공학석사 학위논문

양자 오류 정정 부호의 효율적인 구현을 위한 이온 트랩 칩 구조

2023 년 2 월

서울대학교 대학원
컴퓨터공학부
이정훈

양자 오류 정정 부호의 효율적인 구현을 위한 이온 트랩 칩 구조

지도교수 김 태 현

이 논문을 공학석사 학위논문으로 제출함

2023 년 2 월

서울대학교 대학원

컴퓨터공학부

이정훈

이정훈의 석사 학위논문을 인준함

2023 년 2 월

위 원 장	하 순 회	(인)
부 위 원 장	김 태 현	(인)
위 원	이 창 건	(인)

요약(국문초록)

이온 트랩 기반의 양자컴퓨터는 전하를 띈 입자를 전기장을 이용해 공간상에 포획하고, 외부 전자기장을 통해 이온 상태를 변화시킴으로써 양자정보를 처리한다. 칩 트랩은 DC 전극과 RF 전극을 가진 2차원 칩에 필요한 전압을 가함으로써 이온을 포획하며, 향후 기술 발전에 따라 많은 이온을 포획하도록 구조적 확장이 쉽다는 장점이 있다. 많은 이온이 포획된 칩 트랩을 이용해 효율적인 양자 연산을 하기 위해서는 이온 위치, 이동 순서 등 여러 변수에 대한 최적화가 필요한데, 이때 최적화된 변수는 실행하고자 하는 양자 알고리즘에 따라 다르다. 따라서 범용성, 효율성을 모두 가진 양자컴퓨터 구현을 위해 하드웨어 수준의 구조 최적화와 양자 알고리즘, 스케줄링 등 다양한 수준에서 최적화가 필요하다.

한편 전기 신호를 이용해 비트 단위로 정보를 처리하는 고전컴퓨터와 달리, 양자컴퓨터에 사용되는 qubit는 외부 잡음에 취약하며 물리적 특성상 저장한 정보를 잃기 쉽다. 양자컴퓨터 구현 방법에 따라 차이는 있으나, 물리적인 이온 하나를 실제 qubit 하나에 대응시켜 양자 회로를 구현할 경우 결과에 대한 신뢰도가 매우 떨어진다. 이를 해결하기 위해 여러 이온에 인코딩(encoding) 회로를 가함으로써 한 qubit의 정보를 저장하는 양자 오류 정정 부호를 사용하고, 연산 또한 인코딩된 다수의 qubit에 적절한 논리적 게이트를 가함으로써 이루어진다. 인코딩에 필요한 qubit 개수 최적화, 인코딩에 필요한 게이트 개수 등을 고려해 최적의 양자 오류 정정 부호를 찾기 위한 다양한 연구가 진행 중이며, color 코드는 높은 확장성과 간단한 구조의 논리적 게이트를 가진다는 장점이 있다. Color 코드는 Clifford 게이트 생성자에 해당하는 Hadamard 게이트(H), phase 게이트(S), CNOT 게이트($CNOT$)에 대해 transversal 한 연산을 지원하는데, 인코딩에 사용된 모든 qubit에 대해 얽힘 연산 없이 하고자 하는 게이트를 개별적으로 가하는 것만으로도 원하는 논리적 연산을 할 수 있다. 양자 알고리즘에 따라 사용하는 게이트별 개수와 순서에는 차이가 있으나 정확한 연산을 위해 color 코드를 사용할 경우 하드웨어 수준에서 작동하는 연산에는 일정한 패턴이 존재한다. 본 연구에서

는 color 코드의 효율적인 구현을 위한 이온 트랩 칩 구조를 제안하고, 시뮬레이션을 통해 제안한 칩 구조의 특징에 대해 분석하였다. 표준화된 기술이 존재하지 않기 때문에, 현재 연구실에서 달성 가능한 수치 및 여러 연구에서 검증된 수치를 바탕으로 시뮬레이션하였으며, 1-qubit 게이트 및 2-qubit 게이트를 포함한 다양한 양자 회로를 이용하여 실제 양자 회로가 칩 위에서 작동했을 때 결과를 분석하였다.

.....

주요어 : 양자컴퓨터, 양자 정보, 이온 트랩, 칩 트랩,
양자 오류 정정 부호, color 코드, 칩 시뮬레이션
학 번 : 2021-24050

목차

1. 서론 ····· 1
2. 이론적 배경 ····· 2
2.1 양자 정보 ····· 2
2.1.1 큐비트(Qubit) ····· 2
2.1.2 양자 연산 및 게이트 ····· 3
2.2 양자 오류 정정 부호 ····· 5
(Quantum Error Correction code, QEC code)
2.2.1 Stabilizer Formalism ····· 7
2.2.2 Steane 코드와 Color 코드 ····· 8
2.3 이온 트랩 기반 양자컴퓨터 ····· 11
3 효율적인 양자 오류 정정을 위한 이온 트랩 칩 구조 13
4 이온 트랩 칩 시뮬레이션 ····· 16
4.1 칩 시뮬레이터 구조 ····· 16
4.1.1 양자 회로 생성 및 칩 구조 모델링 ····· 17
4.1.2 칩 시뮬레이션 전략 및 알고리즘 ····· 21
4.1.3 에러 모델 적용 및 성능 평가 ····· 31
4.2 시뮬레이션 결과 ····· 32
5. 결론 ····· 38
참고문헌

1. 서론

1980년대에 양자역학을 이용한 연산과 양자컴퓨터에 대한 개념이 제시된 이래로 양자 연산에 관한 이론적 연구 및 실험적 구현을 위한 다양한 연구가 이루어졌다. 소인수분해 문제, 탐색 공간에서의 Grover 탐색 문제에 있어서는 양자 알고리즘이 고전컴퓨터에서의 알고리즘보다 더 낮은 시간복잡도를 가진다는 것이 증명되었으며, 양자 화학에서의 복잡한 분자 구조 분석, 다물체 시스템(many-body system) 등의 문제에서도 양자컴퓨터가 더 빠를 것으로 기대된다[1-3].

양자컴퓨터 구현을 위해 이온 트랩, 초전도, 질소-공동센터(Nitrogen-Vacancy Center) 등 다양한 기술이 연구되고 있는데, 이온 트랩 기반의 양자컴퓨터는 긴 결맞음 시간(coherence time), 높은 신뢰도(fidelity)를 갖고 확장성이 좋다는 장점을 갖고 있다[4-7]. 특히 칩 기반의 이온 트랩 양자컴퓨터는 밀리미터 규모의 작은 칩을 이용해 전기장 퍼텐셜을 만들어 수십 개의 이온을 포획하고, 퍼텐셜 모양을 바꾸어가며 포획된 이온을 이동시킬 수 있다[8]. 현재까지 수십 개의 이온을 포획한 이온 트랩 기반의 양자컴퓨터가 개발되었는데, 이론적으로는 더 많은 이온을 포획할 수 있으나 상용화 가능한 수준에 도달하기 위해서는 많은 기술적 난관이 존재한다. 특히 임의의 양자 알고리즘을 실행하기 위해서, 양자컴퓨터는 10^{10}개 이상의 양자 게이트를 10^{-10} 이하의 오류율로 실행할 수 있어야 한다[9]. 양자 시스템은 외부 noise에 취약해서 저장된 정보를 잃기 쉬우므로, 낮은 오류율을 달성하기 위해 양자 오류 정정 부호(QEC Code, Quantum Error Correction Code)를 사용한다[10].

양자 오류 정정 부호는 여러 개의 물리적 큐비트(qubit)를 이용해 하나의 논리적 qubit에 해당하는 정보를 저장하고, 일부 qubit에 오류가 발생하면 사용한 오류 정정 부호에 해당하는 양자 회로를 이용해 오류 정보를 추출함으로써 원래 정보를 복원한다. 대표적으로 Shor의 9-qubit 코드는 9개의 물리적 qubit에 한 개 qubit의 정보를 저장하며[11], 사용하는 물리 qubit 개수, 오류 threshold, transversal 한 게이트 개수와 같은 지표를 향상시키기 위해 다양한 양자 오류 정정 부호가 개발되었다.

Color 코드는 높은 확장성을 갖고, 국소적인 qubit 간 연산만으로 이루어진 직관적인 오류 정보 추출과정을 갖기 때문에 향후 사용될 가능성이 큰 오류 정정 부호 중 하나이다[12-14]. 특히 color 코드는 많은 transversal 게이트를 지원하기 때문에 여러 논리적 qubit 간 연산 과정도 쉽다는 장점이 있고, 이온 트랩을 이용한 양자오류정정 과정 실험에서 자주 사용된다[15-17].

본 논문에서는 효율적인 양자 오류 정정 부호 구현을 위한 이온 트랩 칩 구조를 제안하고, 해당 칩을 이용한 양자컴퓨터 상에서의 이온 이동 및 양자 게이트 시퀀스에 대한 시뮬레이션을 통해 회로 실행 시간, 논리적 qubit의 오류율 등을 분석하였다. 이온 트랩 상에서의 qubit 이동 스케줄링에 관한 기존의 연구들은 임의의 양자 회로를 가정하였으나, 실제 양자컴퓨터는 양자 오류 정정 부호로 인코딩된 정보를 활용할 가능성이 크며 이에 따라 이온의 이동 순서 및 이온 간 연산 패턴에 규칙성이 존재할 수 있다. 따라서 이를 고려해 이온 트랩 칩을 설계하면 하드웨어 수준에서 양자 회로를 최적화할 수 있으며, 양자 알고리즘 실행 시간 단축, 신뢰도 향상, 칩 집적도 향상을 기대할 수 있다.

2. 이론적 배경

2.1. 양자 정보

2.1.1. 큐비트(Qubit)

일반적으로 고전컴퓨터에서의 정보는 0 또는 1의 이진수로 표현되며, 셀에 저장된 전자의 개수 또는 wire에 흐르는 전류 값을 이용해 0과 1을 표현한다. 양자컴퓨터에서도 0과 1에 해당하는 basis(기저)를 이용해 정보를 표현하지만 고전컴퓨터와 달리 0과 1의 중첩상태를 허용한다. 양자역학에서 전자와 같은 입자는, 입자가 속한 시스템을 기술하는 Hamiltonian에 대한 basis의 중첩으로 표현되는 파동함수로 기술되고, Hamiltonian 또는 입자에 가해지는 외부의 작용은 basis에 가해지는 행렬로 이해할 수 있다. 양자컴퓨터는 여러 basis 중 두 basis($|0\rangle$,$|1\rangle$)를 선택해 원하는 파동함수를 만들고 정보를 나타내는데, 양자컴퓨터에서

정보를 나타내는 기본 단위를 qubit(큐비트)라고 하고 다음과 같이 나타낼 수 있다.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (\alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1)$$

일반적으로 n-qubit에 해당하는 정보는 2^n-dimension을 가진 Hilbert space상의 벡터로 표현되며, $|00\cdots0\rangle$부터 $|11\cdots1\rangle$까지 2^n개 basis의 합으로 나타낸 일반적인 양자 상태는 다음과 같다.

$$|\psi\rangle = \alpha_1\alpha_2\cdots\alpha_n|00\cdots00\rangle + \alpha_1\alpha_2\cdots\alpha_{n-1}\beta_n|00\cdots01\rangle + \alpha_1\alpha_2\cdots\beta_{n-1}\alpha_n|00\cdots10\rangle$$
$$+\cdots+\beta_1\beta_2\cdots\alpha_n|11\cdots10\rangle + \beta_1\beta_2\cdots\beta_n|11\cdots11\rangle$$

2.1.2. 양자 연산 및 게이트

일반적으로 고전컴퓨터에서 저장된 정보를 수정하고 복잡한 연산을 하기 위해 XOR, OR, AND와 같은 연산을 수행한다. 고전적인 정보는 0과 1의 구분이 명확하므로 주어진 길이의 이진수로 표현할 수 있는 정보의 개수가 유한하고, 정의될 수 있는 연산 또는 함수의 개수도 유한하다. 반면 양자 정보에서 qubit는 $|0\rangle$,$|1\rangle$의 연속적인 중첩상태를 허용하기 때문에 이론적으로 한 개의 qubit만으로도 무한한 정보를 표현할 수 있고, 행렬로 표현되는 연산의 종류도 무한하다.

양자역학의 특성상 파동함수에 가해질 수 있는 연산은 unitary 해야 하는데, 1-qubit에 정의되는 unitary 행렬은 다음과 같이 표현된다.

$$U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \text{ with unitarity } UU^\dagger = U^\dagger U = I$$

$$\Rightarrow U|\psi\rangle = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha u_{11} + \beta u_{12} \\ \alpha u_{21} + \beta u_{22} \end{pmatrix}$$

$$= (\alpha u_{11} + \beta u_{12})|0\rangle + (\alpha u_{21} + \beta u_{12})|1\rangle$$

$$(u_{ij}, \alpha, \beta \in \mathbb{C},\ |\alpha|^2 + |\beta|^2 = 1)$$

1-qubit에 정의된 unitary 행렬 중 대표적으로 Pauli operator가 있으며, Pauli operator의 형태 및 만족하는 성질은 다음과 같다.

$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$, $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

$\Rightarrow I^2 = X^2 = Y^2 = Z^2 = I$ (self-inverse)

For arbitrary U, $U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} = \left(\frac{u_{11}+u_{22}}{2}\right) I + \left(\frac{u_{12}+u_{21}}{2}\right) X + i\left(\frac{u_{12}-u_{21}}{2}\right) Y + \left(\frac{u_{11}-u_{22}}{2}\right) Z$

2-qubit 연산자에는 대표적으로 Controlled-NOT($CNOT$)이 있으며, 첫 번째 qubit(control qubit)가 $|0\rangle$일 때 두 번째 qubit(target qubit)에 I 게이트를, 첫 번째 qubit가 $|1\rangle$일 때 두 번째 qubit에 X 게이트를 가한다.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

일반적으로 n-qubit 연산자는 unitary 성질을 만족하는 $2^n \times 2^n$ 행렬로 표현된다.

Solovay-Kitaev 정리에 따르면, 유한한 단일 qubit 게이트와 $CNOT$ 게이트만으로 임의의 n-qubit 게이트를 임의의 정확도로 근사할 수 있으며 [18], 이때 사용된 단일 qubit 게이트 집합과 $CNOT$ 게이트를 universal quantum 게이트라 한다. Universal 게이트 집합은 유일하지 않으며, 대표적으로 $\{H, S, T, CNOT\}$, $\{H, S, CNOT, Toffoli\}$이 universal 한 게이트 집합이다[19].

또한, Pauli 그룹의 normalizer 그룹을 Clifford 그룹이라 부르며[20, 21], 다음 관계를 만족한다.

$$U \in \mathcal{C}, P \in \mathcal{P} \Rightarrow UPU^\dagger \in \mathcal{P}$$

($\mathcal{C}$: Clifford 그룹, $\mathcal{P}$: Pauli 그룹)

Clifford 게이트에는 대표적으로 Hadamard 게이트(H)와 phase 게이트(S)가 있으며, 위 관계식을 쉽게 확인할 수 있다.

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

$$HXH^\dagger = Z, HYH^\dagger = -Y, HZH^\dagger = X$$

$$SXS^\dagger = Y, SYS^\dagger = -X, SZS^\dagger = Z$$

Gottesmann-Knill 정리에 따르면, Clifford 게이트만으로 이루어진 양자

회로는 고전컴퓨터로도 다항 시간 내에 시뮬레이션할 수 있다[22, 23]. 그러나 Clifford 게이트만으로 구성된 임의의 게이트 집합은 universal 게이트를 구성할 수 없으며, 임의의 양자 회로를 구현하기 위해서는 non-Clifford 게이트에 대한 구현이 필요하다[19].

2.2. 양자 오류 정정 부호(Quantum Error Correction code, QEC code)

소인수분해, Grover 탐색 알고리즘 등 양자 알고리즘에 관한 이론적 연구가 이루어진 데 반해, 실제 양자컴퓨터에서 양자 알고리즘을 구현하는 데에는 많은 기술적 난관이 존재한다. 기술에 따라 차이가 있으나, qubit을 구현하기 위해 사용하는 초전도 회로 또는 이온은 신뢰도가 낮으며, 레이저와 자기장 등을 발생시키는 장치에의 노이즈, 주변 온도에 매우 민감하게 반응한다. 극저온, 진공을 통한 장비의 정밀도 향상, 데이터 후처리 등을 통해 qubit 신뢰도를 어느 정도 올릴 수 있으나[24-29], 정확한 양자 회로 구현을 위해서는 양자 오류 정정 부호의 사용이 필요하다. 양자 오류 정정 부호는 여러 개의 qubit을 이용해 한 개 qubit에 해당하는 정보를 저장하며, 해당 qubit들 중 일부에 오류가 발생한 경우 특정 양자 회로를 가해 오류에 대한 정보를 추출한다. 이때 추출된 오류 정보를 신드롬(syndrome) 이라 부르며, 추출된 신드롬을 바탕으로 보상 게이트 연산을 가해 오류를 상쇄할 수 있다[30, 31].

