Introduction to Quantum Information and Quantum Machine Learning

Project 1

Mateusz Tabaszewski 151945

In [1]:

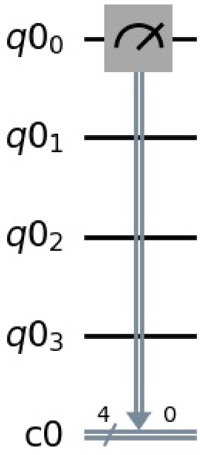
```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit_aer import Aer
from qiskit.compiler import transpile
from qiskit.visualization import *
from numpy import pi
from qiskit.visualization import plot_histogram
from qiskit.transpiler import generate_preset_pass_manager

from qiskit.visualization import plot_state_city, plot_bloch_multivector
from qiskit.visualization import plot_state_paulivec, plot_state_hinton
from qiskit.visualization import plot_state_qsphere
```

Task 1

In [2]:

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

nx=4
shots=2048
qx = QuantumRegister(nx)
cx = ClassicalRegister(nx)
circuitX = QuantumCircuit(qx, cx)
circuitX.measure(qx[0], cx[0])

results = []
for i in range(3):
    job_result = backend.run(transpile(circuitX, backend), shots=shots).result()
    results.append(job_result)

circuitX.draw(output="mpl")
```

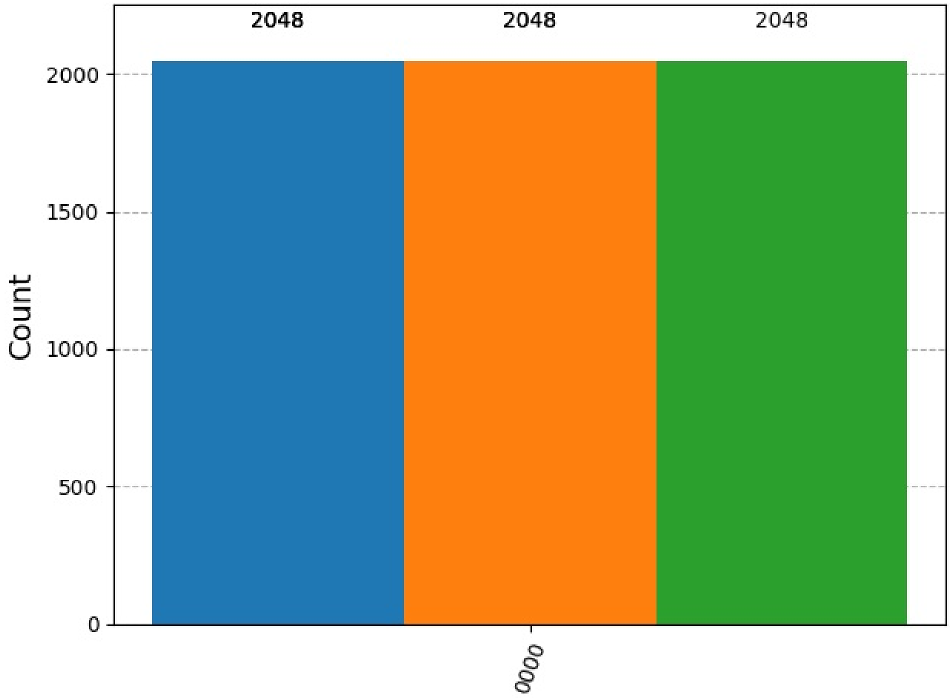
Out[2]:



In [3]:

```
counts_list = []
for job_result in results:
    counts = job_result.get_counts(circuitX)
    counts_list.append(counts)
plot_histogram(counts_list)
```

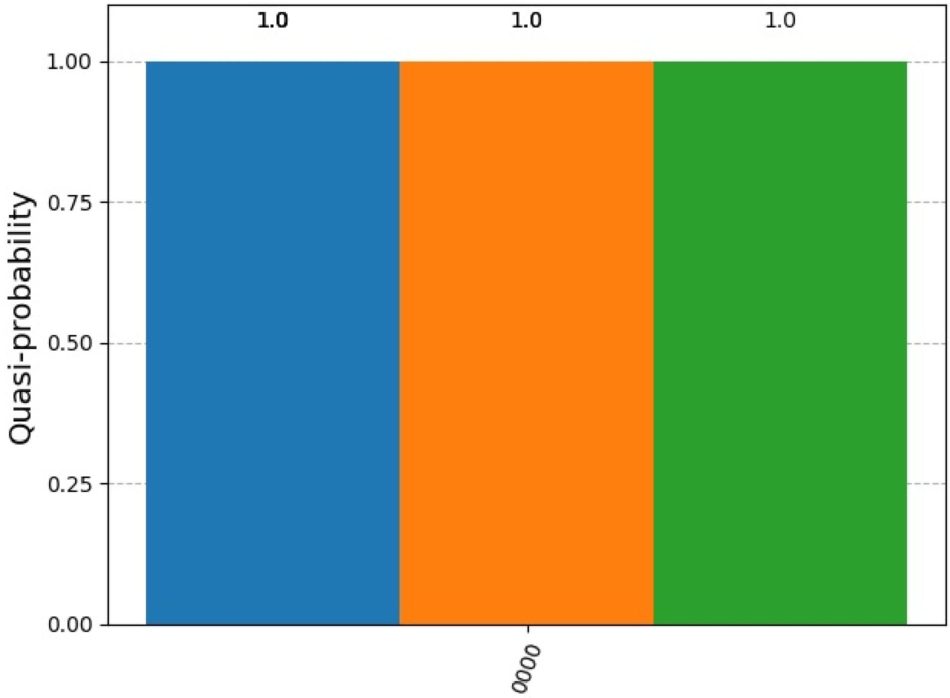
Out[3]:



In [4]:

```
probs_list = []
for counts in counts_list:
    shots = sum(counts.values())
    probs = {state: c / shots for state, c in counts.items()}
    probs_list.append(probs)
plot_histogram(probs_list)
```

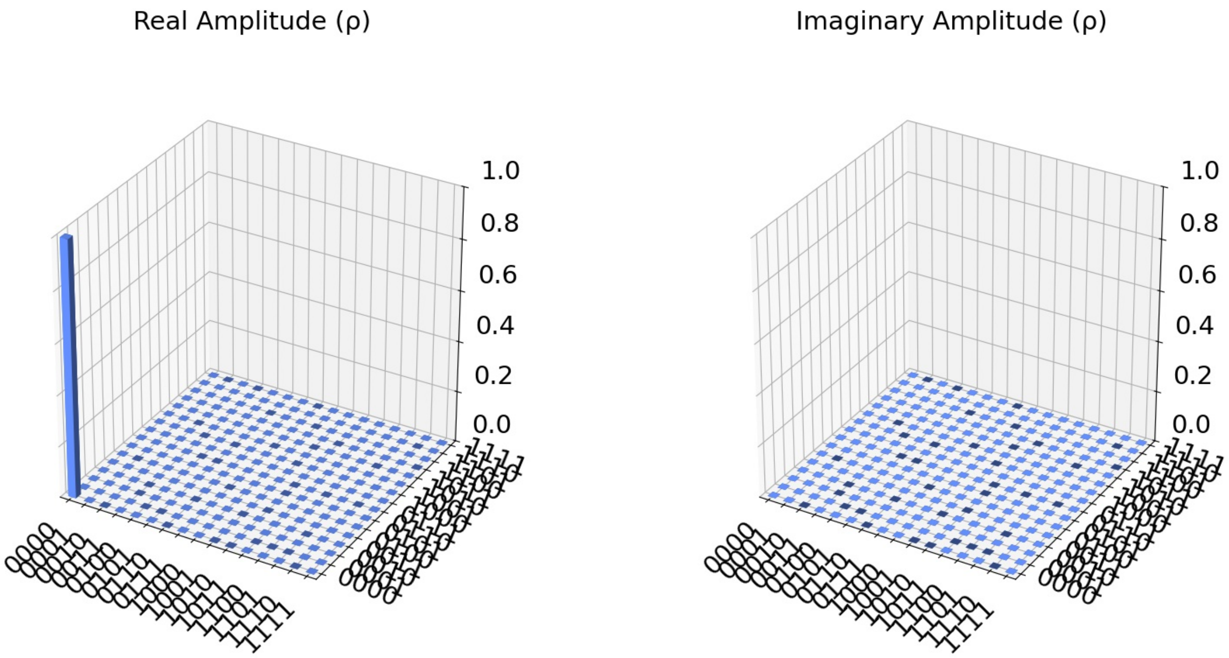
Out[4]:



In [5]:

```
psi = job_result.get_statevector(circuitX)
plot_state_city(psi)
```

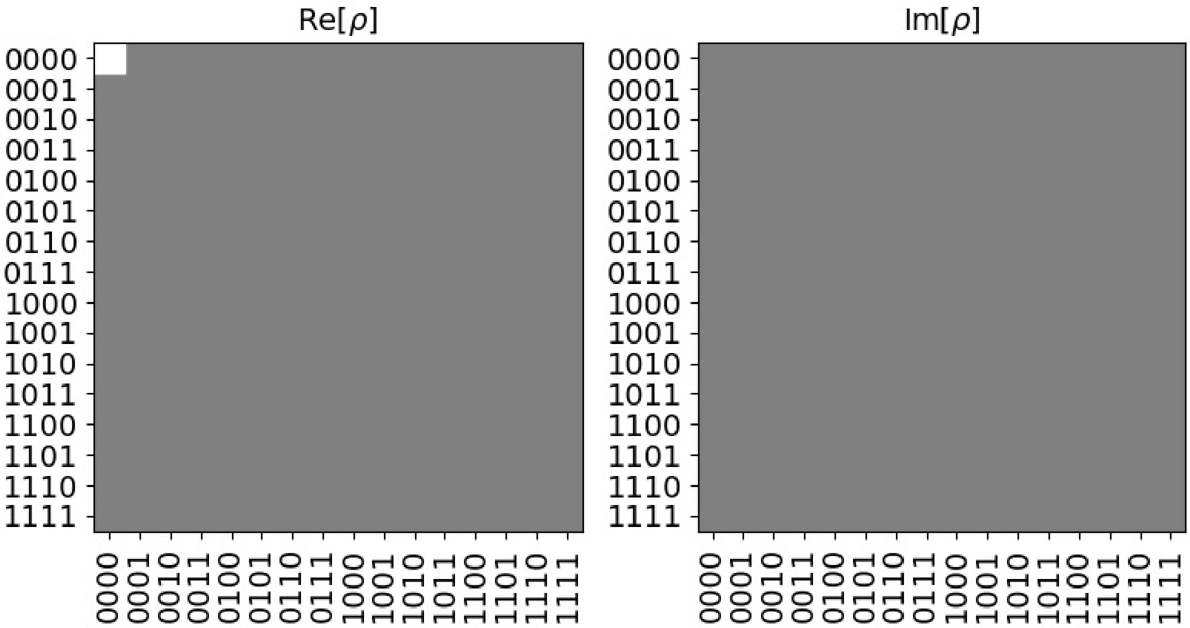
Out[5]:



In [6]:

```
plot_state_hinton(psi)
```

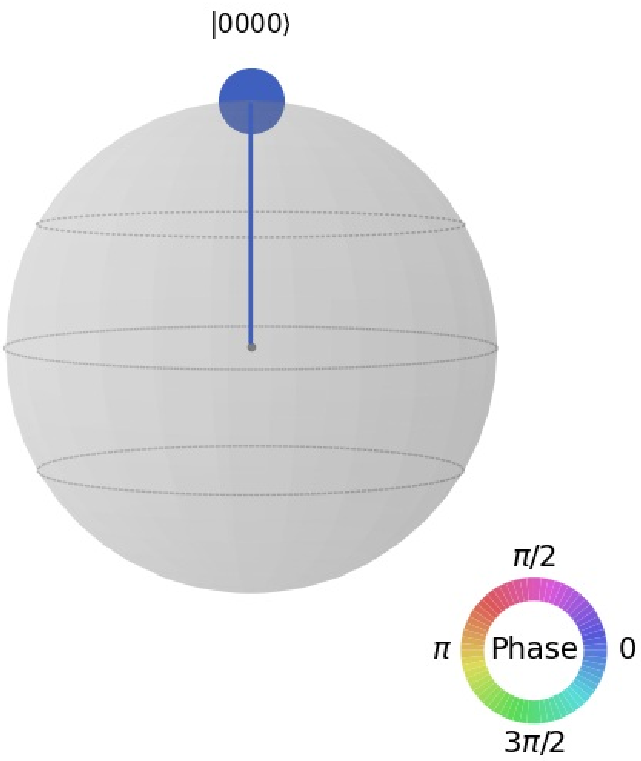
Out[6]:



In [7]:

```
plot_state_qsphere(psi)
```

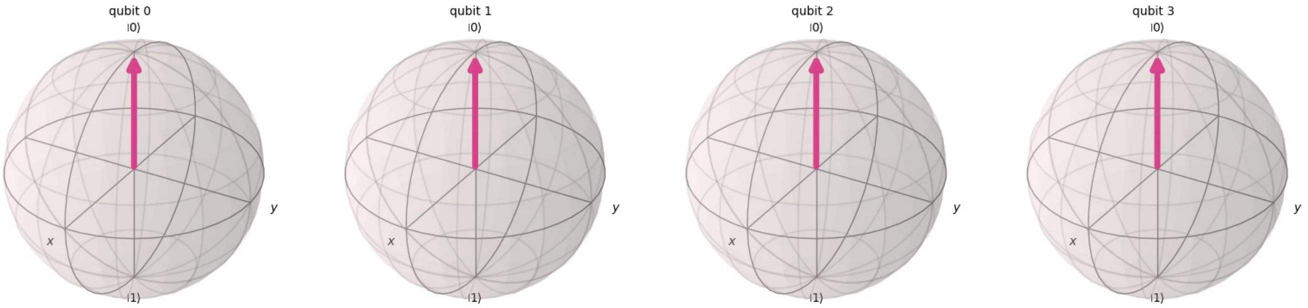
Out[7]:



In [8]:

```
plot_bloch_multivector(psi)
```

Out[8]:

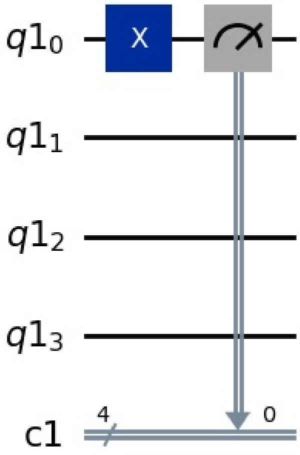


Task 2

In [9]:

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

nx=4
shots=2048
qx = QuantumRegister(nx)
cx = ClassicalRegister(nx)
circuitX = QuantumCircuit(qx, cx)
circuitX.x(qx[0])
circuitX.measure(qx[0], cx[0])

results = []
for i in range(3):
    job_result = backend.run(transpile(circuitX, backend), shots=shots).result()
    results.append(job_result)

circuitX.draw(output="mpl")
```

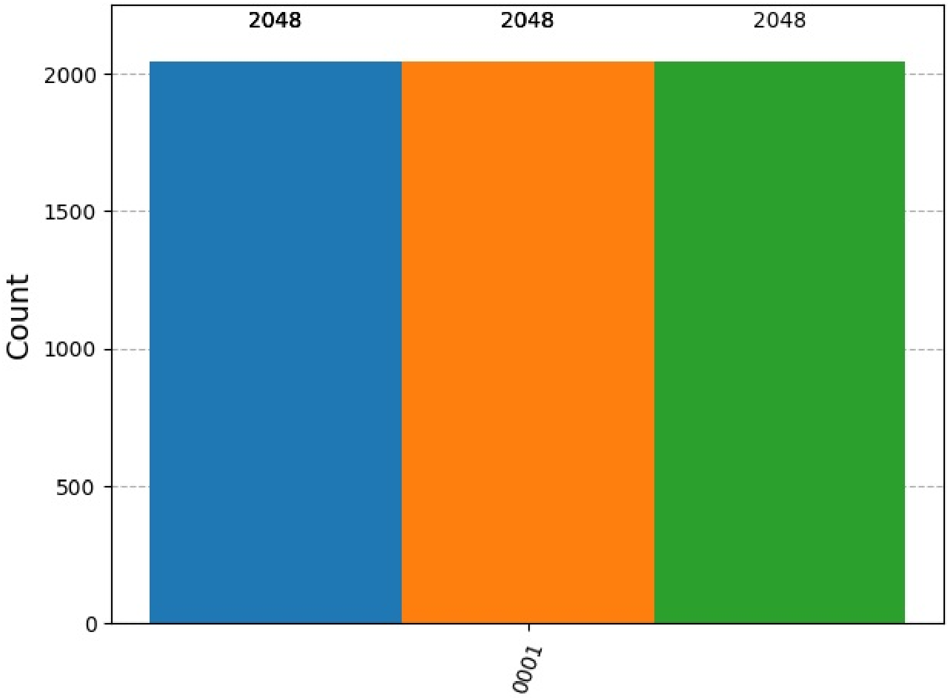
Out[9]:



In [10]:

```
counts_list = []
for job_result in results:
    counts = job_result.get_counts(circuitX)
    counts_list.append(counts)
plot_histogram(counts_list)
```

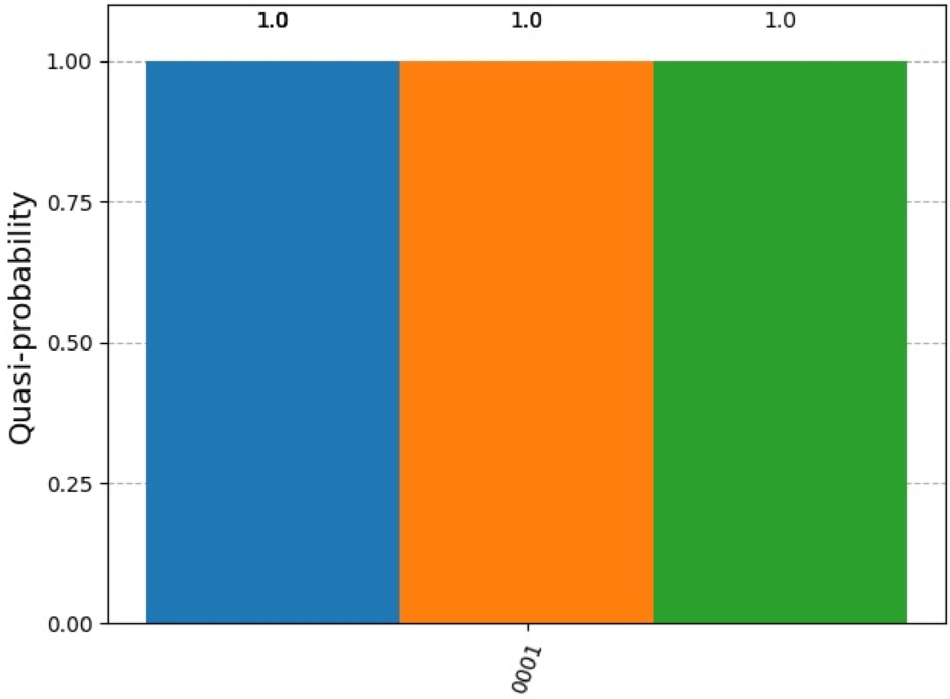
Out[10]:



In [11]:

```
probs_list = []
for counts in counts_list:
    shots = sum(counts.values())
    probs = {state: c / shots for state, c in counts.items()}
    probs_list.append(probs)
plot_histogram(probs_list)
```

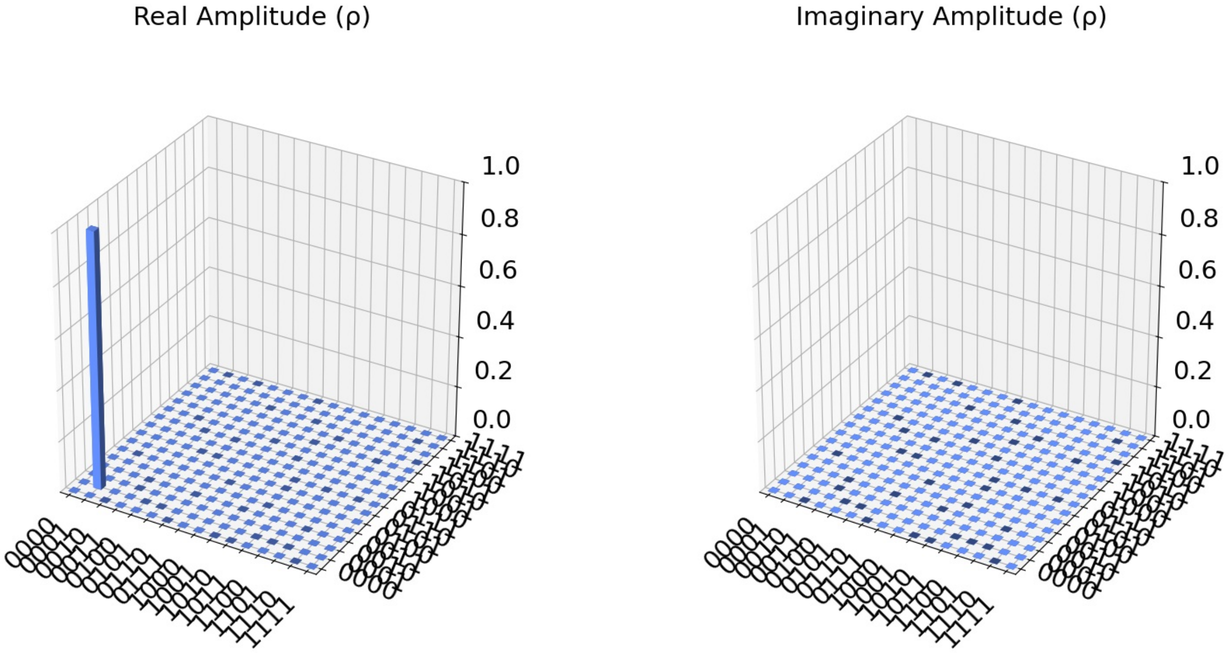
Out[11]:



In [12]:

```
psi = job_result.get_statevector(circuitX)
plot_state_city(psi)
```

Out[12]:

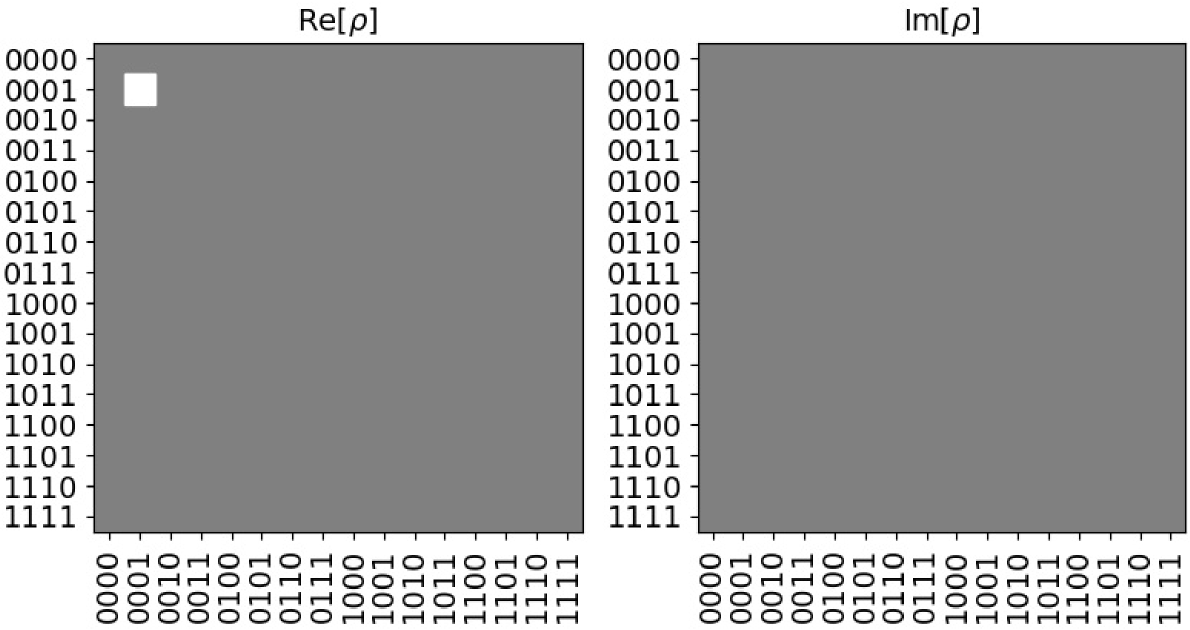

Real Amplitude (ρ)

Imaginary Amplitude (ρ)

In [13]:

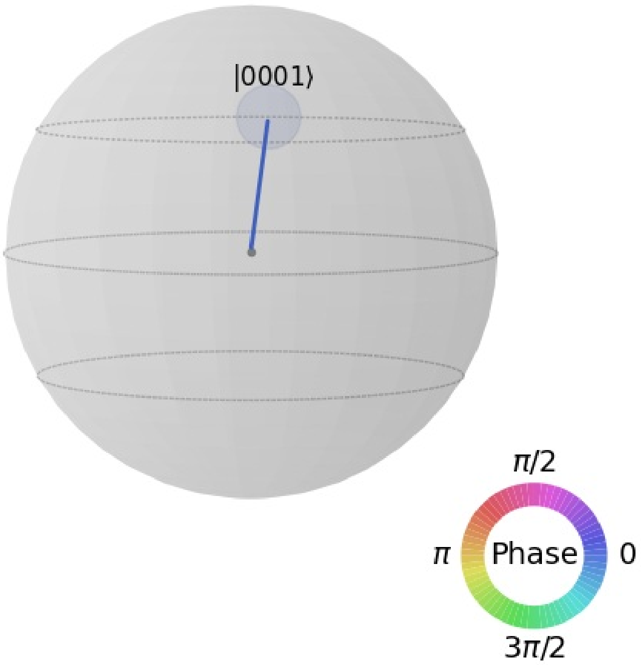
```
plot_state_hinton(psi)
```

Out[13]:



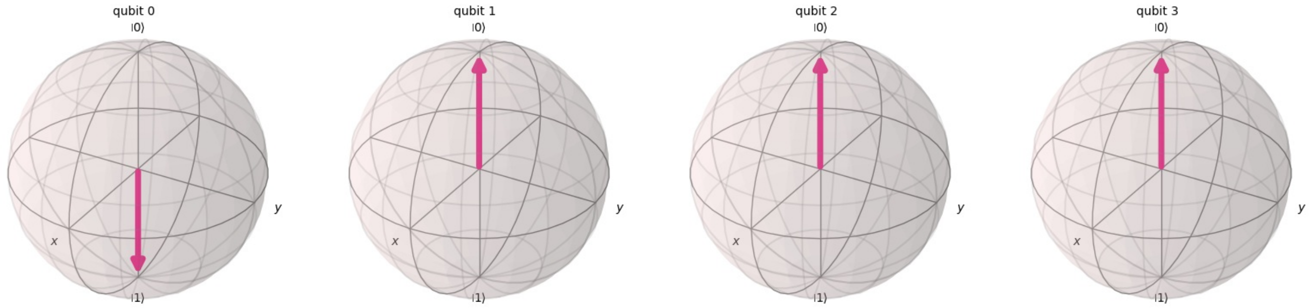
In [14]:
```
plot_state_qsphere(psi)
```

Out[14]:



In [15]:
```
plot_bloch_multivector(psi)
```

Out[15]:

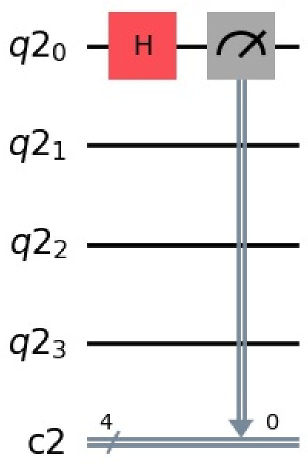


Task 3

In [16]:
```
backend = Aer.get_backend('statevector_simulator')

nx=4
```

```
shots=2048
qx = QuantumRegister(nx)
cx = ClassicalRegister(nx)
circuitX = QuantumCircuit(qx, cx)
circuitX.h(qx[0])
circuitX.measure(qx[0], cx[0])

results = []
for i in range(3):
    job_result = backend.run(transpile(circuitX, backend), shots=shots).result()
    results.append(job_result)

circuitX.draw(output="mpl")
```

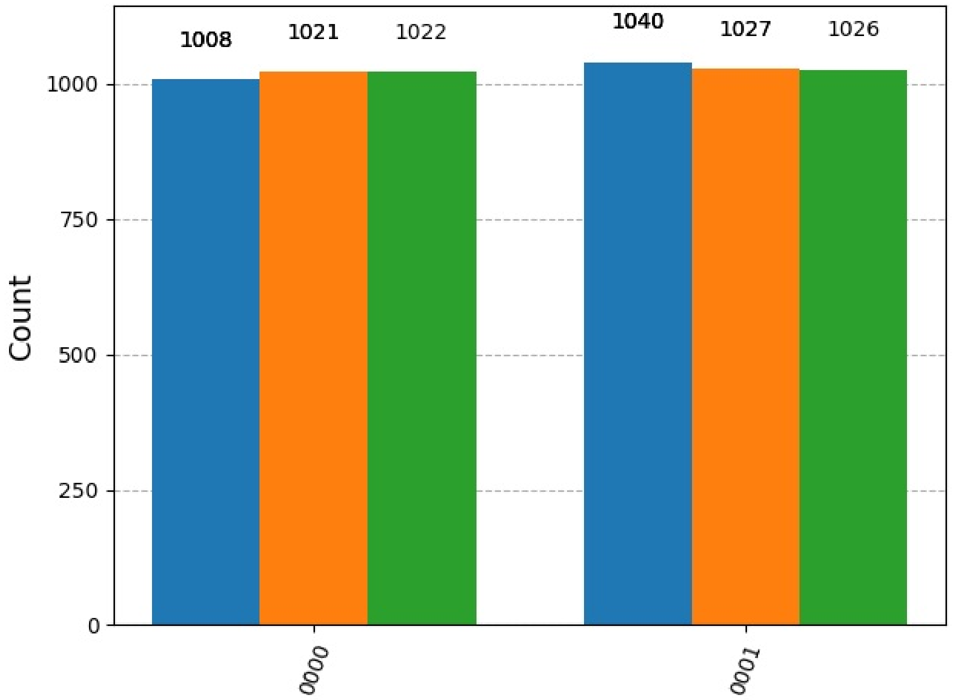
Out[16]:



In [17]:

```
counts_list = []
for job_result in results:
    counts = job_result.get_counts(circuitX)
    counts_list.append(counts)
plot_histogram(counts_list)
```

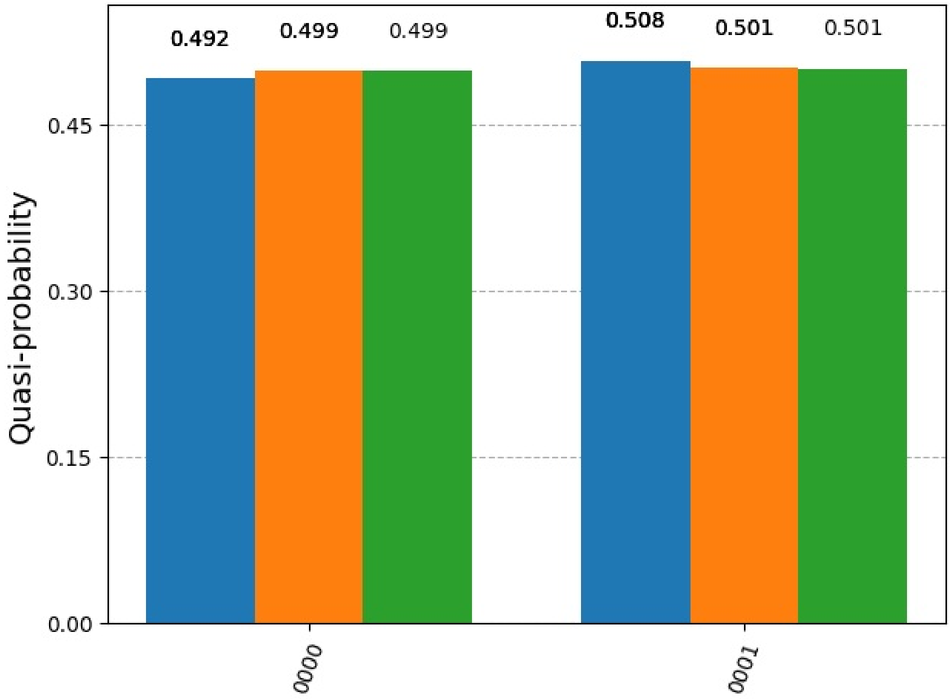
Out[17]:



In [18]:

```
probs_list = []
for counts in counts_list:
    shots = sum(counts.values())
    probs = {state: c / shots for state, c in counts.items()}
    probs_list.append(probs)
plot_histogram(probs_list)
```

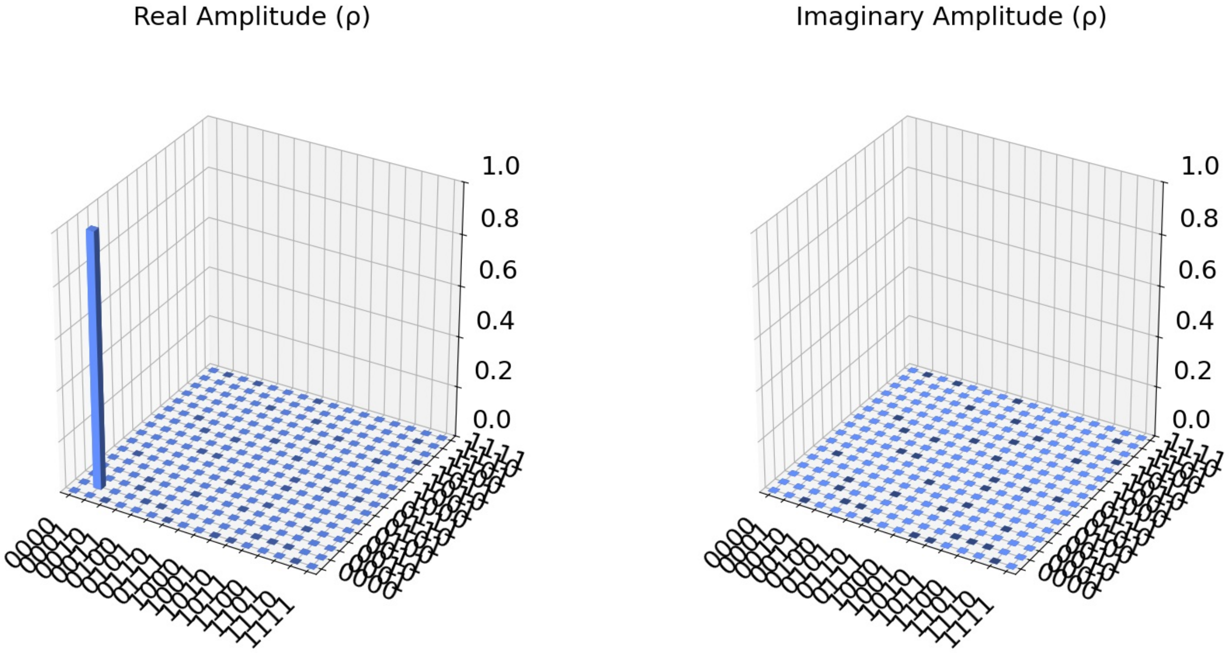
Out[18]:



In [19]:

```
psi = job_result.get_statevector(circuitX)
plot_state_city(psi)
```

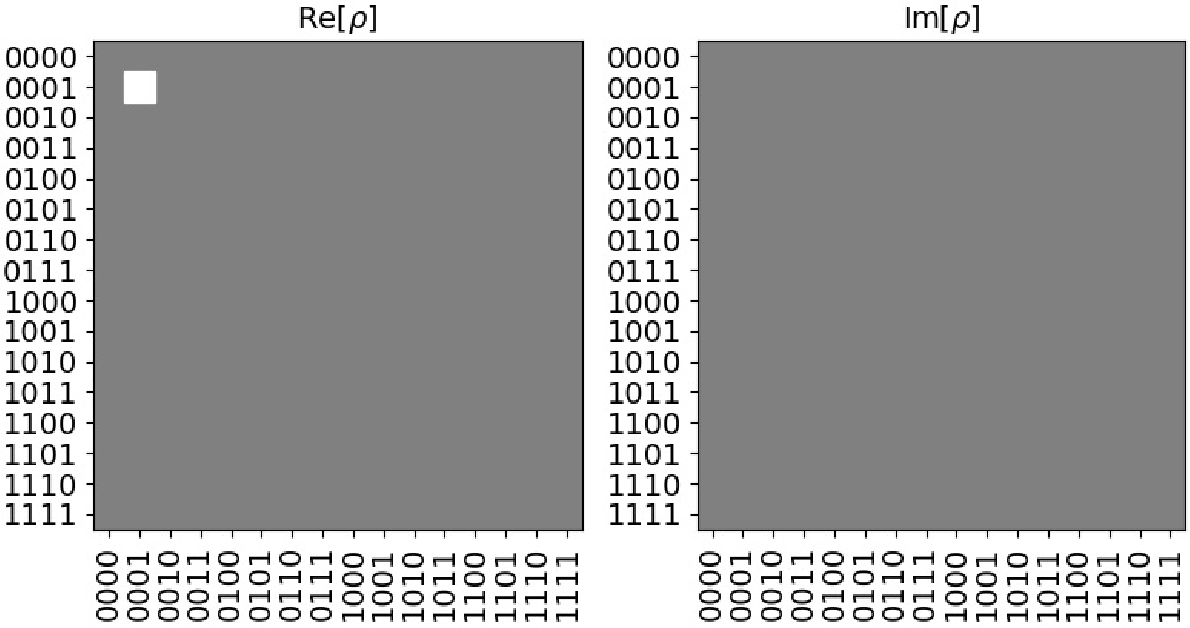
Out[19]:



In [20]:

```
plot_state_hinton(psi)
```

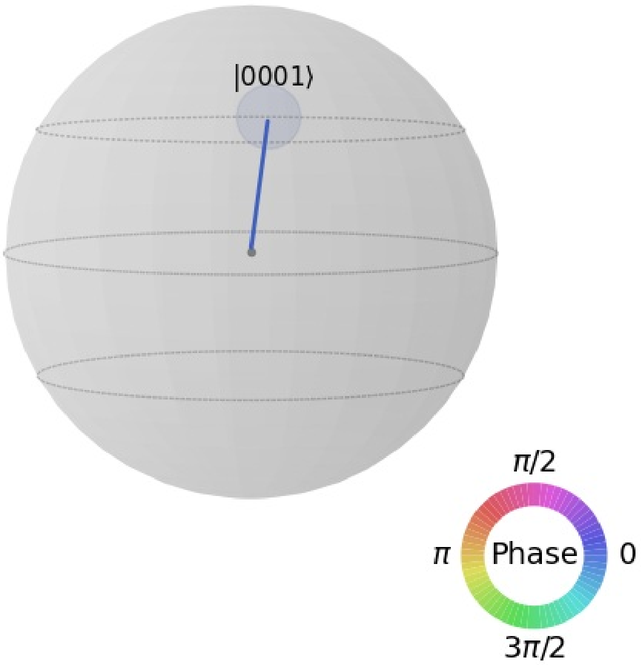
Out[20]:



In [21]:

```
plot_state_qsphere(psi)
```

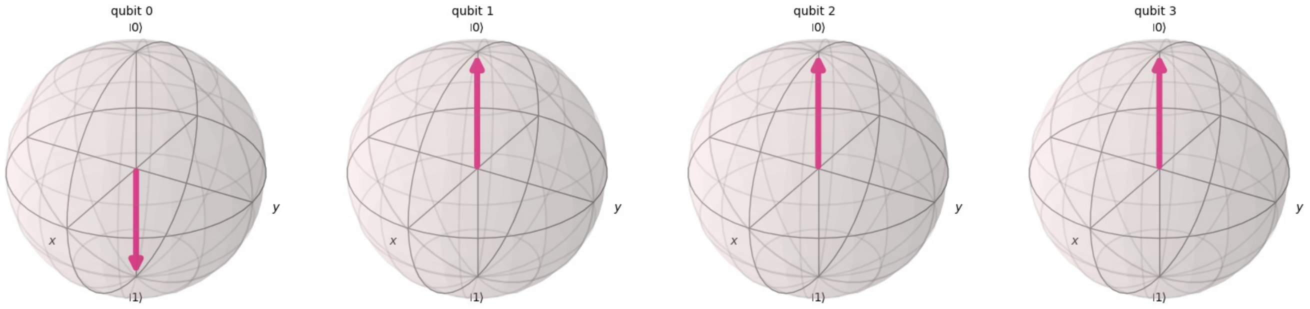
Out[21]:



In [22]:

```
plot_bloch_multivector(psi)
```

Out[22]:



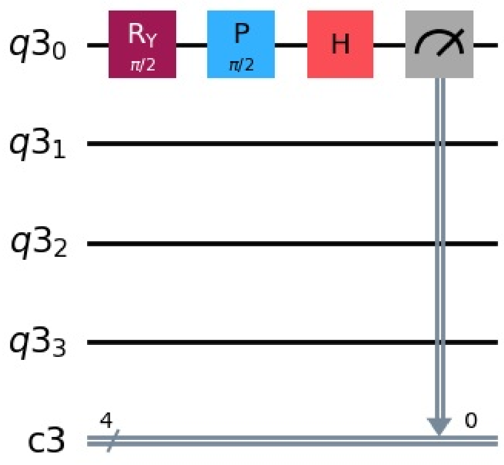
Task 4 - X Base

In [23]:

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

nx=4
```

```
shots=2048
qx = QuantumRegister(nx)
cx = ClassicalRegister(nx)
circuitX = QuantumCircuit(qx, cx)
circuitX.ry(pi / 2, qx[0])
circuitX.p(pi / 2, qx[0])
circuitX.h(qx[0])
circuitX.measure(qx[0], cx[0])

results = []
for i in range(3):
    job_result = backend.run(transpile(circuitX, backend), shots=shots).result()
    results.append(job_result)

circuitX.draw(output="mpl")
```

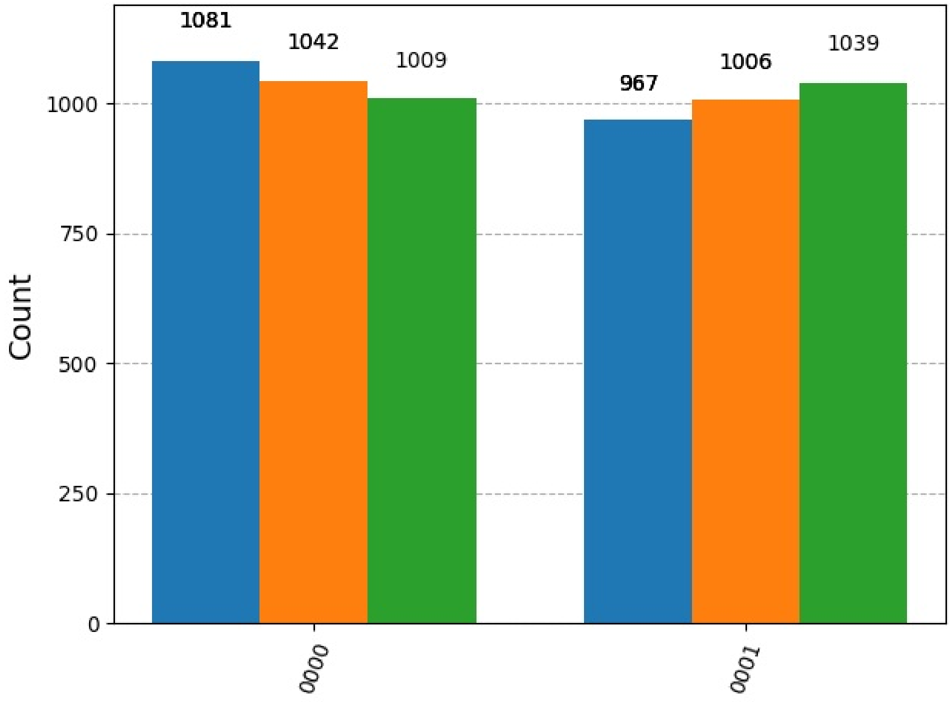
Out[23]:



In [24]:

```
counts_list = []
for job_result in results:
    counts = job_result.get_counts(circuitX)
    counts_list.append(counts)
plot_histogram(counts_list)
```

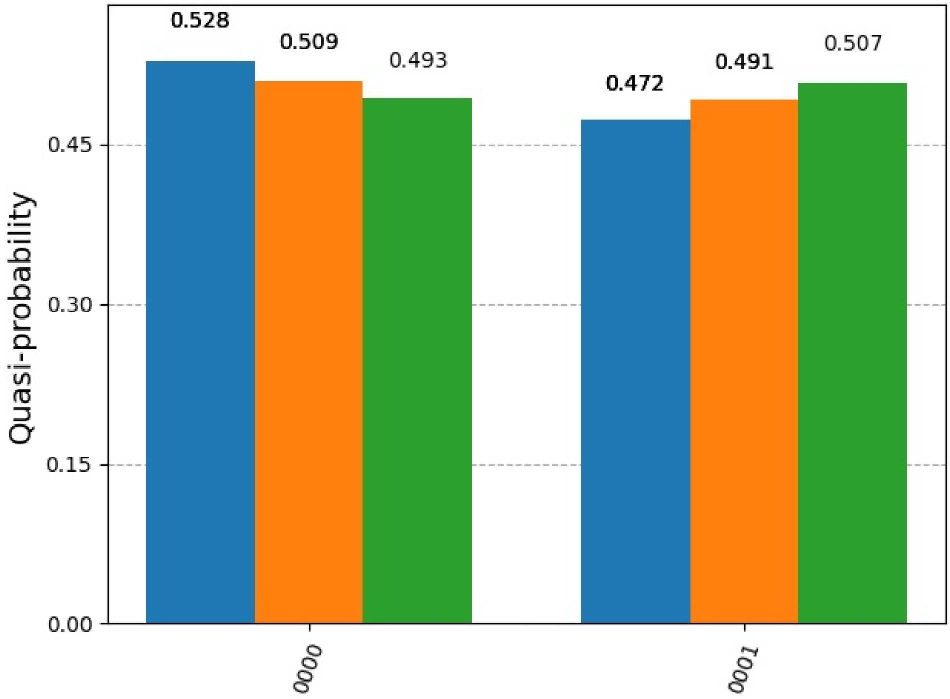
Out[24]:



In [25]:

```
probs_list = []
for counts in counts_list:
    shots = sum(counts.values())
    probs = {state: c / shots for state, c in counts.items()}
    probs_list.append(probs)
plot_histogram(probs_list)
```

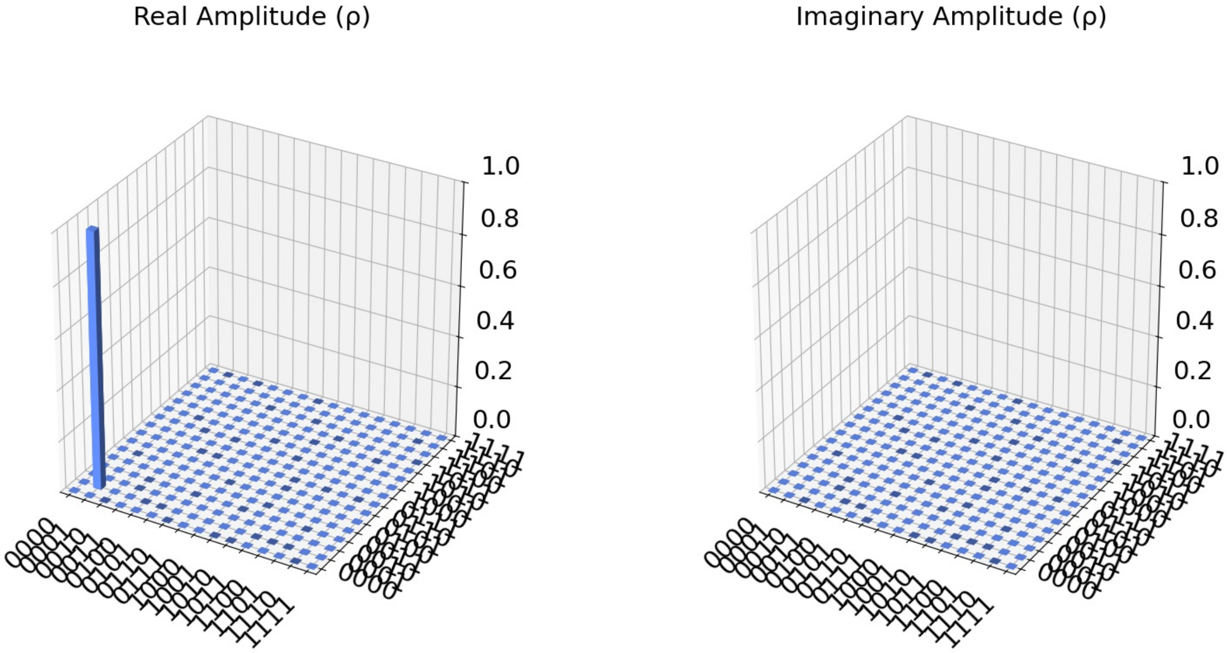
Out[25]:



In [26]:

```
psi = job_result.get_statevector(circuitX)
plot_state_city(psi)
```

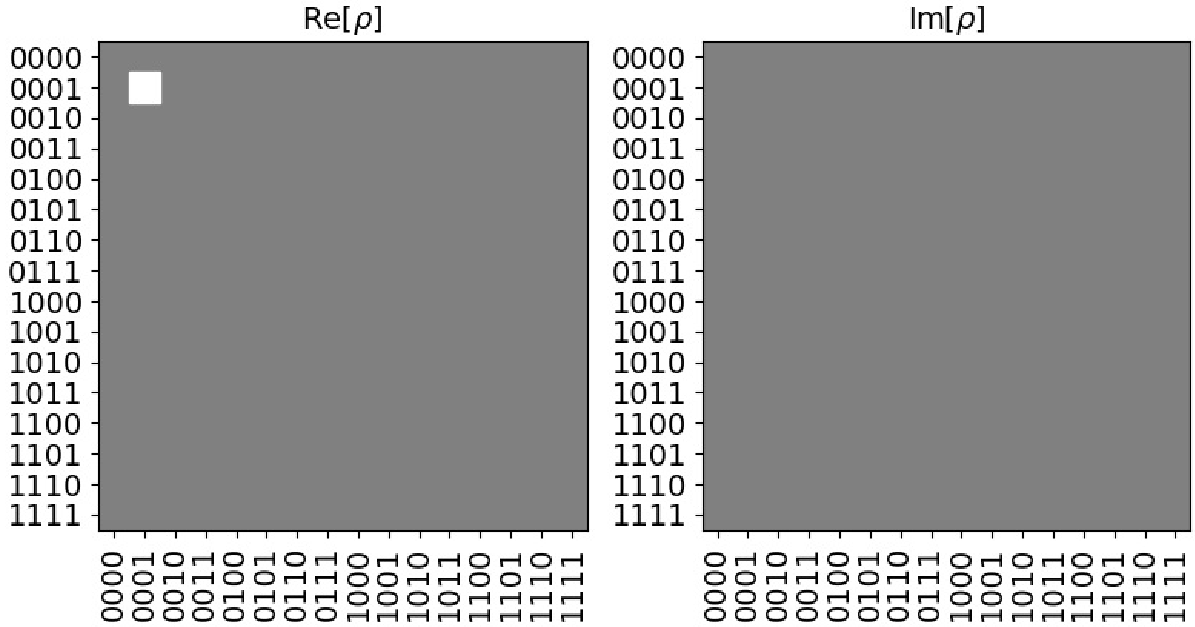
Out[26]:



In [27]:

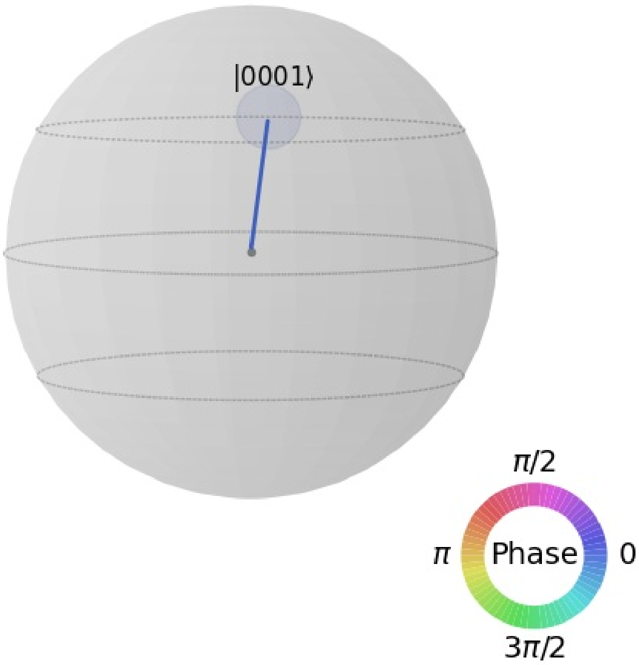
```
plot_state_hinton(psi)
```

```
Out[27]:
```



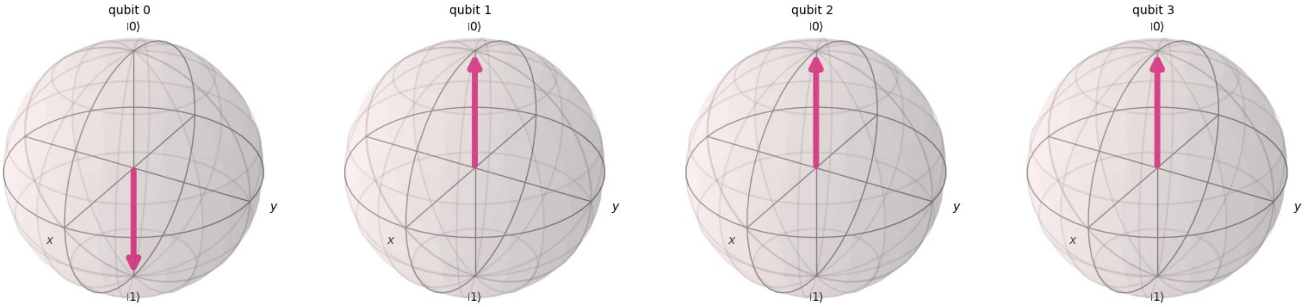
```
In [28]: plot_state_qsphere(psi)
```

```
Out[28]:
```



```
In [29]: plot_bloch_multivector(psi)
```

```
Out[29]:
```

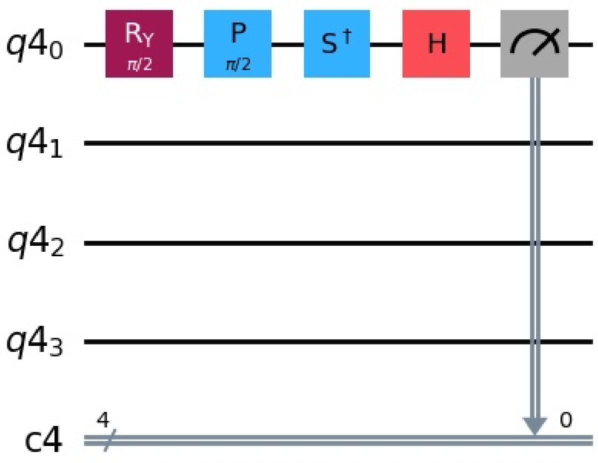


Task 4 - Y Base

```
In [30]: backend = Aer.get_backend('statevector_simulator')

nx=4
```

```
shots=2048
qx = QuantumRegister(nx)
cx = ClassicalRegister(nx)
circuitX = QuantumCircuit(qx, cx)
circuitX.ry(pi / 2, qx[0])
circuitX.p(pi / 2, qx[0])
circuitX.sdg(qx[0])
circuitX.h(qx[0])
circuitX.measure(qx[0], cx[0])

results = []
for i in range(3):
    job_result = backend.run(transpile(circuitX, backend), shots=shots).result()
    results.append(job_result)

circuitX.draw(output="mpl")
```

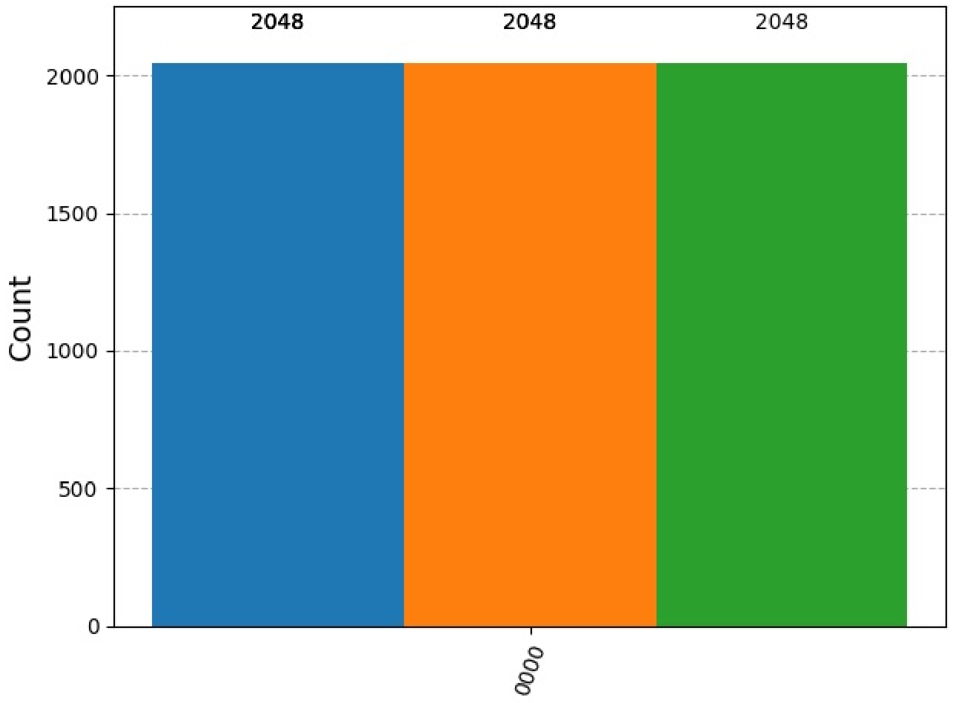
Out[30]:



In [31]:

```
counts_list = []
for job_result in results:
    counts = job_result.get_counts(circuitX)
    counts_list.append(counts)
plot_histogram(counts_list)
```

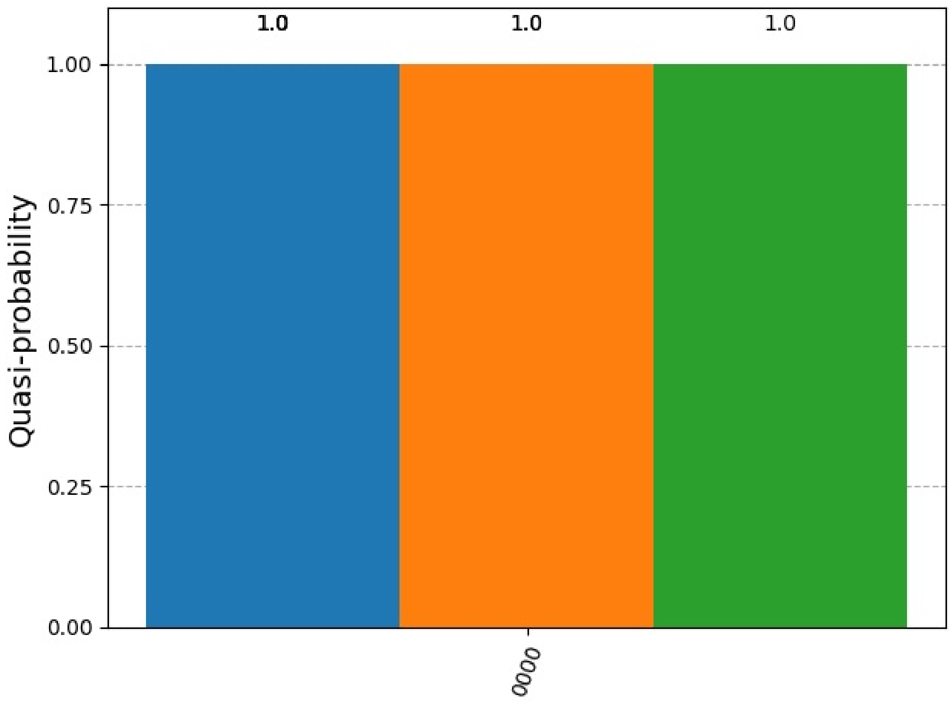
Out[31]:



In [32]:

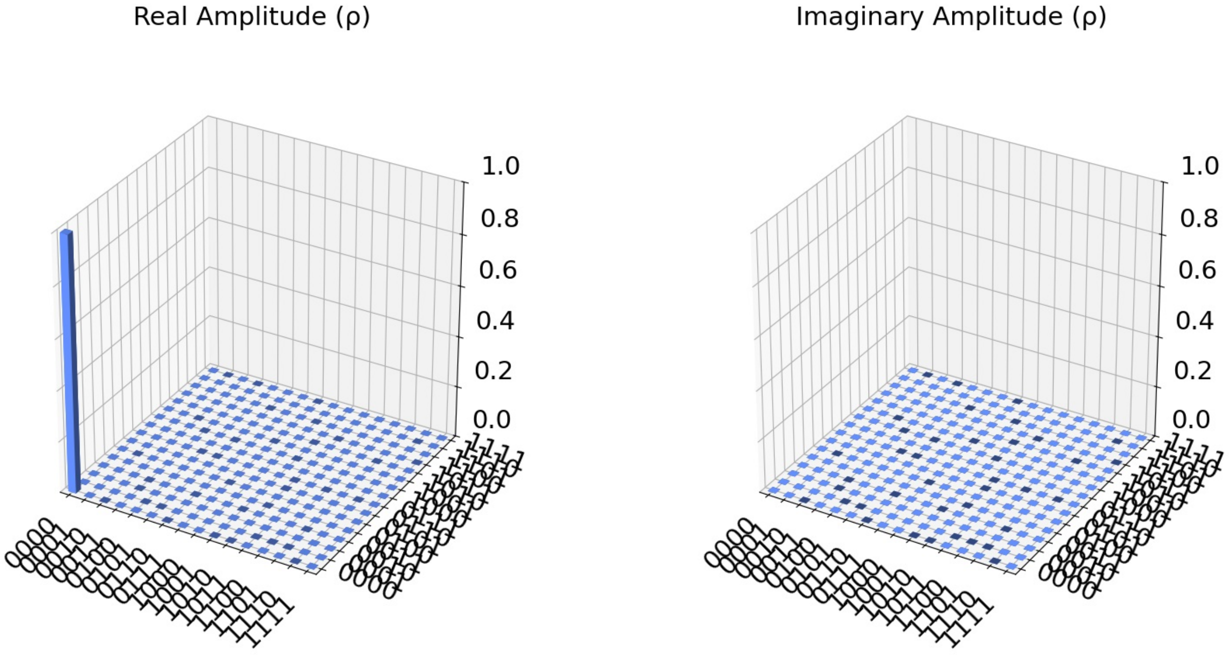
```
probs_list = []
for counts in counts_list:
    shots = sum(counts.values())
    probs = {state: c / shots for state, c in counts.items()}
    probs_list.append(probs)
plot_histogram(probs_list)
```

Out[32]:



In [33]:
```
psi = job_result.get_statevector(circuitX)
plot_state_city(psi)
```

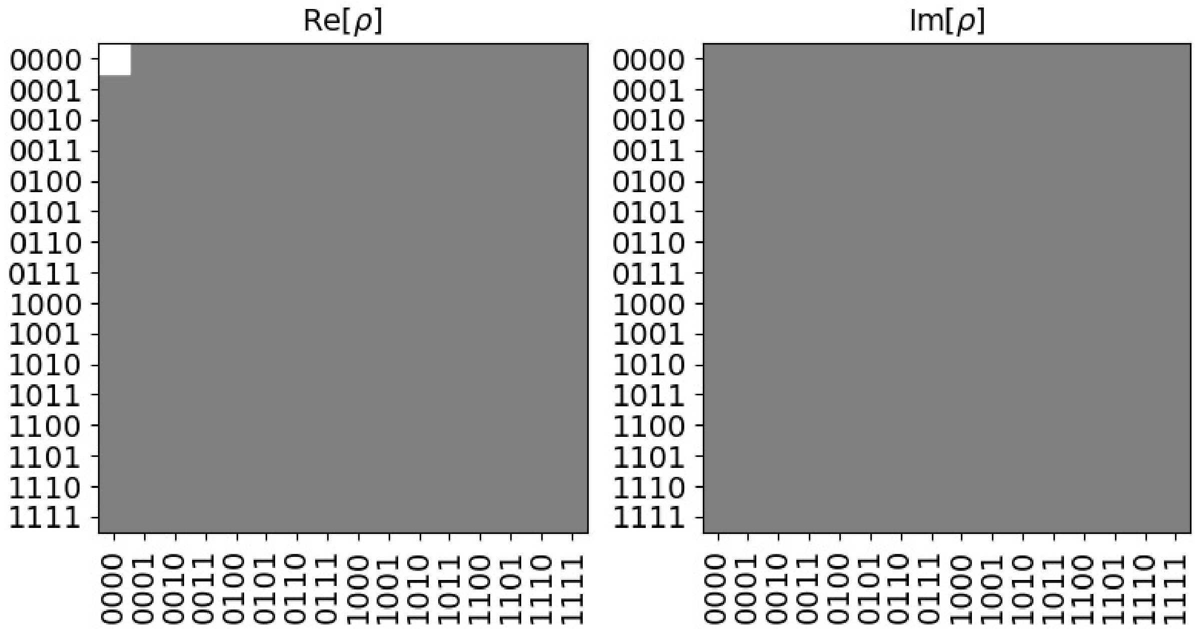
Out[33]:


Real Amplitude (ρ)

Imaginary Amplitude (ρ)

In [34]:
```
plot_state_hinton(psi)
```

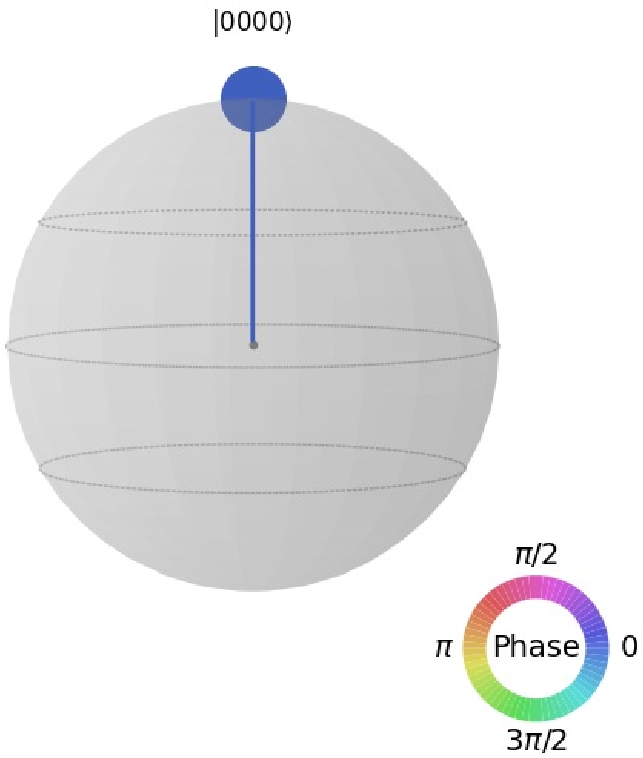
Out[34]:



In [35]:

```
plot_state_qsphere(psi)
```

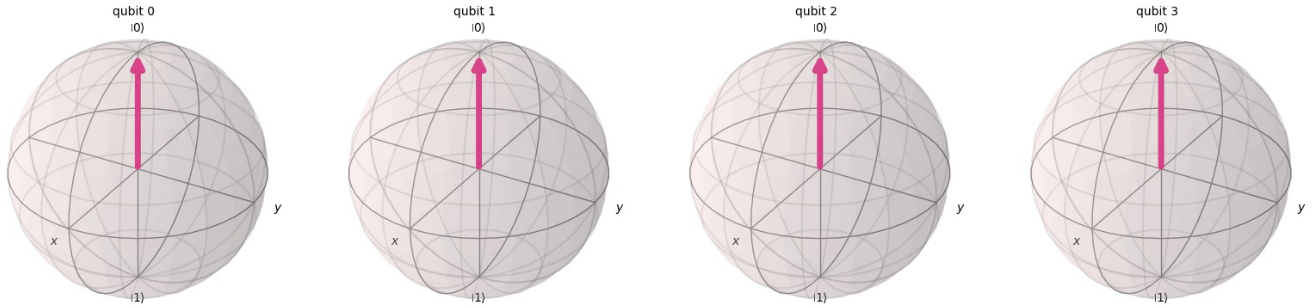
Out[35]:



In [36]:

```
plot_bloch_multivector(psi)
```

Out[36]:



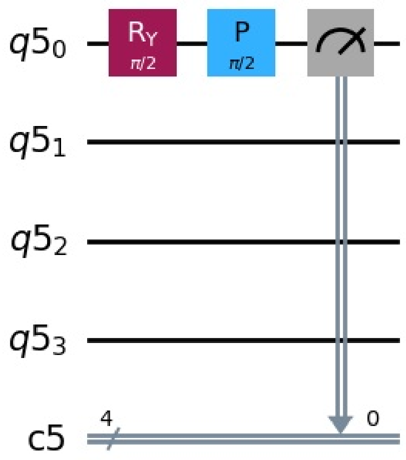
Task 4 - Z Base

In [37]:

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

nx=4
```

```
shots=2048
qx = QuantumRegister(nx)
cx = ClassicalRegister(nx)
circuitX = QuantumCircuit(qx, cx)
circuitX.ry(pi / 2, qx[0])
circuitX.p(pi / 2, qx[0])
circuitX.measure(qx[0], cx[0])

results = []
for i in range(3):
    job_result = backend.run(transpile(circuitX, backend), shots=shots).result()
    results.append(job_result)

circuitX.draw(output="mpl")
```

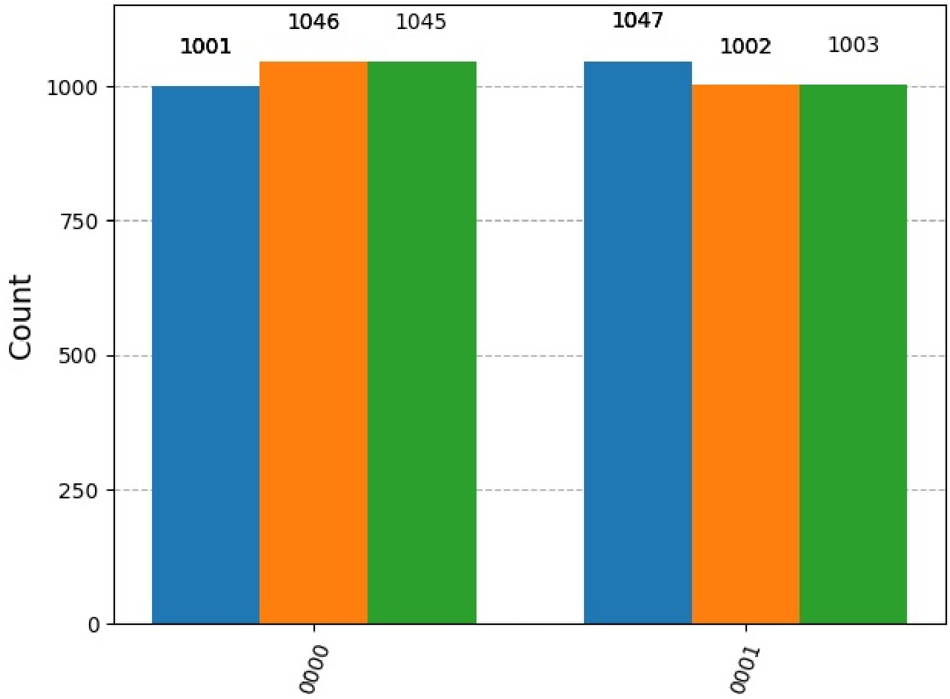
Out[37]:



In [38]:

```
counts_list = []
for job_result in results:
    counts = job_result.get_counts(circuitX)
    counts_list.append(counts)
plot_histogram(counts_list)
```

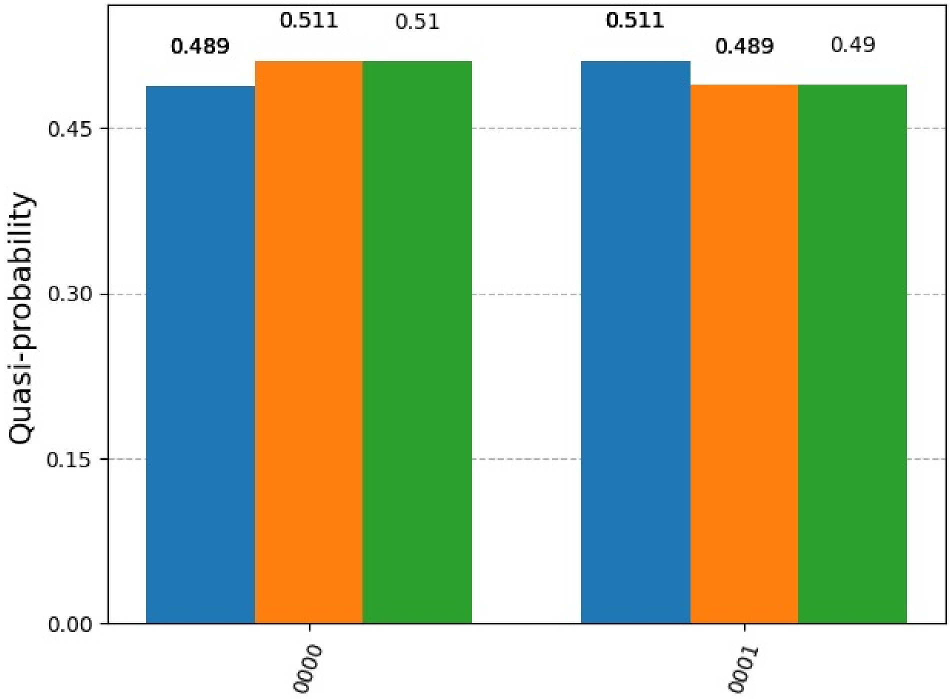
Out[38]:



In [39]:

```
probs_list = []
for counts in counts_list:
    shots = sum(counts.values())
    probs = {state: c / shots for state, c in counts.items()}
    probs_list.append(probs)
plot_histogram(probs_list)
```

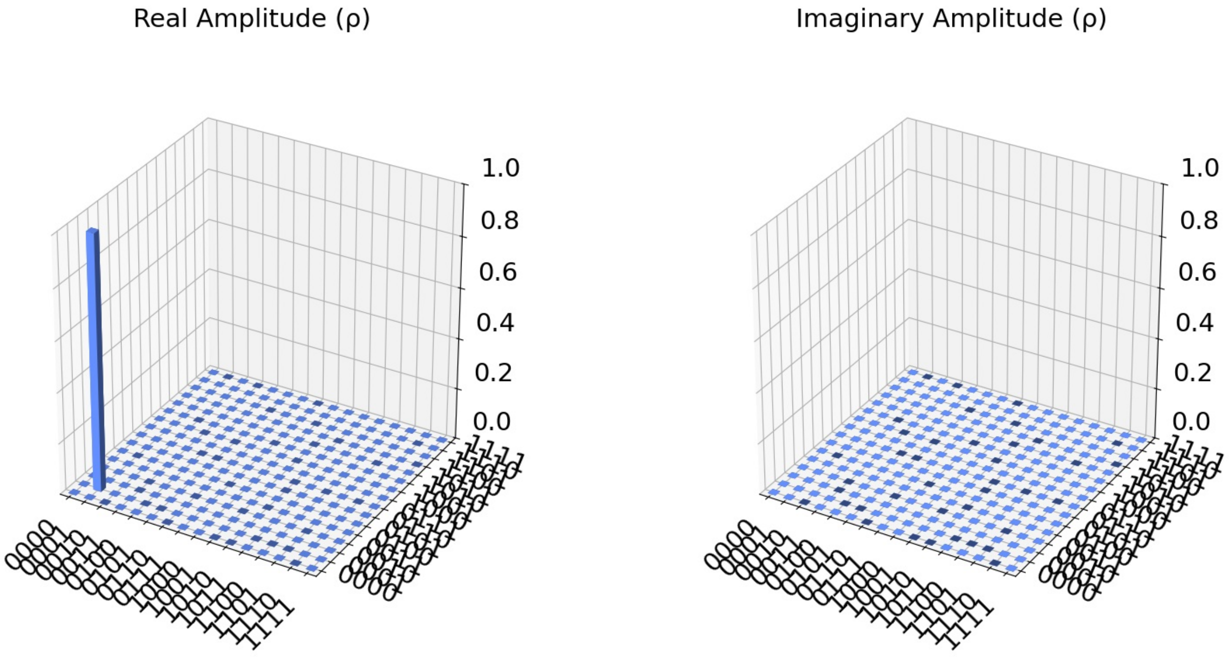
Out[39]:



In [40]:

```
psi = job_result.get_statevector(circuitX)
plot_state_city(psi)
```

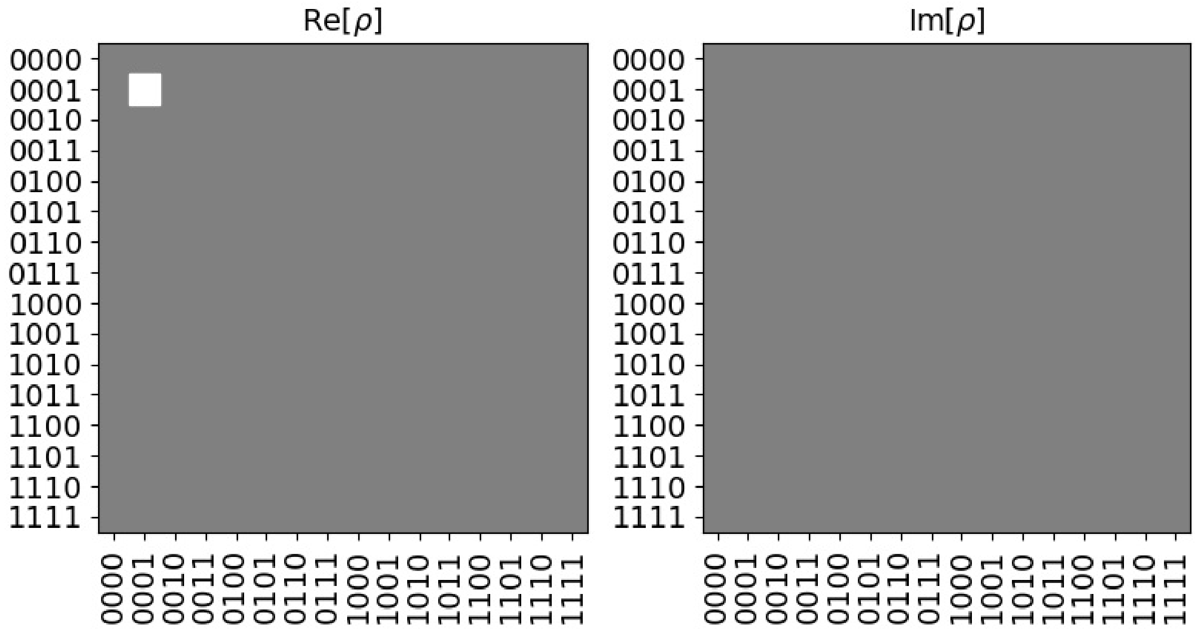
Out[40]:



In [41]:

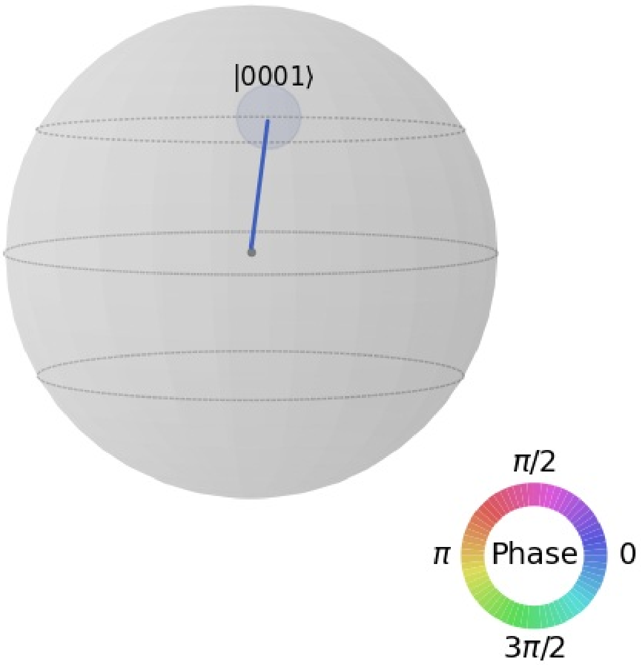
```
plot_state_hinton(psi)
```

Out[41]:



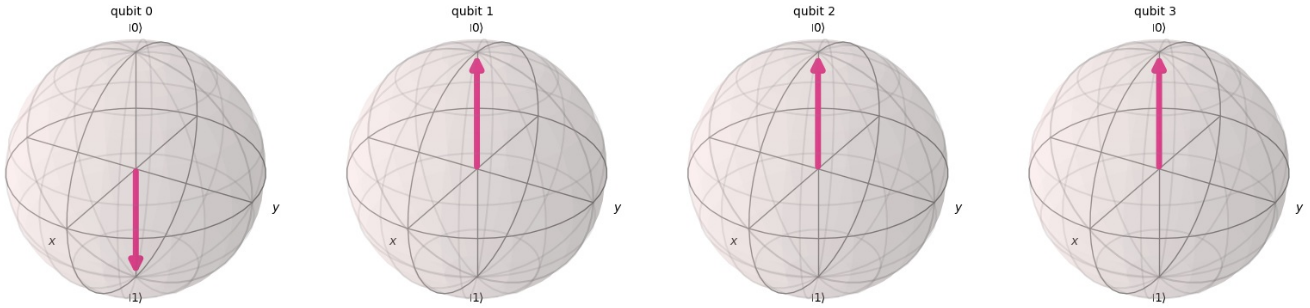
In [42]:
```
plot_state_qsphere(psi)
```

Out[42]:



In [43]:
```
plot_bloch_multivector(psi)
```

Out[43]:


qubit 0
qubit 1
qubit 2
qubit 3
|0⟩
|1⟩
x
y