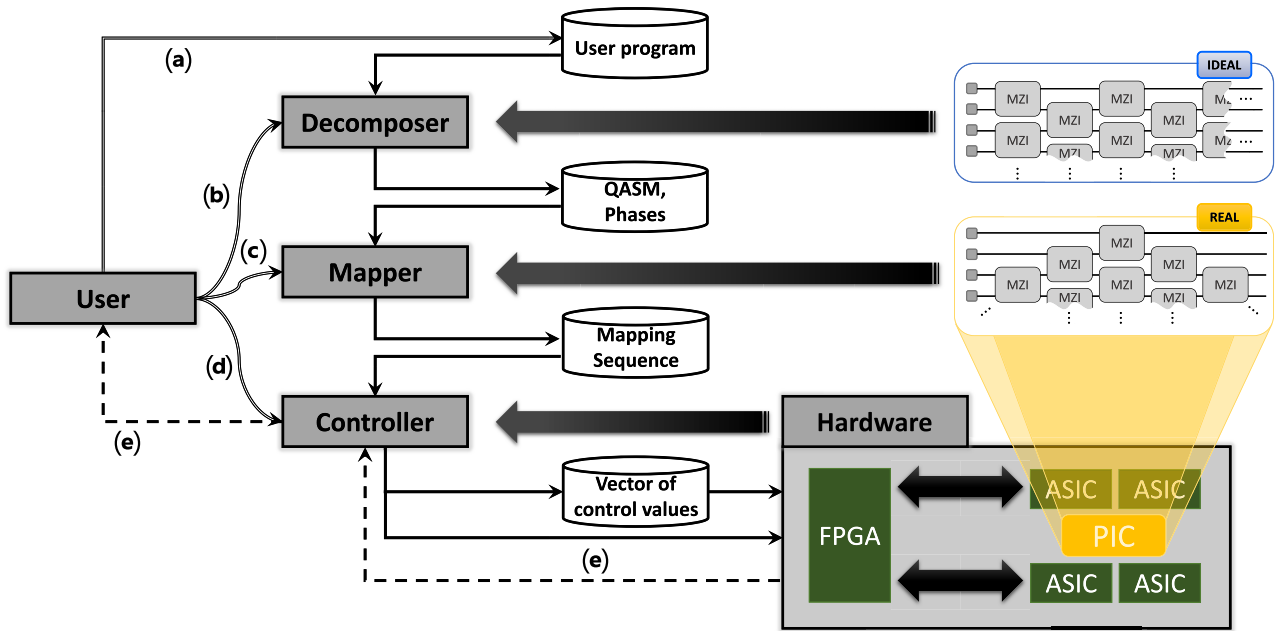
time  $t_{pp}$ . From the perspective of consuming operation time, we imply  $t_{n'} \leq t_n$  since light travels at the speed of light.

### V. LOQC SOFTWARE PLATFORM

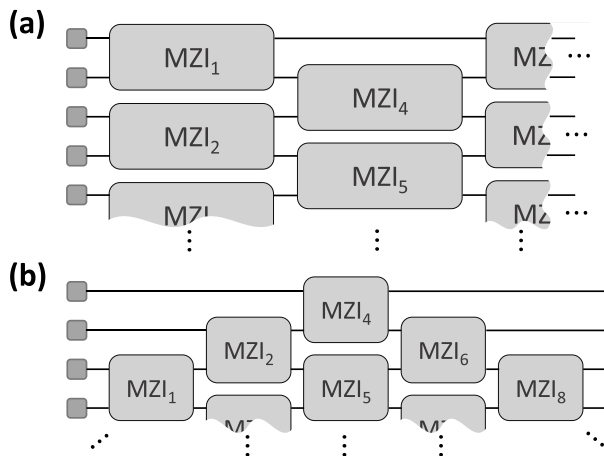
Reflecting on the hardware properties in LOQC and the three essential modules, we propose the structure of software platform tailored to the specific requirements of LOQC-based quantum computing system. In pursuit of a comprehensive understanding, we follow up the elegantly illustrated flowchart in Fig. 5 where we commence the overall structure of the software, which provides a panoramic view of the operation workflow of the platform in Sec. V-A. We then explain the detailed description of inputs and outputs at each module with the figures of data interpretation in Sec. V-B to V-D.

#### A. OVERALL STRUCTURE

The steps are composed of user programming that results from hardware measurement according to the software workflow shown in Fig. 5. Whenever each step progresses, a module stores the results as a database, and a user can obtain customized outputs from the modules. The steps are explained as follows.



