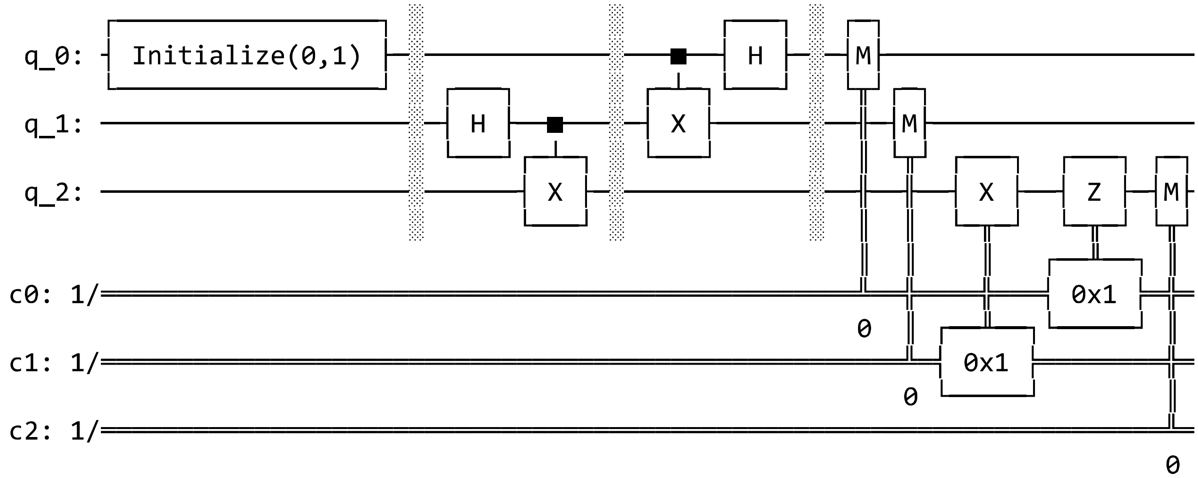
Title : Implement Quantum Teleportation Algorithm

In [29]:
```
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile, QuantumRegister, ClassicalRegiste
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *

# qiskit-ibmq-provider has been deprecated.
# Please see the Migration Guides in https://ibm.biz/provider_migration_guide f
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler, Estimator, Sessic

# Loading your IBM Quantum account(s)
service = QiskitRuntimeService(channel="ibm_quantum")

# Invoke a primitive. For more details see https://qiskit.org/documentation/par
# result = Sampler("ibmq_qasm_simulator").run(circuits).result()
```

In [30]:
```
# Create the quantum circuit with 3 qubits and 3 classical bits
q = QuantumRegister(3, 'q')  # Quantum register
c0 = ClassicalRegister(1, 'c0')  # Classical register for Alice's qubit
c1 = ClassicalRegister(1, 'c1')  # Classical register for Bob's qubit
c2 = ClassicalRegister(1, 'c2')  # Classical register for the result
circuit = QuantumCircuit(q, c0, c1, c2)

# Prepare the initial state to be teleported
circuit.initialize([0, 1], q[0])  # Apply X gate to put in state |1>
circuit.barrier()

# Create an entanglement between Alice's and Bob's qubits
circuit.h(q[1])
circuit.cx(q[1], q[2])
circuit.barrier()

# Teleportation process
circuit.cx(q[0], q[1])
circuit.h(q[0])
circuit.barrier()

# Measure Alice's qubits and send the measurement results to Bob
circuit.measure(q[0], c0[0])
circuit.measure(q[1], c1[0])

# Apply corrective operations on Bob's qubit based on the measurement results
circuit.x(q[2]).c_if(c1, 1)
circuit.z(q[2]).c_if(c0, 1)

# Measure the teleported qubit
circuit.measure(q[2], c2[0])

# Visualize the circuit
print(circuit)
circuit_drawer(circuit, output='mpl')

# Simulate the circuit using the QASM simulator
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1)
result = job.result()
teleported_state = result.get_counts(circuit)

# Print the teleported state
print("Teleported state:", teleported_state)
```




```
Teleported state: {'1 0 1': 1}
```

In [ ]:

In [27]:
```
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, execute, IBMQ
from qiskit.tools.monitor import job_monitor
from qiskit.circuit.library import QFT
import numpy as np

pi = np.pi

# provider = IBMQ.get_provider(hub='ibm-q')

backend = provider.get_backend('ibmq_qasm_simulator')

q = QuantumRegister(5,'q')
c = ClassicalRegister(5,'c')

circuit = QuantumCircuit(q,c)

circuit.x(q[4])
circuit.x(q[2])
circuit.x(q[0])
circuit.append(QFT(num_qubits=5, approximation_degree=0, do_swaps=True, inverse
circuit.measure(q,c)
circuit.draw(output='mpl', filename='qft1.png')
print(circuit)

job = execute(circuit, backend, shots=1000)

job_monitor(job)

counts = job.result().get_counts()

print("\n QFT Output")
print("-------------")
print(counts)
```

```
     ┌───┐┌──────┐┌─┐
q_0: ┤ X ├┤0     ├┤M├────────────
     └───┘│      │└╥┘┌─┐
q_1: ─────┤1     ├─╫─┤M├─────────
     ┌───┐│      │ ║ └╥┘┌─┐
q_2: ┤ X ├┤2 QFT ├─╫──╫─┤M├──────
     └───┘│      │ ║  ║ └╥┘┌─┐
q_3: ─────┤3     ├─╫──╫──╫─┤M├───
     ┌───┐│      │ ║  ║  ║ └╥┘┌─┐
q_4: ┤ X ├┤4     ├─╫──╫──╫──╫─┤M├
     └───┘└──────┘ ║  ║  ║  ║ └╥┘
c: 5/══════════════╩══╩══╩══╩══╩═
                   0  1  2  3  4
Job Status: job has successfully run

 QFT Output
-------------
{'00010': 28, '10000': 26, '10101': 26, '01100': 28, '11001': 30, '01010': 2
5, '10001': 25, '10011': 26, '01000': 40, '11011': 39, '10110': 37, '11111':
26, '10010': 35, '10100': 27, '11101': 22, '00111': 41, '00000': 35, '01011':
27, '00110': 28, '00011': 35, '00100': 37, '00001': 28, '01101': 27, '11010':
33, '01110': 27, '01001': 29, '00101': 39, '11100': 42, '11110': 36, '11000':
24, '10111': 31, '01111': 41}
```

In [ ]:

```
In [28]: from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, execute,IBMQ
from qiskit.tools.monitor import job_monitor
from qiskit.circuit.library import QFT
import numpy as np

pi = np.pi


provider = IBMQ.get_provider(hub='ibm-q')

backend = provider.get_backend('ibmq_qasm_simulator')

q = QuantumRegister(5,'q')
c = ClassicalRegister(5,'c')

circuit = QuantumCircuit(q,c)

circuit.x(q[4])
circuit.x(q[2])
circuit.x(q[0])
circuit.append(QFT(num_qubits=5, approximation_degree=0, do_swaps=True, inverse
circuit.measure(q,c)
circuit.draw(output='mpl', filename='qft1.png')
print(circuit)

job = execute(circuit, backend, shots=1000)

job_monitor(job)

counts = job.result().get_counts()

print("\n QFT Output")
print("-------------")
print(counts)
input()

q = QuantumRegister(5,'q')
c = ClassicalRegister(5,'c')

circuit = QuantumCircuit(q,c)

circuit.x(q[4])
circuit.x(q[2])
circuit.x(q[0])
circuit.append(QFT(num_qubits=5, approximation_degree=0, do_swaps=True, inverse
circuit.measure(q,c)
circuit.draw(output='mpl',filename='qft2.png')

print(circuit)

job = execute(circuit, backend, shots=1000)

job_monitor(job)

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

```
print("\n QFT with inverse QFT Output")
print("------------------------------")
print(counts)
input()
```

```
     ┌───┐┌──────┐┌─┐            
q_0: ┤ X ├┤0     ├┤M├────────────
     └───┘│      │└╥┘┌─┐         
q_1: ─────┤1     ├─╫─┤M├─────────
     ┌───┐│      │ ║ └╥┘┌─┐      
q_2: ┤ X ├┤2 qft ├─╫──╫─┤M├──────
     └───┘│      │ ║  ║ └╥┘┌─┐   
q_3: ─────┤3     ├─╫──╫──╫─┤M├───
     ┌───┐│      │ ║  ║  ║ └╥┘┌─┐
q_4: ┤ X ├┤4     ├─╫──╫──╫──╫─┤M├
     └───┘└──────┘ ║  ║  ║  ║ └╥┘
c: 5/══════════════╩══╩══╩══╩══╩═
                   0  1  2  3  4 
Job Status: job has successfully run

 QFT Output
-------------
{'00110': 24, '00111': 34, '11000': 20, '10111': 29, '00010': 32, '10100': 3
3, '11101': 38, '01100': 36, '01110': 31, '01111': 27, '11001': 47, '11010':
29, '11110': 35, '11100': 23, '10101': 35, '10000': 37, '11111': 19, '10110':
31, '10010': 25, '11011': 33, '01000': 22, '00100': 43, '00011': 38, '01011':
30, '00000': 33, '01001': 27, '00101': 32, '10001': 27, '01010': 37, '01101':
26, '00001': 41, '10011': 26}



     ┌───┐┌──────┐┌─┐            
q_0: ┤ X ├┤0     ├┤M├────────────
     └───┘│      │└╥┘┌─┐         
q_1: ─────┤1     ├─╫─┤M├─────────
     ┌───┐│      │ ║ └╥┘┌─┐      
q_2: ┤ X ├┤2 qft ├─╫──╫─┤M├──────
     └───┘│      │ ║  ║ └╥┘┌─┐   
q_3: ─────┤3     ├─╫──╫──╫─┤M├───
     ┌───┐│      │ ║  ║  ║ └╥┘┌─┐
q_4: ┤ X ├┤4     ├─╫──╫──╫──╫─┤M├
     └───┘└──────┘ ║  ║  ║  ║ └╥┘
c: 5/══════════════╩══╩══╩══╩══╩═
                   0  1  2  3  4 
Job Status: job has successfully run

 QFT with inverse QFT Output
------------------------------
{'00010': 24, '00011': 25, '00100': 30, '11010': 36, '01110': 27, '11000': 3
0, '10111': 31, '01100': 31, '10000': 35, '10101': 22, '00110': 48, '10011':
27, '01000': 30, '11011': 38, '01101': 29, '00001': 31, '11111': 23, '10010':
39, '10110': 39, '11100': 42, '11110': 33, '11001': 35, '00111': 35, '01011':
31, '00000': 31, '01111': 27, '01010': 17, '10001': 35, '01001': 24, '00101':
25, '10100': 41, '11101': 29}
```

Out[28]: ''

In [ ]: