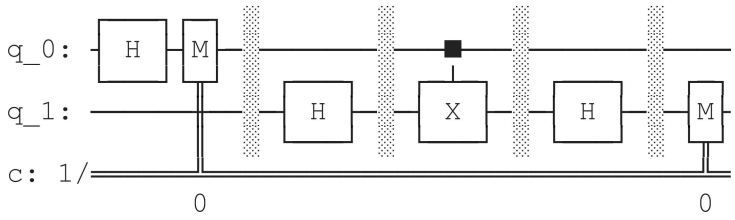
Import Important Libraries

In [2]:
```
import qiskit as q
import qiskit.visualization as qv
from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, transpile, assemble
from qiskit.tools.monitor import job_monitor
import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")
```

Making the Desired Quantum Circuit

In [3]:
```
circuit = QuantumCircuit(2,1)
circuit.h(0); circuit.measure(0,0); circuit.barrier() # Creating and Ensamble of 50
circuit.h(1); circuit.barrier()  # QC Creates a superposition of [1/Sqrt(2)]{ |0> +
circuit.cx(0,1);circuit.barrier() # Randomly decide whether to later flip or not
circuit.h(1);circuit.barrier() # QC again decides whether to flip or not
circuit.measure(1,0) # Measure the Final Result
circuit.draw()
```

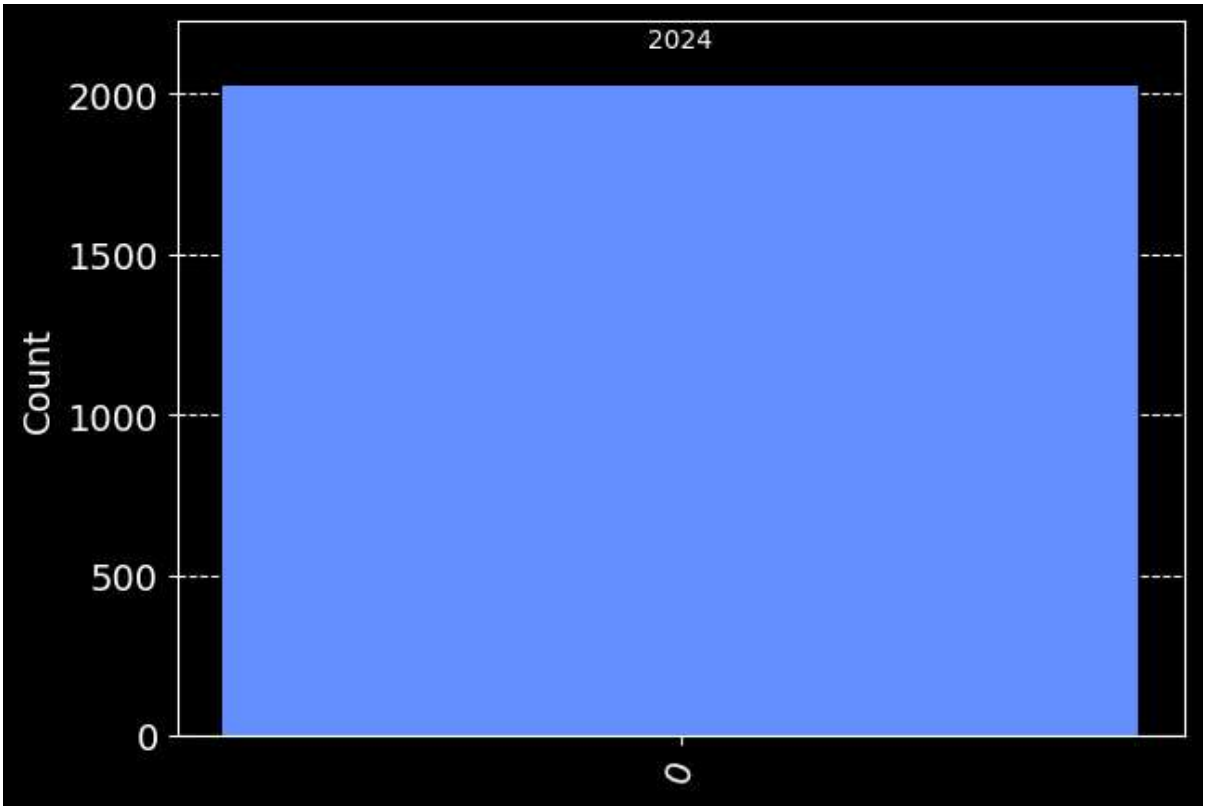
Out[3]:



Running the Circuit via QASM Simulator and checking Results (More the shots, better the probability distribution)

In [4]:
```
qasm_sim = Aer.get_backend("qasm_simulator") # Qiskit simulator backend
result = q.execute(circuit, qasm_sim, shots=2024).result()  # Results
counts = result.get_counts()
qv.plot_histogram(counts) #Visualizing o/p of simulator
```

Out[4]:



Loading your account and Running the Circuit via Least Busy Quantum Computer and checking Results

In [4]:
```
# SAVING AND LOADING YOUR IBM API TOKEN YOU GET WHEN YOU SIGN UP AT IBM QUANTUM COM
IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e975e5
IBMQ.load_account()
provider = IBMQ.get_provider(hub = 'ibm-q')
backend = least_busy(provider.backends(filters = lambda b: b.configuration().n_qubi
```

```
C:\Users\Sanket Lalwani\AppData\Local\Temp\ipykernel_3756\4250883806.py:2: Depreca
tionWarning: The qiskit.IBMQ entrypoint and the qiskit-ibmq-provider package (acce
ssible from 'qiskit.providers.ibmq`) are deprecated and will be removed in a futur
e release. Instead you should use the qiskit-ibm-provider package which is accessi
ble from 'qiskit_ibm_provider'. You can install it with 'pip install qiskit_ibm_pr
ovider'. Just replace 'qiskit.IBMQ' with 'qiskit_ibm_provider.IBMProvider'
  IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e97
5e510fddf7292950f2dd7237518abaa1ef5f53034fc7a726af7ae66ff76ffa9e801")
configrc.store_credentials:WARNING:2023-03-27 01:49:26,450: Credentials already pr
esent. Set overwrite=True to overwrite.
```

In [ ]:
```
#Executing the Circuit on Least Busy Quantum Computer
t_qc = transpile(circuit, backend, optimization_level = 3)
job = backend.run(t_qc)
job_monitor(job)
```

```
Job Status: job is queued (20)
```

Retrieving Job

In [5]:
```
# Load your IBM Quantum account
account = IBMQ.load_account()

# Retrieve the job using the job ID
job = account.get_backend('ibmq_manila').retrieve_job('63f226f62a88b6b2dbe25a02')

# Monitor the job to check its status
job_monitor(job)
# Get the result from the job
result = job.result()
```

```
ibmqfactory.load_account:WARNING:2023-03-27 01:49:43,093: Credentials are already
in use. The existing account in the session will be replaced.
Job Status: job has successfully run
```

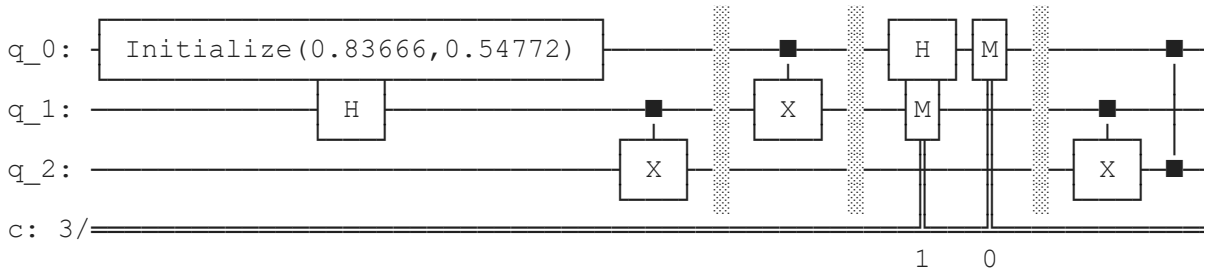
Import Important Libraries

In [1]:
```
import qiskit as q
import qiskit.visualization as qv
import numpy as np

from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, transpile, assemble
from qiskit.tools.monitor import job_monitor

import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")
```

Making the Desired Quantum Circuit

In [4]:
```
circuit = QuantumCircuit(3,3)
initial_state = np.array([np.sqrt(0.7),np.sqrt(0.3)]) #initial state of Alice's qub
circuit.h(1)
circuit.cx(1,2)
circuit.initialize(initial_state,[0]); circuit.barrier()
circuit.cx(0,1); circuit.barrier()
circuit.h(0)
circuit.measure(0,0); circuit.measure(1,1); circuit.barrier()
circuit.cx(1,2); circuit.cz(0,2)
circuit.measure(2,2)
circuit.draw()
```

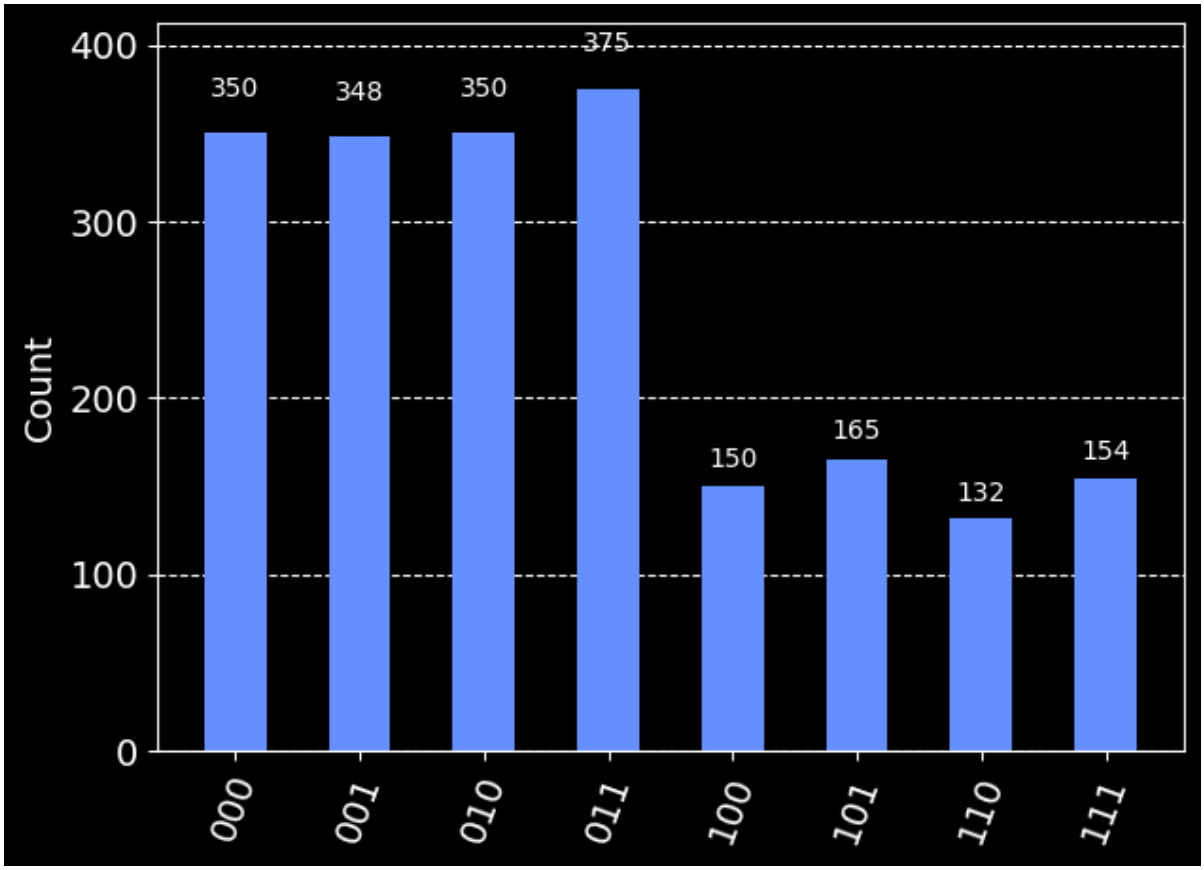
Out[4]:



Running the Circuit via QASM Simulator and checking Results (More the shots, better the probability distribution)

In [5]:
```
qasm_sim = Aer.get_backend("qasm_simulator") # Qiskit simulator backend
result = q.execute(circuit, qasm_sim, shots=2024).result()  # Results
counts = result.get_counts()
qv.plot_histogram(counts) #Visualizing o/p of simulator
```

Out[5]:



Loading your account and Running the Circuit via Least Busy Quantum Computer and checking Results

In [6]:
```
# SAVING AND LOADING YOUR IBM API TOKEN YOU GET WHEN YOU SIGN UP AT IBM QUANTUM COM
IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e975e5
IBMQ.load_account()
provider = IBMQ.get_provider(hub = 'ibm-q')
backend = least_busy(provider.backends(filters = lambda b: b.configuration().n_qubi
```

```
C:\Users\Sanket Lalwani\AppData\Local\Temp\ipykernel_15248\4250883806.py:2: Deprec
ationWarning: The qiskit.IBMQ entrypoint and the qiskit-ibmq-provider package (acc
essible from 'qiskit.providers.ibmq`) are deprecated and will be removed in a futu
re release. Instead you should use the qiskit-ibm-provider package which is access
ible from 'qiskit_ibm_provider'. You can install it with 'pip install qiskit_ibm_p
rovider'. Just replace 'qiskit.IBMQ' with 'qiskit_ibm_provider.IBMProvider'
  IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e97
5e510fddf7292950f2dd7237518abaa1ef5f53034fc7a726af7ae66ff76ffa9e801")
configrc.store_credentials:WARNING:2023-04-27 15:36:49,790: Credentials already pr
esent. Set overwrite=True to overwrite.
```

In [ ]:
```
#Executing the Circuit on Least Busy Quantum Computer
t_qc = transpile(circuit, backend, optimization_level = 3)
job = backend.run(t_qc)
job_monitor(job)
```

```
Job Status: job is queued (None)
```

In [ ]:

```
result = job.result()
counts = result.get_counts()
qv.plot_histogram(counts)
```

Importing Libraries and IBMQ Account

In [2]:
```
import qiskit as q
import qiskit.visualization as qv
import numpy as np

from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, transpile, assemble
from qiskit.tools.monitor import job_monitor

import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")

IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e975e5
IBMQ.load_account()
```

```
C:\Users\Sanket Lalwani\AppData\Local\Temp\ipykernel_14988\104988094.py:14: Deprec
ationWarning: The qiskit.IBMQ entrypoint and the qiskit-ibmq-provider package (acc
essible from 'qiskit.providers.ibmq`) are deprecated and will be removed in a futu
re release. Instead you should use the qiskit-ibm-provider package which is access
ible from 'qiskit_ibm_provider'. You can install it with 'pip install qiskit_ibm_p
rovider'. Just replace 'qiskit.IBMQ' with 'qiskit_ibm_provider.IBMProvider'
  IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e97
5e510fddf7292950f2dd7237518abaa1ef5f53034fc7a726af7ae66ff76ffa9e801")
configrc.store_credentials:WARNING:2023-04-29 00:49:39,512: Credentials already pr
esent. Set overwrite=True to overwrite.
```

Out[2]:
```
<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

Defining balanced and constant function Black Boxes

In [37]:
```
def balanced(n):
    c = QuantumCircuit(n+1,n)
    c.x(n)
    c.barrier()
    for i in range(n+1):
        c.h(i)
    c.barrier()
    for i in range(n):
        c.cx(i,n)
    c.barrier()
    for i in range(n):
        c.h(i)
    for i in range(n):
        c.measure(i,i)
    return c
```

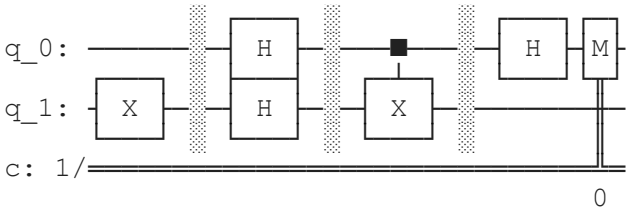
In [38]:
```
def constant(n):
    c = QuantumCircuit(n+1,n)
    c.x(n)
    c.barrier()
```

```
    for i in range(n+1):
        c.h(i)
    c.barrier()

    c.barrier()
    for i in range(n):
        c.h(i)
    for i in range(n):
        c.measure(i,i)
    return c
```

Creating our Quantum Circuits with Hadamard Sandwich for Balanced and Constant respectively

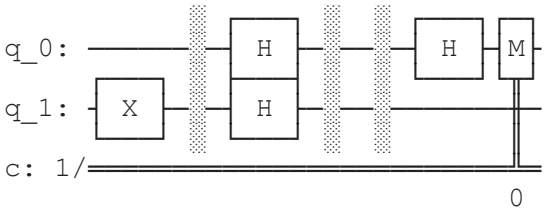
In [43]:
```
cb = balanced(1)
cb.draw()
```

Out[43]:



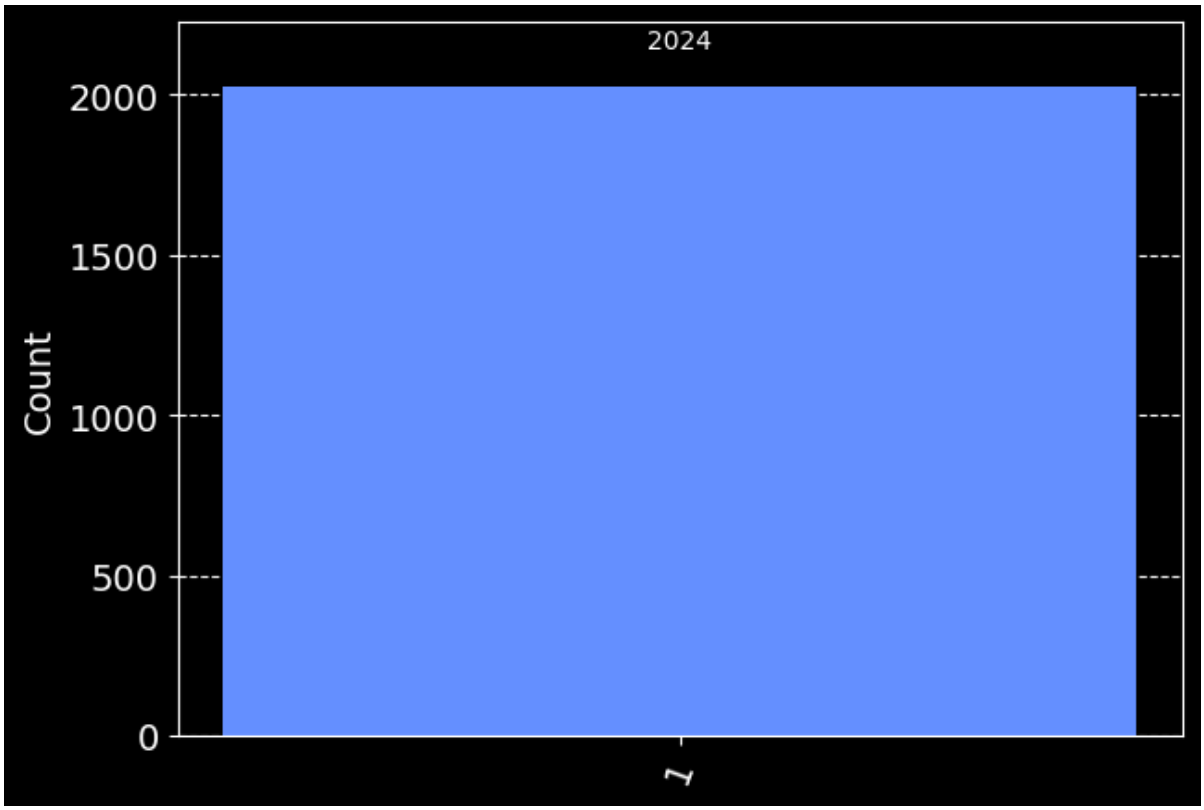
In [44]:
```
cc = constant(1)
cc.draw()
```

Out[44]:



In [45]:
```
qasm_sim = Aer.get_backend("qasm_simulator") # Qiskit simulator backend
result = q.execute(cb, qasm_sim, shots=2024).result()  # Results
counts = result.get_counts()
qv.plot_histogram(counts)
```

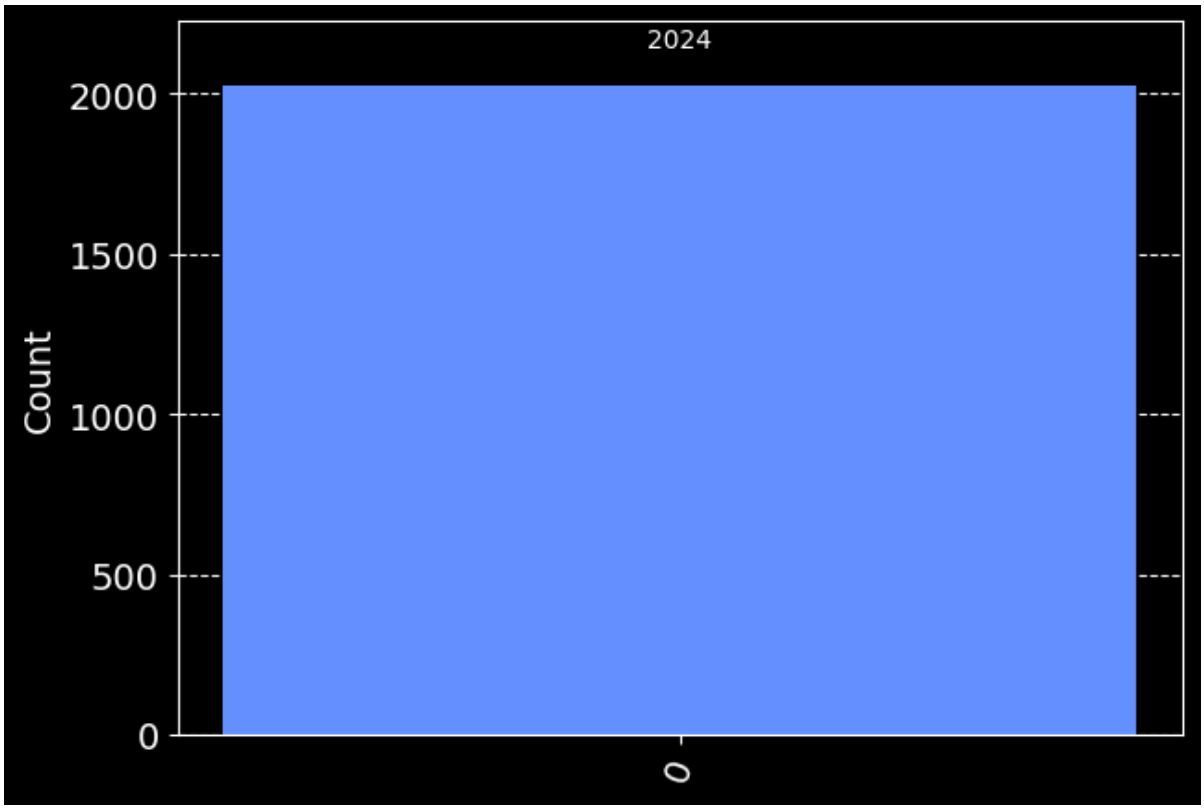
Out[45]:



In [46]:

```
qasm_sim = Aer.get_backend("qasm_simulator") # Qiskit simulator backend
result = q.execute(cc, qasm_sim, shots=2024).result()  # Results
counts = result.get_counts()
qv.plot_histogram(counts)
```

Out[46]:



Importing Libraries and IBMQ Account

In [4]:
```
import qiskit as q
import qiskit.visualization as qv
import numpy as np

from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, transpile, assemble
from qiskit.tools.monitor import job_monitor

import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")

IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e975e5
IBMQ.load_account()
```

```
C:\Users\Sanket Lalwani\AppData\Local\Temp\ipykernel_17796\104988094.py:14: Deprec
ationWarning: The qiskit.IBMQ entrypoint and the qiskit-ibmq-provider package (acc
essible from 'qiskit.providers.ibmq`) are deprecated and will be removed in a futu
re release. Instead you should use the qiskit-ibm-provider package which is access
ible from 'qiskit_ibm_provider'. You can install it with 'pip install qiskit_ibm_p
rovider'. Just replace 'qiskit.IBMQ' with 'qiskit_ibm_provider.IBMProvider'
  IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e97
5e510fddf7292950f2dd7237518abaa1ef5f53034fc7a726af7ae66ff76ffa9e801")
configrc.store_credentials:WARNING:2023-04-29 01:27:47,961: Credentials already pr
esent. Set overwrite=True to overwrite.
```

Out[4]:
```
<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

Defining balanced and constant function Black Boxes

In [10]:
```
def balanced(n):
    c = QuantumCircuit(n+1,n)
    c.x(n)
    c.barrier()
    for i in range(n+1):
        c.h(i)
    c.barrier()
    for i in range(n):
        c.cx(i,n)
    c.barrier()
    for i in range(n):
        c.h(i)
    for i in range(n):
        c.measure(i,i)
    return c
```

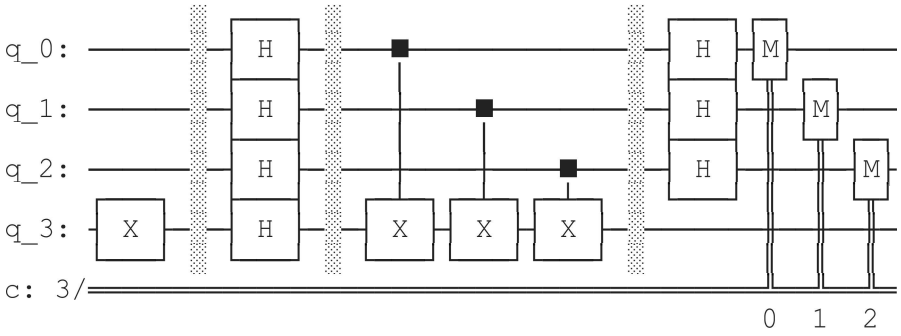
In [11]:
```
def constant(n):
    c = QuantumCircuit(n+1,n)
    c.x(n)
    c.barrier()
```

```
    for i in range(n+1):
        c.h(i)
    c.barrier()

    c.barrier()
    for i in range(n):
        c.h(i)
    for i in range(n):
        c.measure(i,i)
    return c
```

Creating our Quantum Circuits with Hadamard Sandwich for Balanced and Constant respectively

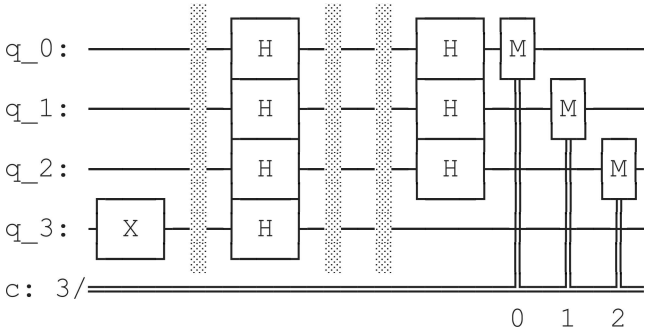
In [15]:
```
cb = balanced(3)
cb.draw()
```

Out[15]:



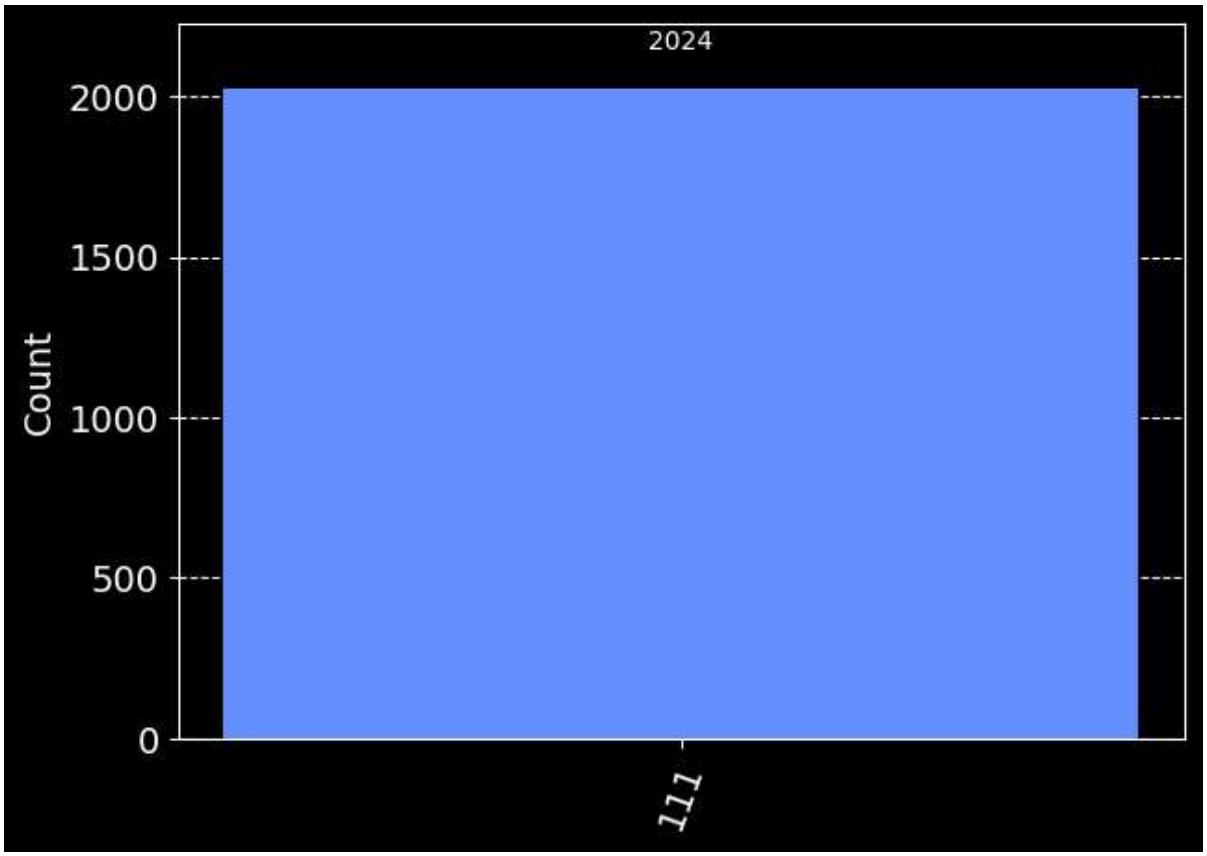
In [16]:
```
cc = constant(3)
cc.draw()
```

Out[16]:



In [17]:
```
qasm_sim = Aer.get_backend("qasm_simulator") # Qiskit simulator backend
result = q.execute(cb, qasm_sim, shots=2024).result()  # Results
counts = result.get_counts()
qv.plot_histogram(counts)
```

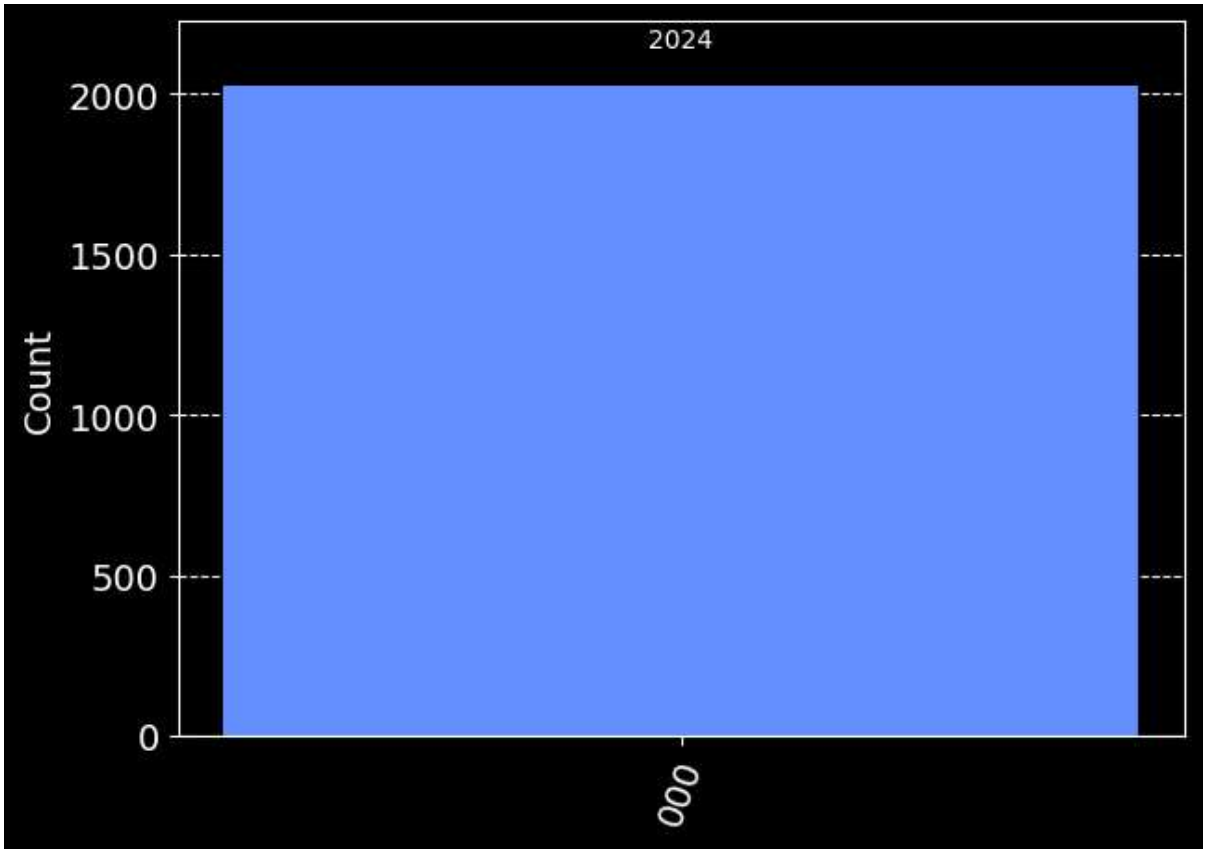
Out[17]:



In [18]:

```
qasm_sim = Aer.get_backend("qasm_simulator") # Qiskit simulator backend
result = q.execute(cc, qasm_sim, shots=2024).result()  # Results
counts = result.get_counts()
qv.plot_histogram(counts)
```

Out[18]:



Importing Libraries and IBMQ Account

In [1]:
```
import qiskit as q
import qiskit.visualization as qv
import numpy as np

from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, transpile, assemble
from qiskit.tools.monitor import job_monitor

import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")

IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e975e5
IBMQ.load_account()
```

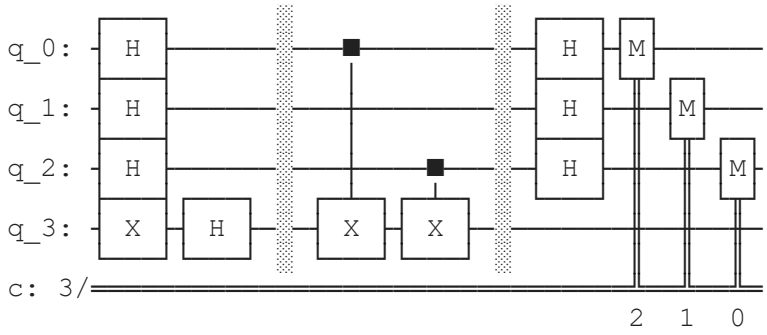
```
C:\Users\Sanket Lalwani\AppData\Local\Temp\ipykernel_7204\104988094.py:14: Depreca
tionWarning: The qiskit.IBMQ entrypoint and the qiskit-ibmq-provider package (acce
ssible from 'qiskit.providers.ibmq`) are deprecated and will be removed in a futur
e release. Instead you should use the qiskit-ibm-provider package which is accessi
ble from 'qiskit_ibm_provider'. You can install it with 'pip install qiskit_ibm_pr
ovider'. Just replace 'qiskit.IBMQ' with 'qiskit_ibm_provider.IBMProvider'
  IBMQ.save_account("8de7277a56e6becca30f1d2cd5957def658c0f26541eec4ffaf33b0224e97
5e510fddf7292950f2dd7237518abaa1ef5f53034fc7a726af7ae66ff76ffa9e801")
configrc.store_credentials:WARNING:2023-04-27 15:16:53,357: Credentials already pr
esent. Set overwrite=True to overwrite.
```

Out[1]:
```
<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

Setting up the Quantum Circuit

In [4]:
```
num = int(input("Enter a Number: "))
x = np.binary_repr(num)
print(x)
c = q.QuantumCircuit(len(x)+1,len(x))
c.h(list(range(len(x))))
c.x(len(x))
c.h(len(x))
c.barrier()
for i, j in enumerate(x):
    if j == "1":
        c.cx(i , len(x))

c.barrier()

c.h(list(range(len(x))))
c.measure(list(range(len(x))) , list(range(len(x)))[::-1] )
c.draw()
```

Out[4]:



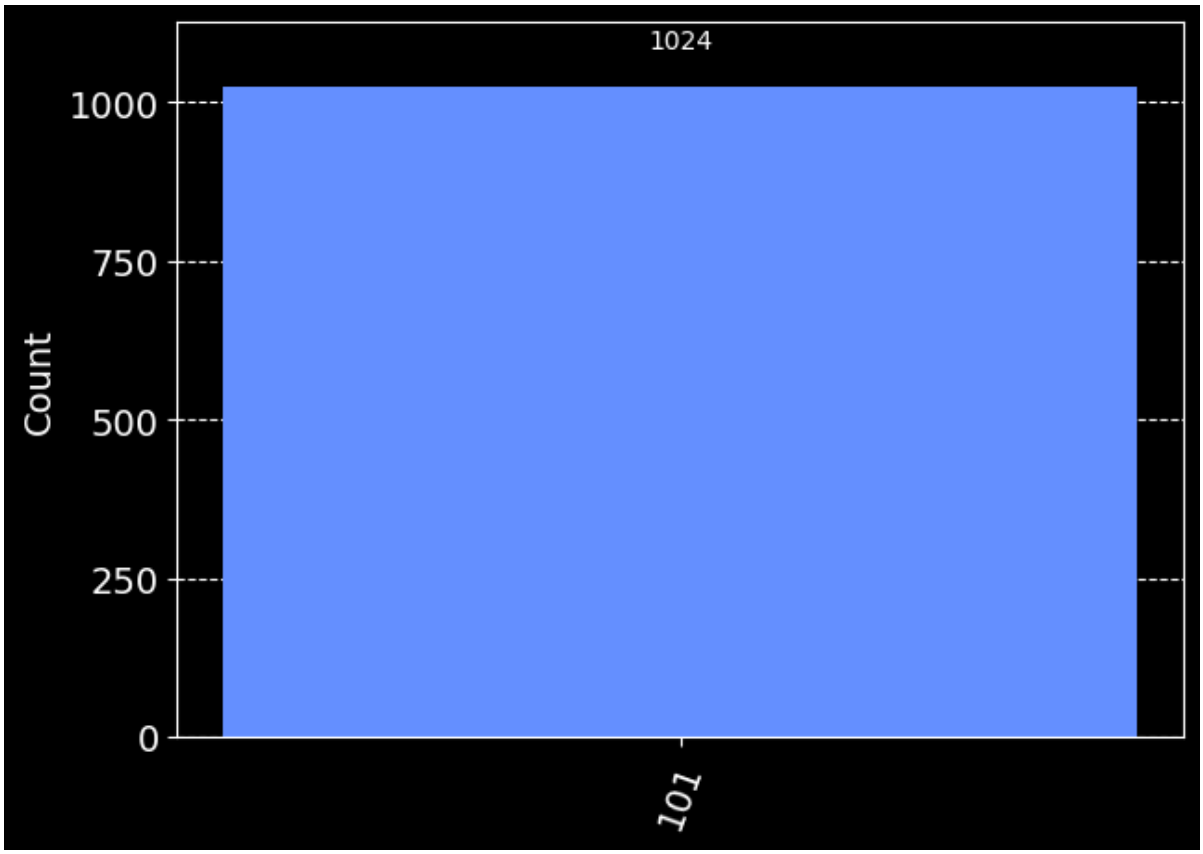
Simulating the Quantum Circuit

In [5]:

```
qasm_b = q.Aer.get_backend("qasm_simulator")
counts = q.execute(c, qasm_b, shots = 1024).result().get_counts()
print("Counts =", counts)
for i in counts:
    a = int(str(i),2)

print("Your number was", a, "and Number of shots =", 1024)
qv.plot_histogram(counts)
```

```
Counts = {'101': 1024}
Your number was 5 and Number of shots = 1024
```

Out[5]:



Running the Quantum Circuit using Quantum Computer

In [18]:

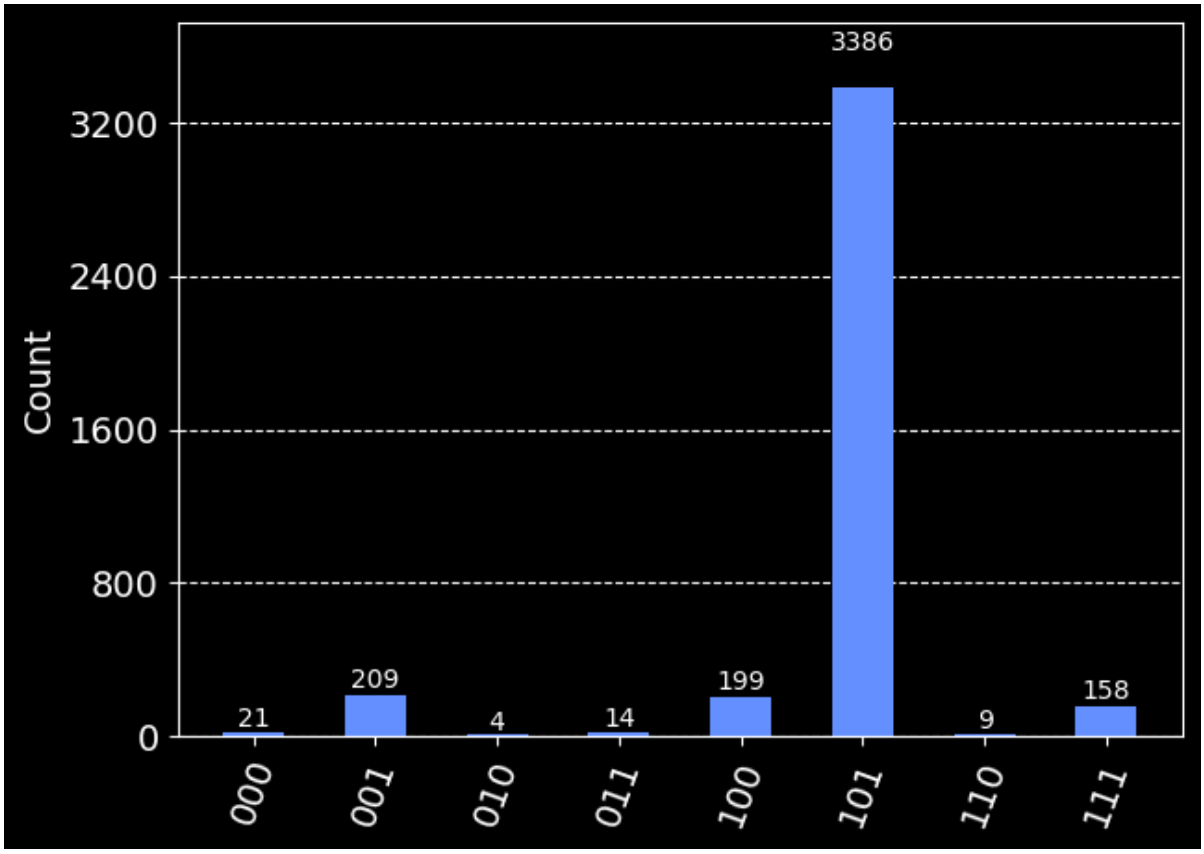
```
provider = IBMQ.get_provider(hub = 'ibm-q')
backend = least_busy(provider.backends(filters = lambda b: b.configuration().n_qubi
```

```
t_qc = transpile(c, backend, optimization_level = 3)
job = backend.run(t_qc)
job_monitor(job)
```

```
Job Status: job has successfully run
```

In [19]:
```
result = job.result()
counts = result.get_counts()
qv.plot_histogram(counts)
```

Out[19]:



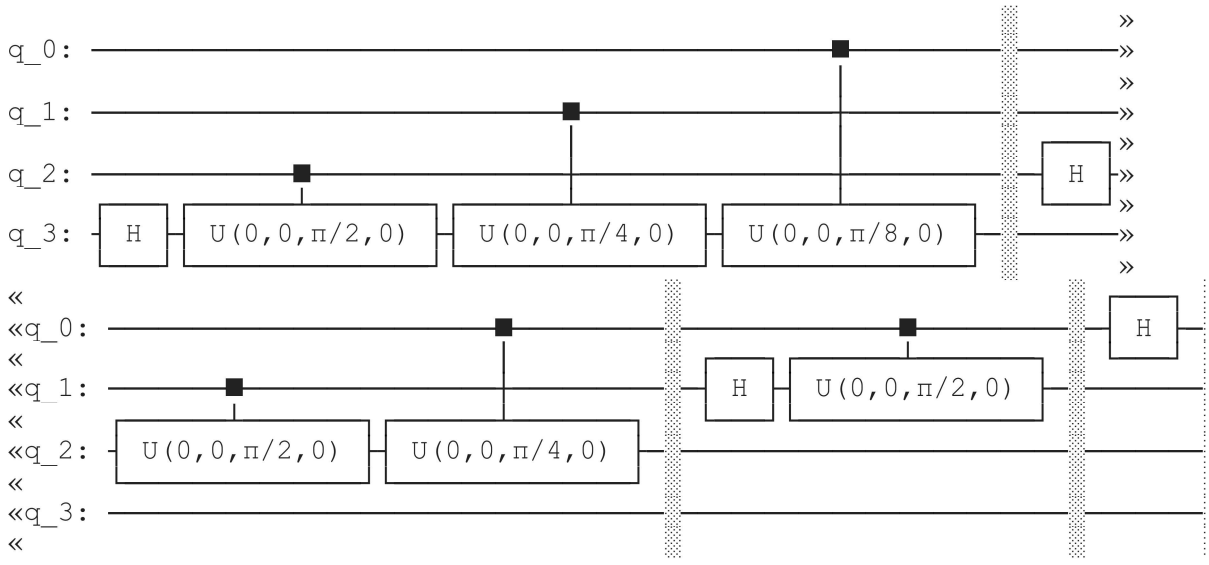
Import Important Libraries

In [1]:
```
import numpy as np

from qiskit.circuit.library import QFT
import qiskit.visualization as qv
from qiskit import IBMQ, Aer, QuantumCircuit, transpile, assemble
from qiskit.providers.ibmq import least_busy
from qiskit.quantum_info import Statevector
from qiskit.tools.monitor import job_monitor

import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")
```

Defining the QFT Circuit Function

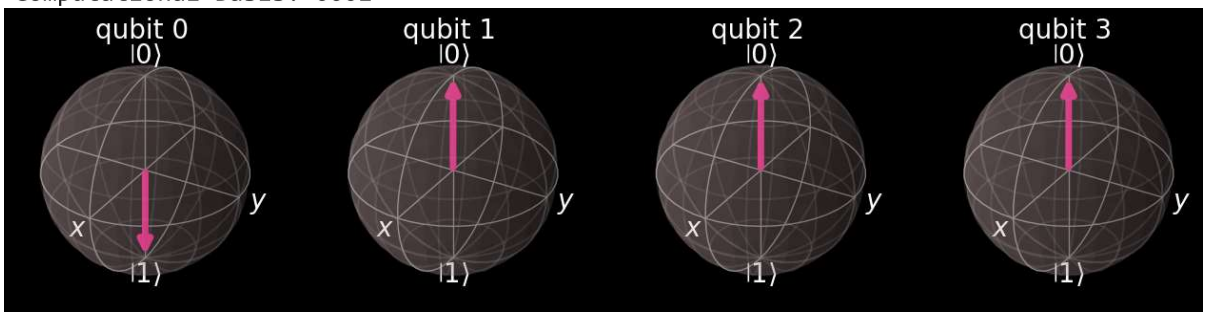
In [2]:
```
def myQFT(n):
    circuit = QuantumCircuit(n)
    for i in range(n):
        circuit.h(n-1-i)
        for j in range(i+1, n):
            circuit.cu(0, 0, np.pi/(2**(j-i)),0 , n-1-j, n-1-i)
        circuit.barrier()

    for k in range(0, int(n/2)):
        circuit.swap(k, n-1-k)

    return circuit

myQFT(4).draw()
```

Out[2]:

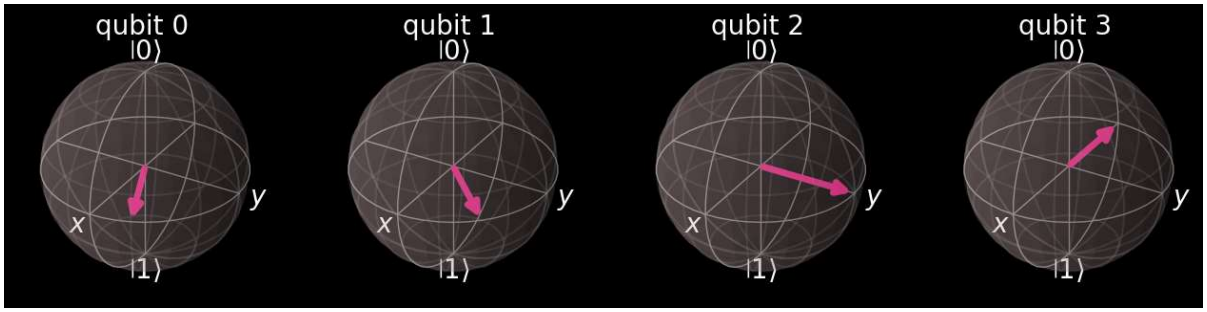


In [10]:
```
state = "0001"; circuit = QuantumCircuit(len(state))

circuit.initialize(Statevector.from_label(state).data)

print("Computational Basis:", state);
display(qv.plot_bloch_multivector(Statevector.from_instruction(circuit).data))

circuit.append(QFT(len(state)), circuit.qubits)

print("Fourier Basis") ;
display(qv.plot_bloch_multivector(Statevector.from_instruction(circuit).data))
```

```
Computational Basis: 0001
```




```
Fourier Basis
```



In [ ]:

Import Important Libraries

In [2]:
```
import numpy as np

from qiskit.circuit.library import QFT
import qiskit.visualization as qv
from qiskit import IBMQ, Aer, QuantumCircuit, transpile, assemble
from qiskit.providers.ibmq import least_busy
from qiskit.quantum_info import Statevector
from qiskit.tools.monitor import job_monitor

import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")
```

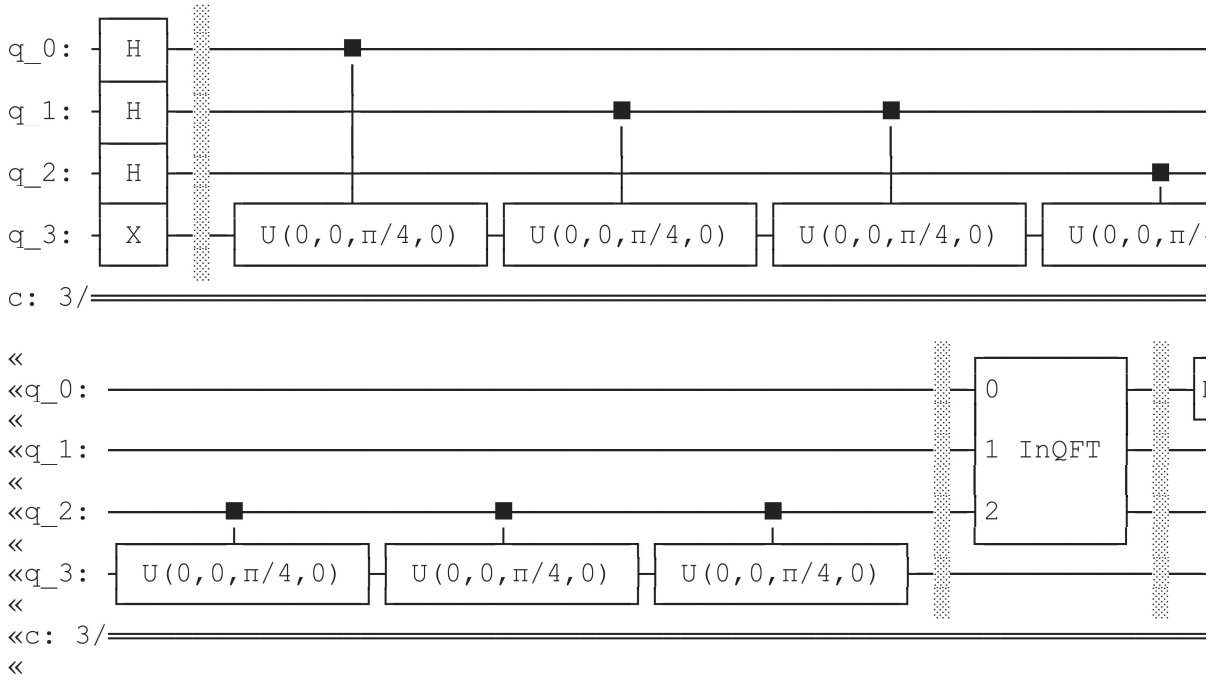
Defining the Inverse QFT Circuit Function

In [3]:
```
def InvQFT(n):
    circuit = QuantumCircuit(n, name ="InQFT")
    circuit.qubits[::-n]
    for k in range(0, int(n/2)):
        circuit.swap(k, n-1-k)
    circuit.barrier()
    for i in range(n):
        circuit.h(i)
        for j in range(i+1, n):
            circuit.cu(0, 0, -np.pi/(2**(j-i)),0 ,j, i)
        circuit.barrier()
    return circuit
```

In [8]:
```
n = 3 # Controlled Qubits
m = 1 # Target Qubit

qpe = QuantumCircuit(n+m,n)
for i in range(n):
    qpe.h(i)

for j in range(m):
    qpe.x(j+n)

qpe.barrier()

lamda = np.pi/4

x = 1
for i in range(n):
    for j in range(x):
        qpe.cu(0, 0, lamda, 0, i, n)
        x = x+1

qpe.barrier()
```

```
qpe.append(InvQFT(n), qpe.qubits[0:n])

qpe.barrier()
for i in range(n):
    qpe.measure(i,i)
qpe.draw()
```

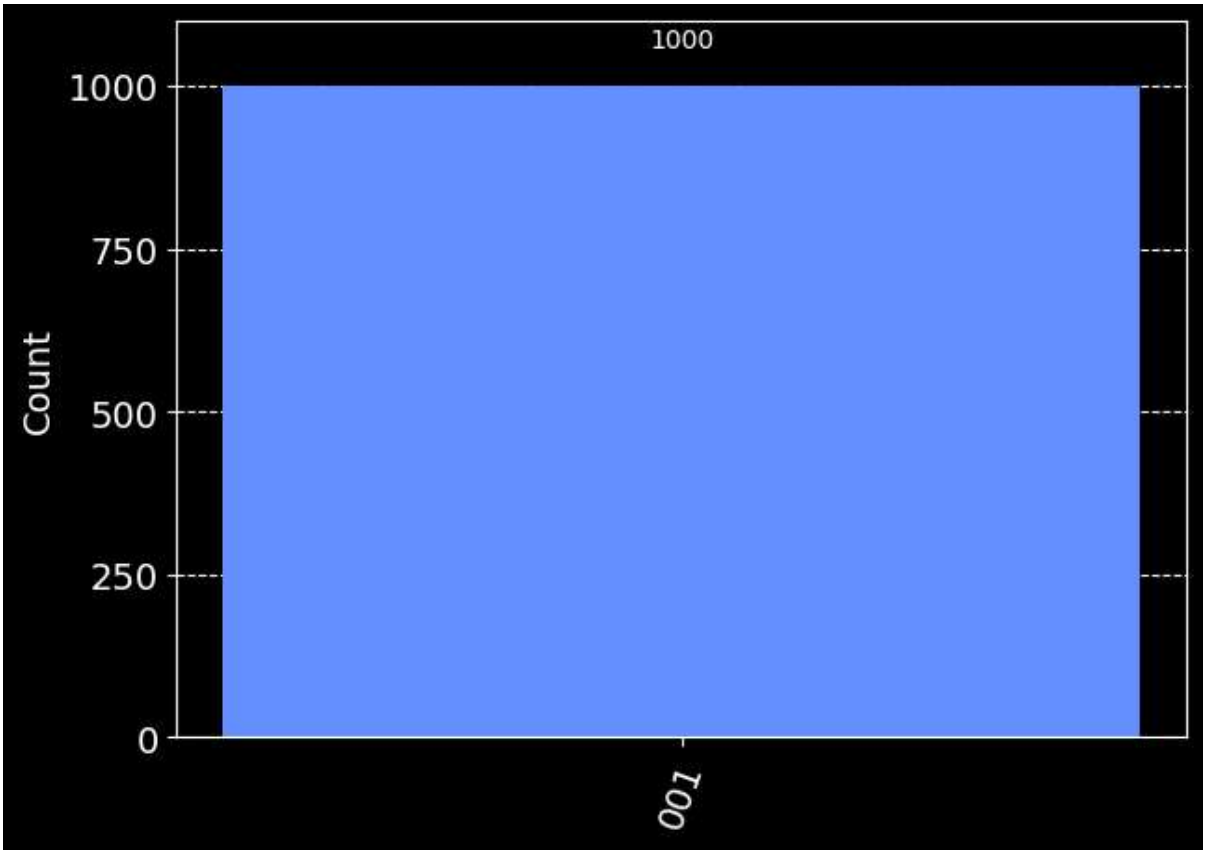
Out[8]:



In [9]:

```
aer_sim = Aer.get_backend('aer_simulator')
shots = 1000
t_qpe = transpile(qpe, aer_sim)
results = aer_sim.run(t_qpe, shots=shots).result()
answer = results.get_counts()

qv.plot_histogram(answer)
```

Out[9]:



In [ ]:

```
#We see we get one result (001) with certainty, which translates to the decimal: 1.
#We now need to divide our result by 2^n to get Theta:
# Theta = Binary(001)/2^n = 1/8
```

Importing Libraries

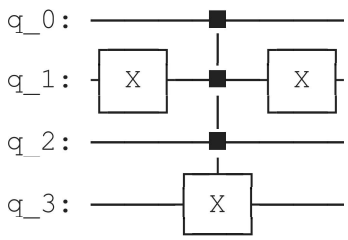
In [2]:
```
import qiskit as q
import qiskit.visualization as qv
import numpy as np

from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, transpile, assemble
from qiskit.tools.monitor import job_monitor
from qiskit.circuit.library import GroverOperator

import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")
```

Defining oracles Uf and Ur

In [3]:
```
def Uf(n, num):
    circuit = QuantumCircuit(n+1,name = "Uf")
    x = np.binary_repr(num, n)[::-1]
    for i, j in enumerate(x):
        if j == "0":
            circuit.x(i)

    circuit.mcx(list(range(n)),n)

    for i, j in enumerate(x):
        if j == "0":
            circuit.x(i)

    return circuit
print("Uf Circuit:")
Uf(3,5).draw()
```

```
Uf Circuit:
```

Out[3]:

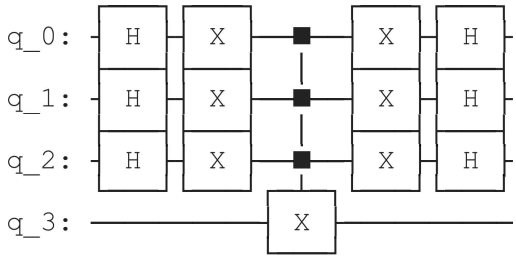


In [4]:
```
def diffuser(n):
    circuit = QuantumCircuit(n+1, name ="diffuser")
    circuit.h(range(n))
    circuit.x(range(n))
    circuit.mcx(list(range(n)),n)
    circuit.x(range(n))
    circuit.h(range(n))
    return circuit
```

```
print("diffuser Circuit:")
diffuser(3).draw()
```

```
diffuser Circuit:
```

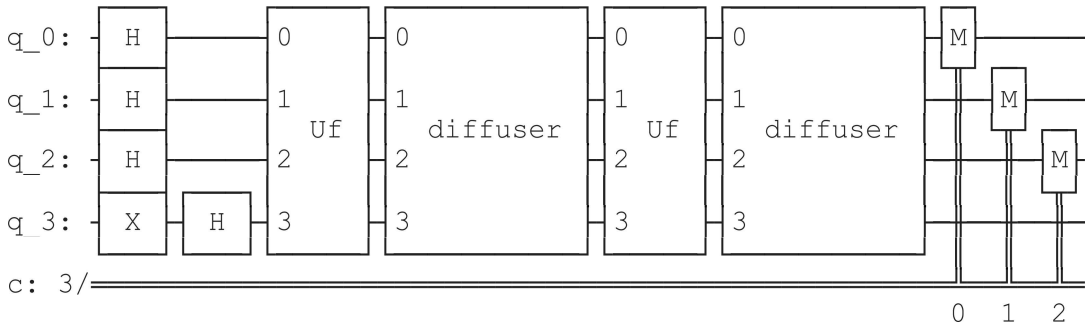
Out[4]:



In [5]:

```
n = 3; num = 6
grover = QuantumCircuit(n+1,n)
nsol = 1 # No. of Solutions

# Initialize the total number of times we pass the oracles
r = int(np.floor(np.pi/4*np.sqrt(2**n)/nsol))

grover.h(range(n))

#Initiate Last Qubit in |-> state
grover.x(n)
grover.h(n)

