https://doi.org/10.7840/kics.2024.49.7.1030

IBM Qiskit을 활용한 양자 회로 구현 소개

민 상 원◆, 이 정 수°

Introduction to Quantum Circuit Implementation Using IBM Qiskit

Sangwon Min◆, Jungsoo Rhee°

요 약

양자 컴퓨팅과 양자 기술은 암호학과 국방산업 등 다양한 분야에서 기존의 한계를 극복하고 새로운 패러다임을 제공하는 혁신적인 도구로 간주 되고 있다. 이러한 큰 영향력이 기대됨에도 불구하고 양자 컴퓨팅과 양자 기술을 사용할 수 있게 도와주는 양자 코딩에 대한 관심은 상대적으로 적은 것이 사실이다. 최근에는 접근성이 용이한 양자 코딩 플랫폼의 등장으로 실제 양자 컴퓨터를 활용하거나 양자 시뮬레이터를 통해 양자 코드를 작성하고 동작시킬 수 있는 환경이 조성되고 있다. 이에 따라 본 논문은 상용화된 양자 코딩 플랫폼들을 소개하고 큐비트와 양자 게이트에 대해 알아본다. 또, 이를 바탕으로 IBM의 Qiskit을 활용하여 고전 리플 캐리 덧셈(RCA) 알고리즘을 양자 RCA 회로로 구현하는 과정을 상세히 소개한다.

Key Words : Qiskit, Quantum Circuit, Quantum Simulator, Quantum Ripple Carry Adder

ABSTRACT

Quantum computing and quantum technology are considered revolutionary tools that overcome existing limitations and provide new paradigms in fields of cryptography and defense. Despite their significant potential impact, quantum coding, which enables the utilization of quantum computers and technologies, has received relatively less attention. Recently, the rise of user-friendly quantum coding platforms has created an environment where users can easily write and execute quantum code using real quantum computers or quantum simulators. This paper introduces commercial quantum coding platforms and explores qubits and quantum gates. Furthermore, it describes the implementation of a Quantum Ripple-Carry Adder(QRCA) circuit using IBM's Qiskit, based on the classical RCA algorithm.

Ⅰ. 서 론

양자 컴퓨팅 기술은 현대 사회의 복잡한 문제를 해결하기 위한 혁신적인 도구로 주목받고 있다. 이러한 기술은 전통적인 컴퓨터의 한계를 극복하고 바이오, 인공지능, 암호학 등 다양한 분야에서 새로운 패러다임을 제공할 것으로 기대되고 있다.

미국 국방부(U.S. Department of Defense)의 최고

※ 본 연구는 과학기술정보통신부 및 정보통신기획평가원의 ICT혁신인재4.0 사업의 연구결과로 수행되었음 (IITP-2024-2020-0-01825)
※ 이 기술은 2023년도 정부(산업통상자원부)의 재원으로 한국산업기술진흥원의 지원을 받아 개발한 결과물입니다. (P0008703, 2023년 산업혁신인재성장지원사업)
◆ First Author : Dept. of Smart Convergence Security, Busan University of Foreign Studies, alstkddnjs98@gmail.com, 학생회원
° Corresponding Author : Dept. of Smart Convergence Security, Busan University of Foreign Studies, rhee@bufs.ac.kr, 정회원
논문번호 : 202401-002-E-RU, Received December 28, 2023; Revised February 14, 2024; Accepted March 25, 2024

기술책임자인 M. D. Griffin은 양자 컴퓨터와 양자 통신 기술이 국방 분야에서 활용 가능하다고 하였다. 특히 양자 시계 및 센서 등의 기술은 상대적으로 더 빠르게 개발 및 구현될 수 있으며, 이는 GPS가 차단된 환경에서 통신을 유지하는 데 적용할 수 있다고 강조하였다.[1]

Mr. Feenstra는 미국 의회에 양자 기술을 활용한 분자 모델링과 시뮬레이션을 강화하는 법안을 제안하여 기존의 컴퓨터에 비해 훨씬 뛰어난 양자 기술의 속도와 정확도를 활용한 합성 비료, 약물 개발, 에너지 저장 기술의 혁신을 촉진하고자 하였다.[2]

암호학 측면에선 특히 큰 변화가 예측되는데, 양자 컴퓨팅이 발전하여 이론적으로만 구현된 Shor 알고리즘[3]과 Grover 알고리즘[4]이 현실적으로 사용 가능한 시점이 오면 현대에 많이 사용하는 RSA 알고리즘(Ron Rivest, Adi Shamir, Leonard Adleman Algorithm)이나 타원 곡선 암호 알고리즘(Elliptic Curve Cryptography Algorithm)과 같은 공개키 알고리즘은 Shor 알고리즘으로 인해 더 이상 사용이 불가능하고, AES(Advanced Encryption Standard)나 LEA(Lightweight Encryption Algorithm)와 같은 대칭키 알고리즘은 Grover 알고리즘의 공격으로부터 안전성을 보장받기 위해서 키 사이즈(bits)를 기존 대비 2배로 늘려야 한다.[5]

위와 같이 양자 컴퓨팅은 다양한 분야에서 혁신적인 변화를 가져올 전망이지만, 이러한 기술들을 작동시킬 양자 코딩에 대해서는 상대적으로 관심도가 적은 것이 사실이다. 과거에는 상용화된 양자 코딩 플랫폼이 존재하지 않았기에 양자 알고리즘 연구만 하였으나, 최근엔 접근성이 용이한 양자 코딩 플랫폼이 등장하여 실제 양자 컴퓨터에 접근하여 가동하거나, 시뮬레이터를 활용하여 양자 코드의 작동 여부를 파악할 수 있다. 이에 따라 본 논문에서는 쉽게 사용할 수 있도록 상용화된 여러 양자 코딩 시뮬레이터 플랫폼을 소개하고, 양자 코딩을 할 때 회로를 구성하는 절차를 소개한다. 또 이를 통해 IBM에서 제작한 Qiskit을 활용하여 고전 RCA(Ripple Carry Adder) 알고리즘을 양자 회로를 활용한 양자 RCA 회로로 구현할 것이다.

Ⅱ. 본 론

2.1 양자 컴퓨팅 시뮬레이터

최근엔 기술의 발전과 이전보다 높은 관심도에 의해 다양한 플랫폼에서 양자 코딩을 위한 양자 시뮬레이터를 제공하고 있다.

IBM Quantum Experience은 IBM이 2017년 3월에 출시한 양자 컴퓨팅 시뮬레이터 플랫폼으로, 양자 회로를 디자인하고 시뮬레이션 할 수 있는 서비스를 제공한다. 본 논문에서는 IBM에서 제공하는 Qiskit 패키지를 활용하여 프로그래밍하고 시뮬레이션을 실행하였다.[6]

Google에서 2018년 7월에 공식 출시한 Cirq는 양자 회로 시뮬레이션과 양자 알고리즘 개발을 위한 프레임워크로, 로컬에서 소스를 다운받거나 Colab에서 양자 시뮬레이션을 가동할 수 있다. Cirq에 대한 자세한 문서와 예제는 Google Quantum AI에서 제공하고 있다.[7]

마이크로소프트는 2017년 12월에 Quantum Development Kit을 처음 공개하였다. 이 툴킷은 마이크로소프트에서 자체 개발한 양자 프로그래밍 언어인 Q# 뿐만 아니라 Qiskit과 Cirq에 대한 지원을 제공하고 있다. 사용자는 Jupyter Notebook을 사용한 Azure Quantum 포털에서 필요한 양자 시뮬레이션을 수행할 수 있다.[8]

이러한 플랫폼들은 양자 회로를 실험하고, 양자 프로그래밍 언어를 학습하는 데에 유용하며 사용자의 선호에 따라 플랫폼을 결정할 수 있다.

2.2 큐비트

고전 컴퓨터는 기본단위로 비트(bit)를 사용하지만 양자 컴퓨터는 큐비트(qubit)를 사용한다. 큐비트를 브라켓 표기법을 활용해 간단한 수식으로 표현하자면, $|\Phi\rangle = \alpha|0\rangle + \beta|1\rangle$ 로 나타낼 수 있다. $|\Phi\rangle$ 은 큐비트의 상태, $\alpha|0\rangle$ 는 큐비트가 0일 확률, $\beta|1\rangle$ 는 큐비트가 1일 확률을 나타낸다. 이는 $|\alpha|^2+|\beta|^2=1$ 식을 항상 성립한다. 그림 1은 큐비트 3개를 할당받아 1번 큐비트에는 X 게이트, 2번 큐비트에는 H 게이트를 적용하고 각 큐비트의 상태를 블로흐 구를 사용하여 나타낸 그림이다. 큐비트를 할당받을 땐 기본적으로 $|0\rangle$ 상태로 초기화되어 주어지기 때문에 아무런 작업을 거치지 않은 0번 큐비트는 $|0\rangle$ 상태(0일 확률이 100%, 1일 확률이 0% 이므로 $\begin{pmatrix}1\\0\end{pmatrix}$ 수식으로 표현 가능), 1번 큐비트는 $|1\rangle$ 상태(0일 확률이 0%, 1일 확률이 100% 이므로 $\begin{pmatrix}0\\1\end{pmatrix}$ 으로 표현 가능), 2번 큐비트는 H gate를 활용하여 중첩(superposition)이 된 상태($\begin{pmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{pmatrix}$ 으로 표현 가능)이다. 큐비트는 이러한 중첩을 활용하여 이진 표현인 0과 1뿐만 아니라 양자 수학적인 성질을 이용하여 여러 상태를 동시에 가질 수 있다. 이들을 측정한다면 0번 큐비트는 항상 $|0\rangle$ 상태, 1번 큐비트는 $|1\rangle$ 항상 상태, 2번 큐비

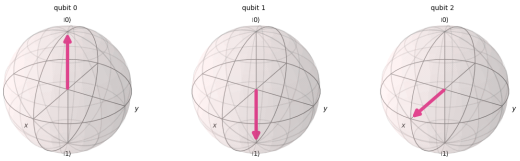

그림 1. 차례대로 $|0\rangle$ 상태, $|1\rangle$ 상태, 중첩 상태의 큐비트를 블로흐 구로 나타낸 모습
Fig. 1. Qubits in each $|0\rangle$ state, $|1\rangle$ state, and superposition state are shown in a Bloch sphere

트는 $|0\rangle$ 일 확률과 $|1\rangle$ 일 확률이 반반이므로 이 큐비트들을 100번 반복하여 측정한다고 가정하면 평균적으로 $|010\rangle$ 이 50번, $|110\rangle$ 이 50번씩 측정될 것이다.

여기서 주목해야 할 점은 큐비트를 '중첩 상태'로 만들면 $|0\rangle$ 과 $|1\rangle$ 의 상태를 동시에 지닐 수 있다는 것이다. 이는 양자 컴퓨터가 고전 컴퓨터와 가장 큰 차이점을 보이는 부분으로, 고전 컴퓨터에서 3 bit가 있다면 000, 001, 010, … , 111까지 총 8개의 정보 중 하나를 나타낼 것이다. 하지만 3 qubit라면 8개의 정보를 동시에 중첩하여 모두 표현하는 것이 가능하다. 또한 '측정'이라는 행위를 하는 순간 큐비트는 중첩 상태를 잃고 $|0\rangle$ 혹은 $|1\rangle$ 의 상태 중 하나를 갖는다는 특징을 활용하여 양자 통신 과정에서 도청이 이루어졌는지에 대한 여부를 파악할 수 있다.[9]

2.3 양자게이트

양자게이트는 전통적인 논리 게이트와 유사하지만 중첩 및 얽힘과 같은 양자의 물리적 특성을 바탕으로 다양한 양자 알고리즘과 기본 연산을 구현할 수 있다.

2.3.1 X 게이트

X 게이트는 양자 컴퓨팅의 기본적인 단일 큐비트 연산으로, 고전 전산의 NOT 게이트와 동일한 역할을 한다. X 게이트는 큐비트의 상태를 바꾸는데 사용되며, X 게이트를 양자 상태에 적용하면 표 1과 같이 $|0\rangle$ 상태는 $|1\rangle$ 로, $|1\rangle$ 상태는 $|0\rangle$ 로 바뀐다. 이러한 성질로 인해, X 게이트는 '비트 반전' 연산이라고도 불린다.

이러한 X 게이트의 수식은 $X=\begin{pmatrix}0&1\\1&0\end{pmatrix}$ 으로 정의된다. 즉, $|0\rangle=\begin{pmatrix}1\\0\end{pmatrix}$ 를 $|1\rangle=\begin{pmatrix}0\\1\end{pmatrix}$ 로, 혹은 $|1\rangle=\begin{pmatrix}0\\1\end{pmatrix}$ 를 $|0\rangle=\begin{pmatrix}1\\0\end{pmatrix}$ 로 바꾼다. 아래는 $|0\rangle$ 과 $|1\rangle$ 을 X 게이트로 반전한 수식이다.

$$X|0\rangle=\begin{pmatrix}0&1\\1&0\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}=\begin{pmatrix}0\\1\end{pmatrix}=|1\rangle \quad (1)$$

$$X|1\rangle=\begin{pmatrix}0&1\\1&0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}=\begin{pmatrix}1\\0\end{pmatrix}=|0\rangle \quad (2)$$

표 1. X 게이트의 효과
Table 1. Effect of X gate

Gate	Input	Output
X gate	$\|0\rangle$	$\|1\rangle$
	$\|1\rangle$	$\|0\rangle$

2.3.2 H 게이트

Hadamard 게이트, 줄여서 H 게이트는 양자 컴퓨팅에서 매우 중요한 게이트이다. 이 게이트는 단일 큐비트에 작용하며, 주로 큐비트를 '중첩 상태'로 변환하는 데 사용된다. 그림 1에서 볼 수 있듯이 $|0\rangle$ 혹은 $|1\rangle$ 상태의 큐비트에 H 게이트를 적용하면 $|0\rangle$ 상태에서는 $|+\rangle$ 로, $|1\rangle$ 상태는 $|-\rangle$ 로 상태가 바뀌고 측정 전까지 $|0\rangle$ 과 $|1\rangle$ 의 중첩 상태를 갖게 된다.

2.3.3 CX 게이트(CNOT 게이트)

CX 게이트는 2-큐비트 게이트로, 제어 비트(control bit; c)와 대상 비트(target bit; t)로 구성된다. 표 2와 같이 제어 비트가 '1' 상태일 때만 대상 비트의 상태에 X gate가 적용되어 반전되는데 이를 통해 양자 얽힘 상태를 생성하거나 조작할 수 있다.

고전 전산에서 가역회로인 CX 게이트는 $CX(c,t)=(c,c\oplus t)$ 로 표현될 수 있다. 이를 수식으로 표현하면 (3)과 같이 정의된다.

$$CX=\begin{pmatrix}I_2&0\\0&X\end{pmatrix}=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix} \quad (3)$$

예를 들어 $|00\rangle$ 는 $CX|00\rangle=|00\rangle$ 으로, $|10\rangle$ 는 $CX|10\rangle=|11\rangle$ 로 계산된다. 이는 다음 (4)(5)와 같이

표 2. CX 게이트의 효과
Table 2. Effect of CX gate

Gate	Input		Output
	control bit(c)	target bit(t)	$(c,c\oplus t)$
CX gate	$\|0\rangle$	$\|0\rangle$	$\|00\rangle$
	$\|1\rangle$	$\|0\rangle$	$\|11\rangle$
	$\|0\rangle$	$\|1\rangle$	$\|01\rangle$
	$\|1\rangle$	$\|1\rangle$	$\|10\rangle$

표현될 수 있다.

$$CX|00\rangle = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\begin{pmatrix}1\\0\\0\\0\end{pmatrix}=\begin{pmatrix}1\\0\\0\\0\end{pmatrix}=|00\rangle \quad (4)$$

$$CX|10\rangle = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\begin{pmatrix}0\\0\\1\\0\end{pmatrix}=\begin{pmatrix}0\\0\\0\\1\end{pmatrix}=|11\rangle \quad (5)$$

2.3.4 CCX 게이트(Toffoli 게이트)

Toffoli 게이트라고도 불리는 CCX 게이트는 양자 컴퓨팅에서 사용되는 3-큐비트 게이트이다. 이 게이트는 두 개의 제어 비트와 하나의 대상 비트로 구성되어 있으며 표 3과 같이 두 제어 비트 모두 1의 상태일 경우에만 대상 비트에 X 게이트를 적용하여 반전시킨다.

고전 전산에서 가역회로인 CCX 게이트는 $CCX(c_1,c_2,t)=(c_1,c_2,c_1c_2\oplus t)$로 표현된다. 단, c_1,c_2는 제어비트이고, t는 대상 비트이다. 이러한 CCX 게이트는 양자 컴퓨팅에서는 수식으로 $CCX=\begin{pmatrix}I_4&0\\0&CX\end{pmatrix}$이며, 8행 8열 사이즈의 행렬로 정의된다. 특히 Toffoli 게이트를 이용하면 고전 전산의 NAND 게이트와 FANOUT 게이트를 구현할 수 있다.[10]

표 3. CCX 게이트의 효과
Table 3. Effect of CCX gate

Gate	Input			Output
	control bit1(c1)	control bit2(c2)	target bit(t)	$(c_1,c_2,c_1c_2\oplus t)$
CCX gate	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|000\rangle$
	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|100\rangle$
	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|010\rangle$
	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	$\|111\rangle$
	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$	$\|001\rangle$
	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|101\rangle$
	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|011\rangle$
	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|110\rangle$

Ⅲ. 구 현

3.1 Qiskit을 활용한 양자 회로 구현

Qiskit의 제작사인 IBM에서 제안하는 양자 프로그램 작성 단계는 다음과 같다.[11]

1. 문제를 양자 형식으로 매핑
2. 회로 및 연산자 최적화
3. 양자 함수를 시용하여 실행
4. 결과 분석

위의 단계를 따르는 예시로 그림 1처럼 큐비트 3개를 할당받고 2번째 큐비트에는 X 게이트를 적용, 3번째 큐비트에는 H 게이트를 적용하고 블로흐 구로 표현하는 코드를 구현하면 다음과 같다.

1. 문제를 양자 형식으로 매핑
```
from qiskit import QuantumCircuit
circuit = QuantumCircuit(3)
circuit.x(1)
circuit.h(2)
```
2. 회로 및 연산자 최적화
   이 예시는 간단한 회로이므로 최적화가 필요하지 않다.
3. 양자 함수를 시용하여 실행
```
from qiskit.quantum_info import Statevector
state = Statevector.from_instruction(circuit)
```
4. 결과 분석
```
from qiskit.visualization import plot_bloch_multivector
plot_bloch_multivector(state)
```

먼저 어느 환경에서든, Qiskit을 활용하기 위해서는 pip install qiskit 명령어를 통해 qiskit 패키지를 우선 설치하여야 한다. 그리고 QuantumCircuit 함수[12]를 이용해 원하는 개수의 큐비트를 할당받고 편의를 위해 circuit 변수로 저장한다. 이 때 큐비트는 기본적으로 할당될 때 0번부터 시작하는 점을 주의해야 한다. 따라서 circuit 회로의 2번째 큐비트에 X 게이트를 적용하고 싶으면 circuit.x(1)과 같이 1번 큐비트를 지정하고, 3번째 큐비트에 H 게이트를 적용하려면 circuit.h(2)를 실행해야 한다. 원하는 문제를 양자 회로로 매핑하는 작업이 끝났으면 이 회로를 실행시키는 작업을 거쳐야 한다. 이 코드의 목적은 블로흐 구를 통해 양자 상태를 시각적으로 표현하는 것이므로 결과 분석 단계에서 블로흐 구를 보여주는 plot_bloch_multivector 함수[13]를 사용해야 한다. [13]에 의하면 블로흐 구를 표현하기 위해선 Statevector 값을 입력해야 하는데 이를 위해선 회로 실행 단계에서 Statevector 함수[14]를 작동해야 한다. 이에 따라 Statevector가 circuit의 큐비트 상태정보를 추출하고, 이를 바탕으로 plot_bloch_multivector함수가 동작하여 사용자가 큐비트의 상태를 블로흐 구 형태로 볼 수 있게 된다.

본 논문에서는 양자 코딩의 한 예시로, Python을 사용하여 IBM에서 개발한 오픈소스 양자 개발 도구인 Qiskit을 통해 앞서 소개한 게이트들을 조합하여 고전 Ripple Carry Adder를 양자 Ripple Carry Adder로 구현할 것이다. 일반적으로 산술연산은 양자 회로에서 가동하는 것보단 고전 회로에서 가동하는 것이 비용 측면에서 더 경제적이라고 알려져 있다.[15] 하지만 Shor 알고리즘의 인수분해 문제를 수행하는 과정에 양자 가산기 회로가 사용되는데, 이 때문에 양자 덧셈 알고리즘의 비용을 최소화하려는 연구 또한 오랫동안 진행되어 왔다.[16-18]

3.2 고전 Ripple Carry Adder 회로

Ripple Carry Adder (RCA)는 기본적인 이진 덧셈기 구조로, 간단하고 직관적인 구현 방법을 제공한다. 이 회로에서 필요한 변수의 종류는 총 4가지로, 두 입력값을 저장할 A와 B, 합계를 나타내는 S, 캐리 값을 저장할 C로 구성된다. S_i는 A_i와 B_i, C_i을 XOR 연산을 한 값으로, 논리식으로 나타내면 $S_i = A_i \oplus B_i \oplus C_i$와 같다. C_{i+1}는 i비트 단계에서 발생한 캐리를 나타내는 값으로, 논리식으로는 $C_{i+1} = A_iB_i + A_iC_i + B_iC_i$ 로 나타낼 수 있다. 단, +는 논리합(OR) 연산을 의미한다.

그림 2는 4bit 값(비트 길이:n=4)을 입력받았을 시 그려지는 RCA 회로이다.[19] 이 그림에서 볼 수 있듯이, RCA 회로는 연속된 덧셈기가 연결된 구조이다. 먼저 첫 번째 덧셈기에서 A_0과 B_0, C_0값을 XOR 연산하여 S_0값을 저장한다. 여기서 최초의 캐리 값인 C_0는 이전 입력값이 없으므로 0으로 초기화한다. 그리고 $A_0B_0 + A_0C_0 + B_0C_0$ 값을 계산하여 C_1로 저장하고 다음 덧셈기로 보낸다. 이 절차를 4번째 덧셈기까지 거쳐 각 덧셈기의 S_i값과 마지막 덧셈기의 C_4값까지 구한다. 최종적으로 이 회로의 출력은 S_0부터 S_3번까지와 C_4이 된다. (Output = (C_n,S_{n-1},S_{n-2}, ⋯, S_0))

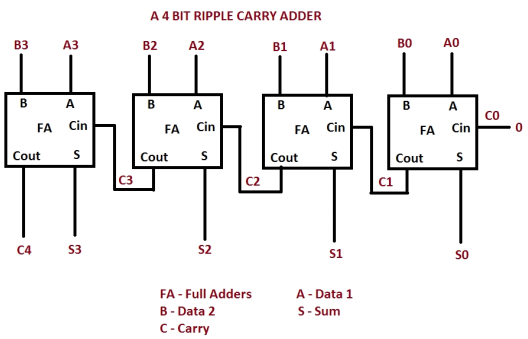

그림 2. 4비트 리플 캐리 가산기 회로도
Fig. 2. 4 bit Ripple Carry Adder circuit diagram

3.3 양자 Ripple Carry Adder 회로

IBM에서 제안하는 양자 프로그램 작성 단계에 따라 본 논문에서 구현할 양자 RCA 회로 프로그램의 작업을 분류하면 다음과 같다.

1-1. 덧셈 작업을 할 두 수(A, B)를 입력받고, A와 B 중 큰 값의 비트 길이(n)를 측정하여 덧셈 회로 구현 함수에 n, A, B 값을 보낸다.

1-2. 덧셈 회로 구현 함수에서 A, B, n 값을 활용하여 A값을 저장할 레지스터, B값을 저장할 레지스터, 연산 중 캐리 값을 저장할 Cin과 Cout 레지스터, 합계를 나타내는 S 레지스터, 결과를 측정하여 저장할 result 레지스터를 선언한다. 준비된 레지스터(A, B, Cin, Cout, S, result)를 양자 회로로 정의한다.

1-3. 입력된 두 수(A, B)를 이진수로 변환하고, 양자 회로상의 A와 B 큐비트에 각 이진수의 자릿수에 따라 X 게이트를 적용한다.

2. 고전 RCA 회로와 같이 $S_i = A_i \oplus B_i \oplus C_i$와 $C_{i+1} = A_iB_i + A_iC_i + B_iC_i$를 계산한다. 그리고 각 단계($i$)가 끝날 때마다 S_i 레지스터의 값을 측정하여 result 레지스터에 저장한다. 마지막 캐리 값(C_n)까지 측정하여 result 레지스터에 저장하고 덧셈 회로를 마친다.

3-1. 덧셈 회로를 시뮬레이션할 시뮬레이터를 설정하고 회로를 시뮬레이터와 transpile한다.

3-2. transpile한 회로를 몇 번 반복할지(shots) 설정하고 덧셈 회로 시뮬레이터를 실질적으로 가동시킨다.

4. 결과물을 원하는 형식으로 출력 받는다.

일반적으로 양자 알고리즘 논문들은 위 단계 중 2단계의 회로 및 연산자 최적화 부분만 언급하지만, 본 논문은 양자 시뮬레이션 코드를 처음부터 작성한다고 가정하고 모두 소개할 것이다.

3.3.1 문제를 양자 형식으로 매핑

3.3.1.1 계산할 값 입력 및 비트 길이 측정

이 단계는 양자 회로를 직접 제어하는 것이 아닌, 양자 형식으로 매핑하기 위한 사전 작업을 하기 위해 단순 파이썬 코드만 사용하는 과정이다.

Code 1과 같이 더하고 싶은 두 자연수를 입력받아 각각 num1, num2로 저장하고 둘 중 큰 값을 기준으로 비트 길이를 정하여 num_qubits 변수에 저장한다. 이

변수는 추후 큐비트를 할당받는 단계에 필요하다. num_qubits, num1, num2 값을 덧셈 회로 구현 함수에 전달하고 결과를 circuit으로 저장한다.

```
num1 = int(input("첫번째 수 입력 : "))
num2 = int(input("두번째 수 입력 : "))
num_qubits = max(num1.bit_length(),num2.bit_len
gth())
circuit = adder_circuit(num_qubits, num1, num2)
```

Code 1. Input value initialization

3.3.1.2 큐비트 할당 및 양자 회로 선언

양자 연산을 위해 큐비트를 할당하는 단계이다. Code 2는 이 단계를 수행하는 코드이다. 다수의 큐비트를 효율적으로 제어할 수 있게 QuantumRegister[20]를 활용하였다. 입력받은 A와 B를 양자 회로 상에 구현하기 위해 각각 num_qubits 만큼의 큐비트를 할당받고 A와 B라고 변수를 지정하였다. 각 단계마다 Cin, Cout, S 값을 계산하기 위해 num_qubits만큼 큐비트를 할당받았다. 마지막 캐리 비트로 인해 총 비트 수가 1개 늘어나는 것을 고려하여 Cin 레지스터는 1개를 추가로 받았다. 이러한 계산의 결과를 측정하여 고전 비트(0과 1)의 상태로 저장할 ClassicalRegister도 num_qubits+1 만큼 비트를 할당받고 result라는 변수로 지정하였다.

```
from qiskit import (QuantumCircuit,
        QuantumRegister, ClassicalRegister)
def adder_circuit(num_qubits, num1, num2):
    A = QuantumRegister(num_qubits, 'a')
    B = QuantumRegister(num_qubits, 'b')
    Cin = QuantumRegister(num_qubits+1, 'cin')
    Cout = QuantumRegister(num_qubits, 'cout')
    S = QuantumRegister(num_qubits, 's')
    result = ClassicalRegister(num_qubits+1,
'result')
    qc = QuantumCircuit(A, B, Cin, Cout, S,
result)
```

Code 2. qubit allocation and quantum circuit declaration

3.3.1.3 양자 레지스터에 입력값 적용

Code 3은 입력받은 A, B값에 맞게 양자 레지스터 A와 B 큐비트에 각 이진수의 자릿수에 따라 X 게이트를 적용하는 코드이다.

예를 들어, A에 4를 입력받으면 이진 표기법 100_2으로 나타낼 수 있고 이를 양자 레지스터 A에 $|100\rangle$으로 표현하기 위해서 1번째, 2번째 큐비트는 건드리지 않고 3번째 큐비트에 X게이트를 적용하여 $|1\rangle$로 바꾸는 작업이다. 그림 3은 이를 시각적으로 표현한 결과이다. 3번째 큐비트(2번 큐비트)에만 X 게이트를 적용하고 이 회로의 큐비트를 차례로 측정하면 $|100\rangle$ 값이 나온다.

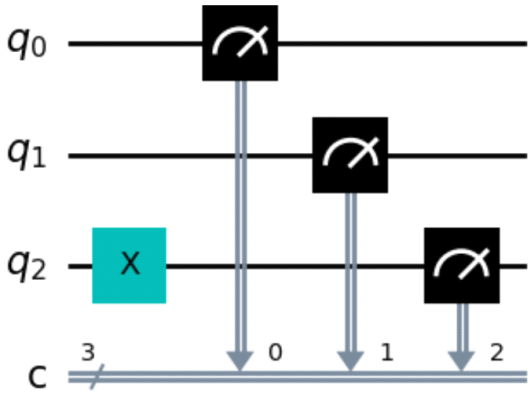

그림 3. 큐비트를 $|100\rangle$로 설정하고 측정하는 회로
Fig. 3. Circuit to set qubit to $|100\rangle$ and measure

```
bin_num1 = bin(num1)[2:].zfill(num_qubits)
bin_num2 = bin(num2)[2:].zfill(num_qubits)
print("비트 수 : ",num_qubits)
for i in range(num_qubits):
    if bin_num1[num_qubits - 1 - i] == '1':
        qc.x(A[i])
    if bin_num2[num_qubits - 1 - i] == '1':
        qc.x(B[i])
```

Code 3. Apply input to quantum register

3.3.2 RCA 회로 구현 및 연산자 최적화

실질적으로 덧셈 작업이 이루어지는 단계이다. 고전 RCA 회로에서 S값은 $S_i = A_i \oplus B_i \oplus C_i$, C_{i+1}값은 $C_{i+1} = A_iB_i + A_iC_i + B_iC_i$ 의 계산을 따랐다. 양자 회로에서도 고전 회로와 같은 결과 도출이 가능하다. Code 4 는 덧셈 작업 및 측정을 수행하는 코드이다.

```
for i in range(num_qubits):
    qc.cx(A[i], S[i])
    qc.cx(B[i], S[i])
    qc.cx(Cin[i], S[i])
    qc.ccx(A[i], B[i], Cout[i])
    qc.ccx(B[i], Cin[i], Cout[i])
    qc.ccx(A[i], Cin[i], Cout[i])
    qc.cx(Cout[i], Cin[i+1])
```

```
        qc.measure(S[i], result[i])
        if i+1 == num_qubits:
            qc.measure(Cout[i], result[num_qubits])
    return qc
```

Code 4. Addition operation implementation code

각 i번째 큐비트의 S_i는 A_i, B_i, C_{in_i}를 제어비트로 하는 CX 연산을 하고 측정되어 $result[i]$에 저장된다. C_{out_i}은 A_i와B_i, B_i와C_{in_i}, A_i와C_{in_i}을 제어비트로 하여 CCX 연산을 수행하고 그 결과를 제어비트로, $C_{in_{i+1}}$을 목표 비트로 하여 CX 연산을 수행한다. 이 작업을 마지막 n-bit까지 수행하면 C_{out_n} 값을 $result[n]$에 측정하고 회로를 마친다. 예를 들어, 큐비트가 초기화될 때 $|0\rangle$ 상태이므로 C_{in_0}는 그대로 $|0\rangle$이고, A_0가 $|1\rangle$,B_0가 $|1\rangle$이라고 가정했을 때, S_0는 A_0와 B_0에 의해 총 2번 CX 게이트가 동작하여 $|0\rangle$이고, C_{out_0}는 A_0와 B_0 모두 $|1\rangle$이기 때문에 CCX 게이트가 1번 동작하여 $|1\rangle$ 상태가 된다.

3.3.3 구현된 양자 회로 실행

양자 코딩은 단순히 회로 구현이 다 되었다고 완성된 것이 아니다. 어떤 결과를 원하는지에 따라 회로 실행 코드가 바뀔 수 있다. 본 논문에서 구현한 양자 RCA 회로는 결과물로 A와 B의 덧셈 결과를 원하기 때문에 측정된 큐비트의 상태가 $|0\rangle$ 인지 $|1\rangle$ 인지를 한눈에 보는 것이 바람직하다. Code 5는 이에 해당하는 코드이다.

```
from qiskit import Aer, transpile, assemble
simulator = Aer.get_backend('qasm_simulator')
transpiled_circuit = transpile(circuit, simulator)
qobj = assemble(transpiled_circuit, shots=10)
job = simulator.run(qobj)
result = job.result().get_counts(circuit)
```

Code 5. output format code

qasm_simulator[21]는 qiskit에서 제공하는 기본적인 백엔드 시뮬레이터로, 양자 회로를 표준 기반 게이트 세트로 변환하여 대부분의 양자 회로에 적합하다. transpile 함수[22]는 회로에서 사용된 게이트의 순서를 특정 시뮬레이션에 맞게 최적화하여 계산 속도나 정확도에 영향을 준다. assemble 함수[23]를 통해 회로의 반복 횟수를 조절할 수 있다. 일반적인 양자 회로에서는 큐비트에 H 게이트를 적용하여 양자 중첩을 발생시키기 때문에 측정을 많이 할수록 정확도가 높아지지만 본 논문의 회로에선 중첩현상을 발생시키는 H 게이트가 사용되지 않았으므로 shots 값에 상관없이 결과가 동일하다. simulator.run 함수로 시뮬레이터를 가동시키고 결과를 출력한다.

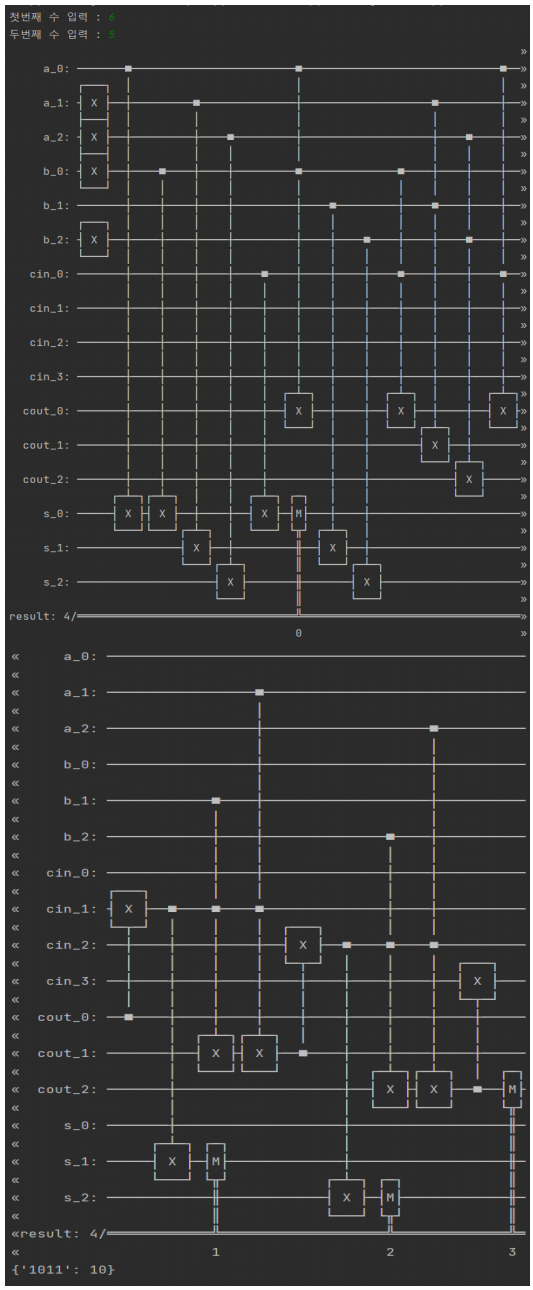

그림 4. 6과 5를 입력했을 때 동작하는 회로의 모습
Fig. 4. The circuit that operates when 6 and 5 are input

3.3.4 결과물 출력

```
print(circuit)
print(result)
```

Code 6. print code

print(circuit)을 통해 회로의 작업 과정을 볼 수 있다. 그림 4는 본 논문에서 구현한 양자 RCA 회로에 6과 5를 입력했을 때 동작한 회로의 모습이다. 6이라는 값을 양자 회로에 적용시키기 위해 A 레지스터의 1번, 2번 큐비트에 X 게이트가 작동하였고, 5라는 값을 B 레지스터에 적용시키기 위해 0번, 2번 큐비트에 X 게이트를 작동된 것을 볼 수 있다.

print(result)에 의해 최종 결과로 {'1011': 10}이라는 값이 출력되었다. '1011'은 10진법으로 계산하면 11을 의미하고 : 표시 뒤의 10은 shots에서 설정한 횟수만큼 반복했을 때 도출된 결과의 횟수를 의미한다.

Ⅳ. 결 론

여러 상용화된 양자 시뮬레이터들과 큐비트의 성질, 간단한 양자 게이트를 설명하였다. 또한 본 논문에서는 양자 프로그램 작성 4단계에 따라, IBM에서 제작한 양자 코딩 툴인 Qiskit을 활용하여 고전 RCA 알고리즘을 양자 RCA 회로로 구현하였다. 그리고 구현한 회로를 직접 양자 시뮬레이터를 활용하여 실행해 보았다. 이를 통해 고전 PC에서 필요한 양자 코딩을 활용하여 양자 회로 시뮬레이션을 어렵지 않게 작동시킬 수 있다는 점도 하나의 시사점이 될 것이다. 이는 Qiskit과 같이 상용화된 플랫폼에서도 고품질의 양자 회로 설계를 할 수 있다는 가능성을 보여준다.

향후 양자 회로 알고리즘을 구현하는 양자 코딩에 대해 할당받는 큐비트의 개수나 게이트의 비용, 회로 깊이 등 양자 회로 비용을 고려한 회로를 개발하는 것을 연구 과제로 삼을 것이다.

References

[1] C. T. Lopez, *DOD Should Focus on Short-Term Goals in Quantum Science*(2020), Retrieved Jan. 17, 2024, from https://www.defense.gov/News/News-Stories/Article/Article/2110617/dod-should-focus-on-short-term-goals-in-quantum-science/

[2] Mr. Feenstra, *H.R.1748 - Quantum in Practice Act*(2023), Retrieved Jan., 22, 2024, from https://www.congress.gov/bill/118th-congress/house-bill/1748/text

[3] P. Shor, "Polynomial time algorithms for discrete logarithms and factoring on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484-1509, Oct. 1997. (https://doi.org/10.1137/S0097539795293172)

[4] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. Twenty-Eighth Annu. ACM Symp. Theory of Comput.*, pp. 212-219, Nov. 1996. (https://doi.org/10.48550/arXiv.quant-ph/9605043)

[5] KISA, *Guide to using cryptography technology in a quantum computing environment*(2016), Retrieved Oct. 13, 2023, from https://www.kisa.or.kr/2060305/form?postSeq=13&page=1

[6] IBM Qiskit, *High-Performance Simulator Tutorials*(2017), Retrieved Nov. 15, 2023, from https://qiskit.org/documentation/stable/0.24/tutorials/simulators/1_aer_provider.html

[7] Google quantum AI, *Cirq Simulate*(2018), Retrieved Nov. 15, 2023, from https://quantumai.google/cirq/simulate

[8] Microsoft Azure, *Azure Quantum Simulators* (2017), Retrieved Nov. 17, 2023, from https://learn.microsoft.com/en-us/azure/quantum/machines/

[9] E. Lee, "A study on the application of quantum key distribution to IPSec," *Convergence Secur. J.*, vol. 21, no. 3, pp. 3-11, Sep. 2021. (http://dx.doi.org/10.33778/kcsa.2021.21.3.003)

[10] H.-W. Lee, *Quantum Information Science Lecture*, Science books, p. 197, 2017.

[11] IBM Quantum, *Hello world*(2023), Retrieved Nov. 17, 2023, from https://docs.quantum.ibm.com/start/hello-world

[12] IBM Quantum, *Quantum Circuit*(2023), Retrieved Nov. 20, 2023, from https://docs.quantum.ibm.com/api/qiskit/circuit

[13] IBM Quantum, *qiskit.visualization.plot_bloch_multivector*(2023), Retrieved Jan. 23, 2024,

from https://docs.quantum.ibm.com/api/qiskit/qiskit.visualization.plot_bloch_multivector
[14] IBM Quantum, *Statevector*(2023), Retrieved Jan. 23, 2024, from https://docs.quantum.ibm.com/api/qiskit/qiskit.quantum_info.Statevector
[15] Microsoft Azure, *Hybrid quantum computing concepts*(2023), Retrieved Jan. 30, 2024, from https://learn.microsoft.com/ko-kr/azure/quantum/hybrid-computing-concepts
[16] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton. "A new quantum ripple-carry addition circuit," *arXiv preprint quant-ph/0410184*, Oct. 2004. (https://doi.org/10.48550/arXiv.quant-ph/0410184)
[17] Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum addition circuits and unbounded fan-out," *Quantum Inf. and Computation*, vol. 10, no. 9, pp. 872-890, Sep. 2010. (https://doi.org/10.48550/arXiv.0910.2530)
[18] T. Häner, M. Roetteler, and K. M. Svore, "Factoring using 2n+2 qubits with toffoli based modular multiplication," *Quantum Inf. and Computation*, vol. 17, no. 7 & 8, pp. 673-684, Jun. 2017. (https://doi.org/10.48550/arXiv.1611.07995)
[19] Allthingsvlsi, *4 Bit Ripple Carry Adder*(2013), Retrieved Aug. 16, 2023, from https://allthingsvlsi.wordpress.com/2013/04/30/4-bit-ripple-carry-adder
[20] IBM Quantum, *QuantumRegister*(2023), Retrieved Jan. 23, 2023, from https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.QuantumRegister
[21] IBM Quantum, *QasmSimulator*(2020), Retrieved Jan. 23, 2023, from https://qiskit.org/ecosystem/aer/stubs/qiskit_aer.QasmSimulator.html#qiskit_aer.QasmSimulator
[22] IBM Quantum, *transpiler*(2023), Retrieved Jan. 23, 2023, from https://docs.quantum.ibm.com/api/qiskit/transpiler
[23] IBM Quantum, *qiskit.compiler.assemble*(2021), Retrieved Jan. 26, 2023, from https://docs.quantum.ibm.com/api/qiskit/0.33/qiskit.compiler.assemble

민 상 원 (Sangwon Min)

2023년 2월 : 부산외국어대학교 스마트융합보안학과 학사 졸업
2023년 3월~현재 : 부산외국어대학교 일반대학원 스마트융합보안학과 석사과정
<관심분야> 사이버보안, 암호학, 양자 프로그래밍
[ORCID:0009-0004-7240-8547]

이 정 수 (Jungsoo Rhee)

1982년 2월 : 경북대학교 사범대학 수학교육과 학사 졸업
1984년 2월 : 경북대학교 일반대학원 수학과 석사 졸업
1993년 8월 : Florida State Univ. 수학과 박사 졸업
1994년~현재 : 부산외국어대학교 스마트융합보안학과 교수
<관심분야> AI/사이버보안, 양자역학, 암호학, Fourier Analysis
[ORCID:0009-0004-9354-5970]