Shor의 9-qubit 코드는 대표적인 양자 오류 정정 부호로, 9개의 qubit을 이용해 한 개 qubit에 해당하는 정보를 저장한 뒤 한 개의 X-에러와 Z-에러를 수정할 수 있다[11]. Shor 코드를 이용한 인코딩에 해당하는 양자 회로는 다음과 같다([그림 1]).

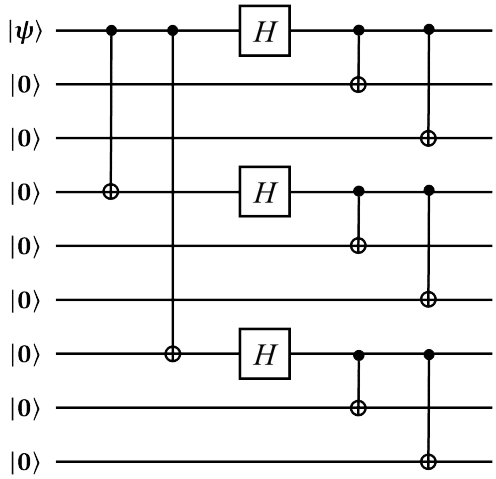


그림 1 Shor 코드를 이용해 9개의 물리적 qubit 에 주어진 양자 상태를 인코딩하는 양자 회로.

[그림 1]의 회로를 이용하여 인코딩된 양자 상태에 한 개의 qubit에 대한 오류가 발생할 경우, 신드롬 추출 회로를 이용해 해당 오류 정보를 얻을 수 있다. 신드롬 추출에는 추가로 4개의 ancilla qubit(또는 측정 qubit)이 필요하며, ancilla qubit은 물리적 qubit들과 적절한 *CNOT*을 통해 얽힘을 만든 뒤 측정을 통해 신드롬을 생성하는 역할을 한다.

일반적으로 n개의 물리적 qubit을 사용해 k개의 (논리적) qubit에 해당하는 정보를 저장하고 distance가 d인 양자 오류 정정 부호를 $[[n,k,d]]$ 코드라 부른다[31]. 이때 distance d는 해당 코드를 이용해 탐지할 수 없는 오류의 최소 크기를 의미하며, 해당 코드를 이용해 올바르게 수정할 수 있는 오류의 개수 t는 $t = \lfloor (d-1)/2 \rfloor$의 관계를 만족한다.

많은 수의 물리적 qubit을 이용해 양자 오류 정정 부호를 구현할수록 많은 수의 오류를 정정할 수 있으나, qubit은 사용할 수 있는 수가 한정된 자원이고 *CNOT*과 같은 2-qubit 게이트를 구현하는 데 있어서 두 qubit의 거리에 따라 구현 난이도가 달라지는 등의 문제가 있다. 따라서 효율적인 양자 오류 정정 부호를 통해 사용하는 물리적 qubit을 최소화하고 오류 threshold 및 transversal 한 논리적 게이트 수 최대화, 디코딩 문제의 효율적인 해결 알고리즘 등에 관한 연구가 진행되고 있다[14, 32-35].

Stabilizer 코드는 매우 간단한 인코딩 및 디코딩 회로를 갖고 있으므로 대표적으로 사용되는 양자 오류 정정 부호의 종류이다.

2.2.1. Stabilizer Formalism

양자 연산 A에 대해 $A|\psi\rangle=|\psi\rangle$를 만족할 경우, A를 $|\psi\rangle$의 stabilizer라 하고, $|\psi\rangle$ 가 A에 의해 stabilize 되었다고 한다[36, 37].

n-qubit 양자 상태에 대해 Pauli 그룹을 다음과 같이 정의할 수 있다.

$$P^{\otimes n} = \{\sigma_1 \otimes \sigma_2 \otimes \cdots \otimes \sigma_n : \sigma_i \in \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm iY, \pm Z, \pm iZ\}\}$$
$$\sigma_1\sigma_2 \cdots \sigma_n \equiv \sigma_1 \otimes \sigma_2 \otimes \cdots \otimes \sigma_n$$

이때 $P^{\otimes n}$의 subgroup $S(\subset P^{\otimes n})$의 모든 $S_i \in S$에 의해 stabilize 되는 양자 상태 $|\psi\rangle$ (혹은 $S_i|\psi\rangle=|\psi\rangle$ $\forall S_i \in S$)는 S에 의해 stabilize 되었다고 말하고, S를 stabilizer group이라 한다. Stabilizer의 정의 및 그룹의 정의에 따라, S가 0 state 이외의 stabilized vector를 갖기 위해서는 다음 두 조건을 만족해야 한다.

$$-I^{\otimes n}, \pm iI^{\otimes n} \notin S$$
$$S_i, S_j \in S \Rightarrow [S_i, S_j] = S_iS_j - S_jS_i = 0 \text{ (S의 모든 원소는 commute 한다)}$$

예를 들어, 5개 qubit에 대해 초기 stabilizer 그룹이 $S_1 = \{IIIII, XZZXI\}$ $(\subset P^{\otimes 5})$와 같이 설정되었다고 하자. S_1에 의해 stabilize 되는 양자 상태의 basis는 다음과 같다.

$$\begin{aligned} T_1 = \{&|+00+0\rangle, |+11+0\rangle, |-00-0\rangle, |-11-0\rangle, \\ &|+01-0\rangle, |+10-0\rangle, |-01+0\rangle, |-10+0\rangle, \\ &|+00+1\rangle, |+11+1\rangle, |-00-1\rangle, |-11-1\rangle, \\ &|+01-1\rangle, |+10-1\rangle, |-01+1\rangle, |-10+1\rangle\} \end{aligned} \qquad |T_1| = 16$$

이때 stabilize 그룹 S_1에 새로운 stabilizer $IXZZX$ 를 추가할 경우 새로운 stabilizer 그룹 S_2와 stabilize 되는 양자 상태 T_2는 다음과 같다.

$$S_2 = \{IIIII, XZZXI, IXZZX, XYIYX\} = \langle XZZXI, IXZZX\rangle$$

$$
\begin{aligned}
T_2 = \{&|00000\rangle + |10010\rangle + |01001\rangle - |11011\rangle, && |11111\rangle + |01101\rangle + |10110\rangle - |00100\rangle, \\
&|00001\rangle + |10011\rangle + |01000\rangle - |11010\rangle, && |00010\rangle + |10000\rangle + |11001\rangle - |01011\rangle, \\
&|00011\rangle + |10001\rangle + |11000\rangle - |01010\rangle, && |00101\rangle + |11110\rangle - |10111\rangle - |01100\rangle, \\
&|00110\rangle + |11101\rangle - |10100\rangle - |01111\rangle, && |01110\rangle + |11100\rangle + |00111\rangle - |10101\rangle\}
\end{aligned}
\qquad |T_2| = 8
$$

즉 stabilizer generator를 하나씩 추가하고 group을 확장할수록 stabilized 되는 basis 개수는 절반씩 줄어든다. 일반적으로 n개의 (물리적) qubit에 대해 $n-k$개의 stabilizer generator를 이용해 양자 오류 정정 부호를 만들면 k개의 논리적 qubit (혹은 2^k개의 basis)를 사용할 수 있다[19].

2.2.2. Steane 코드와 Color 코드

Steane 코드는 대표적인 stabilizer 코드로, 다음과 같은 stabilizer group generator와 논리적 $|0\rangle_L$, $|1\rangle_L$을 갖는다[38].

$$
\begin{aligned}
S &= \langle IIIXXXX, IXXIIXX, XIXIXIX, IIIZZZZ, IZZIIZZ, ZIZIZIZ\rangle \\
|0\rangle_L &= |0000000\rangle + |1010101\rangle + |0110011\rangle + |1100110\rangle + |0001111\rangle + |1011010\rangle + |0111100\rangle + |1101001\rangle \\
|1\rangle_L &= |1111111\rangle + |0101010\rangle + |1001100\rangle + |0011001\rangle + |1110000\rangle + |0100101\rangle + |1000011\rangle + |0010110\rangle
\end{aligned}
$$

또한 [그림 2]는 $|0\rangle$으로 초기화된 7개의 물리적 qubit으로부터 논리적 $|0\rangle_L$을 만들어내는 회로이다.

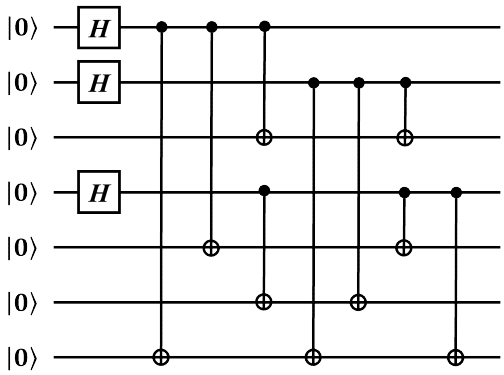


그림 2 Steane 코드를 이용해 7개의 qubit에 논리적 $|0\rangle_L$ 상태를 만드는 양자 회로.

이때 논리적 $|0\rangle_L$, $|1\rangle_L$ 상태에 논리적 게이트를 가하고자 할 경우,

Steane 코드에 맞는 방식으로 물리적 qubit들에 게이트를 가해야 한다. 논리적 Pauli 게이트, Hadamard 게이트 및 *CNOT* 게이트의 구현은 다음과 같다.

$$X_L = X_1X_2X_3X_4X_5X_6X_7, \quad Z_L = Z_1Z_2Z_3Z_4Z_5Z_6Z_7, \quad H_L = H_1H_2H_3H_4H_5H_6H_7$$
$$CNOT_L = CNOT_1CNOT_2CNOT_3CNOT_4CNOT_5CNOT_6CNOT_7$$

즉 많은 게이트에 대해, 물리적 qubit 각각에 같은 종류의 게이트를 가하는 것만으로도 논리적 게이트를 구현할 수 있는데, 이를 양자 오류 정정 부호의 transversality라 한다. 논리적 게이트를 가하는 과정에 *CNOT* 게이트가 필요한 경우, 하나의 물리적 qubit에서 발생한 오류가 다른 물리적 qubit으로 전파될 수 있다. 이 경우 양자 오류 정정 부호의 크기를 늘리더라도 양자 연산 과정에서 많은 오류가 발생해 결과적으로 양자 회로의 신뢰도를 떨어뜨릴 수 있다. 따라서 많은 종류의 transversal 게이트를 가지는 것이 좋으며, Steane 코드는 앞서 언급한 Hadamard, *CNOT* 게이트 이외에 Phase 게이트 또한 transversal 한 방식으로 논리적 게이트를 구현한다[39].

Eastin-Knill 정리에 따르면 모든 논리적 게이트를 transversal 한 방식으로 구현할 수 있는 양자 오류 정정 부호는 존재할 수 없으며[40], universal 게이트 집합인 $\{H, S, T, CNOT\}$ 중 T 게이트는 Steane 코드를 이용해 transversal 하게 구현할 수 없다. T 게이트를 구현하기 위해서는 magic state distillation, 3D color 코드 등의 방식을 사용해야 한다[41-46].

Color 코드는 기하학적 형태를 따라 만든 stabilizer 코드로, 세 가지 색깔이 주어졌을 때 인접한 두 면의 색이 같지 않도록 색칠할 수 있는 격자를 정의한 뒤 물리적 qubit을 배치한다[12]. 가장 작은 형태의 color 코드 예시는 [그림 3]과 같다.

초록, 빨강, 파랑으로 색칠된 각 면은 인접한 면과 다른 색을 가지며, 물리적 qubit들은 각 모서리에 위치한다. 이때 stabilizer는 각 면의 꼭짓점들에 대하여 X 타입, Z 타입을 정의한다. 예를 들어 초록 면에서

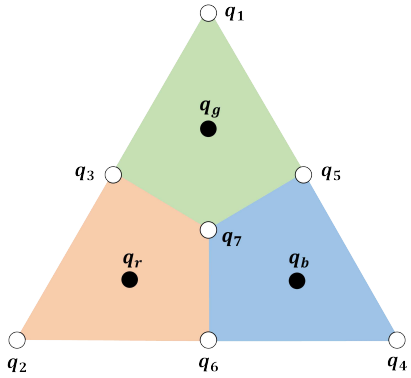


그림 3 제일 작은 color 코드의 물리적 qubit(흰색)들과, color 코드에 따라 정의된 stabilizer generator로 측정하는 ancilla qubit(검은색) 간 관계.

q_1, q_3, q_5, q_7에 대해 모두 X basis로 측정($X_1 I_2 X_3 I_4 X_5 I_6 X_7$)하거나 Z basis로 측정($Z_1 I_2 Z_3 I_4 Z_5 I_6 Z_7$)하는 두 stabilizer를 정의할 수 있다. [그림 3]의 예시에서 정의된 전체 stabilizer generator는 다음과 같다.

$$S = \langle IIIXXXX, IXXIIXX, XIXIXIX, IIIZZZZ, IZZIIZZ, ZIZIZIZ \rangle$$

즉, 각 면에 대해 ancilla qubit(q_g, q_r, q_b)을 두고 인접한 물리적 qubit들과 $CNOT$을 통해 얽힘을 형성한 뒤 ancilla qubit을 측정함으로써 stabilizer 측정을 할 수 있다.

물리적 qubit의 indexing에 따라 달라질 수 있으나, 위 예시에서 얻은 stabilizer 그룹은 앞서 언급한 Steane 코드와 같음을 확인할 수 있으며, Steane 코드는 2D color 코드의 가장 작은 형태라 할 수 있다. 일반적으로 color 코드는 크기와 상관없이 transversality 성질이 유지되기 때문에, 더 많은 수의 qubit을 이용해 만든 color 코드에 대해서도 Pauli 게이트, Hadamard 게이트, phase 게이트, $CNOT$ 게이트는 transversal 하게 구현할 수 있다[39].

많은 transversal 게이트를 가지는 것 외에도, color 코드는 stabilizer가 local 하게 정의되어있기 때문에 stabilizer 측정 과정에서 오류가 발생하더라도 인코딩된 상태 전체로 오류가 쉽게 전파되지 않고 실제 하드웨어 상에서의 구현이 쉽다는 장점이 있다.

2.3. 이온 트랩 기반 양자컴퓨터

양자컴퓨터 구현을 위해서는, 이론적으로 존재하는 양자 알고리즘을 효율적이고 정확하게 구현할 수 있는 물리 시스템이 필요하다. DiVincenzo 조건은 양자컴퓨터를 위한 물리 시스템이 만족해야 할 조건들을 제시한 것으로, 그 조건들은 다음과 같다[47].

a. 잘 정의된 qubit을 가진 확장 가능한 물리 시스템
b. $|00\cdots0\rangle$과 같은 간단한 상태로 쉽게 초기화 가능한 물리 시스템
c. 게이트 연산 시간보다 충분히 긴 결맞음 시간을 갖는 물리 시스템
d. universal quantum 게이트 구현이 가능한 물리 시스템
e. qubit별로 상태 측정이 가능한 물리 시스템

정확도와 구현 난이도에 차이는 있으나, 위 조건을 만족하는 여러 기술이 연구되고 있으며 대표적으로 이온 트랩이 있다.

이온 트랩은 전기장을 이용해 qubit에 해당하는 대전된 전하(혹은 이온)를 공간상에 포획한 뒤, 외부 전자기장을 이용해 전자의 에너지 준위를 변화시켜서 qubit 상태를 변화시킨다[48]. 전극을 이용해 만들어내는 전기장 혹은 전기 퍼텐셜 모양을 변형시켜 여러 개의 이온을 일렬로 포획하고 그중 일부 이온을 이동시키는 것이 가능한데, 일렬로 포획된 이온을 이온 체인(ion chain)이라 하고, 이온을 이동시키는 것을 이온 셔틀링(shuttling)이라 한다[49, 50]. 이온 트랩 기반의 양자컴퓨터는 결맞음 시간이 길고 상온에서도 구현할 수 있으며 게이트 연산 및 qubit 상태 측정 신뢰도가 높다는 장점이 있다. 또한, 같은 이온 체인에 포획된 이온 간에는 떨어진 거리와 상관없이 2-qubit 게이트 연산을 할 수 있으므로 qubit 간 full-connectivity를 가지는 장점이 있다[51].

그러나 실제 구현 시 하나의 이온 체인에 무한히 많은 이온을 포획하는 것은 불가능하므로 수십 개 정도의 이온을 포획하는데, 실제 양자 알고리즘 및 양자 오류 정정 부호의 사용에서 요구되는 만큼의 이온을 사용하기 위해 여러 개의 이온 체인을 만들고, 이온 체인 간 이온 이동이 필요하다. 이 때문에 이온 트랩 기반의 양자컴퓨터는 확장성이 떨어질 수 있으나, 전기장을 형성하는 전극 및 이온이 포획되는 트랩을 평면 칩 위

에 만듦으로써 시스템 크기를 소형화하고 여러 이온 체인을 연결할 수 있도록 하고 있다[52-55]. [그림 4]와 [그림 5]는 각각 이온 트랩 칩의 구조 및 실제 사진이다.

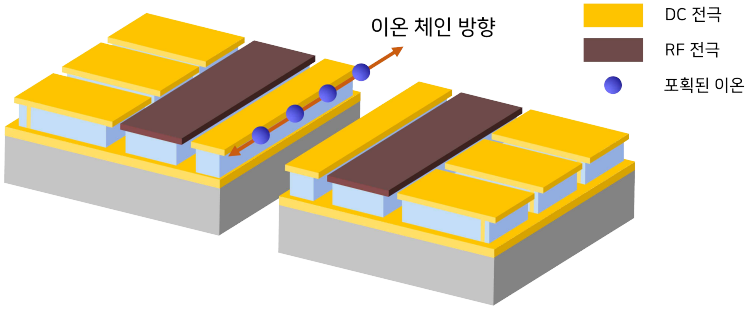


그림 4 칩을 이용한 이온 트랩에서 이온이 포획되는 부분 및 주변 전극들을 도식화한 것. DC 전극과 RF 전극들에 필요한 전압을 가함으로써 원하는 형태의 전기장을 형성하고 이를 이용해 이온을 포획한다. 전극 개수에 따라 더 많은 이온을 포획할 수 있다.

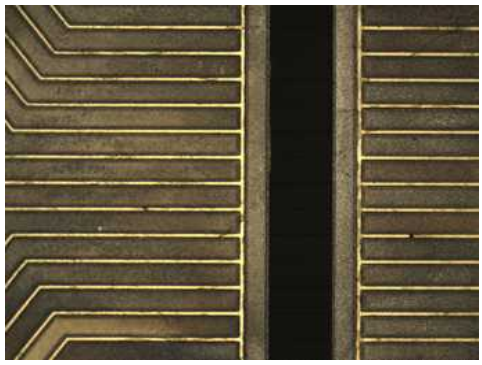


그림 5 이온 트랩 칩 사진.

다만 레이저 방향, 칩 레이어에 사용하는 물질의 종류 등에 따라 구체적인 형태는 달라질 수 있으며 특히 여러 개의 이온 체인을 포획할 수 있는 칩을 설계할 경우, 이온 체인 간 간격, 방향, 연결방법 등 추가로 고려해야 할 요소가 많다. 가로 및 세로 방향으로 배열된 이온 트랩에서 이온 shuttling을 이용해 양자 연산을 진행하는 QCCD(Quantum Charge-Coupled Device)에 대한 아이디어가 제안되었으며[56], 이후 이온 트랩 칩 구조 설계를 최적화하기 위한 연구가 진행되고 있다[57-63].

3. 효율적인 양자 오류 정정을 위한 이온 트랩 칩 구조

참고문헌 [57] 에 따르면 동일한 칩 구조에 대해, 구현되는 양자 알고리즘의 형태에 따라 효율성에 차이가 발생한다. 즉 구현하고자 하는 양자 알고리즘이 가진 2-qubit 게이트 개수와 2-qubit 게이트에 해당하는 두 qubit 간 거리에 따라 주어진 칩 디자인의 효율성이 달라진다. 따라서 이온 트랩 칩을 디자인하는 데 있어서 구현하고자 하는 양자 알고리즘의 형태를 고려해야 하는데, 양자 오류 정정 부호 적용에 따른 양자 회로 구조는 상위 레벨의 양자 알고리즘과 상관없이 하드웨어 제작 단계에서 고려할 수 있다.

Steane 코드 혹은 color 코드를 적용하여 양자 알고리즘을 구현할 경우 알고리즘 형태에 상관없이 많은 transversal 게이트를 가해야 하므로, 많은 게이트가 일정 개수의 물리적 qubit에 동시에 동일하게 적용된다. 칩 구조 설계에 이러한 규칙을 반영하면 하드웨어 수준에서의 최적화가 가능하며, 본 논문에서는 transversal 게이트 연산을 효율적으로 구현하기 위한 이온 트랩 칩 구조를 제안하고자 한다.

7-qubit color 코드 혹은 Steane 코드에 대해 제안한 이온 트랩 칩 구조는 [그림 6]과 같다. 제안한 칩 구조에서 수평 방향의 이온 트랩 구역 한 줄을 트랙(track)으로, 수직 방향 이온 트랩 구역 사이 7개의 병렬적인 이온 트랩 구역들을 섹터(sector)라 부른다([그림 6(a)]). 각 트랙의 같은 위치에 놓인 이온들이 하나의 논리적 qubit을 나타내며, transversal 한 논리적 게이트를 가하고자 할 경우, 해당하는 물리적 qubit들에 동시에 같은 레이저를 조사하면 된다([그림 6(b)]). 이온의 이동이 필요한 경우, 전극에 가해진 전압값을 수정함으로써 수평 방향으로 이온을 shuttling 할 수 있다([그림 6(c)]). 이온이 shuttling 하는 도중 수직 방향의 이온 트랩 구역을 지날 경우, 해당하는 논리적 qubit은 신드롬 추출을 통한 오류 정정을 할 수 있다. 또한, 구체적인 구현 방법에 따라 차이가 있을 수 있으나, 칩 가장 왼쪽에서 이온들을 포획 및 $|0\rangle$ 상태로 초기화하고 shuttling을 통해 중간 섹터에서 게이트 연산을 한 뒤, 마지막 측정 시

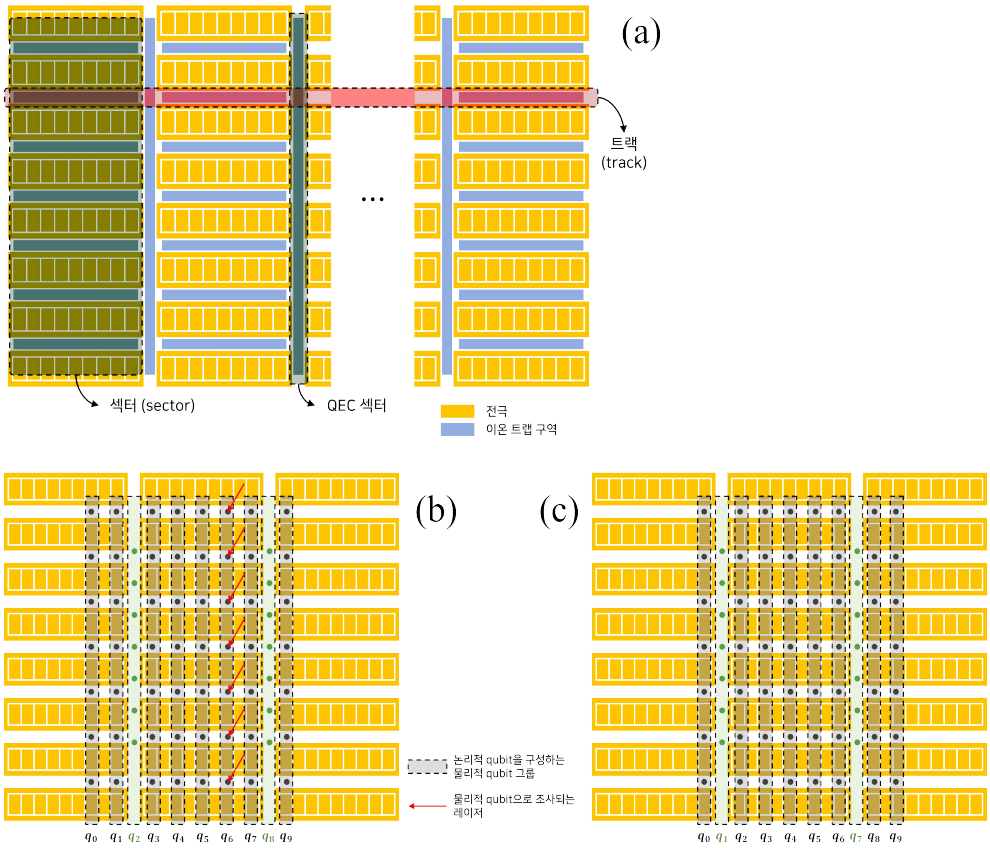


그림 6 **(a)** 격자 형태의 전극 배치를 통해 가로 방향 및 세로 방향 이온 체인을 형성할 수 있는 칩 구조. 가로 방향 이온 체인들 한 줄을 트랙(track), 같은 세로 줄에 있는 가로 방향 이온 체인 그룹을 섹터(sector)라 부르며, 섹터 간 세로 방향 이온 체인에서는 양자 오류 정정 부호에 따른 신드롬 추출이 이루어지므로 QEC 섹터라 부른다. 7개 qubit을 사용하는 Steane 코드를 고려하여 7개의 트랙이 존재한다. **(b)** 한 섹터 내의 트랙들은 같은 수의 이온을 포획하고 있으며, 세로 방향으로 정렬되어있다. 세로 방향의 7개 이온들은 Steane 코드로 인코딩된 하나의 논리적 qubit들을 의미하며, 해당 qubit에 transversal 게이트를 가하면 모든 물리적 qubit에 동일한 레이저를 조사함으로써 동일한 게이트를 가하면 된다(빨강 화살표). QEC 섹터의 7개 물리적 qubit들은 같은 체인에 존재하므로 2-qubit 게이트를 할 수 있으며, 신드롬 추출을 통해 오류를 정정한다. **(c)** (b)에서 오른쪽으로 이온 shuttling 후 이온의 배치. Shuttling 방향 제일 앞 섹터를 제외한 나머지 섹터에서는 오른쪽 이온들이 다음 섹터로 넘어간다.

이온들을 가장 오른쪽으로 shuttling 하여 측정용 레이저를 이용해 최종 상태를 얻는다고 가정한다.

제안한 칩 구조의 장점은 양자 오류 정정 부호 사용에 따른 논리적 게이트의 transversality를 고려해 전극을 배치하고 이온들을 shuttling 하므로, 임의의 전극 구조의 칩을 사용했을 때에 비해 불필요한 이온의 움직임을 없앨 수 있다. 구체적으로, transversal 게이트를 가하는 데 있어서 다른 트랙에 있는 물리적 qubit들 간에는 2-qubit 게이트가 필요 없기 때문에, 같은 이온 체인 내에 존재하지 않아도 된다. 즉 대부분 같은 체인에 있을 필요가 없는 qubit 간 구역(트랙)을 분리했기 때문에 이온이 수직 방향으로 움직이거나 트랙 간 이온 이동이 이루어질 필요가 없다. 신드롬 추출을 위해 다른 트랙의 qubit 간 2-qubit 게이트가 필요한 경우도, 실제로는 하나의 논리적 qubit을 구성하는 물리적 qubit 간에만 2-qubit 게이트가 필요하다. 따라서 모든 트랙의 qubit들이 평행하게 움직이는 상황에서 QEC 섹터에 도달했을 때, 2-qubit 게이트가 필요한 qubit끼리 수직 방향 이온 포획 구역에서 자연스럽게 이온 체인을 형성한다.

트랙별 병렬적인 이온의 이동은, 이온의 움직임을 최소화하는 것 외에 전기장 퍼텐셜 조절을 위해 가해야 하는 전극별 전압의 개수를 줄일 수

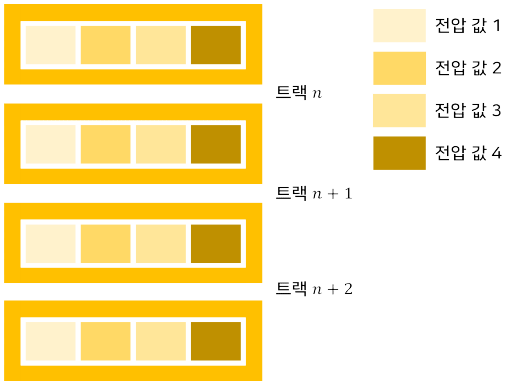


그림 7 수평 방향 전극들의 전압값을 도식화한 그림. 각 트랙 내에서 같은 위치에 존재하는 이온들은 같은 전기장 퍼텐셜을 통해 shuttling 하므로, 평행한 전극들이 서로 같은 전압값을 가진다. 전극 세그먼트에 칠해진 색이 같으면 같은 전압값을 갖는다.

있다는 장점이 있다. 전기장 퍼텐셜 모양을 바꾸기 위해서는 트랙 주변의 DC 전극에 가해지는 전압값을 바꾸어야 하는데, 트랙별 전기장 퍼텐셜 모양이 모두 다르다면 전극에 전압값 또한 트랙별로 상이해야 한다. 많은 DC 전극을 모두 독립적으로 제어한다면 전극과 외부를 연결하는 와이어 개수, 각 전극과 연결된 DAC(Digital to Analog Converter) 채널 개수가 증가하는 문제가 생기는데 제안한 칩 구조상에서는 트랙 개수와 상관없이 트랙 길이에 비례하는 만큼의 전극만 제어하면 된다([그림 7]). 이를 통해 이온 트랩 시스템 제어를 위한 파라미터 개수 감소, 칩의 집적도 증가를 기대할 수 있다.

또한, 주기적인 이온 shuttling 과정에서 qubit들이 주기적으로 QEC 섹터를 지나기 때문에 자연스럽게 모든 논리적 qubit들이 비교적 균등한 신드롬 추출 및 오류 정정 기회를 가지며, 다른 섹터의 논리적 qubit 간 *CNOT* 연산이 필요한 경우, 주기적인 shuttling에 맞는 적절한 SWAP 게이트를 통해 비교적 빠르게 섹터 간 qubit 이동이 가능하다.

제안한 칩 상에서의 1-qubit 게이트, 2-qubit 게이트 연산에 따른 이온의 움직임 및 에러 분석과 전체 알고리즘 구현에 있어서 latency를 조사하기 위해 칩 구조를 모델링하고 벤치마크 양자 회로에 대해 시뮬레이션하였다.

4. 이온 트랩 칩 시뮬레이션

4.1. 칩 시뮬레이터 구조

칩 시뮬레이션의 전체적인 흐름은 다음과 같다([그림 8]).

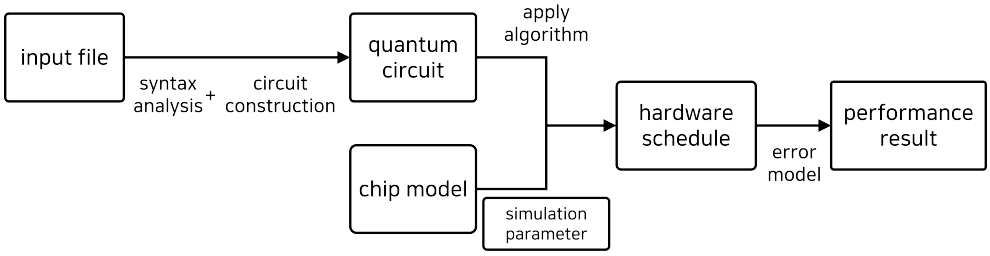


그림 8 칩 시뮬레이션 진행 흐름도.

시뮬레이션은 크게 1) 양자 회로 생성 및 칩 구조 모델링, 2) 해당 구조 상에서 스케줄링 알고리즘을 통한 양자 회로 구현, 3) 구현된 양자 회로에 대한 에러 모델 적용 후 전체 오류율 및 latency를 얻는 순으로 진행된다. 시뮬레이터는 Python 3.8.8을 이용해 구현하였으며, 시뮬레이션 단계별 구체사항은 다음과 같다.

4.1.1. 양자 회로 생성 및 칩 구조 모델링

시뮬레이터에 주어지는 입력 파일은 구현하고자 하는 양자 회로를 기술하고 있으며, 첫 번째 줄에 총 qubit 개수를, 나머지 줄에 양자 회로를 구현하는 게이트 연산에 대한 명령어들을 가진다. 입력 파일 예시와 생성되는 양자 회로는 다음과 같다([그림 9]).

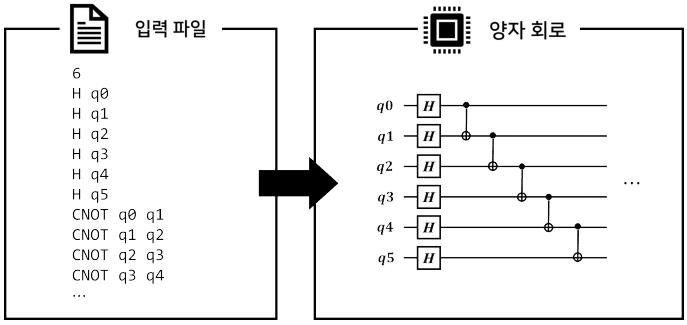


그림 9 입력 파일 형태 및 명령어에 따라 생성되는 양자 회로 예시.

입력 파일로부터 양자 회로를 생성하는 과정에서 잘못된 형식이나 지원하지 않는 게이트 연산 등을 확인하기 위해 구문 분석(syntax analysis)을 한다. 문법이 간단하고 명령어의 종류가 많지 않기 때문에 현재는 비교적 간단한 구문 분석을 하고 있으나, 큰 양자 회로를 기술하거나 향후 시뮬레이션 기능을 확장하는 데 있어서 입력 파일에 대한 체계적인 검사가 필요하다. 따라서 파이썬의 내장 예외 클래스를 상속받아 문법 오류별 예외 클래스를 정의하고 입력 파일 구문 분석 과정에서 사용하였다. 본 시뮬레이션에서 정의한 예외 클래스는 [표 1]과 같다.

표 2 입력 파일 구문 분석을 위해 정의한 오류별 예외 처리 클래스.

예외 클래스	설명
GateTypeError	정의된 게이트(예: H, S, CNOT 등) 이외의 게이트를 사용한 경우
InvalidSymbolError	논리적 qubit을 표현하는데 있어, 'q' 혹은 'Q'로 시작하지 않는 경우
QubitIndexOutOfRangeError	논리적 qubit을 표현하는데 있어, qubit의 인덱스가 입력 파일 첫 줄에 지정된 전체 qubit 개수와 맞지 않는 경우 (음의 인덱스 혹은 전체 qubit 수보다 크거나 같은 인덱스)

오류가 없는 입력 파일에 대해서는 기술하는 양자 회로를 구성하게 되는데, 입력 파일의 각 명령어에 해당하는 양자 게이트 클래스를 생성한 뒤, 게이트가 가해지는 논리적 qubit의 자료구조에 생성된 양자 게이트 클래스를 추가하는 방식으로 양자 회로를 구성한다. 양자 회로 모델의 관리와 효율적인 시뮬레이션을 위해 몇 가지 클래스를 정의하고 있으며, 정의된 클래스 및 이를 이용한 양자 회로 구성 과정은 다음과 같다([그림 10, 11]).

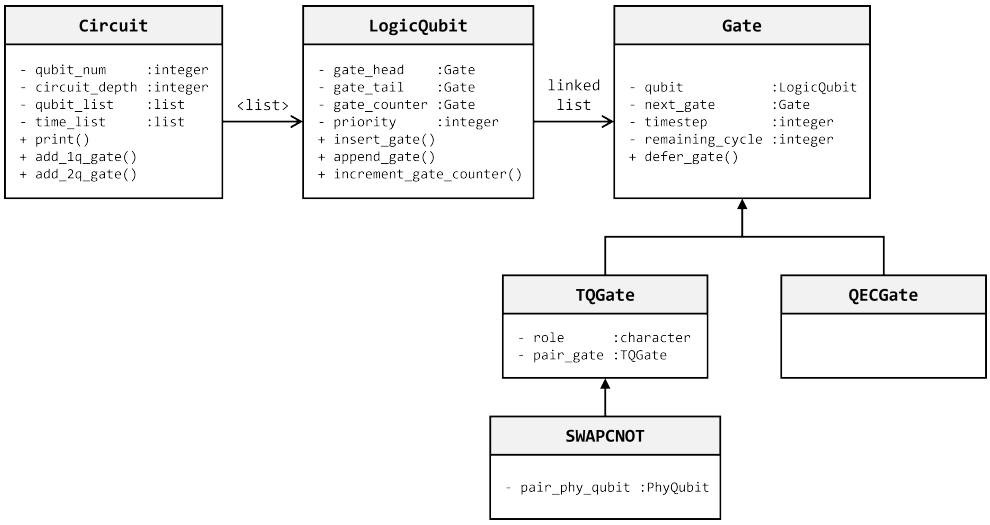


그림 10 양자 회로 모델링을 위해 정의한 클래스에 대한 클래스 다이어그램.

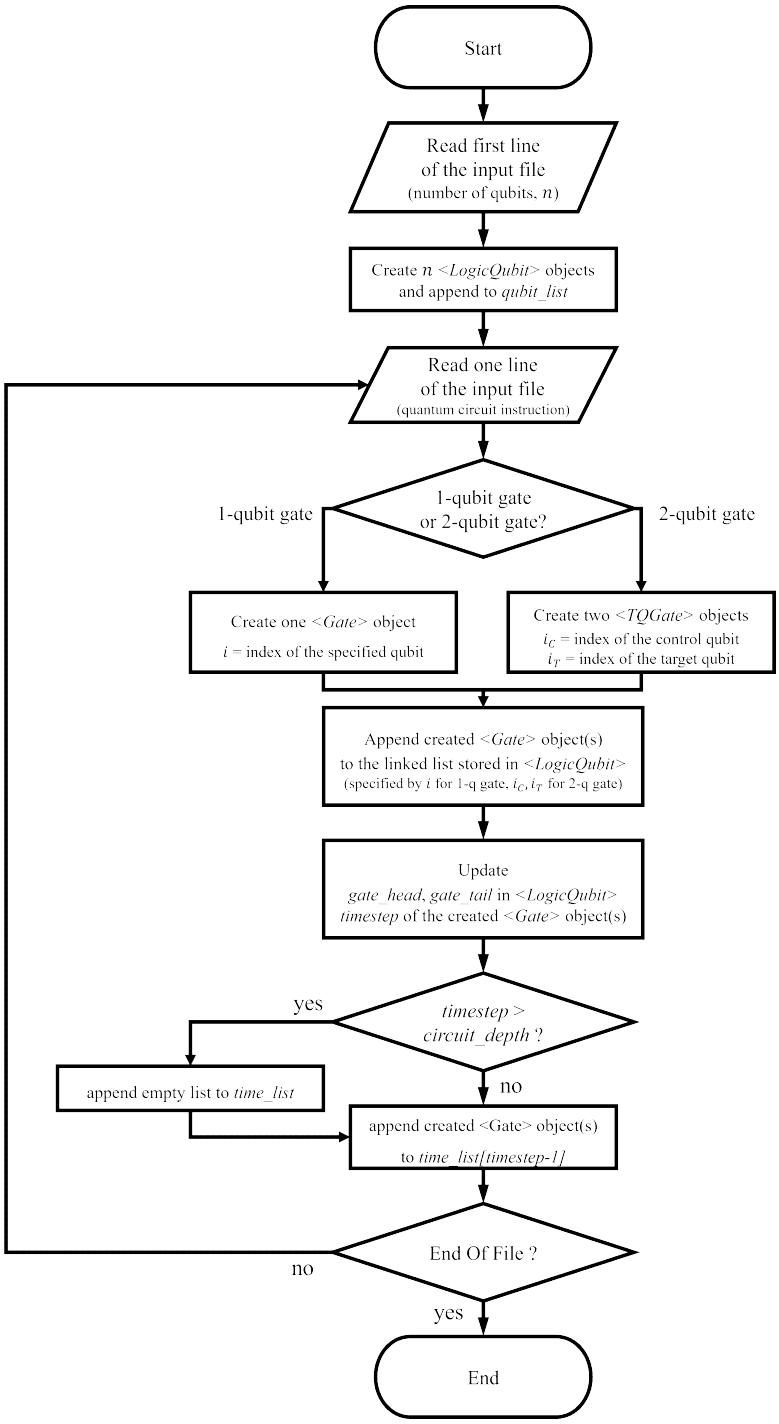


그림 11 입력 파일로부터 양자 회로를 생성하는 과정에 대한 알고리즘 순서도.

<Circuit> 클래스는 입력 파일 첫 줄에 나온 수 만큼의 *<LogicQubit>* 클래스 오브젝트를 갖고 있으며, *<LogicQubit>* 클래스 오브젝트는 해당 논리적 qubit에 가해지는 게이트를 의미하는 *<Gate>* 클래스 오브젝트들을 linked list로 관리한다. 즉 입력 파일을 읽으며 *<Gate>* 클래스 오브젝트를 생성하고, qubit 인덱스를 이용하여 *<LogicQubit>*에 관리하는 linked list에 추가한다. 2-qubit 게이트는 두 개의 *<TQGate>* 클래스 오브젝트를 생성하고, 각 *<LogicQubit>*의 linked list에 추가한다. 두 개의 *<TQGate>* 클래스는 서로에 대한 포인터를 갖고 있어서 향후 스케줄링에서 사용할 수 있다. 이러한 과정을 모두 수행하면 입력 파일에 해당하는 양자 회로에 대한 *<Circuit>* 클래스 오브젝트 하나가 생성된다.

한편 생성된 양자 회로를 제안한 칩 구조상에서 시뮬레이션하기 위해서는 칩을 모델링하는 자료구조 또한 필요하다. 제안한 칩 구조에서 하나의 섹터 당 7개의 트랙이 있으나, 각 트랙에 있는 물리적 qubit들은 개수, 위치, 이동 방향, 가하는 게이트 등에 있어서 완벽히 동일하다. 따라서 시뮬레이터는 하나의 트랙만 모델링하며, 에러 모델 적용 단계에서만 트랙 개수에 따른 실제 물리적 qubit을 고려한다. 칩 모델링을 위해 정의한 클래스 및 칩 내부의 qubit을 모델링하는 주요 함수는 다음과 같다([그림 12, 13]).

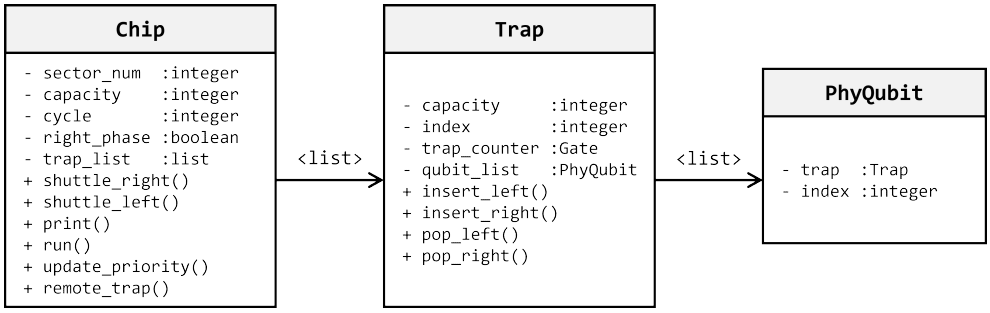


그림 12 칩 구조 모델링을 위해 정의한 클래스에 대한 클래스 다이어그램.

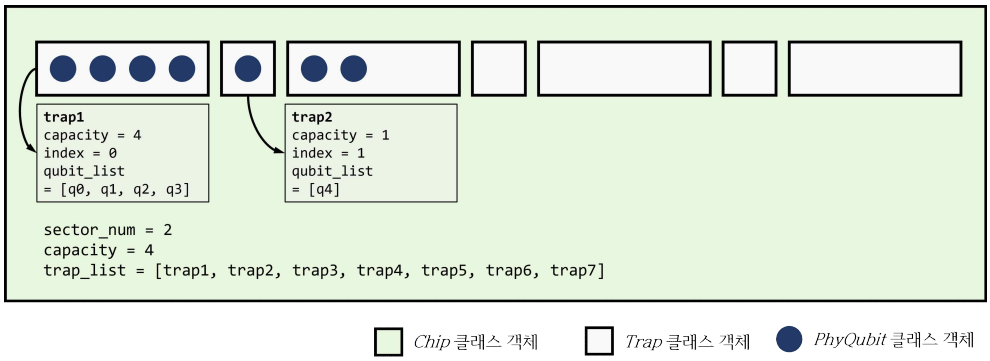


그림 13 *<Chip>* 클래스 오브젝트 생성, 초기화에 따라 모델링한 칩 구조 도식화.

시뮬레이션에서는 [그림 13]에서와 같이 *<Chip>* 클래스 오브젝트를 하나 생성한 뒤, 앞서 생성한 *<Circuit>* 클래스 오브젝트를 *<Chip>* 클래스에 정의된 *run()* 함수에 인자로 넘겨줌으로써 칩 위에서 양자 회로 구현을 위한 스케줄링을 시작한다.

4.1.2. 칩 시뮬레이션 전략 및 알고리즘

<Chip> 클래스에 정의된 *run()* 함수는 *<Circuit>* 클래스 오브젝트를 인자로 받은 뒤, 해당 양자 회로를 칩에서 구현할 수 있도록 스케줄링하고 그 결과를 반환한다. 구체적인 스케줄링 알고리즘은 바뀔 수 있으나, 이온 트랩의 물리적 특성을 고려해 공통적으로 적용해야 할 가정들이 존재한다.

a. 섹터별 포획 가능한 이온 개수가 정해지면 필요한 논리적 qubit 개수와 상관없이 항상 고정된 개수의 물리적 qubit을 포획한다. 최대 이온 체인의 개수가 N개일 경우, N보다 적은 물리적 qubit을 포획하는 것 또한 가능하다. 그러나 포획된 이온들에 레이저 조사를 통한 게이트 연산을 하고자 할 때 레이저가 정확히 이온을 지나야 하며, 인접한 이온들에 영향을 주는 크로스톡(crosstalk)을 최소화하기 위해 실험 단계에서의 최적화가 필요하다. 임의 개수의 이온이 포획된 이온 체인을 사용할 경우 이온 개수에 따른 성능 차이가 발생하기 때문에 실제 이온 트랩 시스템 제작 시 고정된 수의 이온을 포획할 가능성이 크며, 본 시뮬레이션에서도 하나의 섹터별로 포획된 이온의 개수는 시뮬레이션 과정에서 변하지

않는다고 가정한다. 포획된 이온 개수보다 적은 qubit을 이용하는 양자 연산이 필요한 경우, 이온의 왼쪽부터 차례대로 논리적 qubit을 할당하며, 나머지 이온들은 포획된 뒤 사용하지 않는다고 가정한다.

b. 고전컴퓨터의 CPU가 clock cycle 단위로 명령어를 처리하는 것과 유사하게, 본 시뮬레이터가 시뮬레이션하는 이온 트랩 시스템도 이산화된 시간 단위로 양자 게이트를 실행하거나 이온을 shuttling 한다. 일반적으로 이온 트랩에서 게이트 연산을 하거나 이온을 움직이는데 수십에서 수백 마이크로초 정도의 시간이 걸리는데, 구체적인 수치는 사용한 물리 기술과 구현 환경에 따라 달라진다. 아직 표준화된 기술이 없으므로 대략적인 스케일을 고려하여 time unit 단위로 게이트 연산을 진행했으며, [표 2]는 게이트 별로 필요한 time unit 정보이다.

표 3 게이트별 소요되는 time unit

게이트	소요되는 time unit
1-qubit 게이트	1
2-qubit 게이트	5
이온 shuttling	1

c. 주기적인 신드롬 추출과 효율적인 SWAP 연산을 위해 포획된 이온들을 주기적으로 shuttling 한다. [그림 6(c)]에서와 같이 오른쪽으로 이온을 shuttling 하면 섹터 가장 오른쪽 이온은 다음 섹터로 넘어가며, 다른 이온들은 섹터 내에서의 위치만 옮겨가게 된다. 옮겨간 섹터가 QEC 섹터일 경우, 해당 논리적 qubit은 다음 게이트 연산이 아닌 신드롬 추출을 통한 오류 정정 과정을 거치며, 다음 shuttling 때 인접 섹터로 넘어가 게이트 연산을 계속한다. 가장 오른쪽 3개 섹터(1개 QEC 섹터 포함)를 비워두고 나머지 섹터를 모두 채우는 식으로 이온을 포획했을 때, 주기적으로 오른쪽으로 shuttling 한 뒤 가장 오른쪽 끝에 도달했을 때 방향을 바꾸어 주기적으로 왼쪽으로 shuttling을 이어간다. 위 과정을 반복함으로써 모든 이온이 비교적 공평하게 신드롬 추출 기회를 얻을 수 있으며, 향후 SWAP 연산 시 주기적 shuttling을 이용해 논리적 qubit을 다른 섹터로 빠르게 이동시킬 수 있다.

구현 방식에 따라 차이가 있으나, [표 2]에 따라 이온 shuttling에 걸리는 시간은 1 time unit으로 가정했다. Shuttling 이후 QEC 섹터로 들어간 논리적 qubit은 신드롬 추출이 완료되기 전까지 다음 섹터로 이동할 수 없으므로, shuttling의 최소 주기는 신드롬 추출에 필요한 time unit이다. Steane 코드의 신드롬 추출에 5개 2-qubit 게이트 연산이 필요하므로, 본 시뮬레이션에서는 shuttling 주기를 2-qubit 게이트에 필요한 time unit의 5배인 25로 설정했다.

d. 한 time unit 동안, 하나의 섹터에서는 한 개의 논리적 게이트 연산만 가능하다. 하나의 이온 체인에 포획된 수십 개 이온에 대해 동시에 게이트 연산을 가하는 것도 가능하나, 각 이온에 조사되는 레이저 세기가 약해짐에 따라 게이트 시간이 오래 걸리고 2-qubit 게이트의 경우 두 개 이상의 게이트를 동시에 진행했을 때 신뢰도가 급격히 떨어진다. Acoustic-Optic Deflectors(AOD) 등을 이용하면 레이저를 원하는 이온 위치로 조사하여 빠르게 하나의 게이트 연산을 끝낼 수 있으며, 본 시뮬레이터에서도 이를 가정하여 한 time unit 당 한 섹터에서는 한 개의 1-qubit 게이트 또는 2-qubit 게이트만 가능하다고 가정한다.

위 가정들에 따른 전반적인 스케줄링 과정을 도식화하면 [그림 14]와 같다.

위의 가정에 따라 제안한 칩 구조에서 주어진 양자 회로를 실행하기 위해 time unit마다 각 섹터별로 실행해야 할 게이트를 선택해야 한다. 섹터별로 실행하는 게이트 순서에 따라 전체 양자 회로 구현에 필요한 시간과 SWAP에 따른 qubit 이동 양상이 바뀌며, 따라서 다음 게이트를 선택하기 위한 알고리즘을 효율적으로 구성할 필요가 있다.

고전컴퓨터에서 프로그램 카운터(Program Counter, PC)를 이용해 다음 실행 명령어에 대한 주소를 저장하는 것과 비슷하게, 본 시뮬레이터에서는 논리적 qubit별로 다음 실행해야 할 게이트를 가리키는 게이트 카운터(Gate Counter) 개념을 도입하여 스케줄링에 활용하였다. 시뮬레이터는 time unit마다 각 논리적 qubit에 대해 우선순위(priority)를 계산한 뒤, 섹터별로 가장 높은 우선순위를 가진 qubit을 뽑아 해당 qubit의 게

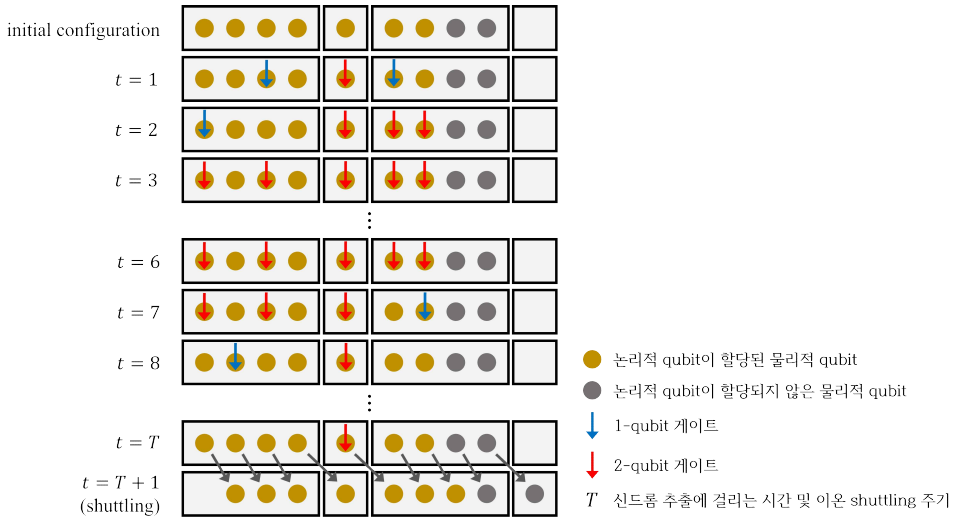


그림 14 네 가지 조건을 반영한 스케줄링 결과의 한 예시. 두 개의 섹터와 하나의 QEC 섹터에 63(=9×7)개의 이온이 포획되어있고, 7개의 논리적 qubit만 필요해서 49(=7×7)개의 물리적 qubit에만 논리적 qubit이 할당된 상황을 가정하였다. 1-qubit 게이트에 1 time unit, 2-qubit 게이트에 5 time unit이 필요하고 shuttling 주기는 5개의 2-qubit 게이트 필요한 25 time unit이라 가정하였다.

이트 카운터가 가리키는 게이트를 실행한다. 구체적인 게이트 선택 알고리즘은 [Algorithm 1]과 같다.

대부분의 경우에 qubit의 우선순위를 계산하고, 가장 우선순위가 높은 게이트를 실행할 수 있는지 판단한 뒤 실행 가능할 경우 해당 time unit에 실행을, 실행 불가능할 경우 해당 qubit을 제외한 나머지 qubit에 대해 다시 우선순위를 뽑아 다음 실행 게이트를 선택한다. 그러나 선택된 게이트가 2-qubit 게이트이고 해당하는 두 물리적 qubit이 다른 섹터에 있으면 해당 time unit에 곧바로 2-qubit 게이트를 할 수 없다. 이 경우 하나의 물리적 qubit을 다른 물리적 qubit과 같은 이온 체인에 위치시키기 위해 SWAP을 통해 섹터 사이를 건너가야 하며, 구체적인 알고리즘은 [Algorithm 2]와 같다.

Algorithm 1 TrapCounterFinder

Input: *trap* object (having *trap_counter, qubit_list* as class members)
Output: None
Behavior: set next gate as a *trap.trap_counter*, or tell that no possible next gate exists

```
next_logical_qubit_exists = True
while trap_counter is None and next_logical_qubit_exists is True do
    // iterate while trap_counter is not determined but candidate logical qubit(s) exist
    next_logical_qubit_exists = False
    max_priority = MIN
    next_qubit = None
    for each physical_qubit in trap.qubit_list do
        if physical_qubit does not contain logical qubit or
         physical_qubit.logical_qubit.is_complete() or
         physical_qubit.logical_qubit.priority = 0 or
         (physical_qubit.logical_qubit.wait_cycle ≠ 0 and
          physical_qubit.logical_qubit.last_decrement = current_cycle) then
            // skip the physical qubit that cannot be chosen
            continue
        endif
        if physical_qubit.logical_qubit.wait_cycle ≠ 0 and
         physical_qubit.logical_qubit.last_decrement ≠ current_cycle then
            // for the logical qubit that can be executed after the next shuttling occurs
            // simply decrement the wait_cycle variable and continue next iteration
            physical_qubit.logical_qubit.wait_cycle = phy_qubit.logical_qubit.wait_cycle - 1
            physical_qubit.logical_qubit.last_decrement = current_cycle
            continue
        end
        // for this physical qubit, it contains meaningful logical qubit so set the flag as True
        next_logical_qubit_exists = True
        // calculate priority of the qubit by calling update_priority() function, and update max
        priority = update_priority(physical_qubit)
        if priority > max_priority then
            max_priority = priority
            next_qubit = physical_qubit.logical_qubit
        endif
    end
    if next_qubit.gate_counter is type of TQGate and
     two qubits of the next_qubit.gate_counter belongs to the same trap and
     not enough cycles remain to finish next_qubit.gate_counter before the next shuttling then
        // If the chosen next_qubit requires 2-qubit gate for the qubits in the same trap
        // but there are not enough cycles to finish 2-qubit gate before the next ion shuttling,
        // initialize the wait_cycle variable and choose the next_qubit again
        phy_qubit.logical_qubit.wait_cycle = remaining cycle
        phy_qubit.logical_qubit.last_decrement = current_cycle
        continue
    else if next_qubit.gate_counter is type of TQgate and
      two qubits of the next_qubit.gate_counter belongs to the different traps then
```

```
// If the chosen next_qubit requires 2-qubit gate for the qubits in the different trap
        // call remote_trap() function to do proper instructions (e.g., insert SWAPCNOT gate)
        // and choose the next_qubit again
        remote_trap(next_qubit.gate_counter)
        continue
    else if next_qubit.priority > 0 then
        // If the chosen next_qubit is executable, finally set its gate_counter as the trap_counter
        // so that the current trap starts executing it
        trap_counter = next_qubit.gate_counter
            if trap_counter is type of SWAPCNOT then
                trap_counter.remaining_cycle = 15
            else if trap_counter is type of TQGate then
                trap_counter.remaining_cycle = 5
            else if trap_counter is type of QECGate then
                trap_counter.remaining_cycle = 25
            else then
                trap_counter.remaining_cycle = 1
        trap_counter.qubit.priority = MAX
    endif
end
```

Algorithm 2 RemoteSectorHandler

Input: *physical_qubit1, physical_qubit2* (having *logical_qubit, trap* as class members)
Output: None
Behavior: To make logical_qubit1 reach the sector of logical_qubit2, make logical_qubit1 sleep until the SWAPCNOT is possible, or insert SWAPCNOT if it is currently possible

```
logical_qubit1 = physical_qubit1.logical_qubit
trap1 = physical_qubit1.trap
logical_qubit2 = physical_qubit2.logical_qubit
trap2 = physical_qubit2.trap

// if other SWAPCNOT gate already scheduled in this sector, make the current logical qubit
// sleep and append to pending_swap_list to let the trap wake up this gate later
if trap1.pending_swap_list is not empty then
    logical_qubit1.prioritiy = 0
    append logical_qubit1 to trap1.pending_swap_list
    return
// if 1) the shuttling phase and required SWAP are in opposite direction or
//    2) not enough cycles are left for SWAPCNOT operate before the next shuttling or
//    3) physical qubit is already at the end of the sector
// make the current logical qubit sleep and higher the priority of the pair gate instead
else if (right_phase=True and trap1.index > trap2.index) or
 (right_phase=False and trap1.index < trap2.index) or
 (not enough cycles left for SWAPCNOT operation) or
 (right_phase=False and trap1.index(physical_qubit1)=0) or
 (right_phase=True and trap1.index(physical_qubit1)=len(trap1)-1) then
    logical_qubit1.priority = 0
    higher logical_qubit2.priority
    return
```

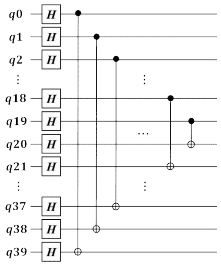
```
// if the SWAPCNOT is possible, determine the proper physical qubit to swap with
else if (right_phase and trap1.index > trap2.index) then
    physical_qubit_to_swap_with = trap1[len(trap1)-1]
else then
    physical_qubit_to_swap_with = trap1[0]
endif

// insert the SWAPCNOT gate to the logical_qubit1 and update the gate_counter to it
logical_qubit1.insert_gate(SWAPCNOT1, logical_qubit1.gate_counter.timestep)
logical_qubit1.gate_counter = SWAPCNOT1

// if the physical qubit to be swapped with also includes the meaningful logical qubit,
// insert the SWAPCNOT gate to that logical qubit and update the gate_counter to it
if physical_qubit_to_swap_with.logical_qubit ≠ None then
    logical_qubit_to_swap_with = physical_qubit_to_swap_with.logical_qubit
    logical_qubit_to_swap_with.insert_gate \
     (SWAPCNOT2, logical_qubit_to_swap_with.gate_counter.timestep)
    logical_qubit_to_swap_with.gate_counter = SWAPCNOT2
endif
```

즉 입력 파일로부터 주어진 양자 회로를 칩 상에서 실행하되, QEC 섹터로 들어간 qubit에 대해서는 신드롬 추출에 해당하는 게이트 연산을, 다른 섹터 qubit과의 2-qubit 게이트를 위해 SWAP 연산이 필요한 경우 해당하는 게이트 연산을 추가해가며 실제 하드웨어 상에서 이루어지는 게이트 연산을 도출해낸다.

위와 같은 전략을 바탕으로 간단한 양자 회로에 대해 알고리즘을 적용해 보았고, 도출된 게이트 연산 스케줄 결과는 [그림 15]와 같다.

[그림 15]의 예시에서는 q0와 q39, q1과 q38 등 멀리 떨어진 두 qubit 간 2-qubit 게이트가 다수 존재하기 때문에 Hadamard 연산 이후 많은 SWAP을 통해 2-qubit 게이트를 해야 하는 두 qubit을 같은 섹터로 모으는 작업이 필요하다. q10을 예시로 들면, 첫 25 time unit 동안 QEC 섹터에서 신드롬 추출과정을 거친 뒤, 26번째 time unit에 이온 shuttling을 통해 두 번째 섹터로 들어간다. 27~32 time unit 동안은 해당 섹터에서 q10에 대한 Hadamard 및 q19, q20 간 *CNOT*이 이루어지고 33~47 time unit 동안 SWAP을 통해 q10과 섹터 맨 마지막의 q19 위치가 바뀐다. 이후 52번째 time unit에서 shuttling을 통해 QEC 섹터로 들어가 53~77번째 time unit 동안 신드롬 추출과정을 거친 뒤, 78번째 time unit

(a)



(b)

QEC sector 1 QEC sector 2 QEC sector 3 QEC sector 4

time unit	sector 1		sector 2		sector 3		sector 4		
1 ~ 24	0 1 2 3 4 5 6 7 8 9	10	11 12 13 14 15 16 17 18 19 20	21	22 23 24 25 26 27 28 29 30 31	32	33 34 35 36 37 38 39		
25	9 1 2 3 4 5 6 7 8 0	10	20 12 13 14 15 16 17 18 19 11	21	22 23 24 25 26 27 28 29 30 31	32	33 34 35 36 37 38 39		
26	9 1 2 3 4 5 6 7 8	0	10 20 12 13 14 15 16 17 18 19	11	21 22 23 24 25 26 27 28 29 30	31	32 33 34 35 36 37 38 39		
27 ~ 32	9 1 2 3 4 5 6 7 8	0	10 20 12 13 14 15 16 17 18 19	11	21 22 23 24 25 26 27 28 29 30	31	32 33 34 35 36 37 38 39		
41 ~ 47	9 1 2 3 4 5 6 7 8	0	10 20 12 13 14 15 16 17 18 19	11	21 22 23 24 25 26 27 28 29 30	31	32 33 34 35 36 37 38 39		
48 ~ 51	8 1 2 3 4 5 6 7 9	0	19 20 12 13 14 15 16 17 18 10	11	21 22 23 24 25 26 27 28 29 30	31	32 33 34 35 36 37 38 39		
52	8 1 2 3 4 5 6 7	9	0 19 20 12 13 14 15 16 17 18	10	11 21 22 23 24 25 26 27 28 29	30	31 32 33 34 35 36 37 38 39		
53 ~ 66	8 1 2 3 4 5 6 7	9	0 19 20 12 13 14 15 16 17 18	10	11 21 22 23 24 25 26 27 28 29	30	31 32 33 34 35 36 37 38 39		
67 ~ 77	7 1 2 3 4 5 6 8	9	0 19 20 18 13 14 15 16 17 12	10	11 21 22 23 24 25 26 27 28 29	30	31 32 33 34 35 36 37 38 39		
78	7 1 2 3 4 5 6	8	9 0 19 20 18 13 14 15 16 17	12	10 11 21 22 23 24 25 26 27 28	29	30 31 32 33 34 35 36 37 38 39		
79 ~ 92	7 1 2 3 4 5 6	8	9 0 19 20 18 13 14 15 16 17	12	10 11 21 22 23 24 25 26 27 28	29	30 31 32 33 34 35 36 37 38 39		
93 ~ 103	6 1 2 3 4 5 7	8	9 17 19 20 18 13 14 15 16 0	12	28 11 21 22 23 24 25 26 27 10	29	30 31 32 33 34 35 36 37 38 39		
104	6 1 2 3 4 5	7	8 9 17 19 20 18 13 14 15 16	0	12 28 11 21 22 23 24 25 26 27	10	29 30 31 32 33 34 35 36 37 38	39	
105 ~ 118	6 1 2 3 4 5	7	8 9 17 19 20 18 13 14 15 16	0	12 28 11 21 22 23 24 25 26 27	10	29 30 31 32 33 34 35 36 37 38	39	
119 ~ 129	5 1 2 3 4 6	7	8 16 17 19 20 18 13 14 15 9	0	12 28 11 21 22 23 24 25 26 27	10	29 30 31 32 33 34 35 36 37 38	39	
130	5 1 2 3 4	6	7 8 16 17 19 20 18 13 14 15	9	0 12 28 11 21 22 23 24 25 26	27	10 29 30 31 32 33 34 35 36 37	38	39
131 ~ 144	5 1 2 3 4	6	7 8 16 17 19 20 18 13 14 15	9	0 12 28 11 21 22 23 24 25 26	27	10 29 30 31 32 33 34 35 36 37	38	39
145 ~ 155	4 1 2 3 5	6	7 15 16 17 19 20 18 13 14 8	9	26 12 28 11 21 22 23 24 25 0	27	10 29 30 31 32 33 34 35 36 37	38	39
156	4 1 2 3	5	6 7 15 16 17 19 20 18 13 14	8	9 26 12 28 11 21 22 23 24 25	0	27 10 29 30 31 32 33 34 35 36	37	38 39

n : qn에 Hadamard 연산

n : qn에 SWAPCNOT 연산

n : qn에 Hadamard 연산과 SWAPCNOT 연산 모두

n : SWAPCNOT을 통해 두 qubit 위치가 바뀐 것

n : 해당 qubit에 양자 회로상 CNOT 연산

그림 15 (a) 40개 qubit에 모두 Hadamard 게이트를 가한 뒤 대칭적으로 CNOT 게이트를 가하는 양자 회로. (b) (a)의 회로를 sector별 qubit 개수가 10개인 칩 상에서 구현했을 때 time unit별 qubit 위치 및 qubit에 가해지는 게이트 스케줄. 26, 52, 78, 104, 130, 156번째 time unit에서는 이온 shuttling을 통해 전체 이온 위치를 오른쪽으로 한 칸 씩 옮겨가며, 나머지 time unit동안 sector별로 필요한 연산을 진행한다. Ion shuttling간 25 time unit이 있는데, 여러 개의 게이트 연산이 가능할 경우 여러 개의 연산을 진행할 수 있다.

에 다음 섹터로 들어간다. 위와 같은 과정을 거쳐 130번째 time unit에 *CNOT*을 하고자 하는 q29와 같은 섹터에 들어가게 되며, 131~135번째 time unit 동안 원래 하고자 했던 *CNOT*을 수행하게 된다. 다른 qubit들에 대해서도 같은 방식으로 같은 섹터에 모은 뒤 *CNOT*을 가하는 방식으로 동작하는 것을 확인할 수 있다.

논리적 qubit 별 우선순위를 계산하는 데 있어서 무수히 많은 전략이 존재할 수 있다. 본 논문에서는 양자 오류 정정에 최적화된 칩 구조를 제안하고 검증하는 것이 목표이므로 다양한 전략들을 비교하지 않았으나, qubit별 게이트 카운터의 게이트 종류, 양자 회로상에서 해당 게이트의 실행 시간, 다음 2-qubit 게이트 혹은 회로 종료 시점까지 남은 시간 등 다양한 변수를 고려하여 우선순위 계산에 활용할 수 있다. 즉 게이트 스케줄링 문제 특성을 반영한 변수 및 함수 선택, 함수 해석을 통한 최적화 및 강화학습을 통한 최적화를 통해 스케줄링 알고리즘을 효율적으로 구성할 수 있다. 본 시뮬레이션에서 사용한 우선순위 계산 식은 다음과 같다.

$$p = f(t_{curr}, t_{CNOT}) = w_{time} \cdot t_{curr} + w_{CNOT} \cdot t_{CNOT} \quad (w_{time} < 0, w_{CNOT} > 0)$$

p : 게이트의 우선순위

t_{curr} : 양자 회로 상의 게이트 time unit

t_{CNOT} : 다음 CNOT 혹은 게이트 종료까지 남은 time unit

첫 번째 항은 양자 회로상에서 게이트가 위치한 time unit(t_{curr})을 고려한 것으로, 이른 시간에 실행되는 게이트일수록 높은 우선순위를 갖는다. 예를 들어 두 개 qubit q_A, q_B 각각의 게이트 카운터에 해당하는 게이트 G_A, G_B 의 time unit이 t_A, t_B이고 $t_A > t_B$ 라 하자. 양자 회로를 실행하는 데 있어서 두 qubit에 대해 비슷한 속도로 연산을 실행하는 것이 좋으므로, q_B에 해당하는 게이트를 실행함으로써 두 qubit의 게이트 연산 진행상태를 비슷하게 맞출 수 있다. 이때 q_B가 높은 우선순위를 갖기 위해서는 t가 작을수록 높은 값을 가져야 하므로 음의 가중치를 준다. ($w_{time} < 0$)

두 번째 항은 계산하고자 하는 게이트 이후 처음으로 등장하는 *CNOT* 혹은 양자 회로 끝까지의 거리를 고려한 것으로, 다음 *CNOT* 혹은 양자 회로 끝까지 거리가 클수록 높은 우선순위를 갖는다. 첫 번째 항과 마찬가지로 모든 qubit 간 연산을 최대한 균등하게 하기 위한 전략으로, *CNOT*과 같은 2-qubit 게이트에서 두 qubit이 *CNOT* 게이트까지 도달한 시간 차이가 크면 먼저 도달한 qubit은 오랜 시간 동안 연산을 할 수 없다. 이처럼 2-qubit 게이트에서 연산을 완료하지 못한 다른 qubit에 의존성이 생기기 때문에 최대한 비슷한 시간에 2-qubit 게이트에 도달함으로써 모든 qubit 연산을 균등히 한다. 다음 *CNOT* 연산이 없는 qubit에 대해서는, 해당 qubit이 다른 qubit들보다 너무 이른 시간에 연산을 끝내는 것을 막기 위해 위와 같은 식으로 우선순위를 계산한다.
이 외에도 다양한 변수를 고려해 우선순위를 정할 수 있으며, 최적의 전략을 찾기 위한 여러 연구가 필요하다. 본 시뮬레이터는 Parameter 클래스를 별도로 만들어 qubit 별 우선순위를 계산할 때 사용할 수 있게 하였으며, 추가한 가중치 항목([표 3]) 및 이를 모두 고려한 우선순위 계산식은 다음과 같다.

$$p_{1Q} = f_{1Q}(t_{curr}, t_{CNOT}) = w_{time} \cdot t_{curr} + w_{CNOT} \cdot t_{CNOT} \quad (w_{time} < 0, w_{CNOT} > 0)$$

$$\begin{aligned} p_{2Q} &= f_{2Q}^{A,B}\left(t_{curr}^{A}, t_{CNOT}^{A}, t_{shuttlemargin}^{A}, t_{toCNOT}^{B}, d_{ion}^{A,B}, d_{trap}^{A,B}\right) \\ &= w_{time} \cdot t_{curr}^{A} + w_{CNOT} \cdot t_{CNOT}^{A} + w_{shuttlemargin} \cdot t_{shuttlemargin}^{A} + w_{iondist} \cdot d_{ion}^{A,B} \\ &\quad + w_{trapdist}^{remote} \cdot d_{trap}^{A,B} + \delta_{trapA,trapB} \cdot w_{CNOT}^{remote} \cdot t_{toCNOT}^{B} \end{aligned}$$

양자 회로의 모든 게이트를 완료할 때까지 위의 알고리즘을 이용해 게이트를 선택하고 실행하는 과정을 반복하면 최종 하드웨어 스케줄을 얻게 된다. 생성된 하드웨어 스케줄은 실제 이온 트랩 시스템에서 일어난 연산 순서들을 의미하므로, 확률적으로 게이트들에 오류를 발생시키고 신드롬 추출 및 오류 정정 단계를 이용해 최종적인 오류 발생 여부를 알아내는데 사용할 수 있다. 또한, 하드웨어 스케줄의 길이는 실제 양자 회로 구현에 필요한 시간(latency)을 의미하며, 한 time unit에 해당하는 시간을 곱하면 구체적인 실행 시간(running time)을 얻을 수 있다.

표 4 qubit 게이트의 우선순위 계산을 위해 추가로 설정 가능한 가중치 및 의미.

가중치 항목	설명
$w_{shuttlemargin}$ (negative)	같은 이온 체인의 두 qubit에 대한 2-qubit 게이트에 대해 현재 time unit으로부터 다음 shuttling까지 남은 time unit에 대한 가중치이다. Shuttling은 이온의 위치를 바꾸는데, 이에 따라 같은 이온 체인의 두 qubit이 다른 이온 체인에 위치하게 될 수 있다. 따라서 다음 shuttling까지 충분한 시간이 남지 않은 경우 우선순위를 높임으로써 빨리 시작하는 전략을 취할 수 있다. 다만 shuttling 이후에도 같은 이온 체인에 있을 경우가 더 많고, 2-qubit 게이트를 실행하기에 충분한 시간이 남지 않은 경우 오히려 실행을 미루는 것이 유리하기 때문에 향후 비선형적인 계산식을 적용할 수 있다.
$w_{iondist}$ (positive)	같은 이온 체인의 두 qubit에 대한 2-qubit 게이트에 대해 두 이온 간 거리에 따른 가중치이다. 두 이온간 거리가 멀수록 shuttling에 따라 다른 이온 체인으로 분리될 가능성이 높기 때문에, 우선 순위를 높게 함으로써 SWAP 없이 같은 이온 체인에서 2-qubit 연산을 할 수 있게 한다.
$w_{trapdist}^{remote}$ (positive)	다른 이온 체인의 두 qubit에 대한 2-qubit 게이트에 대해 두 이온 체인 간 거리에 따른 가중치이다. 두 이온 체인간 거리가 멀수록 많은 SWAP 연산을 필요하기 때문에, 우선순위를 높임으로써 빨리 SWAP을 시작하고 해당 2-qubit 게이트로 인해 다른 qubit들에 대한 연산을 모두 할 수 없는 상황을 방지한다.
w_{CNOT}^{remote} (negative)	다른 이온 체인의 두 qubit에 대한 2-qubit 게이트에 대해, 상대 qubit이 해당 2-qubit 게이트까지 도달하기 위해 소요되는 시간에 대한 가중치이다. 추가적인 SWAP을 통해 두 qubit이 같은 이온 체인에 모여야 하는데, 상대 qubit이 해당 2-qubit 게이트까지 도달하기 전 많은 선행 게이트 연산이 필요할수록 SWAP을 수행하기 위한 여유가 많다. 따라서 우선 순위를 낮추고 SWAP 시작 시점을 뒤로 미룰 수 있다.

4.1.3. 에러 모델 적용 및 성능 평가

하드웨어 스케줄을 따라 게이트를 스캔하면서 확률적으로 오류를 발생시키면 양자 회로 마지막 단계에서 qubit 별 오류 또한 확률적으로 발생한다. 이전까지 시뮬레이션 단계에서는 물리적 qubit 하나가 논리적 qubit 하나를 의미했으나, 에러 모델 적용 단계에서는 논리적 qubit을 구성하는 여러 개의 물리적 qubit을 개별적으로 고려해야한다. Steane 코드를 예시로 들면, 7개의 물리적 qubit이 1개의 논리적 qubit을 구성하는데 한 개의 오류까지는 신드롬 측정 및 오류 정정 단계에서 정확히 검출할 수 있다. 반면 두 개 이상의 오류는 올바르게 정정할 수 없거나 검출이 불가능하다. 따라서 본 시뮬레이션에서는 하드웨어 스케줄 상의 게이트마

다 7번의 난수 생성 및 설정한 오류율과의 비교를 통해 물리적 qubit들에 오류를 발생시키고, 해당 qubit들이 QEC 섹터에 진입했을 때 총 발생한 오류 개수를 세어 최종 오류 여부를 판단하였다. Steane 코드에서는 7개의 물리적 qubit으로 한 개 qubit에 대한 오류를 수정할 수 있으나, 확장된 color 코드에서는 더 많은 물리적 qubit을 사용해 여러 qubit에 대한 오류를 수정할 수 있고 시뮬레이터에서의 파라미터값 조절만으로 이를 구현할 수 있다. 구체적인 에러 모델 구현 방법은 [그림 16]과 같다.

이러한 알고리즘을 이용하여 몇 가지 양자 회로에 대해 게이트별로 오류를 생성하고, 양자 회로의 실행이 끝난 뒤 최종적인 오류율을 분석해보았다. 구체적인 실험 내용은 다음과 같다.

4.2. 칩 시뮬레이션 결과

첫 번째 실험에서는 게이트별 오류가 발생할 확률을 변수로 하여 최종 양자 회로 오류율과의 상관관계를 분석했으며, 총 섹터 개수는 10개, 섹터별 이온 개수는 20개로 가정하였다. 실제 구현에서는 차이가 있을 수 있으나, 2-qubit 게이트 실행 시간이 1-qubit 게이트 실행 시간의 5배라 가정했으므로 게이트별 오류율 또한 2-qubit 게이트가 1-qubit 게이트의 5배라 가정하였다. 향후 실제 실험값에 맞게 해당 오류율을 조절하고자 할 경우, 파라미터 수정을 통해 재실험할 수 있다. 실험에 사용한 양자 회로는 멀리 떨어진 2-qubit 게이트 검증을 위해 사용한 양자 회로(RemoteCNOT), 1-qubit 게이트로만 구성된 양자 회로(wide200q, deep50q), qubit 개수와 회로 깊이(circuit depth)를 조절하여 1-qubit, 2-qubit 게이트를 5:1 비율로 랜덤하게 생성한 임의의 양자 회로들(10q;250d, 20q;125d)이며, 각 회로의 그림은 [그림 17]과 같다. 각 회로에 대해 500번의 오류 생성 및 최종 오류를 판단하여 오류율을 계산했으며, 해당 과정을 다시 20번 반복하여 평균 및 표준편차를 구했다. 아래는 회로별 하드웨어 스케줄에 따른 latency([표 4]) 및 실험 결과([그림 18])이다.

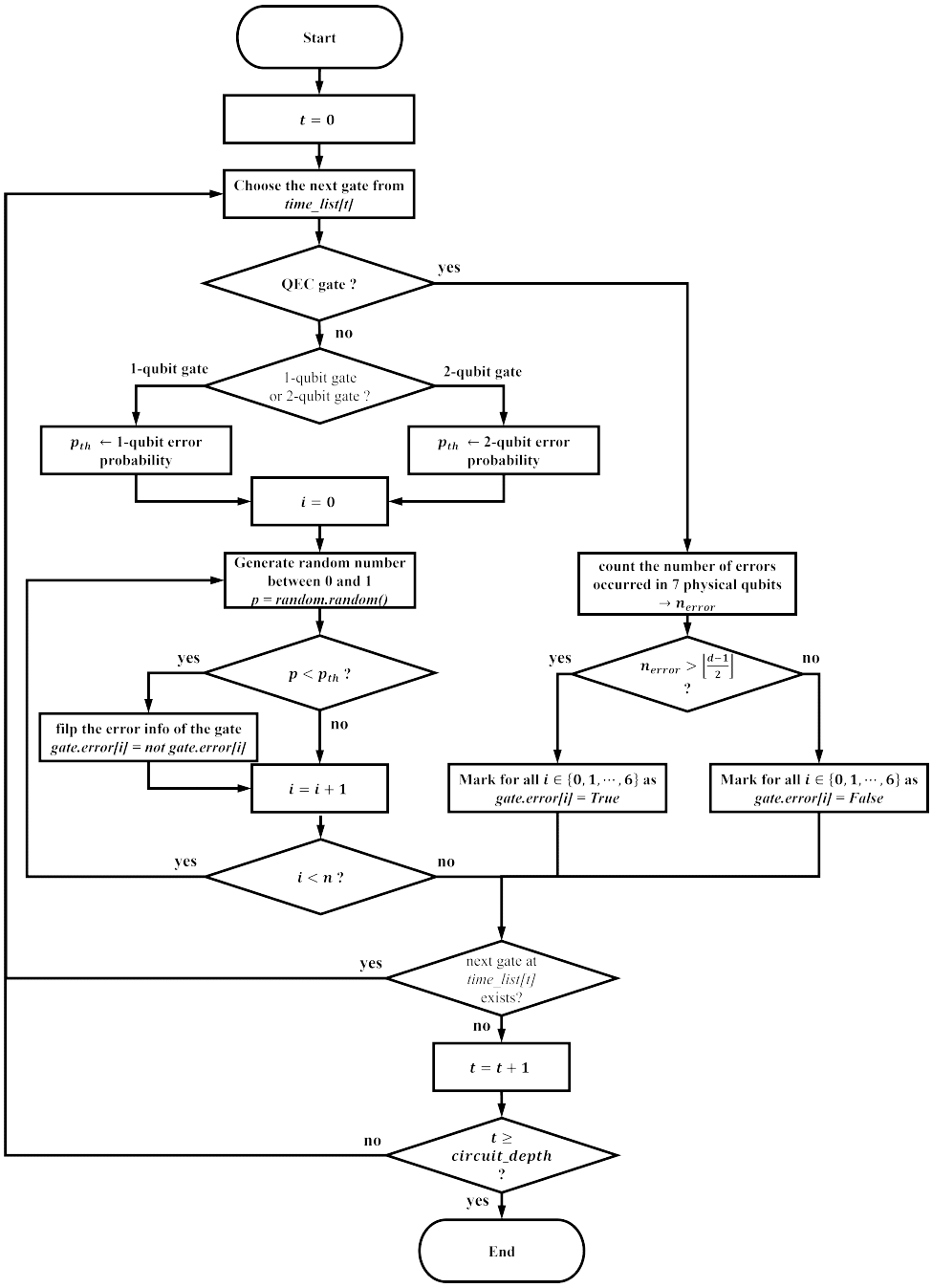


그림 16 생성된 칩 상의 하드웨어 스케줄에 대해, 1-qubit 또는 2-qubit 게이트에 대해서 확률적으로 발생시키고, 신드롬 추출과정에서는 오류정정부호에 따라 오류를 수정하는 알고리즘 순서도.

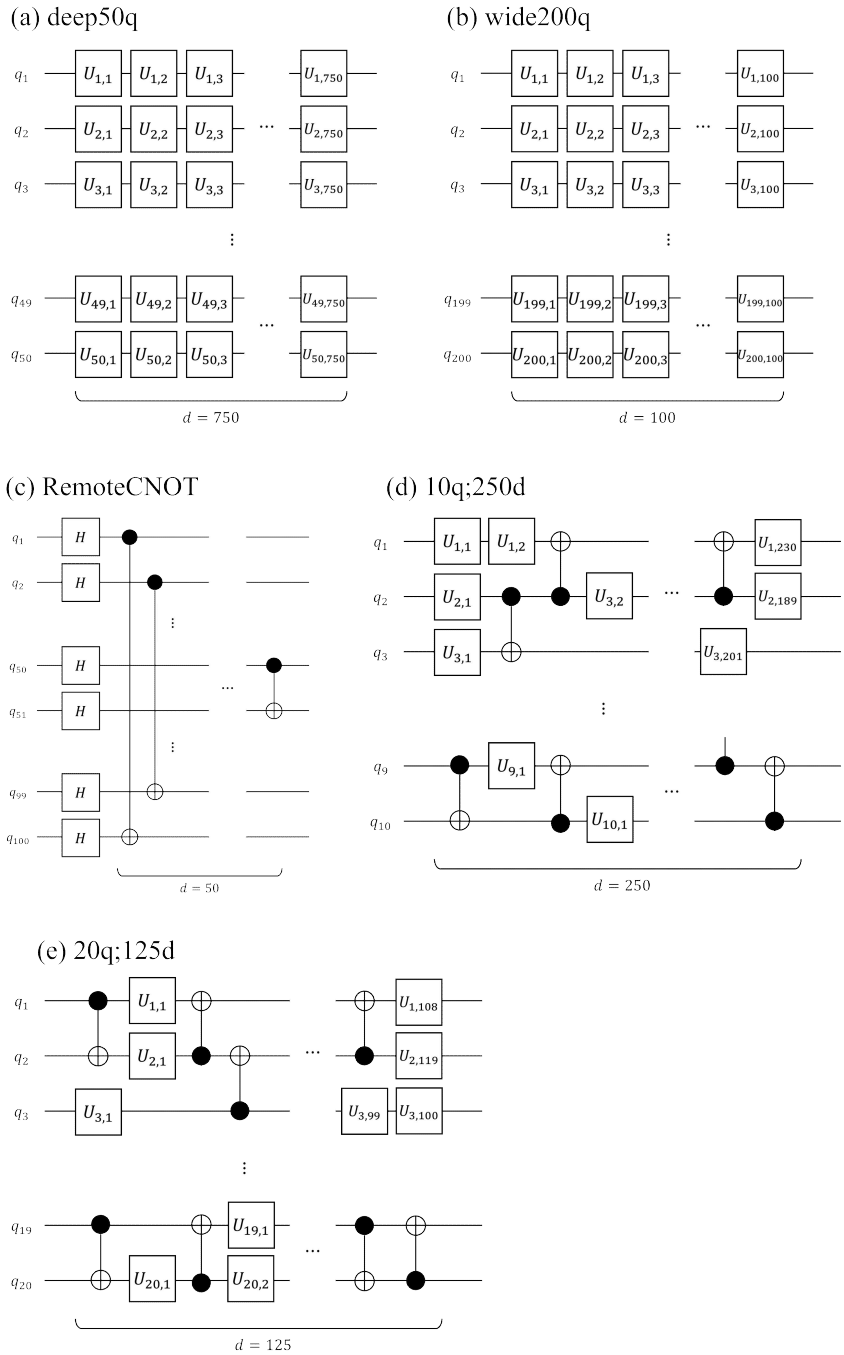


그림 17 시뮬레이션에 사용한 다섯 개 회로의 형태. (a) deep50q, (b) wide200q, (c) RemoteCNOT, (d) 10q;250d, (e) 20q;125d.

표 5 회로별 실제 하드웨어 스케줄의 latency와 한 time unit이 50μs라 가정했을 때 실제 running time.

양자 회로	latency (time unit)	running time (s)
deep50q	12,409	0.62045
wide200q	2,185	0.10925
RemoteCNOT	1,279	0.06395
10q;250d	5,501	0.27505
20q;125d	14,785	0.73925

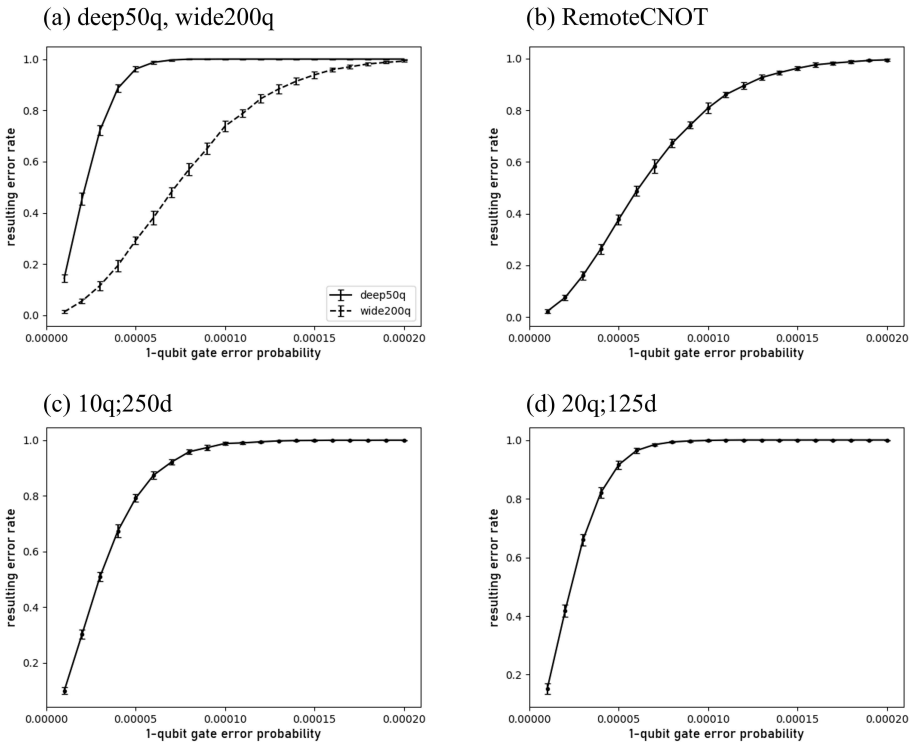


그림 18 (a) 50개의 qubit에 대한 1-qubit 게이트 37500개로 구성된 deep50q (실선), 200개의 qubit에 대한 1-qubit 게이트 20000개로 구성된 wide200q(점선)에 대한 결과. (b) 100개 qubit에 대해 Hadamard 게이트를 가한 뒤 $(q0, q99)$, $(q1, q98)$, $\cdots$, (q_{49}, q_{50}) 쌍에 대해 CNOT 연산을 하는 RemoteCNOT 회로에 대한 결과. (c) 10개 qubit에 대해 약 250의 회로 깊이를 가지고 1-qubit 게이트와 2-qubit 게이트의 비율이 1:5인 랜덤 양자 회로에 대한 결과. (d) 20개 qubit에 대해 약 125의 회로 깊이를 가지고 1-qubit 게이트와 2-qubit 게이트의 비율이 1:5인 랜덤 양자 회로에 대한 결과.

게이트별 오류율이 증가함에 따라 최종 결과에 오류가 발생하는 비율이 증가하는 추세에는 알고리즘별로 차이가 있으나, 1-qubit 게이트별 오류가 10^{-5} 정도 스케일을 가질 때 뚜렷한 변화를 보였고 10^{-4} 수준에서는 최종 오류율이 50%를 넘었다. 특히 랜덤으로 생성된 회로는 다른 회로에 비해 매우 빠른 속도로 오류율이 증가했는데, 특정 규칙 없이 게이트 연산이 수행되었기 때문에 멀리 떨어진 qubit 간 2-qubit 게이트에 따른 많은 SWAP 연산이 필요했을 것으로 보인다. 이 경우 입력된 양자 회로에 비해 실제 하드웨어에서 작동하는 게이트 연산은 훨씬 많아지고, 오류 발생확률도 증가한다.

두 번째 실험에서는 섹터별 이온 개수를 변수로 하여 최종 양자 회로 오류율과의 상관관계를 분석하였다. 1-qubit 게이트의 오류 발생확률은 5×10^{-5}로, 2-qubit 게이트의 오류 발생확률은 2.5×10^{-4}로 가정하였으며, 첫 번째 실험과 마찬가지로 500번 오류 생성을 통해 오류율을 계산하고 이를 20번 반복하여 평균 및 표준편차를 구했다. 실험은 2-qubit 게이트를 포함한 회로(RemoteCNOT, 10q;250d, 20q;125d)에 대해서만 이루어졌으며, 실험 결과는 다음과 같다([그림 19]).

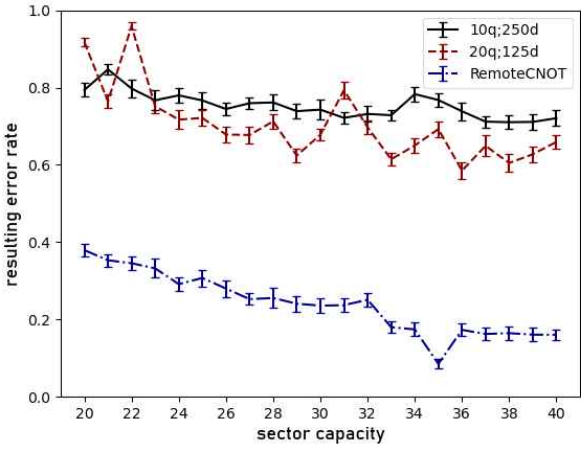


그림 19 10q;250d, 20q;125d, RemoteCNOT 세 개 회로에 대해 섹터별 최대 이온 개수(sector capacity)를 바꾸어가며 최종 오류율을 분석한 결과. 1-qubit 게이트 오류 발생확률은 5×10^{-5}, 2-qubit 게이트 오류 발생확률은 2.5×10^{-4}로 시뮬레이션하였다.

위 결과에서 섹터별 이온 개수가 증가함에 따라 최종 오류율이 감소하는 것을 볼 수 있다. 이는 멀리 떨어진 두 qubit 간 2-qubit 게이트를 함에 있어서, 섹터 당 많은 이온이 있을수록 적은 SWAP이 필요하거나 SWAP 없이 바로 2-qubit 게이트가 가능할 수 있기 때문으로 보인다. 따라서 본 실험의 조건에서는 하나의 섹터에 최대한 많은 이온을 포획하는 것이 최종 오류율 측면에서 유리하다고 할 수 있다. 다만, 극단적으로 하나의 섹터에 모든 qubit을 포획할 경우 SWAP 없이 모든 양자 회로를 구현할 수 있으나 이 경우 shuttling을 통해 QEC 섹터에서의 오류 정정 기회를 가질 수 없다. 또한, 실제 칩을 이용한 이온 트랩 기반 양자컴퓨터 구현에 있어서 너무 많은 이온을 하나의 이온 체인에 포획할 경우 이온 간 간격이 달라지거나 가까운 이온 간의 크로스톡과 같은 문제가 발생한다. 따라서 본 실험 결과를 baseline으로 하여, 물리적 구현 단계에서 허용하는 한계, 섹터별 이온 개수에 따라 달라지는 게이트별 오류율 등의 조건을 추가하면 최적의 섹터별 이온 개수를 도출해낼 수 있다.

제안한 칩 구조는 Steane 코드 또는 color 코드 사용에 따른 기본적인 양자 회로 구조를 고려했으나, 트랙 내에서 이온의 움직임은 여전히 양자 알고리즘의 구조에 의존적이다. 두 번째 실험의 결과([그림 19])의 일부 구간에서 섹터별 이온 개수가 증가함에 따라 최종 오류율도 같이 증가하는 부분이 있고, 해당 구간의 위치는 구현한 알고리즘에 따라 다른 것을 확인할 수 있다. 즉 하드웨어 파라미터를 최적화하는 데 있어서 물리적 한계부터 양자 알고리즘까지 여러 계층에서의 제약사항을 고려할 필요가 있다.

위 실험들을 통해 제안한 칩 구조상에서 다양한 양자 알고리즘을 구현하고, 우선순위 기반의 하드웨어 스케줄링이 가능하다는 것을 확인하였다. 또한, 시뮬레이션 과정에서 수정할 수 있는 변수를 파라미터로 구성하고 몇 가지 파라미터를 달리 하며 반복적으로 시뮬레이션해봄으로써 최종 결과 오류율과의 상관관계를 파악했다. 게이트별 오류 발생확률과 전체 오류 발생확률 간 관계를 통해 게이트별 오류 확률을 10^{-5} 이하로 줄일 필요가 있다는 것을 알 수 있으며, 이는 현재 실험적으로 달성한 확률에

비해 다소 낮은 수치이다. 다만 섹터별 이온 개수에 따른 전체 오류 발생확률의 변화를 통해 하나의 이온 체인에 많은 이온을 포획할수록 유리하다는 것을 알 수 있으며, 이를 고려한다면 목표 오류율을 달성하기 쉬운 수준으로 올릴 수 있을 것으로 예상한다.

향후 구체적이고 고도화된 칩 구조 설계를 위해서는 모든 파라미터에 대한 탐색과 함께 현실적인 제약사항을 반영한 시뮬레이터 구현이 필요하다. 특히 다양한 파라미터에 대한 많은 시뮬레이션을 위해, 시뮬레이션 시간 단축을 위한 효율적인 시뮬레이션 알고리즘 구현이 필요하다. 나아가 우선순위 기반의 스케줄링 외에 다양한 스케줄링 규칙이 존재하며 CPU의 프로세스 및 스레드 스케줄링, HDD 및 SSD의 읽기/쓰기 스케줄링 등 고전컴퓨터에서 사용하는 전략을 양자컴퓨터 게이트 스케줄링에 맞게 수정하는 것도 가능하다.

5. 결론

본 연구에서는 효율적인 양자 오류 정정 부호 구현을 위한 이온 트랩 칩 구조를 제안하고, 해당 칩에서의 양자 회로 구현 모델링 및 결과 스케줄에 대한 실행시간과 여러 오류 모델에 대한 성능 분석을 하였다. 제안한 이온 트랩 칩은 가로 방향으로 배열된 평행한 여러 이온 체인(트랙)들을 병렬적으로 처리함으로써, Steane 코드 및 color 코드에서의 transversal 게이트의 실행을 하드웨어 구조적으로 최적화한다. 또한, 주기적인 이온 shuttling을 통해 모든 논리적 qubit들이 주기적인 신드롬 측정 과정을 거칠 수 있고, 적절한 SWAP 연산을 넣으면 다른 섹터에 존재하는 논리적 qubit간 2-qubit 게이트 연산이 가능하다. 특히 많은 qubit을 사용하는 확장성 있는 이온 트랩 칩에 관한 기존 연구들의 경우, 격자 형태로 배치된 이온 shuttling 통로에서 다양한 방식으로 이온을 움직이며 2-qubit 연산을 진행하였다. 그러나 해당 구조들에서는 하나의 이온 체인에 속한 이온 수가 많지 않기 때문에 full-connectivity라는 이온 트랩 기반 양자컴퓨터의 장점을 살릴 수 없으며, 양자 오류 정정 부호를 사용할 경우 그에 따른 회로 구조를 고려한 최적화를 하기 어렵다. 제안 칩

구조는 이러한 문제를 반영해 설계되었고, 실제 양자 알고리즘 구현에 따른 칩 상에서의 하드웨어의 스케줄과 오류에 대한 분석을 위해 이온의 움직임 및 게이트 스케줄을 모델링하는 시뮬레이터를 구현하였다. 이를 통해 다양한 회로를 입력으로 하여 스케줄링 결과에 대한 분석을 진행하였다.

실험은 게이트별 오류 발생확률에 따른 최종 알고리즘 오류 확률과 칩을 구성하는 섹터별 이온 개수에 따른 최종 알고리즘 오류 확률을 분석하였다. 실험 결과 10^{-5} 이하의 게이트별 오류 확률이 요구되고 현실적인 오류 모델 도입에 따라 해당 수치는 더 내려갈 수 있다. 다만 섹터별 이온 개수에 따른 오류 확률 실험에서 파라미터 수정을 통한 최적화의 가능성을 볼 수 있었고, 하드웨어 칩 구조를 정의하는 수많은 파라미터에 대한 최적화를 통해 요구되는 게이트 오류 확률을 높일 수 있다.

향후 제안한 칩 구조를 설계하는 데 있어서 위 실험들을 바탕으로 최적의 양자 오류 정정 부호 크기(혹은 트랙 개수), 이온 체인당 이온 개수, 논리적 qubit의 물리적 qubit으로의 초기화 전략 등을 수정하여 시뮬레이터를 활용할 수 있다. 특히 SWAP 연산에 따른 전체 오류의 증가가 클 것으로 예상하므로, 우선순위를 계산하는 식과 가중치를 최적화하여 섹터 간 이온의 움직임을 최소화하는 것이 중요하다. 또한, 시뮬레이터 자체의 알고리즘 개선을 통해 많은 파라미터를 탐색하는데 필요한 시간을 줄일 수 있다.

참고문헌

[1] Smith, A., Kim, M.S., Pollmann, F. et al. Simulating quantum many-body dynamics on a current digital quantum computer. npj Quantum Inf 5, 106 (2019).

[2] Lanyon, B., Whitfield, J., Gillett, G. et al. Towards quantum chemistry on a quantum computer. Nature Chem 2, 106 - 111 (2010).

[3] Brown, K., Trout, C. Magic State Distillation and Gate Compilation in Quantum Algorithms for Quantum Chemistry. Preprint at: https://arxiv.org/abs/1501.01298 (2015).

[4] Wang, P., Luan, CY., Qiao, M. et al. Single ion qubit with estimated coherence time exceeding one hour. Nat Commun 12, 233 (2021).

[5] Schäfer, V., Ballance, C., Thirumalai, K. et al. Fast quantum logic gates with trapped-ion qubits. Nature 555, 75 - 78 (2018).

[6] Reens, D., C, Michael., C, Joseph. et al. High-Fidelity Ion State Detection Using Trap-Integrated Avalanche Photodiodes. Phys. Rev. Lett. 129, 100502 (2022).

[7] Wang, Y., Crain, S., Fang, C. et al. High-Fidelity Two-Qubit Gates Using a MEMS based Beam Steering System for Individual Qubit Addressing. Phys. Rev. Lett. 125, 150505 (2020).

[8] Kaushal, V., Lekitsch, B., Stah, A. et al. Shuttling-Based Trapped-Ion Quantum Information Processing. Preprint at: https://arxiv.org/abs/1912.04712 (2019).

[9] Suzuki, Y., Endo, S., Fujii, K. et al. Quantum Error Mitigation as a Universal Error Reduction Technique: Applications from the NISQ to the Fault-Tolerant Quantum Computing Eras. PRX Quantum 3, 010345 (2022).

[10] Calderbank, A., Shor, P. Good Quantum Error Correcting Codes Exist. Phys. Rev. A 54, 1098 (1996).

[11] Shor, P. Scheme for Reducing Decoherence in Quantum Computer Memory. Phys. Rev. A 52, R2493(R) (1995).
[12] Kubica, A. The ABCs of the Color Code: A Study of Topological Quantum Codes as Toy Models for Fault-Tolerant Quantum Computation and Quantum Phases Of Matter. (California Institute of Technology, 2018)
[13] Fowler, A. Two-dimensional color-code quantum computation. Phy. Rev. A 83, 042310 (2011).
[14] Landahl, A., Anderson, J., Rice, P. Fault-Tolerant Quantum Computing with Color Codes. Preprint at: https://arxiv.org/abs/1108.5738 (2011).
[15] Nigg, D., Muller, M., Martinez, E. et al. Quantum Computations on a Topologically Encoded Qubit. Science 345, 302-305 (2014).
[16] Postler, L., Heuβen, S., Pogorelov, I. et al. Demonstration of fault-tolerant universal quantum gate operations. Nature 605, 675 – 680 (2022).
[17] Anderson, C., Bohnet, J., Lee, K. et al. Realization of Real-Time Fault-Tolerant Quantum Error Correction. Phys. Rev. X 11, 041058 (2021).
[18] Kitaev, A. Quantum Computations: Algorithms and Error Correction. Russ. Math. Surv. 52 1191 (1997).
[19] Nielsen, A., Chuang, L. Quantum Computation and Quantum Information (Cambridge University Press, 2010).
[20] Tolar, J. On Clifford Groups in Quantum Computing. J. Phys.: Conf. Ser. 1071 012022 (2018).
[21] Gottesman, D. Theory of fault-tolerant quantum computation. Phys. Rev. A 57, 127 (1998).
[22] Gottesman, D. The Heisenberg Representation of Quantum Computers. Preprint at: https://arxiv.org/abs/quant-ph/9807006 (1998).

[23] Aaronson, S., Gottesman, D. Improved simulation of stabilizer circuits. Phy. Rev. A 70, 052328 (2004).
[24] Kandala, A., Temme, K., Córcoles, A.D. et al. Error mitigation extends the computational reach of a noisy quantum processor. Nature 567, 491 - 495 (2019).
[25] Piveteau, C., Sutter, D., Bravyi, S. et al. Error Mitigation for Universal Gates on Encoded Qubits. Phys. Rev. Lett 127, 200505 (2021).
[26] Zhang, S., Lu, Y., Zhang, K. et al. Error-mitigated quantum gates exceeding physical fidelities in a trapped-ion system. Nat Commun 11, 587 (2020).
[27] Mari, A., Shammah, N., Zeng, W. Extending quantum probabilistic error cancellation by noise scaling. Phys. Rev. A 104, 052607 (2021).
[28] Ueno, Y., Kondo, M., Tanaka, M. et al. QECOOL: On-Line Quantum Error Correction with a Superconducting Decoder for Surface Code. 2021 58th ACM/IEEE Design Automation Conference (DAC), 451-456 (2021).
[29] Ueno, Y., Kondo, M., Tanaka, et al. QULATIS: A Quantum Error Correction Methodology toward Lattice Surgery. 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 274-287 (2022).
[30] Devitt, S., Nemoto, K., Munro, W. Quantum Error Correction for Beginners. Preprint at: https://arxiv.org/abs/0905.2794 (2013).
[31] Roffe, J. Quantum Error Correction; An Introductory Guide. Preprint at: https://arxiv.org/abs/1907.11157 (2019).
[32] Reichardt, B. Fault-tolerant quantum error correction for Steane's seven-qubit color code with few or no extra qubits. Quantum Sci. Technol. 6 015007 (2021).

[33] Cianci, G., Poulin, D. Fast Decoders for Topological Quantum Codes. Phys. Rev. Lett. 104, 050504 (2010).
[34] Huang, S., Newman, M., Brown, K. Fault-tolerant weighted union-find decoding on the toric code. Phys. Rev. A 102, 012419 (2020).
[35] Baireuther, P., Caio, M., Criger, B. et al. Neural network decoder for topological color codes with circuit level noise. New J. Phys. 21 013003 (2019).
[36] Gottesman, D. Stabilizer Codes and Quantum Error Correction. (California Institute of Technology, 1997).
[37] Poulin, D. Stabilizer Formalism for Operator Quantum Error Correction. Phys. Rev. Lett. 95, 230504 (2005).
[38] Steane, A. Simple quantum error-correcting codes. Phys. Rev. A 54, 4741 (1996).
[39] Campbell, E., Terhal, B., Vuillot, C. Roads towards fault-tolerant universal quantum computation. Nature 549, 172 - 179 (2017).
[40] Eastin, B., Knill, E. Restrictions on Transversal Encoded Quantum Gate Sets. Preprint at: https://arxiv.org/abs/0811.4262 (2009).
[41] Rengaswamy, N., Calderbank, R., Newman, M. et al. Classical Coding Problem from Transversal T Gates. 2020 IEEE International Symposium on Information Theory (ISIT), 1891-1896 (2020).
[42] Bombin, H. Dimensional jump in quantum error correction. New J. Phys. 18 043038 (2016).
[43] Brown, B., Nickerson, N., Browne, D. Fault-tolerant error correction with the gauge color code. Nat Commun 7, 12302 (2016).
[44] Rengaswamy, N., Calderbank, R., Newman, M. et al. On Optimality of CSS Codes for Transversal T. IEEE Journal on Selected Areas in Information Theory 1 (2), 499-514 (2020).
[45] Lao, L., Criger, B. Magic state injection on the rotated surface

code. Proceedings of the 19th ACM International Conference on Computing Frontiers (2022).
[46] Vasmer, M., Browne, D. Three-dimensional surface codes: Transversal gates and fault-tolerant architectures. Phys. Rev. A 100, 012312 (2019).
[47] DiVincenzo, D. The Physical Implementation of Quantum Computation. Fortschritte der Physik 48, 9-11, 771-783 (2000).
[48] Zoller, P., Cirac, J. Quantum Computations with Cold Trapped Ions. Phys. Rev. Lett. 74, 4091 (1995).
[49] Kaushal, V., Lekitsch, B., Stahl, A. et al. Shuttling-Based Trapped-Ion Quantum Information Processing. Preprint at: https://arxiv.org/abs/1912.04712 (2019).
[50] Lee, M., Jeong, J., Park, Y. et al. Ion shuttling method for long-range shuttling of trapped ions in MEMS-fabricated ion traps. Jpn. J. Appl. Phys. 60 027004 (2021).
[51] Brown, K., Kim, J., Monroe, C. Co-designing a scalable quantum computer with trapped atomic ions. npj Quantum Inf 2, 16034 (2016).
[52] Cho, D., Hong, S., Lee, M. et al. A review of silicon microfabricated ion traps for quantum information processing. Micro and Nano Syst Lett 3, 2 (2015).
[53] Stick, D., Fortier, K., Haltli, R. et al. Demonstration of a microfabricated surface electrode ion trap. Preprint at: https://arxiv.org/abs/1008.0990 (2010).
[54] Hong, S., Lee, M., Kwon, Y. et al. Experimental Methods for Trapping Ions Using Microfabricated Surface Ion Traps. J. Vis. Exp 126, 56060 (2017).
[55] Chiaverini, J., Blakestad, R., Britton, J. et al. Surface-electrode architecture for ion-trap quantum information processing. Quantum Information & Computation 5 (3), 419-439 (2005).

[56] Kielpinski, D., Monroe, C., Wineland, D. Architecture for a large-scale ion-trap quantum computer. Nature 417, 709 - 711 (2002).
[57] Murali, P., Debroy, D., Brown, K. et al. Toward Systematic Architectural Design of Near-Term Trapped Ion Quantum Computers. Communications of the ACM, 65 (3), 101-109 (2022).
[58] Balensiefer, S., Stickles, L., Oskin, M. An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures. 32nd International Symposium on Computer Architecture (ISCA'05), 186-196 (2005).
[59] Metodi, T., Thaker, D., Cross, A. et al. Scheduling physical operations in a quantum information processor. Quantum Information and Computation IV 6244 (2006).
[60] Whitney, M., Isailovic, N., Patel, Y. et al. Automated generation of layout and control for quantum circuits. Proceedings of the 4th international conference on Computing frontiers, 83 - 94 (2007).
[61] Dousti, M., Pedram, M. Minimizing the Latency of Quantum Circuits during Mapping to the Ion-Trap Circuit Fabric. Preprint at: https://arxiv.org/abs/1412.8003 (2014).
[62] Raeisi, M., Mohammadzadeh, N. Scheduling Physical Operations in Quantum Circuits Using a Greedy Algorithm. International Journal of Electrical Energy 2 (3) (2014).
[63] Sargaran, S., Mohammadzadeh, N. SAQIP: A Scalable Architecture for Quantum Information Processors. ACM Transactions on Architecture and Code Optimization 16 (2), 1-21 (2019).

Abstract

Ion-Trap Chip Architecture for Efficient Quantum Error Correction Implementation

Lee Jeonghoon
Department of Computer Science and Engineering
College of Engineering
The Graduate School
Seoul National University

Ion-trap based quantum computer traps ions using electric field, and manipulates quantum information using external electromagnetic field. Especially, chip trap has DC electrodes and RF electrodes on the surface of the chip to make EM field by applying voltages on them. Chip trap has high scalability, but to trap many ions and use them efficiently, algorithm to initialize ions and schedule moving ions needs to be optimized. Also, optimal values of parameters depend on the algorithm running on the chip, so optimization should consider various architecture layers including hardware design and quantum algorithm.
On the other hand, unlike the bit in classical computers, qubit in quantum computers is fragile to the external noise due to its physical properties. Implementing a desired quantum circuit by mapping one logical qubit to one physical qubit makes the result state incorrect with high probability. To make high-fidelity quantum circuit, quantum error correcting (QEC) codes use corresponding gates to encode one

qubit of information in multiple physical qubits. Various QEC codes are studied to find efficient encoding scheme, such as minimizing the number of physical qubits or gates used for encoding. Color code is one type of quantum error correcting code with high scalability and simple logical gate. Color code has Hadamard gate(H), phase gate(S), and CNOT($CNOT$) gate as a transversal gate set, so that applying Hadamard gate to each physical is equivalent to applying logical Hadamard gate to the encoded state. Number and order of logical gates depend on the algorithm, but under the QEC, there are some patterns in the behavior of physical qubits.

This paper proposes the ion-trap chip architecture for efficiently implementing the color code, and analyzes its properties with chip simulation. The parameters in simulation are determined based on the level achievable in the laboratory, and several quantum circuits consisting of 1-qubit gates and 2-qubit gates are used for simulation.

.....

keywords : quantum computer, quantum information, ion-trap, chip trap, quantum error correction code, color code, chip simulation

Student Number : 2021-24050