**FIGURE 5.** Overall software structure and module diagrams of LOQC. Grey boxes mean the main components and cylinders depict the databases from software output. Each step is alternately connected with modules and databases by solid lines, except for user web control and results from hardware by double solid lines and dashed lines. The explanations for alphabetical lists (a) to (e) are provided in Sec. V-A.

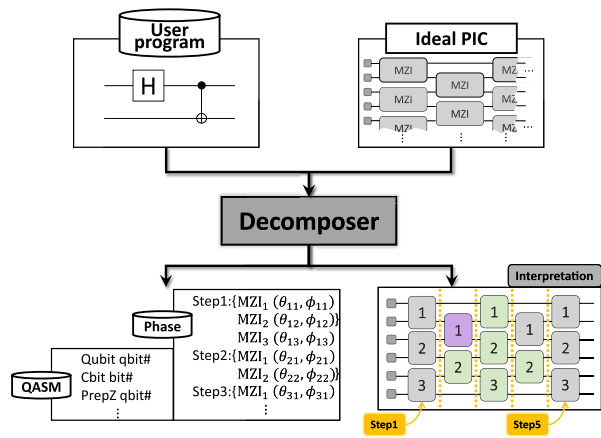


**FIGURE 6.** (a) Staggered MZI network by Clements et al.. (b) Triangle MZI network by Reck et al.

- (a) Initially, users input various quantum algorithms which they wish to employ in this step, such as unitary matrix decomposition, quantum Fourier transform, and Grover’s quantum search algorithm. It is important to note that users should write their algorithm code using quantum gate sets, since the software interprets the script at the circuit level [6]. Upon completion of their programming, the software stores the user program and transmits the data to the decomposer.
- (b) Taking the user-programmed code as input, users run the LOQC decomposer that outputs the quantum assembly language (QASM) code and the phases, which are the

- standard output at the quantum circuit level and the main data in the software platform, respectively. By an additional assumption for its functionality, the module designates the scheme of Clements et al. [27] as an ideal PIC that can be illustrated in Fig. 6(a).
- (c) LOQC Mapper receives the phases from the decomposer and generates mapping sequences. It considers the real PIC structure to assign output to the best location, so the actual PIC structure must be provided as an additional input.
- (d) Controller is the final module employed by users in the software. It receives the laid-out sequence from the mapper and processes it into data to be transferred to the hardware. The module converts the data into an appropriate format for successful transmission, resulting in a vector of control values.
- (e) The remaining part of the structure explains the execution of actual hardware. After receiving the converted data, the hardware set performs the actual experiment. It consists of a field programmable gate array (FPGA), application-specific integrated circuits (ASICs), and a PIC. FPGA and ASIC process information applied to the physical behavior acting on the PIC. PIC uses signals from these electronic devices to perform quantum experiments governed by the photonic properties [28], [29]. After quantum experiments, the hardware provides experiment results to users and ultimately obtains the measurement results of LOQC-based quantum computer.

So far, we have shown the software structure for a holistic view of LOQC-based computer manipulation. In forthcoming subsections, we explore each module in more depth.



**FIGURE 7.** Decomposition flow on LOQC. Decomposer loads user program and ideal PIC structure composed of MZI network in Fig. 6(a). The module provides the output of QASM and phase information. The interpretation box shows a pictorial representation of phase data. Steps are divided into yellow dotted lines and MZIs are numbered in plates. One-qubit gate and two-qubit gate are colored in purple and green, respectively.

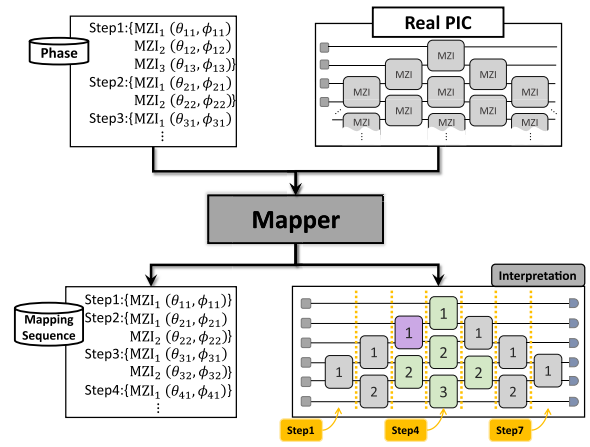
**B. DECOMPOSER**

Recalling from Sec. IV-B, where we described the functional differences between two decomposers, the LOQC decomposer produces QASM and a sequence of phase values as results. Since qubits are occupied at the starting point on PIC and quantum gates are allocated to MZIs in PIC, the decomposer has to consider a reconfigurable PIC structure that can represent a set of single and two-qubit gates. As discussed in the list (b) in Sec. IV, we adapt the *Clements* scheme as an ideal PIC depicted in Fig. 6(a). The scheme has the most densely packed MZI structure so it makes sense for the decomposer to operate with this network by default.

Reflecting the notable properties, the proposed decomposer is summarized on the flowchart in Fig. 7. Workflow of the decomposer organizes two inputs and outputs centered on the module. One of the input blocks “User program” denotes any quantum algorithms that we mentioned in the Sec. IV. Another one “Ideal PIC” is a necessary assumption for a decomposition on LOQC. The “Decomposer” interprets a quantum circuit of a user’s algorithm and transforms proper MZI phases for each quantum operation. Consequently, the module returns “QASM” and “Phase” information. The former is the general language of common quantum compilers, and the latter gives the result of decomposing quantum gates in LOQC into minimal units (phases). This means that the QASM is required to interpret the user data at the quantum circuit level as a standard format, and the phase data is actually used by the LOQC software platform to operate. The data constitutes hierarchical steps, MZI numbers, and phase information for each MZI. It can be graphically interpreted as the right box in Fig. 7.

**C. MAPPER**

LOQC mapper finds the best layout to execute the algorithm written by users, also allocating the phase-level decomposed data as close to the starting point as possible for optimization.



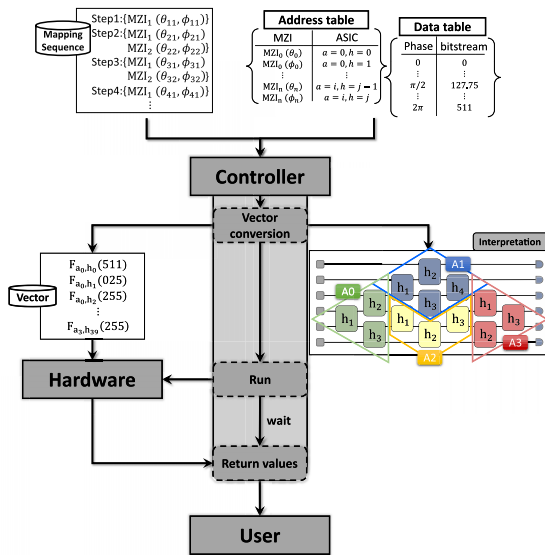
**FIGURE 8.** Mapper flow on LOQC. Phase results from decomposer and real PIC provide inputs to the mapper. The module creates a mapping sequence as a result, which can be interpreted as shown in the right box. As in the case of the decomposer, steps are divided into yellow dotted lines and MZIs are numbered in plates. One-qubit gate and two-qubit gate are colored in purple and green, respectively.

The module effectively assigns the phase information to the proper location of the MZIs according to a network of PIC. As we mentioned in Sec. IV-A, the actual PIC structure must be provided as the additional input of the mapper to fulfill its role. Various MZI networks can configure different PICs, in which one of the proposed structures by Reck et al. [30] formulates a triangular shape depicted in Fig. 6(b). This module provides and stores the mapping sequence.

Figure 8 illustrates the flowchart of the mapper. Mapper operates with “Phase” data received from the decomposer and physical structure of “Real PIC”. Since the positions of MZIs are physically fixed, applying quantum gates is strictly time-dependent. Once the module maps phases into optimized locations of the MZI network, it provides the “mapping sequence” as a result. The sequence indexes every MZI location according to the waveguide and time steps for satisfying the constraint. On the right box of the mapping sequence, we graphically illustrate the mapping result on real PIC. For example, to run *CNOT* gate on a real PIC, the mapper should place the gate in the center of the quantum chip.

**D. CONTROLLER**

The last module of the LOQC platform, the controller, translates the mapping sequence generated by the LOQC mapper into a vector of control values. Since the quantum algorithms written by users have to be run by a photonic processor, PC should reformat the data into a suitable form, which is then delivered to electronic devices that assist in the operation of a PIC. Here, we examine the necessary condition of controlled hardware to transform the data into the exact form. In our design, the auxiliary electronics FPGA and ASIC are positioned on the top of the quantum processor PIC for operation purposes, as illustrated in Fig. 5. The FPGA can receive the data in the form of a bytearray, which is then processed by a customized ASIC designed specifically for the PIC. Phase values are directly controlled by ASICs tailored

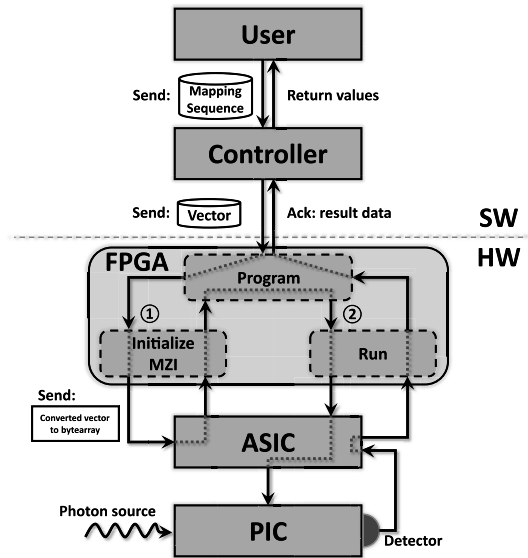


**FIGURE 9.** Controller flow on LOQC. Mapping sequence from mapper, address table given by hardware environment, and pre-written data conversion table act as inputs. Three dashed line boxes are subsets of the controller, each operation is controlled by this module. In the step of vector conversion, the output is a vector, which is represented by the interpretation box on the right. The colored MZIs in the box separate the sectors of the ASICs.

for the PIC, which operates as a quantum gate. Therefore, the controller has to satisfy the two necessities (a) to convert to an appropriate data format, and (b) to know addresses which the contact points of ASIC are connected to the MZI. For the former one, we need to prepare the table for data conversion to transmit into the form of bytearray. The latter one is necessary to have the address table between each MZI and each pin header that connects to it since the pin header of the ASIC connects to one of the MZIs in the PIC.

Figure 9 shows the input/output flow to satisfy the requirements for the controller. The module processes the received data “Mapping sequence” from the result of the mapper. Also, “Address table” of MZIs and ASICs and “Data table” of phases and bitstream behave as additional data for the module. In the case of the latter, since the phase data should be converted to convey the values of ASIC, the values must be normalized and discretized into bits. Once the controller satisfies the previous requirements and operates, it stores the “Vector” data as output and prepares them for sending to the FPGA. The box on the right in Fig. 9 represents the data allocation to the chip. The elements grouped into one vector are physically allocated to the colored sectors that identify the four ASICs used.

The remaining part describes how the data generated by the software flows to the hardware. Specifically, we focus on the role of hardware rather than the operation based on the exact components of the hardware. The rest of the part works as follows. Figure 10 shows the interconnectedness of user data workflow and hardware. After the controller sends the successfully converted data in the vector form, the “Program” in FPGA first is set up for preprocessing, which receives the data and sends the address related to the



**FIGURE 10.** Hardware data flow in LOQC. The flow consists of the user sending data to the hardware and receiving data back. The LOQC FPGA contains two states, each communicating with the rest of the hardware. The module between the user and the controller is omitted.

numbering of the ASICs and each pin header. This process occurs in the “Initialize MZI”. If the process is successfully completed and an ack is sent by the ASIC, the “Program” commands “Run” which is now prepared for operation and the state executes the data to the remaining hardware. When the data arrives at PIC, the quantum experiment for the data begins and photons are collected by detectors. The result is passed to the FPGA, which processes the data into qubit state counts and sends them to the controller. Finally, the controller returns the values to the users and the one cycle of quantum computation is complete.

## VI. CORRECTNESS AND PERFORMANCE

### A. CHECKING REVERSIBILITY

Unlike classical computers, quantum computers possess the reversibility of operations, which follows the unitarity in quantum regime [31]. LOQC platform must also satisfy the reversible operations by quantum gates, as the PIC reproduces quantum computations by the movement of photons along light paths. Proposed software framework prepares the conjugate transpose matrix for each unitary gate and the user can confirm the reversibility by applying the inverse operation to their quantum algorithm at the user programming stage.

Unfortunately, actual operation with hardware has not yet been tested, we can only investigate the reversibility program at the decomposition process. For an arbitrary single qubit operation, we have to check the reversibility  $AB B^\dagger A^\dagger = I$  with the operator

$$A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \quad A^\dagger = \begin{pmatrix} a_{00}^* & a_{10}^* \\ a_{01}^* & a_{11}^* \end{pmatrix} \quad (4)$$

$$B = \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} \quad B^\dagger = \begin{pmatrix} b_{00}^* & b_{10}^* \\ b_{01}^* & b_{11}^* \end{pmatrix} \quad (5)$$



by constructing the reversible checking circuit. In LOQC, we combine the operators with the MZI unitary matrix in (2) for applying to MZIs in PIC. The example pseudocode for checking reversibility can be written in Algorithm 1.

---

**Algorithm 1** Checking Reversibility
 

---

```

initialize libraries
n ← 2
for i = 1, ..., n do
  initialize qri
  initialize cri
  align qr on z-axis
end for
perform A to qr0
perform B to qr0
perform B† to qr0
perform A† to qr0
for i = 1, ..., n do
  Measure qri
  Store qri to cri
end for
  
```

---

### B. MAPPING COMPLEXITY

Whenever the LOQC software proceeds a computing process, it takes a certain amount of time to execute a user program within each module. In particular, the LOQC mapper has the highest time complexity among the software modules, since its main role - finding the suitable layout to the processor architecture. Therefore, we need to calculate the number of resources for minimal quantum operation at first and then analyze how long the mapping spends time.

We simply investigate how LOQC requires the computing resources on qubits and quantum gates. Let us define the notations  $n_{lc,Q}$ ,  $n_{lc,1q}$ ,  $n_{lc,2q}$  as the number of qubits, one-qubit gates, and two-qubit gates on logical circuit representation. A direct correlation between these logical elements and the resource consumption on waveguides and quantum gates emerges:

$$\begin{aligned} n_w &= 2n_{lc,Q} & n_{1Q} &= n_{lc,1q} \\ n_{CZ} &= 3n_{lc,2q} & n_{CNOT} &= 3n_{lc,2q} + 2n_{lc,1q} \end{aligned} \quad (6)$$

Hence, the minimal number of MZIs required per quantum gate is  $n_{MZI} = n_{lc,1q} + 3n_{lc,2q}$  or  $3n_{lc,2q} + 3n_{lc,1q}$ .

Assume that the worst scenario occurs on a mapping process, and all quantum gates from a decomposed result can be mapped into the *Clements* structure in Fig. 6(a). Let us call  $d$  as the circuit depth in LOQC. The mapper swipes all possible MZIs to be applied for  $n_{lc,Q}$  qubits until the end of  $d$ . This empirical insight allows us to estimate the time complexity of the photonic processor as  $\mathcal{O}(n_{lc,Q}^2 d)$ .

### VII. USE CASE

We introduce a simulation of the Bell state generation algorithm on LOQC as a use case. Bell-state generation algorithm represents an elementary algorithm that anyone

can clearly understand quantum phenomena. It could easily accept how our proposed quantum computing on optical system works.

Before demonstrating the Bell-state generation in the proposed software for LOQC, we describe the algorithm mathematically. A generalized equation of Bell-states is

$$|\beta(x, y)\rangle = \frac{|0, y\rangle + (-1)^x |1, \bar{y}\rangle}{\sqrt{2}} \quad (7)$$

where  $\bar{y}$  is the negation of  $y$  as described in [32]. This algorithm has four basis constituted in superposition combinations of qubit eigenvector sets  $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ . We only take one of the basis sets

$$|\beta(0, 0)\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) \quad (8)$$

constructing for the simplified circuit. This state requires only two quantum gates, *Hadamard* and *CNOT*. Explicitly, the *Hadamard* gate transforms  $|0\rangle$  into a superposition of  $(|0\rangle + |1\rangle)/\sqrt{2}$ . The *CNOT* operation on the superposed state yields  $|\beta(0, 0)\rangle$ .

We present the simplified algorithm for the Bell state generation. At first, the algorithm can be depicted as a quantum circuit on the user program box in Fig. 7. The algorithm requires a combination of quantum operators  $H$  and *CNOT*,  $n$ ,  $qr$ , and  $cr$  as a number of qubits, quantum registers, and classical registers. Lastly, to receive the result of Bell measurement, we call *Measure* for quantum measurement and *Store* to save information to classical registers. Hence, demonstration of the algorithm for the Bell-state generation is represented in Algorithm 2.

---

**Algorithm 2** Bell State Generation
 

---

```

initialize libraries
n ← 2
for i = 1, ..., n do
  initialize qri
  initialize cri
  align qr on z-axis
end for
perform H to qr0
perform CNOT to qr0 and qr1
for i = 1, ..., n do
  Measure qri
  Store qri to cri
end for
  
```

---

When the user finishes writing the quantum algorithm, the program is ready to perform the decomposer. The output of module provides QASM and phases, and phase data can be listed hierarchically as mentioned in Sec. V-B.

```

{
  "step1": {
    "MZI1": [0, π],
    "MZI2": [π, π]},
  "step2": {
    "MZI1": [π, π],
  
```

```

    "MZI2": [ $\pi$ ,  $\pi$ ],
    "MZI3": [ $\pi$ ,  $\pi$ ],
  "step3": {
    "MZI1": [ $\pi$ ,  $\pi$ ],
    "MZI2": [0,  $\pi$ ],
  "step4": {
    "MZI1": [ $\pi$ ,  $\pi$ ],
    "MZI2": [ $\pi$ ,  $\pi$ ],
    "MZI3": [ $\pi$ ,  $\pi$ ],
  "step5": {
    "MZI1": [ $\pi$ ,  $\pi$ ],
    "MZI2": [0,  $\pi$ ]}

```

The phase values  $\theta$  and  $\phi$  are evaluated as the unitary matrix in (2). This data shows that  $H$  is contained in “step1,” “step3,” and “step5,” and  $CZ$  is configured in “step4.”  $H$  in “step3” and “step5” is the decomposition result because of CNOT implementation in the MZI network as depicted in Fig. 3. The other steps are represented in the *Identity* gate. With this decomposer data, the mapper conducts its role as explained in Sec. V-C and depicted in Fig. 8.

```

{  "step1": {
    "MZI1": [ $\pi$ ,  $\pi$ ],
  "step2": {
    "MZI1": [ $\pi$ ,  $\pi$ ],
    "MZI2": [ $\pi$ ,  $\pi$ ],
  "step3": {
    "MZI1": [0,  $\pi$ ],
    "MZI2": [0,  $\pi$ ],
  "step4": {
    "MZI1": [ $\pi$ ,  $\pi$ ],
    "MZI2": [ $\pi$ ,  $\pi$ ],
    "MZI3": [ $\pi$ ,  $\pi$ ],
  "step5": {
    "MZI1": [ $\pi$ ,  $\pi$ ],
    "MZI2": [0,  $\pi$ ],
  "step6": {
    "MZI1": [ $\pi$ ,  $\pi$ ],
    "MZI2": [ $\pi$ ,  $\pi$ ],
  "step7": {
    "MZI1": [ $\pi$ ,  $\pi$ ]}

```

The result data has the same form as the decomposer output. However, this result gives the mapped data according to the MZI network depicted in Fig. 6(b), which shows a different number of namespace MZIs under steps. It can be interpreted as shown in the interpretation box in Fig. 8.  $H$  and CNOT are allocated in the purple and light green plates, respectively. The controller imports the mapper output, an address table, and a bitstream data conversion table explained in Sec. V-D. The output data consists of two levels, ASIC and pin header.

```

{  "a1": {
    "h1": [255, 255],
    "h2": [0, 255],
    "h3": [255, 255],
  },

```

```

  "a2": {
    "h1": [255, 255],
    "h2": [255, 255],
    "h3": [255, 255],
    "h4": [255, 255]},
  "a3": {
    "h1": [255, 255],
    "h2": [0, 255],
    "h3": [255, 255]},
  "a4": {
    "h1": [255, 255],
    "h2": [255, 255],
    "h3": [255, 255]}

```

This output assumes four ASICs and the required quantity of pin headers according to the number of MZIs. The result divides four sectors, graphically shown in the interpretation box in Fig. 9. The figure depicts MZI plates with different colors, green, blue, yellow, and red corresponding to sectors 1, 2, 3 and 4, respectively. And the phase value of each pin header is normalized for sending as a bitstream, where the maximum value is set to 511, corresponding to the maximum angle  $2\pi$ .

### VIII. CONCLUSION AND DISCUSSION

Throughout this work, we propose a software framework for making universal programmable LOQC. To run any LOQC hardware, our framework has four components from the user-level to the device-level control. An example use of this framework has been shown for the Bell state generation circuit.

Since this is just beginning, we have lots of future work as follows. First, we will test the software framework under the real device. During this time, we will confirm the interface of each module and their interoperability. After that, we will add another capability such as quantum verification. Quantum hardware verification can be done by using the programming as a systematic way, but we want to make a separate module which is dedicated to quantum characterization by using quantum state tomography and process tomography methods. In addition, we need to consider the analysis of energy consumption and fault-tolerance property of LOQC hardware. Finally, we should optimize each module to speed up and migrate to the hardware module in order to reduce the user-level feedback time.

### ACKNOWLEDGMENT

The authors would like to thank Dongmin Kim (Pukyong National University) for assistance on the mapper and the controller. They also like to thank Martino Bernard and Alessandro Tontini (Fondazione Bruno Kessler) for helpful discussions on photonic hardware. They also like to thank Alessio Baldazzi (University of Trento) for theoretical background explanations of LOQC.

### REFERENCES

[1] R. P. Feynman, “Simulating physics with computers,” *Int. J. Theor. Phys.*, vol. 21, nos. 6–7, pp. 467–488, Jun. 1982.

- [2] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proc. Roy. Soc. London A, Math. Phys. Sci.*, vol. 439, no. 1907, pp. 553–558, Dec. 1992.
- [3] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct. 2019.
- [4] Google Quantum AI, "Non-Abelian braiding of graph vertices in a superconducting processor," *Nature*, vol. 618, no. 7964, pp. 264–269, May 2023.
- [5] *Quantum Computing: Progress and Prospects*, E. Grumbling and M. Horowitz, Eds., Nat. Academies Sci., Eng., Med., Washington, DC, USA, 2019.
- [6] R. LaRose, "Overview and comparison of gate level quantum software platforms," *Quantum*, vol. 3, p. 130, Mar. 2019.
- [7] C. C. Gerry and P. L. Knight, *Introductory Quantum Optics*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
- [8] C. Adami and N. J. Cerf, "Quantum computation with linear optics," in *Quantum Computing and Quantum Communications* (Lecture Notes in Computer Science), vol. 1509. Cham, Switzerland: Springer, 1999, pp. 391–401.
- [9] J. Carolan, C. Harrold, C. Sparrow, E. Martín-López, N. J. Russell, J. W. Silverstone, P. J. Shadbolt, N. Matsuda, M. Oguma, M. Itoh, G. D. Marshall, M. G. Thompson, J. C. F. Matthews, T. Hashimoto, J. L. O'Brien, and A. Laing, "Universal linear optics," *Science*, vol. 349, pp. 711–716, Aug. 2015.
- [10] J. L. O'Brien, G. J. Pryde, A. G. White, T. C. Ralph, and D. Branning, "Demonstration of an all-optical quantum controlled-NOT gate," *Nature*, vol. 426, no. 6964, pp. 264–267, Nov. 2003.
- [11] X. Qiang, X. Zhou, J. Wang, C. M. Wilkes, T. Loke, S. O'Gara, L. Kling, G. D. Marshall, R. Santagati, T. C. Ralph, J. B. Wang, J. L. O'Brien, M. G. Thompson, and J. C. F. Matthews, "Large-scale silicon quantum photonics implementing arbitrary two-qubit processing," *Nature Photon.*, vol. 12, no. 9, pp. 534–539, Sep. 2018.
- [12] Q. Zhang, M. Li, Y. Chen, X. Ren, R. Osellame, Q. Gong, and Y. Li, "Femtosecond laser direct writing of an integrated path-encoded CNOT quantum gate," *Opt. Mater. Exp.*, vol. 9, no. 5, pp. 2318–2326, 2019.
- [13] Y. Li, L. Wan, H. Zhang, H. Zhu, Y. Shi, L. K. Chin, X. Zhou, L. C. Kwek, and A. Q. Liu, "Quantum Fredkin and Toffoli gates on a versatile programmable silicon photonic chip," *npj Quantum Inf.*, vol. 8, no. 1, pp. 1–7, Sep. 2022.
- [14] J. Q. You, J. S. Tsai, and F. Nori, "Scalable quantum computing with Josephson charge qubits," *Phys. Rev. Lett.*, vol. 89, no. 19, Oct. 2002, Art. no. 197902.
- [15] C. Rigetti and M. Devoret, "Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies," *Phys. Rev. B, Condens. Matter*, vol. 81, no. 13, Apr. 2010, Art. no. 134507.
- [16] J. M. Chow, L. DiCarlo, J. M. Gambetta, F. Motzoi, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, "Optimized driving of superconducting artificial atoms for improved single-qubit gates," *Phys. Rev. A, Gen. Phys.*, vol. 82, no. 4, Oct. 2010, Art. no. 040305.
- [17] J. M. Chow, A. D. Córcoles, J. M. Gambetta, C. Rigetti, B. R. Johnson, J. A. Smolin, J. R. Rozen, G. A. Keefe, M. B. Rothwell, M. B. Ketchen, and M. Steffen, "Simple all-microwave entangling gate for fixed-frequency superconducting qubits," *Phys. Rev. Lett.*, vol. 107, no. 8, Aug. 2011, Art. no. 080502.
- [18] T. Alexander, N. Kanazawa, D. J. Egger, L. Capelluto, C. J. Wood, A. Javadi-Abhari, and D. C. McKay, "Qiskit pulse: Programming quantum computers through the cloud with pulses," *Quantum Sci. Technol.*, vol. 5, no. 4, Aug. 2020, Art. no. 044006.
- [19] F. Zilk, K. Staudacher, T. Guggemos, K. Füllinger, D. Krantz Müller, and P. Walther, "A compiler for universal photonic quantum computers," in *Proc. IEEE/ACM 3rd Int. Workshop Quantum Comput. Softw. (QCS)*, Nov. 2022, pp. 57–67.
- [20] N. Heurtel, A. Fryllas, G. D. Gliniasty, R. Le Bihan, S. Malherbe, M. Pailhas, E. Bertasi, B. Bourdoncle, P.-E. Emeriau, R. Mezher, L. Music, N. Belabas, B. Valiron, P. Senellart, S. Mansfield, and J. Senellart, "Perceval: A software platform for discrete variable photonic quantum computing," *Quantum*, vol. 7, p. 931, Feb. 2023.
- [21] P. Kok, W. J. Munro, K. Nemoto, T. C. Ralph, J. P. Dowling, and G. J. Milburn, "Linear optical quantum computing with photonic qubits," *Rev. Modern Phys.*, vol. 79, no. 1, pp. 135–174, Jan. 2007.
- [22] E. Knill, R. Laflamme, and G. J. Milburn, "A scheme for efficient quantum computation with linear optics," *Nature*, vol. 409, no. 6816, pp. 46–52, Jan. 2001.
- [23] N. C. Harris, J. Carolan, D. Bunandar, M. Prabhu, M. Hochberg, T. Baehr-Jones, M. L. Fanto, A. M. Smith, C. C. Tison, P. M. Alsing, and D. Englund, "Linear programmable nanophotonic processors," *Optica*, vol. 5, no. 12, pp. 1623–1631, Dec. 2018.
- [24] M. Patel, J. B. Altepeter, M. A. Hall, M. Medic, and P. Kumar, "Experimental characterization of a telecommunications-band quantum controlled-not gate," *IEEE J. Sel. Topics Quantum Electron.*, vol. 15, no. 6, pp. 1685–1693, Nov. 2009.
- [25] M. Maronese, L. Moro, L. Rocutto, and E. Prati, "Quantum compiling," in *Quantum Computing Environments*. Cham, Switzerland: Springer, 2022, pp. 39–74.
- [26] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Phys. Rev. A, Gen. Phys.*, vol. 52, no. 5, pp. 3457–3467, Nov. 1995.
- [27] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley, "Optimal design for universal multiport interferometers," *Optica*, vol. 3, no. 12, pp. 1460–1465, 2016.
- [28] L. Gemma, M. Bernard, and D. Brunelli, "An optical tool to optimize the output of a photonic integrated chip architecture," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 12, no. 3, pp. 694–702, Sep. 2022.
- [29] M. Bernard, M. Ghulinyan, and F. Acerbi, "Photonic circuits monolithically integrated with silicon photodiodes," in *Proc. Italian Conf. Opt. Photon. (ICOP)*, Jun. 2022, pp. 1–5.
- [30] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani, "Experimental realization of any discrete unitary operator," *Phys. Rev. Lett.*, vol. 73, no. 1, pp. 58–61, Jul. 1994.
- [31] C. Liu, "Reverse checking of quantum algorithm execution," *IEEE Access*, vol. 8, pp. 228702–228710, 2020.
- [32] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th ed. Cambridge, U.K.: Cambridge Univ. Press, 2010.



**YONG KWON** received the B.S. degree in physics from Pukyong National University, Busan, Republic of Korea, in 2022, where he is currently pursuing the M.S. degree in physics through the integrated M.S. program.

From 2020 to 2022, he started a research on quantum information with the Statistical Physics Laboratory, Pukyong National University. During the period, he studied in mutual information and quantum concurrence and published one article on *Journal of Statistical Mechanics: Theory and Experiment*. Since 2022, he has been studying a research on quantum computing with the Quantum Computational Science Laboratory, Pukyong National University.



**BYUNG-SOO CHOI** (Member, IEEE) received the B.S. degree in computer engineering from Chungnam National University, South Korea, in 1996, and the M.S. and Ph.D. degrees in information and communication from the Gwangju Institute of Science and Technology (GIST), South Korea, in 1998 and 2004, respectively. From 2004 to 2006, he was a Postdoctoral Researcher with the University of York, U.K., where he was involved in quantum algorithms.

From 2011 to 2013, he was a Research Scientist with Duke University, USA, where he was involved in fault-tolerant quantum computation. From 2013 to 2015, he was a Researcher with The University of Tokyo, Japan, where he was involved in quantum-dot qubit device. From 2015 to 2022, he was a Principal Researcher with the Electronics and Telecommunications Research Institute (ETRI), where he was involved in quantum computing research and development. Since 2022, he has been an Assistant Professor with the Department of Scientific Computing, Pukyong National University. His research interest includes all areas of quantum computing from device implementation to quantum algorithms.

...