#Apply r rounds of Uf and Diffuser Circuits
for i in range(r):
    grover.append( Uf(n, num), grover.qubits[0:n+1] )
    grover.append( diffuser(n), grover.qubits[0:n+1] )

grover.measure(range(n),range(n))
grover.draw()
```

Out[5]:

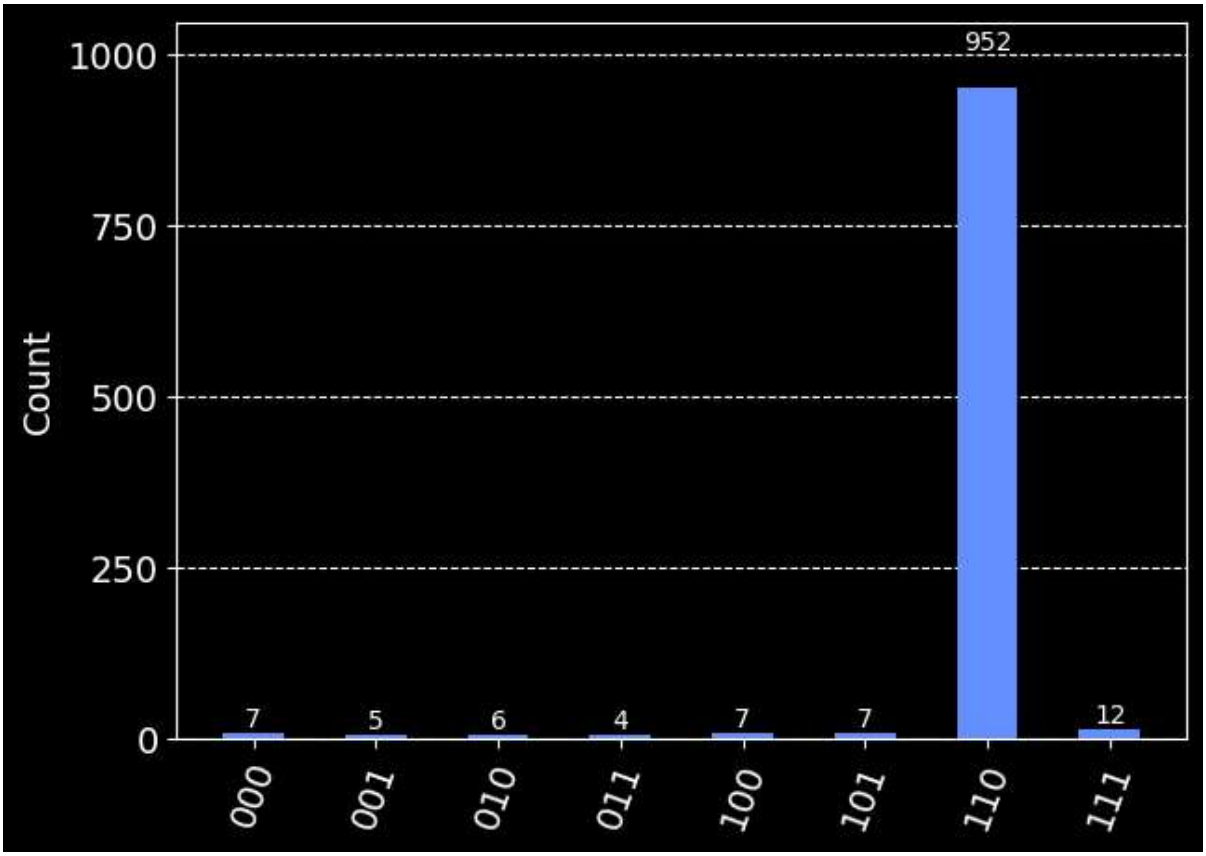


In [8]:

```
aer_sim = Aer.get_backend('aer_simulator')
shots = 1000
t_grover = transpile(grover, aer_sim)
results = aer_sim.run(t_grover, shots=shots).result()
answer = results.get_counts()

print("number:",num,"=",np.binary_repr(num))
qv.plot_histogram(answer)
```

```
number: 6 = 110
```

Out[8]:


1000
750
500
250
0
Count
952
7
5
6
4
7
7
12
000
001
010
011
100
101
110
111

Importing Libraries

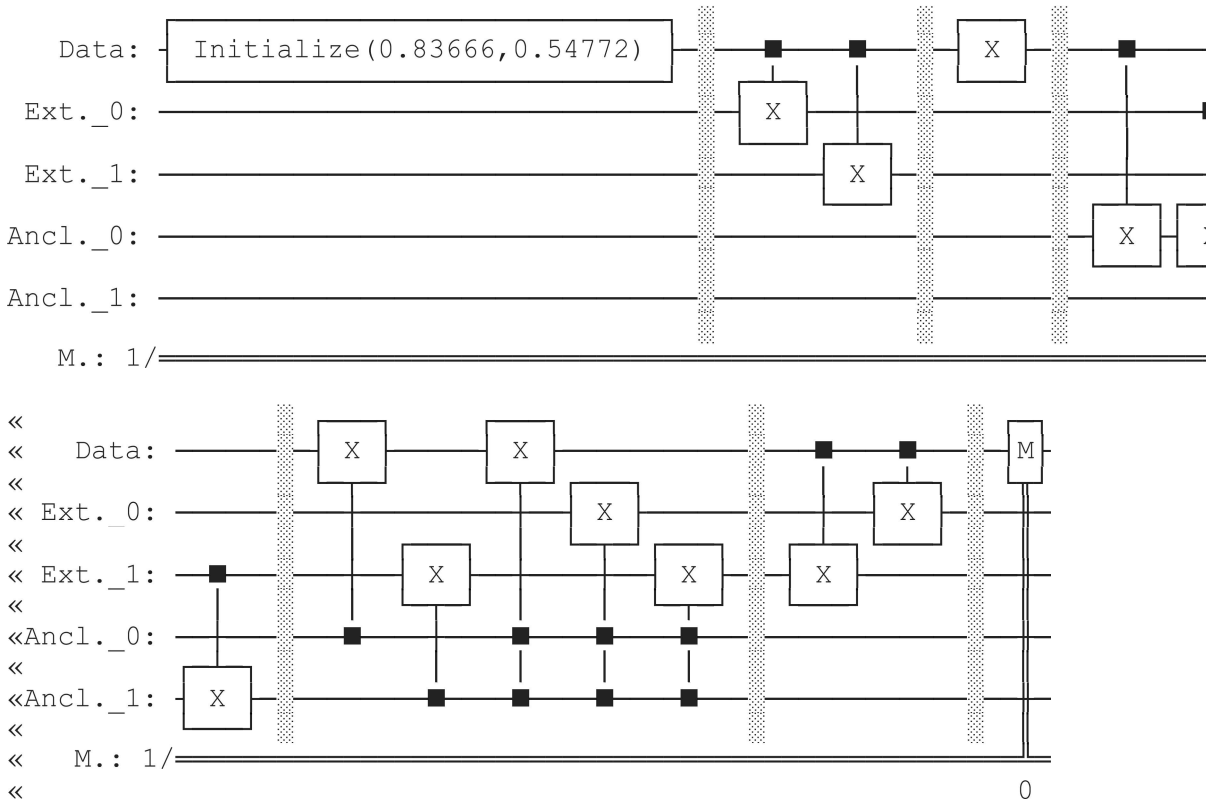
In [2]:
```
import qiskit as q
import qiskit.visualization as qv
import numpy as np

from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, transpile, assemble
from qiskit.tools.monitor import job_monitor
from qiskit.circuit.library import GroverOperator

import matplotlib.style
import matplotlib as plt
plt.style.use("dark_background")
```

Defining our Quantum Circuit

In [56]:
```
n = q.QuantumRegister(1, name = "Data")
m = q.QuantumRegister(2, name = "Ext.")
qa = q.QuantumRegister(2, name = "Ancl.")
cl = q.ClassicalRegister(1, name = "M.")
circuit = QuantumCircuit(n, m, qa, cl)

#Initializing our information qubit
initial_state = np.array([np.sqrt(0.7),np.sqrt(0.3)])
circuit.initialize(initial_state,0)
circuit.barrier()

#Encoding Circuit: a|0> + b|1> => a|000> + b|111>
circuit.cx(0,1); circuit.cx(0,2)

#Bit Flip Error
circuit.barrier()
circuit.x(0)
circuit.barrier()

#Syndrom Circuit
circuit.cx(0,3); circuit.cx(1,3)
circuit.cx(1,4); circuit.cx(2,4)
circuit.barrier()

#Recovery Circuit
circuit.cx(3,0); circuit.cx(4,2)
circuit.ccx(4,3,0)
circuit.ccx(4,3,1)
circuit.ccx(4,3,2)
circuit.barrier()

#Decoding Circuit: a|000> + b|111> => a|0> + b|1>
circuit.cx(0,2); circuit.cx(0,1)
circuit.barrier()
```

```
#Measurement
circuit.measure(0,0)

circuit.draw()
```

Out[56]:



In [57]:

```
aer_sim = Aer.get_backend('aer_simulator')
shots = 1000
t_c = transpile(circuit, aer_sim)
results = aer_sim.run(t_c, shots=shots).result()
answer = results.get_counts()

qv.plot_histogram(answer)
```

Out[57]: