

Notebook

October 24, 2023

1 Introduction

1.1 Some lecture notes

One online tool very useful to emulate quantum circuits is the [IBM Quantum Learning](#) platform.

It is also available a library which deals with quantum mechanics: [QuTiP](#)

1.2 Prerequisites

Here we will implement our first quantum circuit using the python library `Qiskit`. In order to run the code, remember to install the dependencies:

```
pip install qiskit qiskit[visualization] qiskit-aer
```

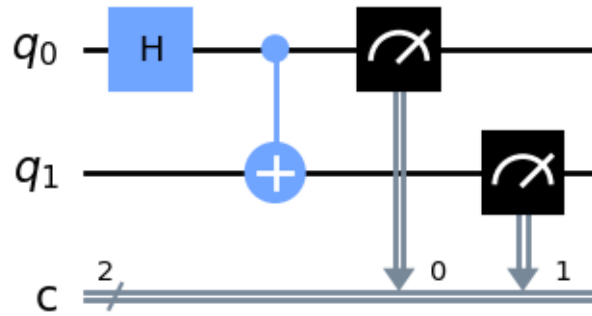
Let's import the function we will use:

```
[1]: from qiskit import QuantumCircuit, transpile
     from qiskit.providers.aer import QasmSimulator
     from qiskit.visualization import plot_histogram
```

```
[2]: simulator = QasmSimulator()
     # declare a circuit with 2 qubits and 2 classical bits
     circuit = QuantumCircuit(2, 2)
```

```
[3]: # apply a Hadamard gate to the qubit 0
     circuit.h(0)
     # apply a C-X gate on qubit 1 using qubit 0 as control
     circuit.cx(0, 1)
     # measure both qubits, storing values into classical bits
     circuit.measure([0, 1], [0, 1])
     circuit.draw(output='mpl')
```

```
[3]:
```



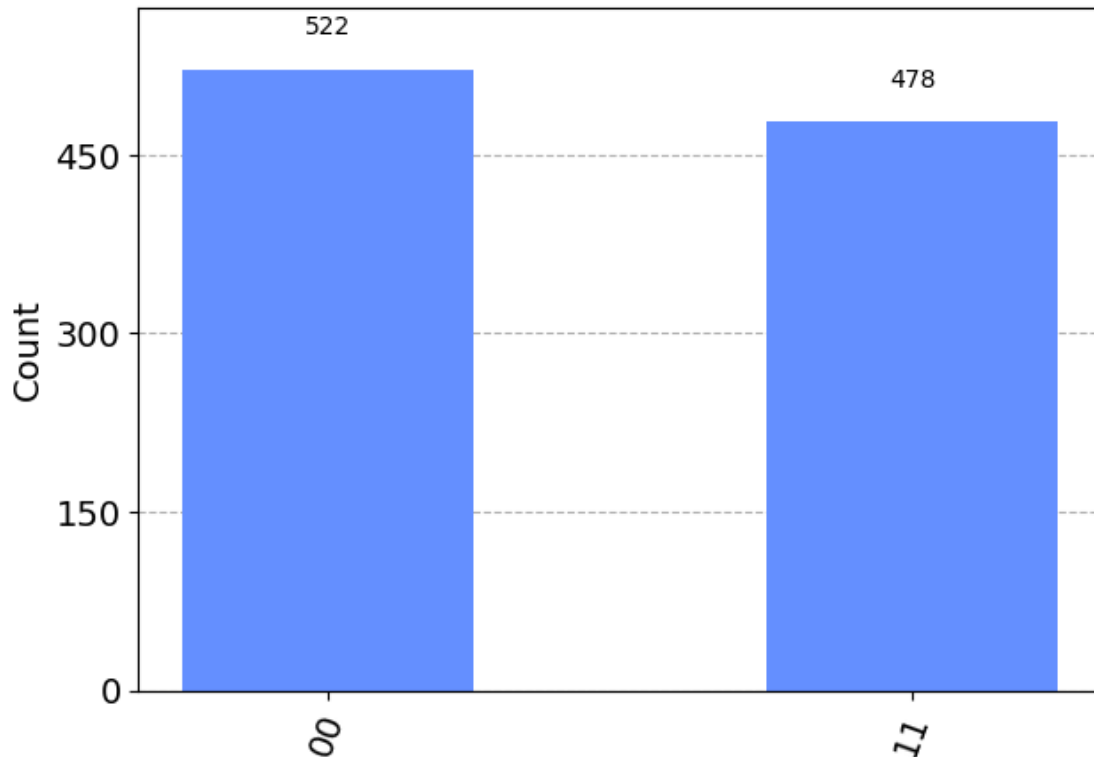
```
[4]: # transpile (i.e. "compile") the circuit to run on the simulator
compiled_circuit = transpile(circuit, simulator)
# run the circuit on the simulator and get the results
job = simulator.run(compiled_circuit, shots=1000)
result = job.result()
```

```
[5]: counts = result.get_counts(compiled_circuit)
print("\nTotal count for 00 and 11 are:", counts)

plot_histogram(counts)
```

Total count for 00 and 11 are: {'11': 478, '00': 522}

[5]:



2 Quantum Teleportation

2.1 The circuit

In this first lab lecture we will see how to simulate quantum teleportation using Qiskit.

First, let's create the quantum circuit we need:

```
[15]: from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

# one register of 3 qubits
qr1 = QuantumRegister(2, name='a')
qr2 = QuantumRegister(1, name='b')
# 2 registers of 1 bit each
cr1 = ClassicalRegister(1, name='cr1')
cr2 = ClassicalRegister(1, name='cr2')
# quantum circuit
teleportation_circuit = QuantumCircuit(qr1, qr2, cr1, cr2)
```

In order to do teleportation we must give Alice and Bob an entangled pair.

```
[16]: def bellPair(qc, a, b):
    '''
```

```

Creates a bell pair in qc using qubits a & b
'''
qc.h(a) # Put qubit a into state |+>
qc.cx(a,b) # CNOT with a as control and b as target

```

```

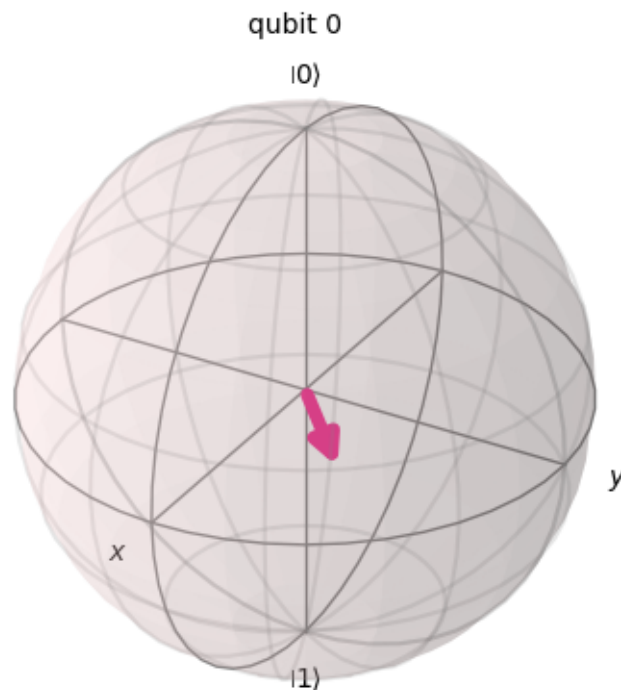
[17]: from qiskit.quantum_info import random_statevector
      from qiskit.visualization import array_to_latex, plot_bloch_multivector

      # Initialize a random state
      psi = random_statevector(2)
      # Display it nicely
      display(array_to_latex(psi, prefix="|\\psi\\rangle ="))
      # Show it on a Bloch sphere
      plot_bloch_multivector(psi)

```

$$|\psi\rangle = [-0.7682519717 - 0.1879997459i \quad -0.3909643146 - 0.4707354972i]$$

[17]:



Let's initialize $|\psi\rangle$ starting from $|0\rangle$. Notice that `Initialize` is technically not a gate since it contains a reset operation, so it's not invertible.

```

[18]: from qiskit.extensions import Initialize

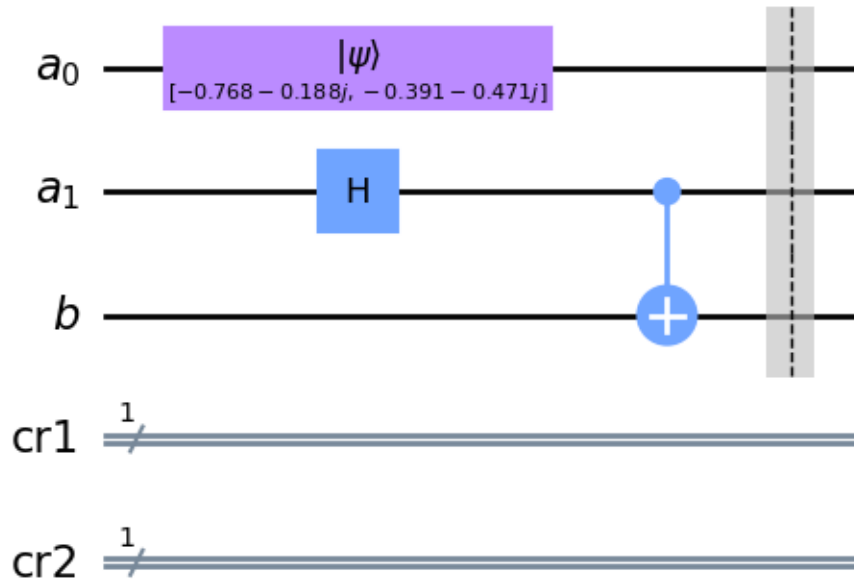
```

```
teleportation_circuit.append(Initialize(psi), [0])
```

[18]: <qiskit.circuit.instructionset.InstructionSet at 0x7f0fa46a2e00>

```
[19]: bellPair(teleportation_circuit, 1, 2)
teleportation_circuit.barrier()
teleportation_circuit.draw(output='mpl')
```

[19]:

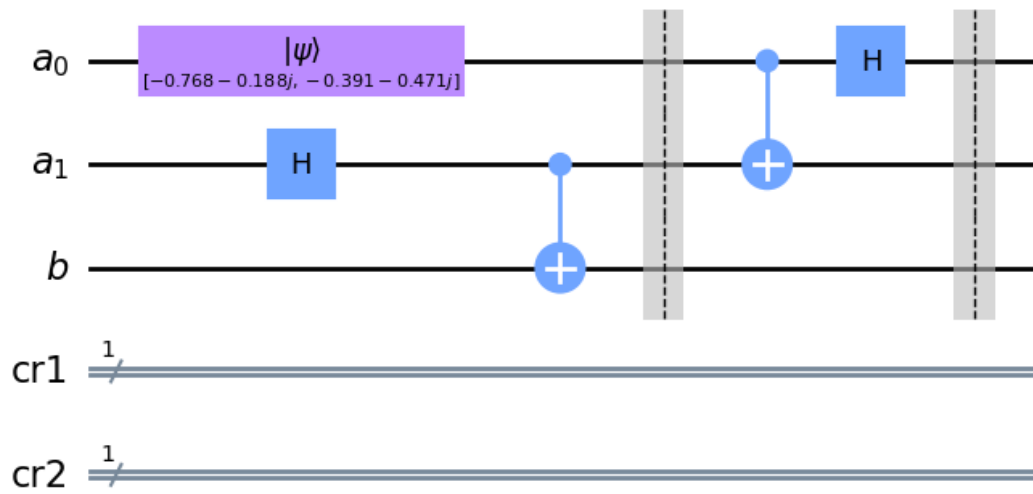


Now Alice and Bob shares the entangled couple $|a_1\rangle, |b\rangle$.

The goal of Alice is to send the qubit $|a_0\rangle$ to Bob. In order to do this, Alice will do:

```
[20]: teleportation_circuit.cx(0,1)
teleportation_circuit.h(0)
teleportation_circuit.barrier()
teleportation_circuit.draw(output='mpl')
```

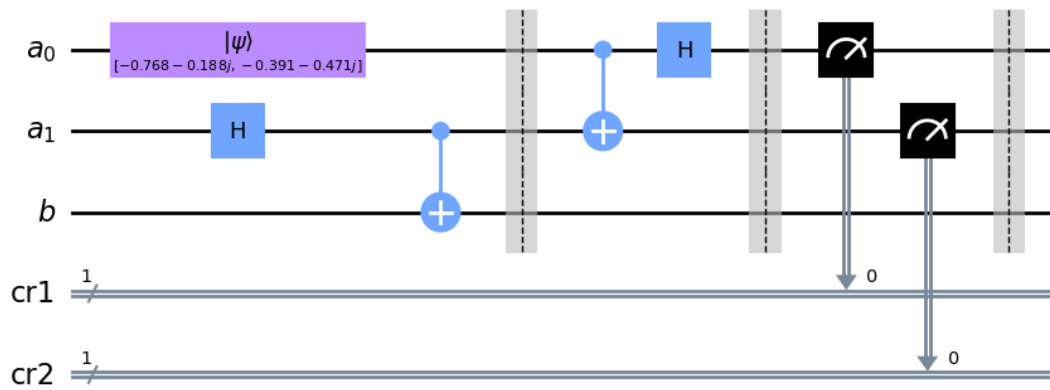
[20]:



Now, Alice needs to send also two classical bits to Bob in order to permit him to adjust the received bit.

```
[21]: teleportation_circuit.measure(0,0)
      teleportation_circuit.measure(1,1)
      teleportation_circuit.barrier()
      teleportation_circuit.draw(output='mpl')
```

[21]:

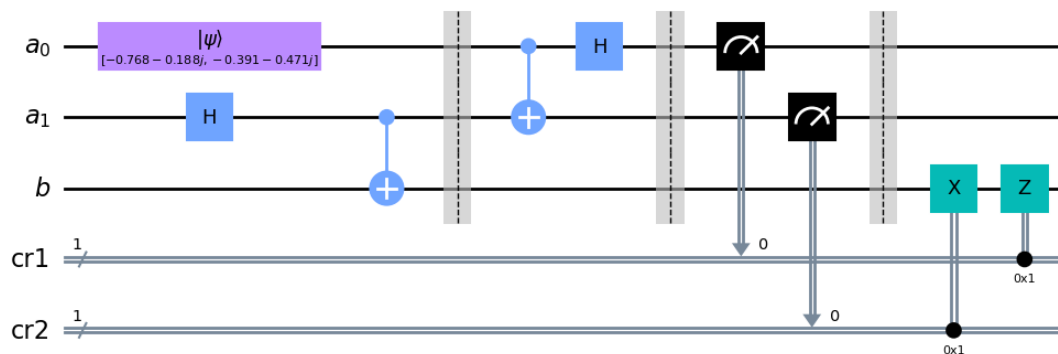


At this moment Bob has all he needs in order to get the original state $|a_0\rangle$.

```
[22]: teleportation_circuit.x(2).c_if(cr2, 1)
      teleportation_circuit.z(2).c_if(cr1, 1)
```

```
teleportation_circuit.draw(output='mpl')
```

[22]:



[23]:

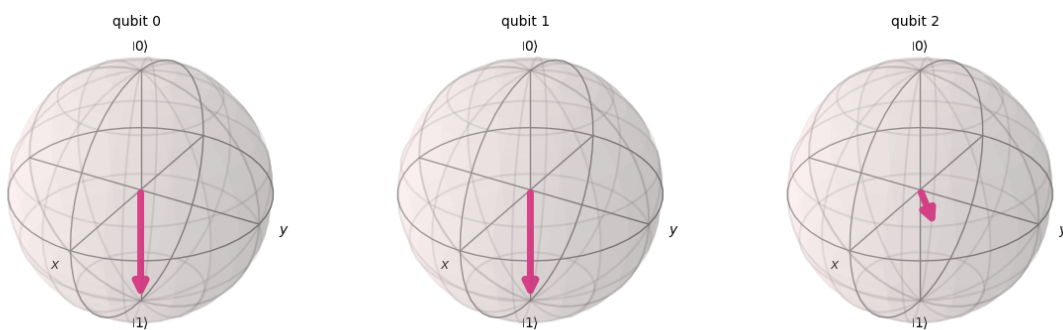
```
from qiskit import Aer
import numpy as np

simulator = Aer.get_backend('statevector_simulator')
result = simulator.run(teleportation_circuit).result()
psi_out = result.get_statevector()

display(array_to_latex(psi_out, prefix="\\psi\\rangle ="))
plot_bloch_multivector(psi_out)
```

$$|\psi\rangle = [0 \ 0 \ 0 \ -0.7682519717 - 0.1879997459i \ 0 \ 0 \ 0 \ -0.3909643146 - 0.4707354972i]$$

[23]:



The circuit is complete: Alice's qubit has correctly been sent to Bob.

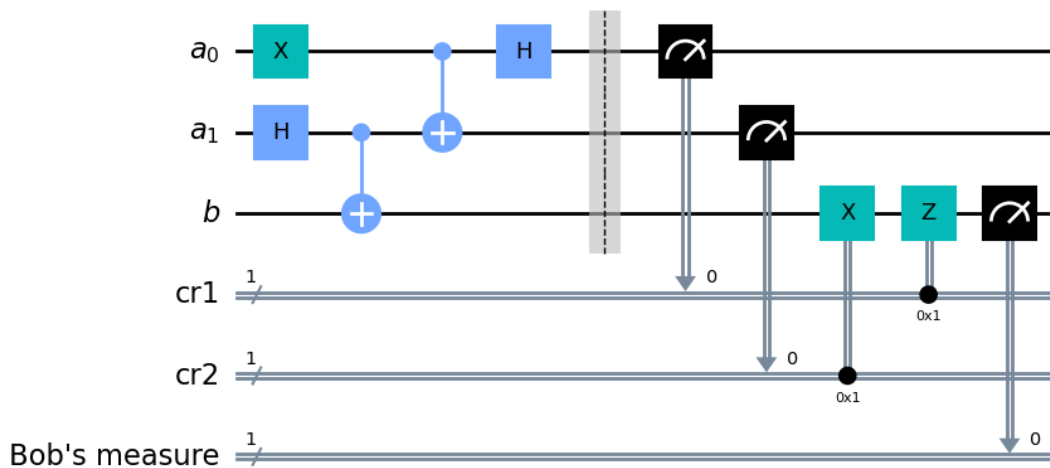
Let's save the circuit in a function, in order to reuse it.

```
[24]: def tpCircuit(initState = None):
    """
    Returns a teleportation circuit with the given initial state
    """
    qr1 = QuantumRegister(2, name='a')
    qr2 = QuantumRegister(1, name='b')
    cr1 = ClassicalRegister(1, name='cr1')
    cr2 = ClassicalRegister(1, name='cr2')
    cr3 = ClassicalRegister(1, name='Bob\'s measure')
    qc = QuantumCircuit(qr1, qr2, cr1, cr2, cr3)
    if initState is not None:
        qc.append(initState, [0])
    else:
        qc.x(0)
    bellPair(qc, 1, 2)
    qc.cx(0,1)
    qc.h(0)
    qc.barrier()
    qc.measure(0,0)
    qc.measure(1,1)
    qc.x(2).c_if(cr2, 1)
    qc.z(2).c_if(cr1, 1)
    return qc
```

2.2 Simulate the teleportation protocol

```
[25]: qc = tpCircuit()
qc.measure(2, 2)
qc.draw(output='mpl')
```

[25]:

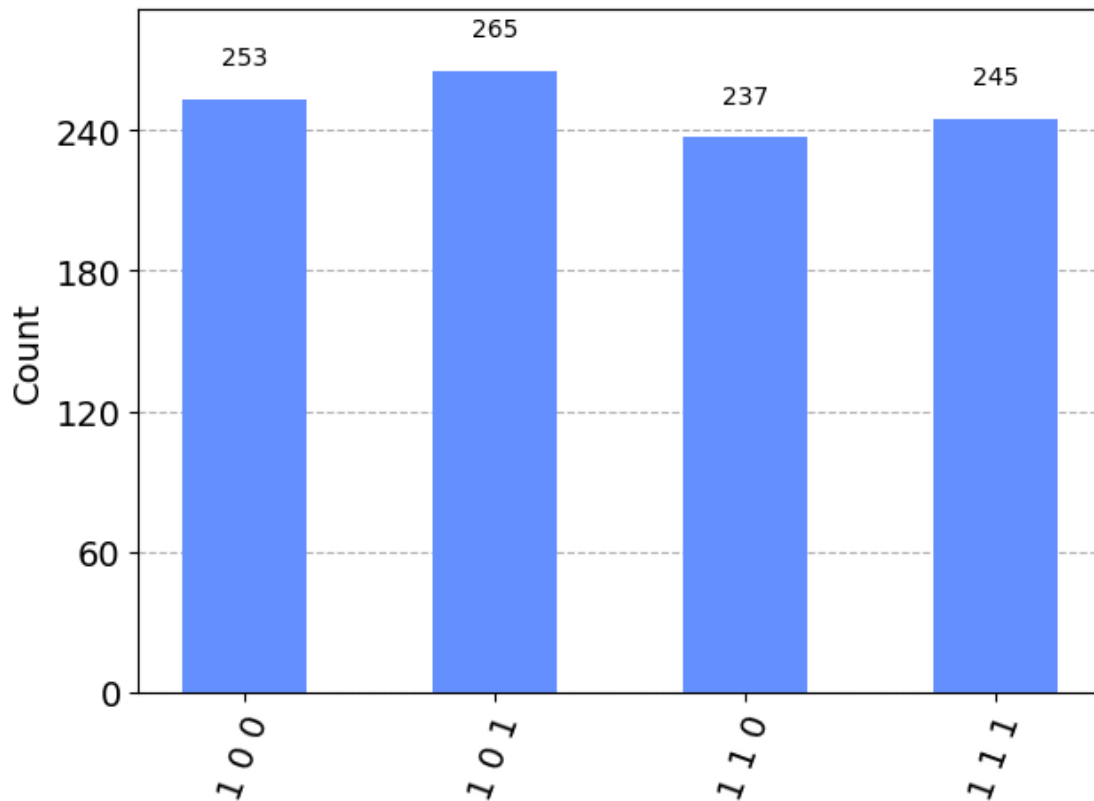



```
[26]: simulator = Aer.get_backend('aer_simulator')
      N = 10**3 # Number of shots
      job = simulator.run(qc, shots=N)
```

```
[27]: from qiskit.visualization import plot_histogram

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

[27]:



As we can see, each string of bits begins with a 1, which means that we have 100% probability to get the one state.

3 Quantum Phase Estimation

3.1 The circuit

We have a goal: given a unitary operator U , we want to estimate θ from $U|\psi\rangle = e^{2\pi\theta}|\psi\rangle$.

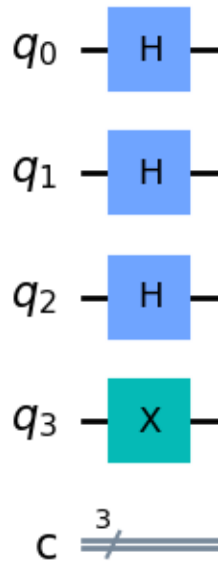
Notice that in this case $|\psi\rangle$ is an eigenvector of eigenvalue $e^{2\pi\theta}$. Since U is unitary, every eigenvalue has norm 1.

We will use three qubits as *counting qubits*, and a fourth one as eigenstate of the unitary operator T . We initialize the last one in $|1\rangle$ by applying the X gate.

```
[1]: from qiskit import QuantumCircuit

qpe_circuit = QuantumCircuit(4, 3)
for qbit in range(3):
    qpe_circuit.h(qbit)
qpe_circuit.x(3)
qpe_circuit.draw(output='mpl')
```

[1]:

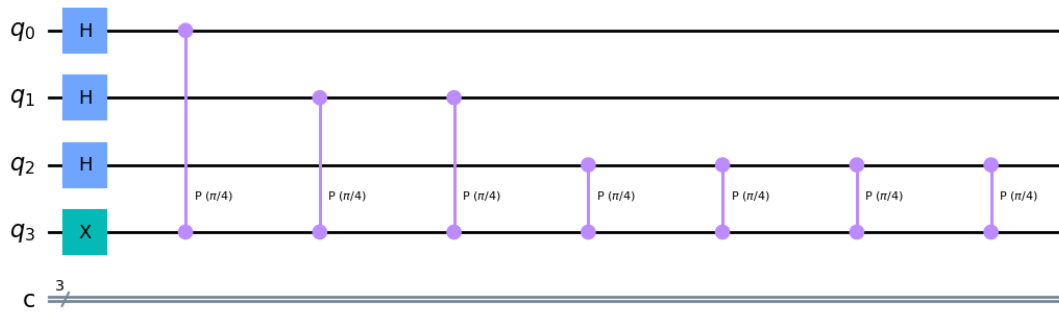


Now we want to perform controlled unitary operations

```
[2]: import numpy as np

repetitions = 1
for counting_qubit in range(3):
    for i in range(repetitions):
        qpe_circuit.cp(np.pi/4, counting_qubit, 3) # This is the Controlled Ugate
    repetitions *= 2
qpe_circuit.draw(output='mpl')
```

[2]:



Now we apply the **Inverse Quantum Fourier Transform** in order to convert the counting register state.

The code for the IQFT is:

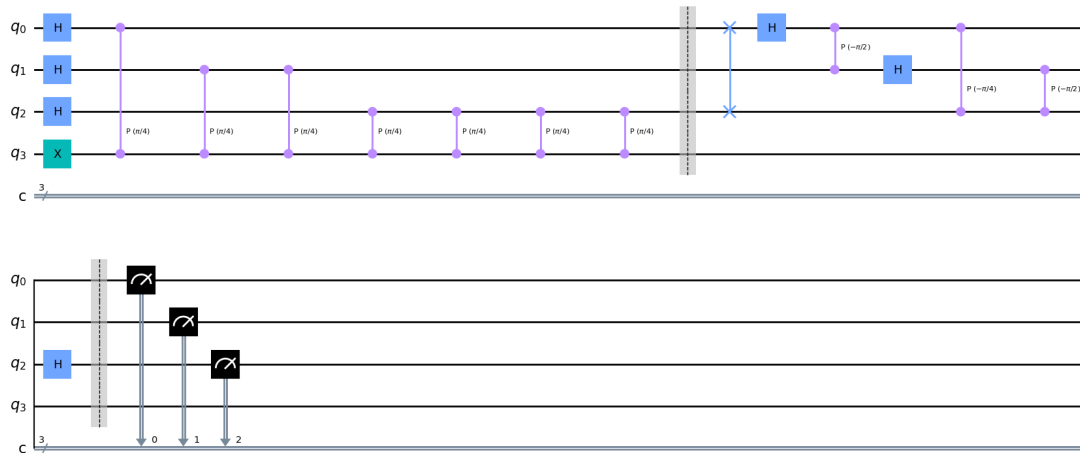
```
[3]: def iqft(qCircuit, n):
    """
    Apply the inverse quantum Fourier transform to the first n qubits in
    qCircuit
    """
    for qubit in range(n//2):
        qCircuit.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qCircuit.cp(-np.pi/float(2**(j-m)), m, j)
        qCircuit.h(j)
```

```
[4]: qpe_circuit.barrier()
      iqft(qpe_circuit, 3)
      qpe_circuit.barrier()

      for n in range(3):
          qpe_circuit.measure(n,n)

      qpe_circuit.draw(output='mpl')
```

[4]:



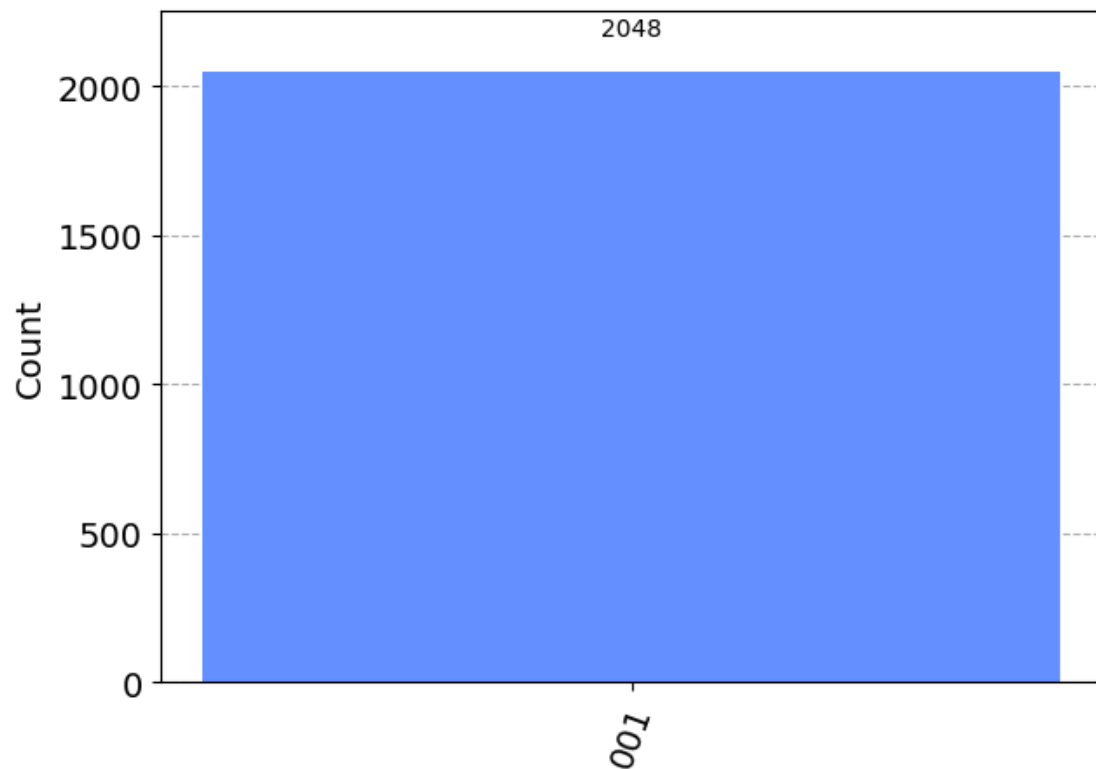
3.2 Simulate the IQFT

```
[5]: from qiskit import Aer, transpile
from qiskit.visualization import plot_histogram

simulator = Aer.get_backend('aer_simulator')
N = 2**11
t_qpe = transpile(qpe_circuit, simulator)

result = simulator.run(t_qpe, shots=N).result()
plot_histogram(result.get_counts())
```

[5]:



Notice that we always get (001) as result: it translates into decimal 1.

In order to get θ we need to divide our result by 2^n , i.e.

$$\theta = \frac{1}{2^3} = \frac{1}{8}$$

as expected.