Qiskit 개발자 자격 시험

Inho Choi

Qiskit Advocate

Qiskit

구성

Qiskit

- Lecture 1: 게이트와 양자 회로
- Lecture 2: 양자 회로의 측정과 OpenQasm
- Lecture 3: 양자 백엔드에 양자회로 실행하기
- Lecture 4: 양자 회로 및 회로의 정보와 실행결과를 해석하기
- Lecture 5: 유용한 기능들

Lecture 3: 양자 회로 실행과 백엔드

1. 양자 회로 실행
2. 백엔드

Qiskit

양자 회로 실행

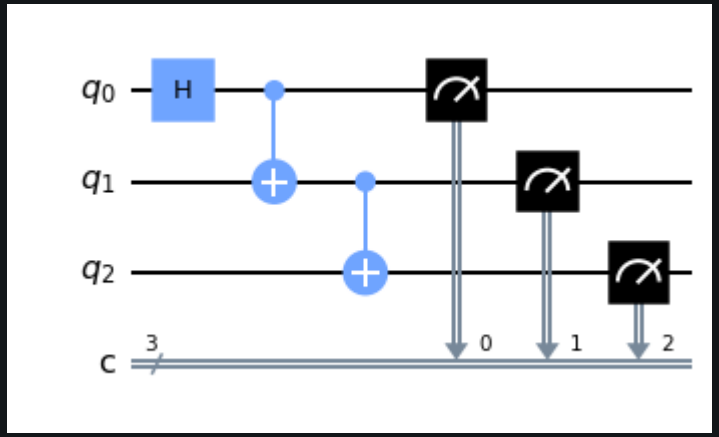
시뮬레이션에서 양자 회로 실행

Qiskit

1. 백엔드 불러오기
2. 실행 함수 사용하여 양자회로, 샷의 개수를 지정해주기
3. 결과값 불러오기
4. 결과값에서 필요한 값을 추출하기

양자 회로 실행

1. 백엔드 불러오기
2. 실행 함수 사용하여 양자회로, 샷의 개수를 지정해주기
3. 결과값 불러오기
4. 결과값에서 필요한 값을 추출하기

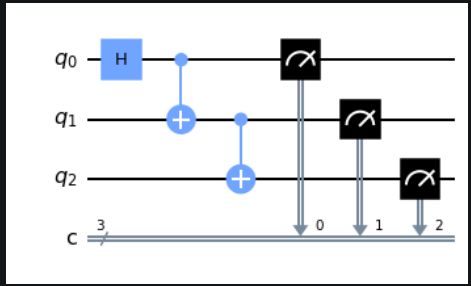


qc

백엔드 불러오기

Qiskit

```
backend = Aer.get_backend('qasm_simulator')
```

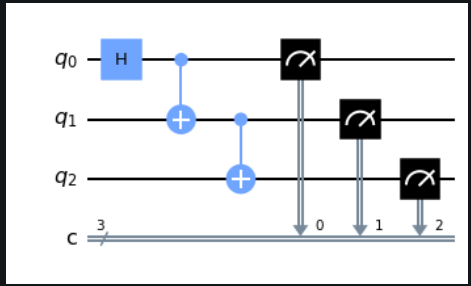


qc

실행 함수 사용

Qiskit

```
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
```

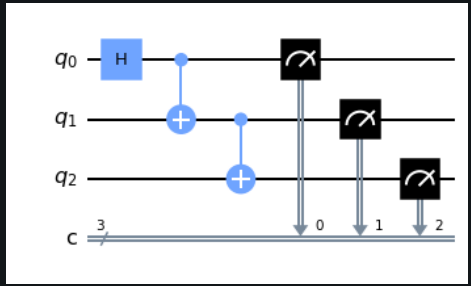


qc

결과값 불러오기

Qiskit

```
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
result = job.result()
```

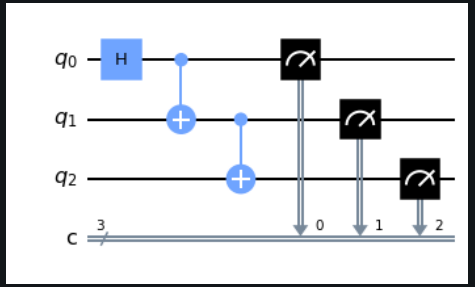


qc

값 추출하기

Qiskit

```
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
result = job.result()
counts = result.get_counts(qc)
```



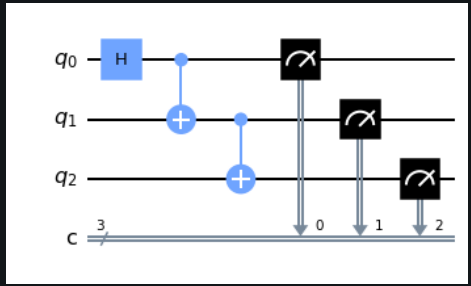
qc

값 출력

Qiskit

```
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
result = job.result()
counts = result.get_counts(qc)

print(counts)
```

```
{'111': 525, '000': 499}
```



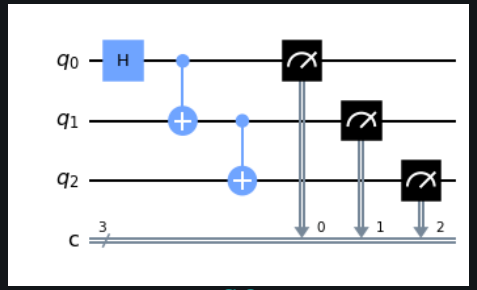
qc

값 출력

Qiskit

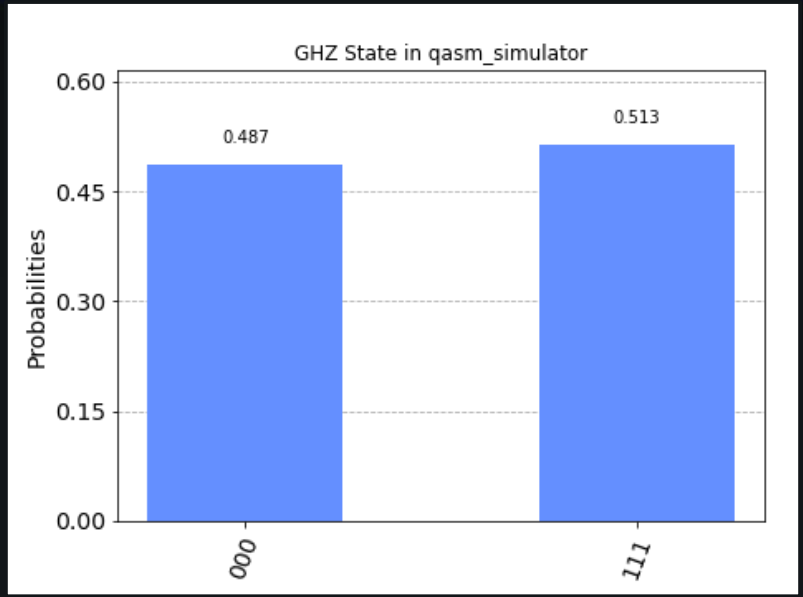
```
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
result = job.result()
counts = result.get_counts(qc)

print(counts)
```

```
{'111': 525, '000': 499}
```



qc

```
plot_histogram(counts, title="GHZ State in qasm_simulator")
```



실제 양자 장치에서 양자 회로 실행

Qiskit

Graphically build circuits with

IBM Quantum Composer

Launch Composer

Develop quantum experiments in

IBM Quantum Lab

Launch Lab

Jump back in:

- Untitled circuit
- Untitled circuit
- quantum-challenge/2022-qgss/lab4/lab-4.ipynb
- quantum-challenge/2022-spring/exercise1/01.CM_...

API token ⓘ

****

View account details

실제 양자 장치에서 양자 회로 실행

Qiskit

```
TOKEN = "YOUR_TOKEN"
IBMQ.save_account(TOKEN)
```

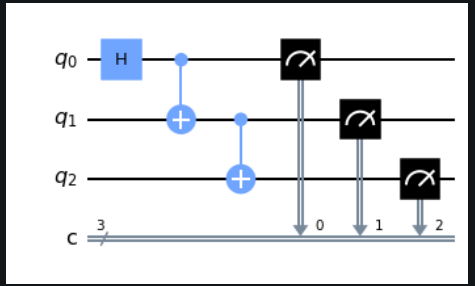
IBMQ 계정을 불러온적이 없고 IBM Quantum Lab에서
노트북을 사용하지 않고 있다면 이 코드를 한번만 실행하세요

실제 양자 장치에서 양자 회로 실행

Qiskit

```
backend=provider.get_backend('ibmq_lima')
job = execute(qc, backend, shots=1024)
result = job.result()
counts = result.get_counts(qc)
print(counts)
```

```
{'000': 480, '001': 14, '010': 2, '011': 19, '100': 10, '101': 15, '110': 22, '111': 462}
```



qc

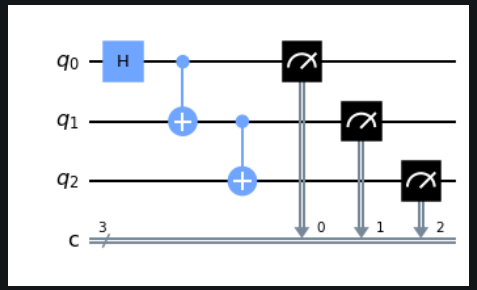
실제 양자 장치에서 양자 회로 실행

Qiskit

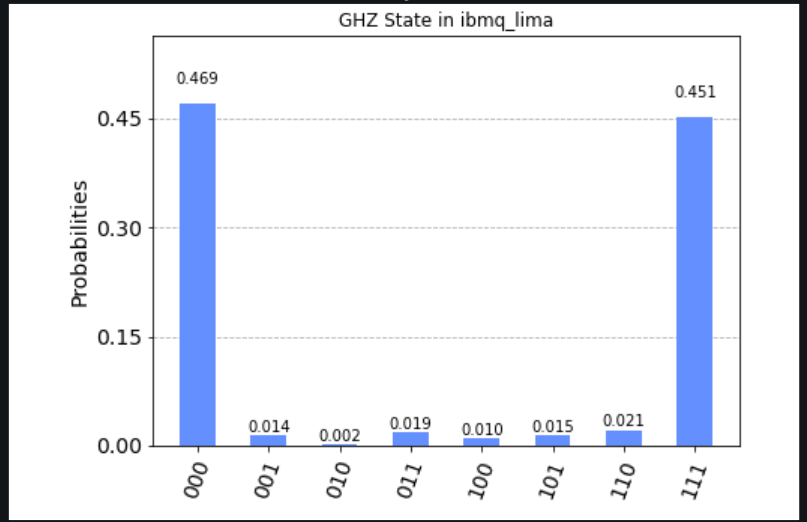
```
backend=provider.get_backend('ibmq_lima')
job = execute(qc, backend, shots=1024)
result = job.result()
counts = result.get_counts(qc)
print(counts)
```

```
plot_histogram(counts, title="GHZ State in ibmq_lima")
```

```
{'000': 480, '001': 14, '010': 2, '011': 19, '100': 10, '101': 15, '110': 22, '111': 462}
```



qc



실제 양자 장치에서 양자 회로 실행

Qiskit

```
backend=provider.get_backend('ibmq_lima')
job = execute(qc, backend, shots=1024)
result = job.result()
counts = result.get_counts(qc)
print(counts)
```

job.cancel(): 보낸 양자 회로 작업 취소
job.status(): 작업 상태 출력
job_monitor(job): 작업을 실시간 모니터링
job.job_id(): 보낸 작업 id 출력

트랜스파일

- 지정한 백엔드의 basis gate에 맞게 양자회로를 트랜스파일

트랜스파일

Qiskit

```
from qiskit.compiler import transpile, assemble
```

트랜스파일

Qiskit

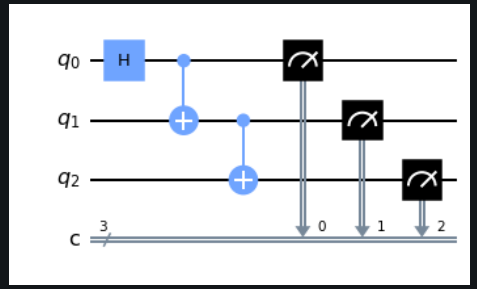
```
from qiskit.compiler import transpile, assemble
```

```
backend = provider.get_backend('ibmq_lima')
backend.configuration().basis_gates
```

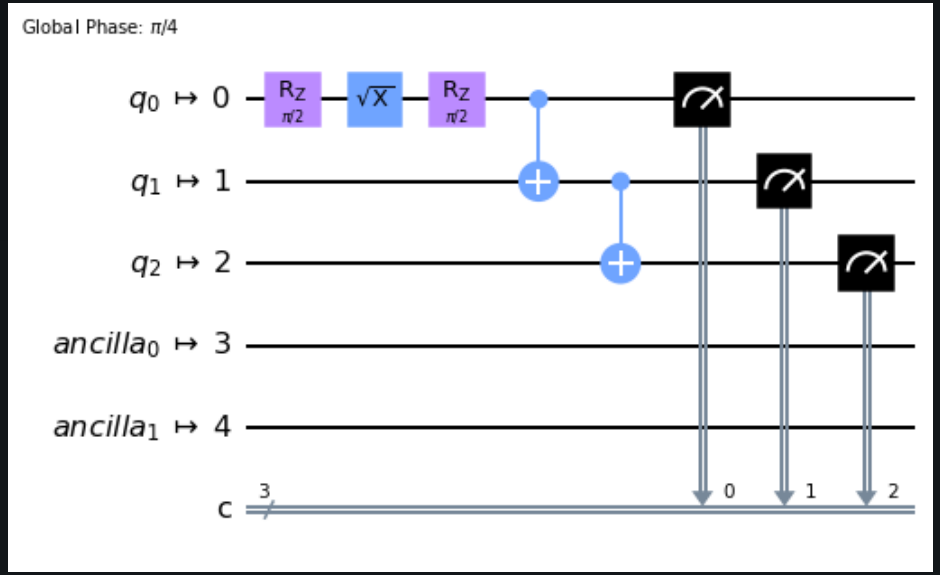
```
['id', 'rz', 'sx', 'x', 'cx', 'reset']
```

트랜스파일

```
backend = provider.get_backend('ibmq_lima')
transpile_circuit = transpile(qc, backend)
```



qc



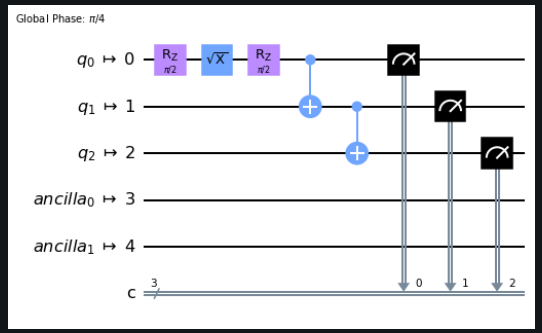
transpile_circuit

트랜스파일

Qiskit

```
backend = provider.get_backend('ibmq_lima')
transpile_circuit = transpile(qc, backend)
job = backend.run(transpile_circuit)
result = job.result()
counts = result.get_counts(transpile_circuit)
print(counts)
```

```
{'000': 1903, '001': 31, '010': 11, '011': 82, '100': 26, '101': 72, '110': 96, '111': 1779}
```



transpile_circuit

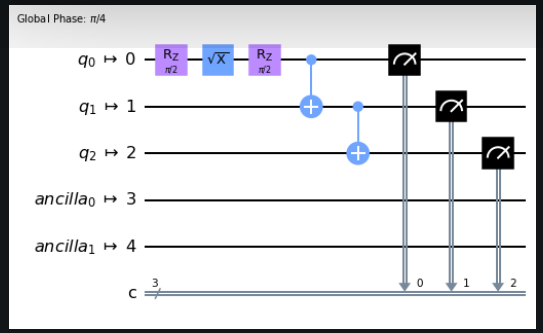
트랜스파일

Qiskit

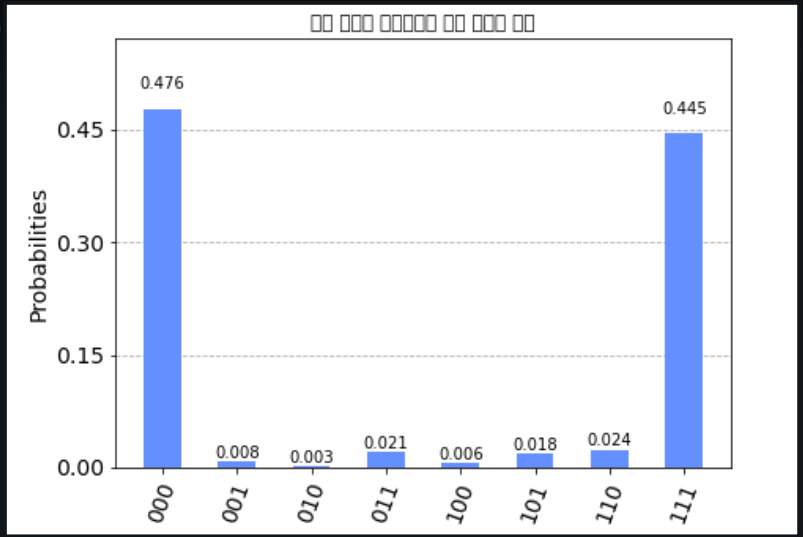
```python
backend = provider.get_backend('ibmq_lima')
transpile_circuit = transpile(qc, backend)
job = backend.run(transpile_circuit)
result = job.result()
counts = result.get_counts(transpile_circuit)
print(counts)
```

```python
plot_histogram(counts, title="양자 회로를 트랜스파일 하여 실행한 결과")
```

```
{'000': 1903, '001': 31, '010': 11, '011': 82, '100': 26, '10
```



transpile_circuit



최적화 트랜스파일

Qiskit

```
transpile(qc, backend, optimization_level=optional, initial_layout=optional)
```

양자회로를 지정해준 단계만큼 최적화 하여 트랜스파일 합니다
initial_layout: 원하는 큐비트를 실제 장치 큐비트에 지정할 수 있습니다.

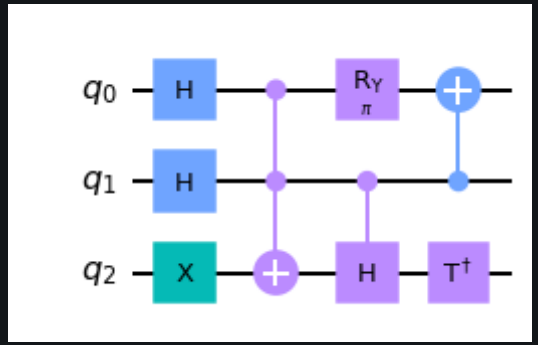
```
plot_circuit_layout(circuit, backend)
```

양자 회로의 layout을 그려줍니다

최적화 트랜스파일

```
plot_circuit_layout(circuit, backend)
```

양자 회로의 layout을 그려줍니다



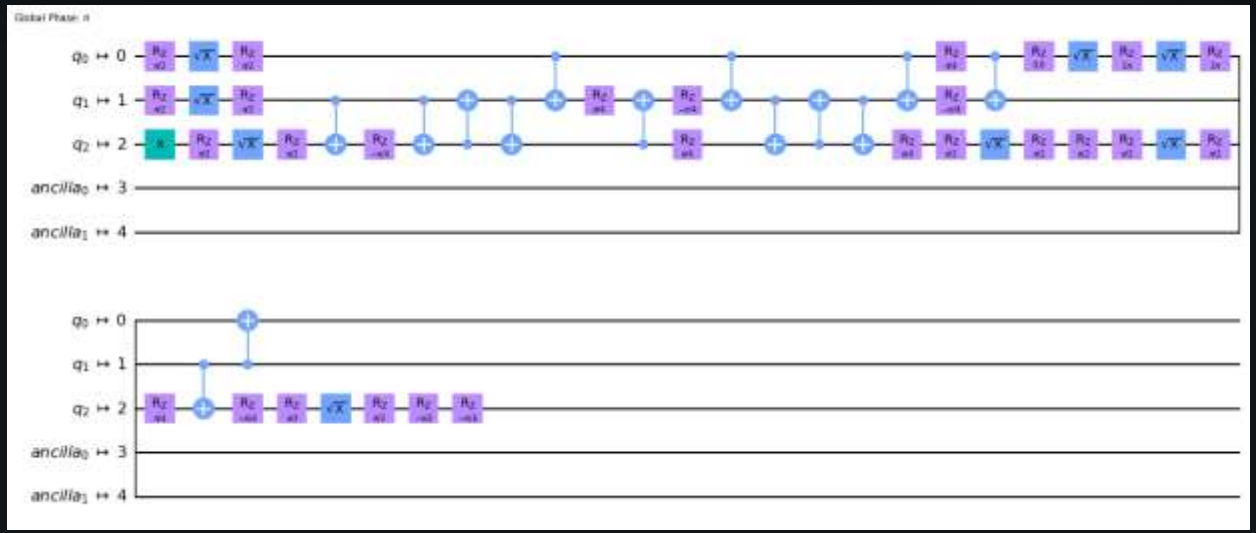
qc_trans

```
transpile(qc, backend, optimization_level=optional, initial_layout=optional)
```

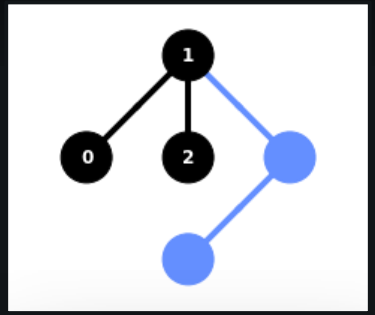
양자회로를 지정해준 단계만큼 최적화 하여 트랜스파일 합니다
initial_layout: 원하는 큐비트를 실제 장치 큐비트에 지정할 수 있습니다.

최적화 0단계 트랜스파일

```
transpile_circuit_0 = transpile(qc_trans, backend, optimization_level=0)
```




```
plot_circuit_layout(transpile_circuit_0, backend)
```




```
transpile_circuit_0.depth()
```

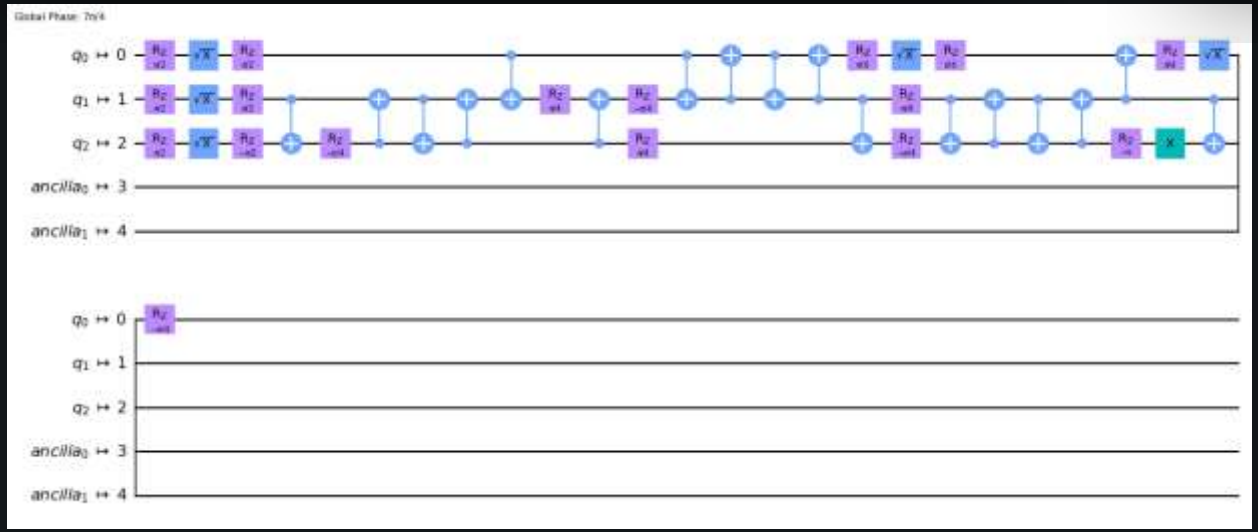
33

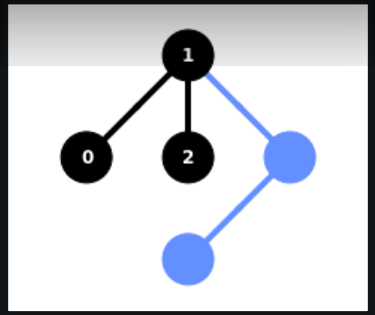
최적화 1단계 트랜스파일

Qiskit

```
transpile_circuit_1 = transpile(qc_trans, backend, optimization_level=1)
```

```
plot_circuit_layout(transpile_circuit_1, backend)
```





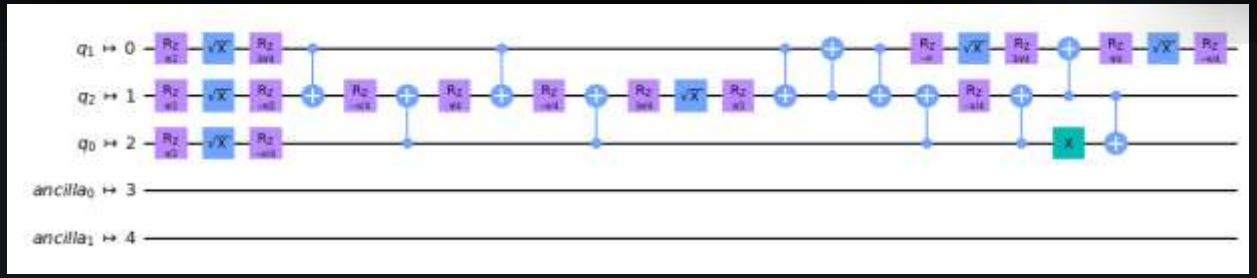

```
transpile_circuit_1.depth()
```

26

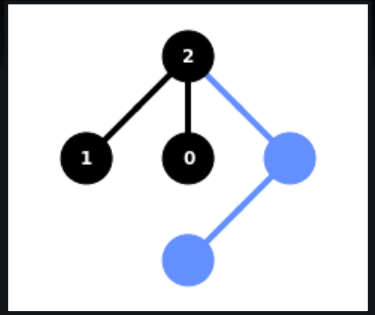
최적화 3단계 트랜스파일

Qiskit

```
transpile_circuit_3 = transpile(qc_trans, backend, optimization_level=3)
```




```
plot_circuit_layout(transpile_circuit_3, backend)
```




```
transpile_circuit_3.depth()
```

23

Qiskit

백엔드

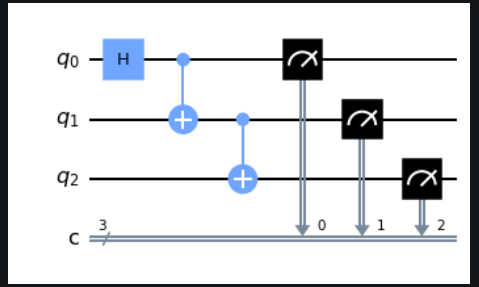
백엔드 시뮬레이터

Qiskit

- Qiskit Aer
- Basic Aer
- 실제 양자 장치

Qiskit Aer

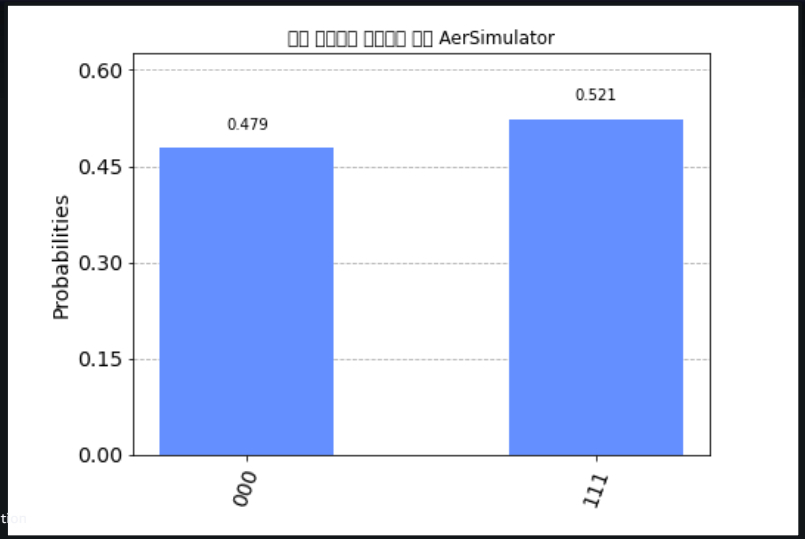
```
from qiskit.provider.aer import AerSimulator
```

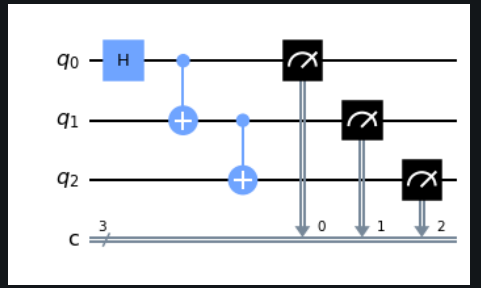


qc

노이즈 없는 Qiskit Aer

```
backend = AerSimulator()
result = backend.run(qc).result()
counts = result.get_counts(qc)
plot_histogram(counts, title="가상 백엔드를 지정하지 않은 AerSimulator"
```





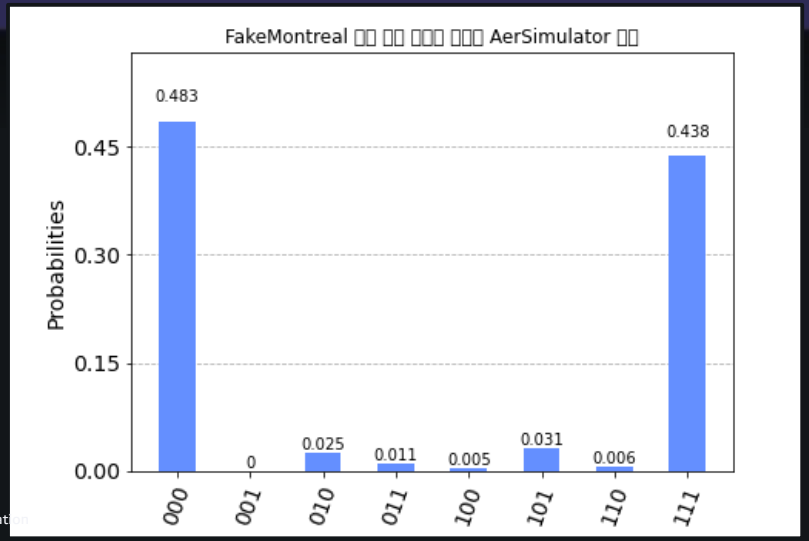
qc

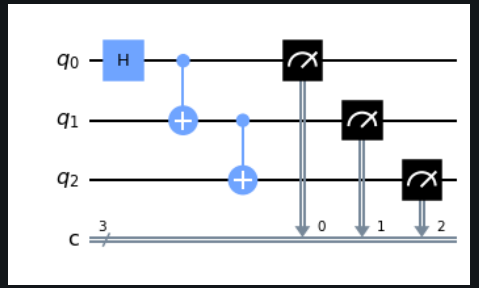
실제 장치 모방한 노이즈 있는 Qiskit Aer

Qiskit

```
from qiskit.test.mock import FakeMontreal
```

```
backend = AerSimulator.from_backend(FakeMontreal())
result = backend.run(qc).result()
counts = result.get_counts(qc)
plot_histogram(counts, title="FakeMontreal 가상 양자 장치를 이용한 AerSimulator 결과")
```





qc

Qiskit Aer 백엔드로 설정하기

```
provider = AerProvider()
backend = provider.backend(name='aer_simulator')
```

[AerSimulator('aer_simulator')]

```
backend = Aer.backends(name='aer_simulator')
```

[AerSimulator('aer_simulator')]

```
backend = AerSimulator()
```

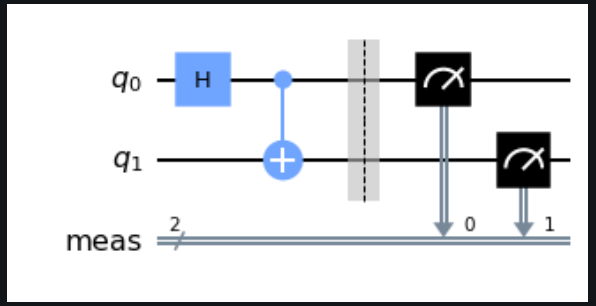
AerSimulator('aer_simulator')

Basic Aer 백엔드

- `qasm simulator`
- `statevector simulator`
- `unitary simulator`

qasm simulator

- 노이즈를 설정하여 만들 수 있음
- 여러 시뮬레이션 방법을 지원함
- 파이썬 dict 형태의 결과값을 출력




```
{'counts': {'0x3': 534, '0x0': 490}}
```

statevector simulator

- 노이즈가 없음
- CPU와 GPU 시뮬레이션 방법을 지원함
- Statevector을 결과값으로 출력

```
{'statevector': Statevector([ ...
}
```

$$\left[\frac{1}{\sqrt{2}} \quad 0 \quad 0 \quad \frac{1}{\sqrt{2}}\right]$$

unitary simulator

- 노이즈가 없음
- CPU와 GPU 시뮬레이션 방법을 지원함
- Operator을 결과값으로 출력

```
{'unitary': Operator([[ ...
}
```

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \end{bmatrix}$$

Basic Aer 백엔드로 설정하기 (qasm_simulator)

```
backend = BasicAer.backends(name='qasm_simulator')
```

```
[<QasmSimulatorPy('qasm_simulator')>]
```

```
from qiskit.providers.basicaer import BasicAerProvider
provider = BasicAerProvider()
backend = provider.backends(name='qasm_simulator')
```

```
[QasmSimulator('qasm_simulator')]
```

```
from qiskit.providers.basicaer import QasmSimulatorPy
backend = QasmSimulatorPy
```

```
<QasmSimulatorPy('qasm_simulator')>
```

실제 양자 장치

```
ibmq_account = IBMQ.load_account()
```

Using ‘get_backend’

```
backend = ibmq_account.get_backend('BACKEND_NAME')
```

Using ‘backend’

```
backend = ibmq_account.backend.BACKEND_NAME
```