Practica3_APCC

April 18, 2021

1 La QFT y sumadores cuánticos

Mario Quiñones Pérez

1.1 tKet

En esta práctica profundizaremos en la Transformada Cuántica de Fourier (QFT), que es fundamental para aplicar la phase kickback. Además, aprovecharemos el diseño de este componente para implementar un sumador cuántico.

[1]:
```
from pytket import Circuit
from pytket.extensions.qiskit import AerBackend
import numpy as np
```

1.1.1 QFT

La Transformada Cuántica de Fourier (QFT) es un método para trabajar en el dominio de la fase, esto nos permite aprovechar el fenómeno de la superposición, visto en la práctica anterior. Con esta puerta pasamos del dominio de la amplitud al dominio de la fase para poder así influir en la salida al modificar las amplitudes antes de realizar la medición favoreciendo así aquella salida que nos interesa, en esta práctica la de la suma.

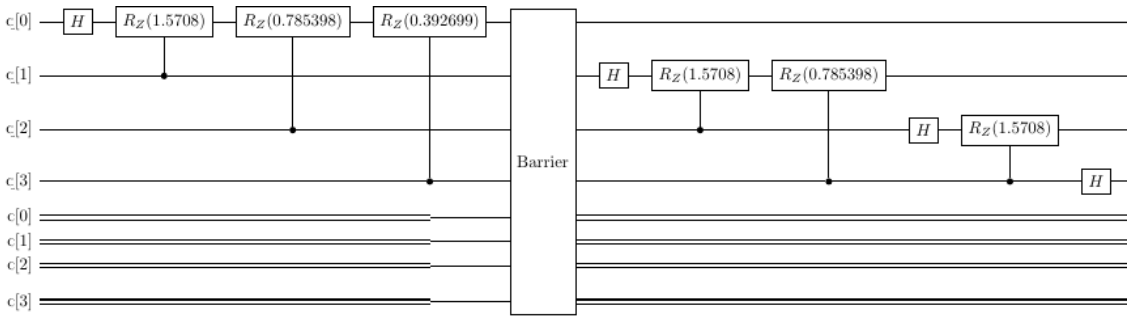
En primer lugar hay que aplicar una puerta Hadamard a todos los qubits y, a continuación, hay que aplicar rotaciones controladas. Dichas rotaciones siguen la siguiente regla: el qubit xi controla una rotación R /2^(i−j) sobre el qubit objetivo xi−j.

[2]:
```
##QFT

# Esta función crea una transformada cuántica de Fourier para un número de␣
 ↪qubits dado,
# si no se especifica se creará para dos qubits
def QFT(qc, qbits = 2):
    qbits2 = qbits - 1

    for i in range(qbits):
        # Por cada qubit se aplica una puerta hadamard
        qc.H(i)
        # Y se calcula en control en qué posición se empieza el control de␣
 ↪rotaciones
```

```
        control = i + 1

        # Se realizan dichas rotaciones , tantas como qbits2 diga
        for j in range(qbits2):
            qc.CRz(np.pi/np.power(2, (j+1)), control, i)
            control = control + 1

        # Se añade entre grupos de rotaciones por claridad
        if (i < qbits - 1):
            c.add_barrier([0,1,2,3])
        # Se calcula cuantas rotaciones tendrá el siguiente qubit (una menos
 ↪que la anterior)
        qbits2 = qbits2 - 1
```

En esta práctica y para la representación de los circuitos en tket se ha utilizado tanto la función to_latex_file('name.tex') como la aplicación TexWorks para la creación de pdfs con las imágenes de los circuitos que luego se añadirán al código como png. Se adjuntaran los png devueltos para la correcta visualización de los notebook.

[3]:
```
c = Circuit(4,4)
QFT(c, qbits = 4)

# Función utilizada para recuperar el circuito como archivo latex, para pasarlo
 ↪pdf y por ultimo a png.
c.to_latex_file('QFT.tex')
```



1.1.2 AQFT

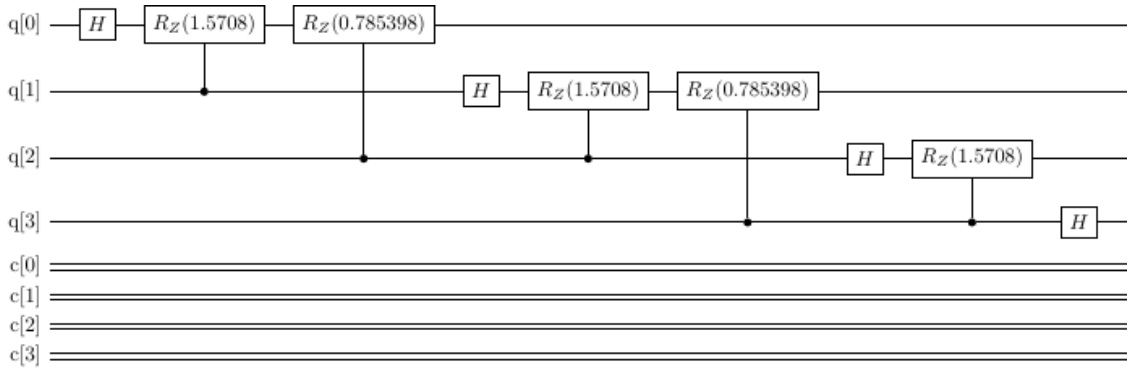
Como se ha mencionado anteriormente, la QFT requiere realizar rotaciones controladas. El ángulo de rotación puede llegar a ser muy pequeño si tenemos muchos qubits de entrada, por lo que controlar físicamente dichas rotaciones con precisión absoluta es muy complicado.

Debido a que realmente estas rotaciones tan pequeñas no contribuyen en gran medida al resultado final, se puede utilizar la QFT aproximada (AQFT). Con la que se permite un número máximo de rotaciones controladas, por ejemplo 2, como aparece en la función AQFT.

[4]:
```
##AQFT
```

```
# Esta función crea una transformada cuántica de Fourier aproximada para un␣
 ↪número de qubits dado y
# con una cantidad máxima de rotaciones 'maxc', si no se especifica se creará␣
 ↪para dos qubits y con maxc = 2
def AQFT(qc, qbits = 2, maxc = 2):
    qbits2 = qbits - 1

    for i in range(qbits):
        qc.H(i)
        control = i + 1

        for j in range(qbits2):
            # Solo se hacen más rotaciones controladas si no nos pasamos del␣
 ↪máximo dado
            if(j < maxc):
                qc.CRz(np.pi/np.power(2, (j+1)), control, i)
                control = control + 1
        qbits2 = qbits2 - 1
```

[5]:
```
c = Circuit(4,4)
AQFT(c, qbits = 4)

# Función utilizada para recuperar el circuito como archivo tex para pasarlo␣
 ↪pdf y luego a png
c.to_latex_file('AQFT.tex')
```



1.1.3 Sumador Cuántico

Dados dos operandos de entrada A y B, de n qubits cada uno, la manera de realizar dicha operación es conservar uno de los dos, en este caso B y en el otro acumular los qubits para que el resultado sea A + B, así dicho circuito es reversible ya que tenemos B para aplicar la resta y recuperar A.

Este sumador de Draper tiene tres fases:

- Aplicar la QFT o AQFT a A para pasar al dominio de la fase y así con el sumador y sus rotaciones controladas poder influir en la salida del circuito

- La suma en el dominio de la fase. Realizada mediante rotaciones controladas, siguiendo la regla en la que Bj aplica una rotación controlada al qubit Ai con una rotación de /(2^j)

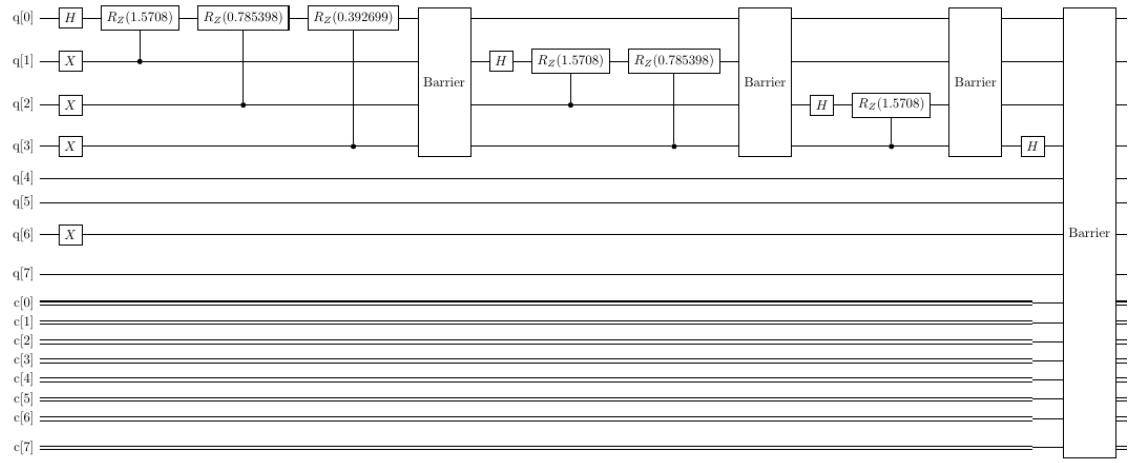
- La QFT inversa (IQFT) al resultado de la suma que es lo mismo que aplicar la QFT pero las puertas en el orden contrario a esta. Se utiliza para pasar de nuevo al dominio de la amplitud y medir los resultados obtenidos del sumador que modifica la fase.

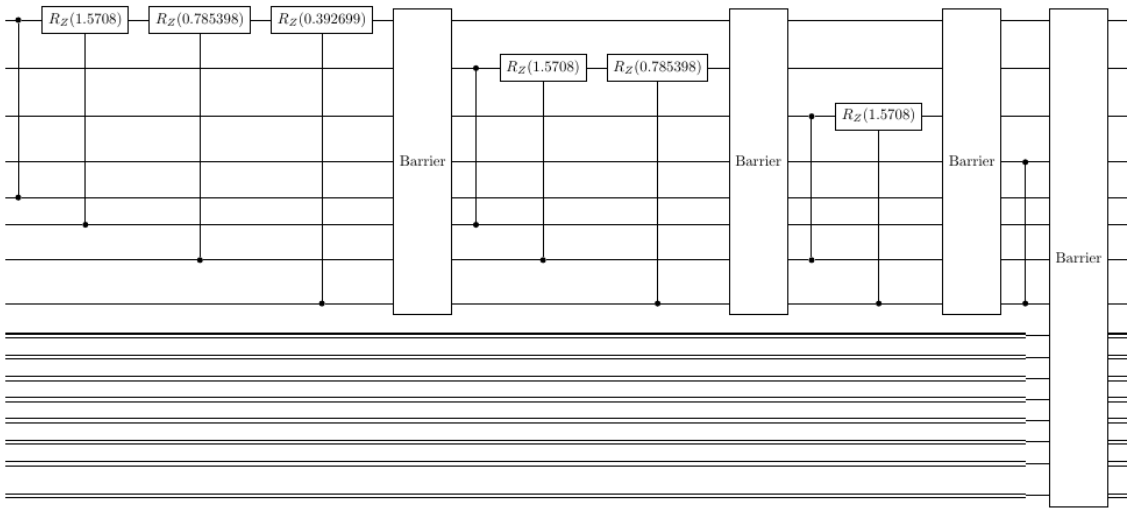
[6]:
```
##adder

# Esta se realiza mediante rotaciones controladas a dos elementos con el mismo
# número de qubits A y B siguiendo la regla en la que Bj aplica una rotación
# controlada al qubit Ai con una rotación de /(2^j)


# Esta función crea un sumador para un número de qubits dado, si no se␣
 ↪especifica
# se creará para un sumador de dos qubits
def adder(qc, qbits = 2):
    qbits2 = qbits - 1

    for i in range(qbits):
        # Se calcula en control en que qubit empezará el control de los qubits
        control = i + qbits
        # Se aplica la rotación Z a cada qubit de A con el que está en su misma␣
 ↪posición en B
        qc.CZ(control,i)

        # Se realizan el resto de rotaciones por qubit tantas como diga qbits2
        for j in range(qbits2):
            control = control + 1
            qc.CRz(np.pi/np.power(2, (j+1)), control, i)

        # Se añade una barrera al final de cada grupo de rotaciones por claridad
        if (i < qbits - 1):
            c.add_barrier([0,1,2,3,4,5,6,7])
        # Se calcula cuantas rotaciones tendrá el siguiente qubit (una menos␣
 ↪que la anterior)
        qbits2 = qbits2 - 1
```

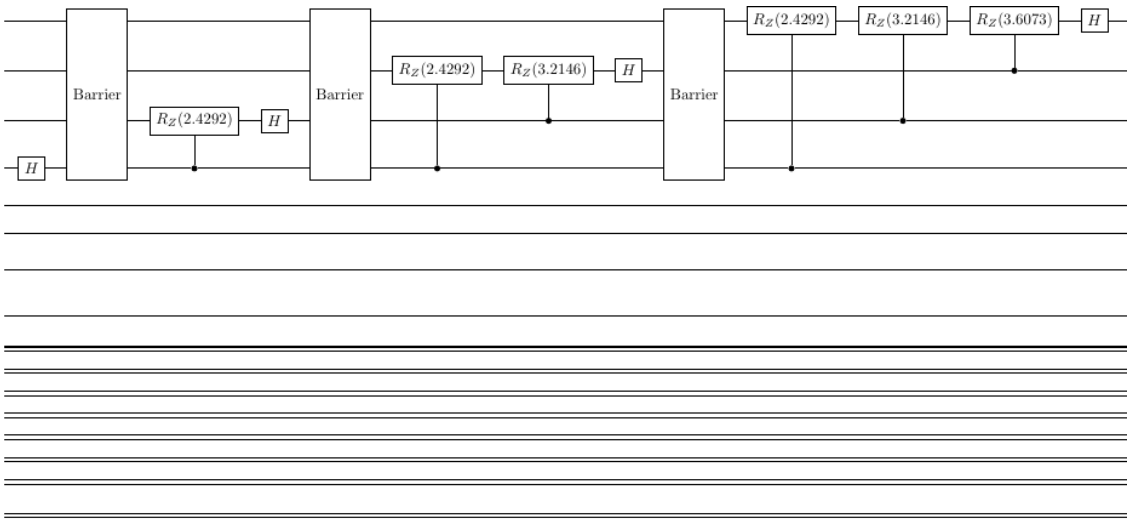
[7]:
```
##IQFT

# Esta función crea una transformada cuántica de Fourier inversa para un número␣
 ↪de qubits dado,
```

```
# si no se especifica se creará para dos qubits. Ademas añadiremos un maximo de␣
 ↪rotaciones
# controladas 'maxc' para poder utilizar esta función con la AQFT tambien
def IQFT(qc, qbits = 2, maxc = 2):
    qbits2 = 0

    for i in range(qbits):
        control = qbits - 1

        for j in range(qbits2):
            if(j < maxc):
                qc.CRz(-np.pi/np.power(2, (j+1)), control, qbits - i - 1)
                control = control - 1

        qc.H(qbits - i - 1)
        if (i < qbits - 1):
            c.add_barrier([0,1,2,3])
        qbits2 = qbits2 + 1
```

[8]:
```
c = Circuit(8,8)

c.X(1)
c.X(2)
c.X(3)
c.X(6)

QFT(c, qbits = 4)

c.add_barrier([0,1,2,3,4,5,6,7], [0,1,2,3,4,5,6,7])

adder(c, qbits = 4)

c.add_barrier([0,1,2,3,4,5,6,7], [0,1,2,3,4,5,6,7])

IQFT(c, qbits = 4, maxc = 3)

# Función utilizada para recuperar el circuito como archivo tex para pasarlo␣
 ↪pdf y luego a png
c.to_latex_file('complete_adder.tex')
```


q[0]
q[1]
q[2]
q[3]
q[4]
q[5]
q[6]
q[7]
c[0]
c[1]
c[2]
c[3]
c[4]
c[5]
c[6]
c[7]
H
X
$R_Z(1.5708)$
$R_Z(0.785398)$
$R_Z(0.392699)$
Barrier


$R_Z(1.5708)$
$R_Z(0.785398)$
$R_Z(0.392699)$
Barrier


H
Barrier
$R_Z(2.4292)$
$R_Z(3.2146)$
$R_Z(3.6073)$

1.1.4 Simulaciones

Se realiza la simulación de la misma suma 0111 y 1000 con la AQFT y la QFT y vemos que ambas dan siempre la solución correcta ya que la AQFT es una aproximación bastante buena da la QFT en la mayoría de situaciones a partir de un determinado número de qubits ya que las rotaciones que se harán serán muy pequeñas como para afectar a la solución

[9]:
```
c = Circuit(8,8)

c.X(1)
c.X(2)
c.X(3)
c.X(4)

QFT(c, qbits = 4)
adder(c, qbits = 4)
IQFT(c, qbits = 4)
c.measure_all()  # medir todos los qbits(1 en este caso)

b = AerBackend()                # conectar al backend
b.compile_circuit(c)            # compilar el circuito para satisfacer las␣
 ↪condiciones del backend
handle = b.process_circuit(c, n_shots = 1000)  # ejecutar 1000 veces
counts = b.get_result(handle).get_counts()   # recuperar los resultados
print(counts)
```

```
Counter({(1, 1, 1, 1, 1, 0, 0, 0): 1000})
```

[10]:
```
c = Circuit(8,8)

c.X(1)
c.X(2)
c.X(3)
c.X(4)

AQFT(c, qbits = 4)
adder(c, qbits = 4)
IQFT(c, qbits = 4)
c.measure_all()  # medir todos los qbits(1 en este caso)

b = AerBackend()                # conectar al backend
b.compile_circuit(c)            # compilar el circuito para satisfacer las␣
 ↪condiciones del backend
handle = b.process_circuit(c, n_shots = 1000)  # ejecutar 1000 veces
counts = b.get_result(handle).get_counts()   # recuperar los resultados
print(counts)
```

```
Counter({(1, 1, 1, 1, 1, 0, 0, 0): 1000})
```

1.2 Quiskit

El codigo para la creación de este circuito en qiskit sera casi igual que en tket con pequeños cambios como minusculas en vez de mayusculas en las puertas, por tanto las partes iguales no serán comentadas ya que ya lo fueron en la parte anterior

[1]:
```
from qiskit import QuantumCircuit
from qiskit import IBMQ, Aer, transpile, assemble
from qiskit.visualization import plot_histogram
from qiskit.extensions import Initialize
import numpy as np
```

1.2.1 QFT

[2]:
```
def QFT(qc, qbits = 2):
    qbits2 = qbits - 1

    for i in range(qbits):
        qc.h(i)
        control = i + 1

        for j in range(qbits2):
            qc.crz(np.pi/np.power(2, (j+1)), control, i)
            control = control + 1

        qc.barrier()
        qbits2 = qbits2 - 1
```

[3]:
```
qc = QuantumCircuit(4)
QFT(qc, qbits = 4)
qc.draw()
```

[3]:
```
                                                    »
q_0:   H   RZ(/2)   RZ(/4)   RZ(/8)                  »
                                              »
q_1:                        H   RZ(/2)   RZ(/4)   »
                                                    »
q_2:                                        »
                                                          »
q_3:                                        »
                                                           »

«
«q_0:
«
«q_1:
«
«q_2:   H   RZ(/2)
«
```

```
«q_3:             H
«
```

1.2.2 AQFT

[4]:
```
def AQFT(qc, qbits = 2, maxc = 2):
    qbits2 = qbits - 1

    for i in range(qbits):
        qc.h(i)
        control = i + 1

        for j in range(qbits2):
            if(j < maxc):
                qc.crz(np.pi/np.power(2, (j+1)), control, i)
                control = control + 1

        qc.barrier()
        qbits2 = qbits2 - 1
```

[5]:
```
qc = QuantumCircuit(4)
AQFT(qc, qbits = 4)
qc.draw()
```

[5]:
```
                                                    »
q_0:   H   RZ(/2)   RZ(/4)                          »
                                                »
q_1:                  H   RZ(/2)   RZ(/4)       »
                                                      »
q_2:                                 H  »
                                                          »
q_3:                                   »
                                                             »
«
«q_0:
«
«q_1:
«
«q_2:   RZ(/2)
«
«q_3:          H
«
```

1.2.3 Sumador Cuántico

[6]:
```
def adder(qc, qbits = 2):
    qbits2 = qbits - 1

    for i in range(qbits):
        control = i + qbits
        qc.cz(control,i)

        for j in range(qbits2):
            control = control + 1
            qc.crz(np.pi/np.power(2, (j+1)), control, i)

        qc.barrier()
        qbits2 = qbits2 - 1
```

[7]:
```
def QFTI(qc, qbits = 2, maxc = 2):
    qbits2 = 0

    for i in range(qbits):
        control = qbits - 1

        for j in range(qbits2):
            if(j < maxc):
                qc.crz(-np.pi/np.power(2, (j+1)), control, qbits - i - 1)
                control = control - 1

        qc.h(qbits - i - 1)
        qc.barrier()
        qbits2 = qbits2 + 1
```

[8]:
```
# Dibujamos el circuito de lsumador completo con 8 qbits

qc = QuantumCircuit(8)

QFT(qc, qbits = 4)

adder(qc, qbits = 4)

QFTI(qc, qbits = 4, maxc = 3)

qc.draw()
```

[8]:
```
                                                  »
q_0:  H  RZ(π/2)  RZ(π/4)  RZ(π/8)                »
                                                  »
q_1:                     H  RZ(π/2)  RZ(π/4)      »
```

```
                                                                  »
q_2:                                        »
                                                                     »
q_3:                                        »
                                                                      »
q_4:                                        »
                                                                      »
q_5:                                        »
                                                                      »
q_6:                                        »
                                                                      »
q_7:                                        »
                                                                      »
«                                                          »
«q_0:                  RZ(/2)   RZ(/4)   RZ(/8)     »
«                                                              »
«q_1:                                       »
«                                                                 »
«q_2:  H   RZ(/2)                               »
«                                                              »
«q_3:            H                            »
«                                                                    »
«q_4:                                       »
«                                                                       »
«q_5:                                       »
«                                                                    »
«q_6:                                       »
«                                                                     »
«q_7:                                       »
«                                                                     »
«                                                                        »
«q_0:                                         »
«                                                                  »
«q_1:  RZ(/2)   RZ(/4)                                  »
«                                                        »
«q_2:                RZ(/2)            RZ(-/2)   H  »
«                                                                 »
«q_3:                            H            »
«                                                                    »
«q_4:                                         »
«                                                                      »
«q_5:                                         »
«                                                                      »
«q_6:                                         »
«                                                                       »
«q_7:                                         »
«                                                                         »
```

```
«                                                                  »
«q_0:                      RZ(-/2)   RZ(-/4)   RZ(-/8)  »
«                                                        »
«q_1:    RZ(-/2)   RZ(-/4)   H                           »
«                                                              »
«q_2:                                              »
«                                                                    »
«q_3:                                              »
«                                                                     »
«q_4:                                              »
«                                                                     »
«q_5:                                              »
«                                                                     »
«q_6:                                              »
«                                                                     »
«q_7:                                              »
«                                                                     »
«
«q_0:   H
«
«q_1:
«
«q_2:
«
«q_3:
«
«q_4:
«
«q_5:
«
«q_6:
«
«q_7:
«
```

[9]:
```
# Dibujamos el circuito del sumador completo con 8 qbits usando una AQFT

qc = QuantumCircuit(8)

AQFT(qc, qbits = 4)

adder(qc, qbits = 4)

QFTI(qc, qbits = 4)

qc.draw()
```

[9]:
```
                                                                  »
q_0:  H   RZ(/2)   RZ(/4)                      »
                                                »
q_1:                H   RZ(/2)   RZ(/4)        »
                                                      »
q_2:                                H  »
                                                            »
q_3:                                   »
                                                               »
q_4:                                   »
                                                               »
q_5:                                   »
                                                               »
q_6:                                   »
                                                               »
q_7:                                   »
                                                               »
«                                                        »
«q_0:               RZ(/2)   RZ(/4)   RZ(/8)     »
«                                                        »
«q_1:                                  »
«                                                           »
«q_2:  RZ(/2)                              »
«                                                        »
«q_3:         H                              »
«                                                           »
«q_4:                                  »
«                                                              »
«q_5:                                  »
«                                                               »
«q_6:                                  »
«                                                               »
«q_7:                                  »
«                                                               »
«                                                                        »
«q_0:                                        »
«                                                              »
«q_1:  RZ(/2)   RZ(/4)                                  »
«                                                         »
«q_2:                RZ(/2)            RZ(-/2)   H  »
«                                                              »
«q_3:                            H            »
«                                                                     »
«q_4:                                        »
«                                                                        »
«q_5:                                        »
«                                                                        »
```

```
«q_6:                                          »
«                                                                                »
«q_7:                                          »
«                                                                                 »
«
«q_0:                      RZ(-/2)    RZ(-/4)    H
«
«q_1:     RZ(-/2)    RZ(-/4)    H
«
«q_2:
«
«q_3:
«
«q_4:
«
«q_5:
«
«q_6:
«
«q_7:
«
```

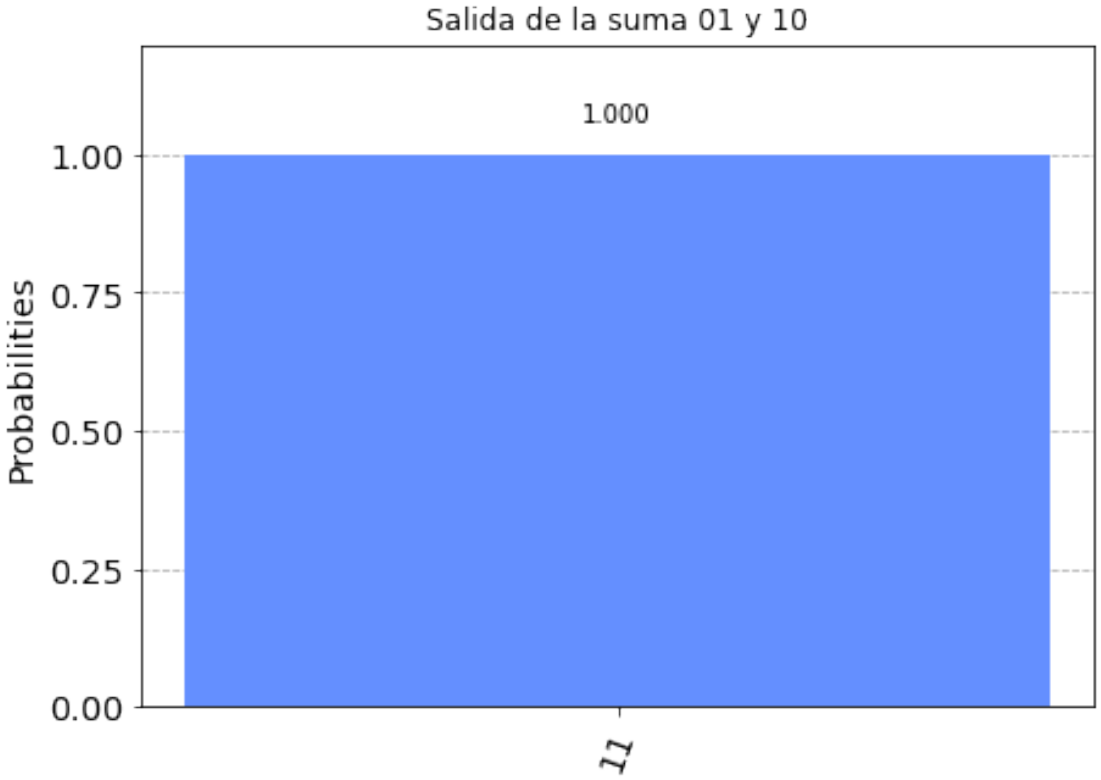
1.2.4 Simulaciones

En las siguientes simulaciones comprovamos que en condiciones ideales el cicuito con QFT relizara la suma perfectamente en todas las situaciones, pero la AQFT, aunque la mayoría sean correctas, no todas sus soluciones seran las esperadas, esto es debido a que no se realizan todas las rotaciones necesarias sino que solo una cantidad fija de ellas como maximo (2 en este caso), en una simulación ideal dara paso a problemas , lo que será compensado en una simulación en backends reales ya que estas rotaciones que se obvian son muy pequeñas y dificiles de implementar lo que hace a la AQFT una buena solución, aunque no en simulaciones ideales

[10]:
```
qc = QuantumCircuit(4, 2)

# Inicializamos a uno varias variables para realizar una suma que implique␣
 ↪tanto sumas de unos y ceros como de unos y unos
# Representan A = 01 B = 10
qc.x(1)
qc.x(2)


# Aplicamos la QFT al circuito
QFT(qc, qbits = 2)
# Usamos un sumador de Draper
adder(qc, qbits = 2)
# Aplicamos la QFT inversa para devolver los valores
QFTI(qc, qbits = 2)
```

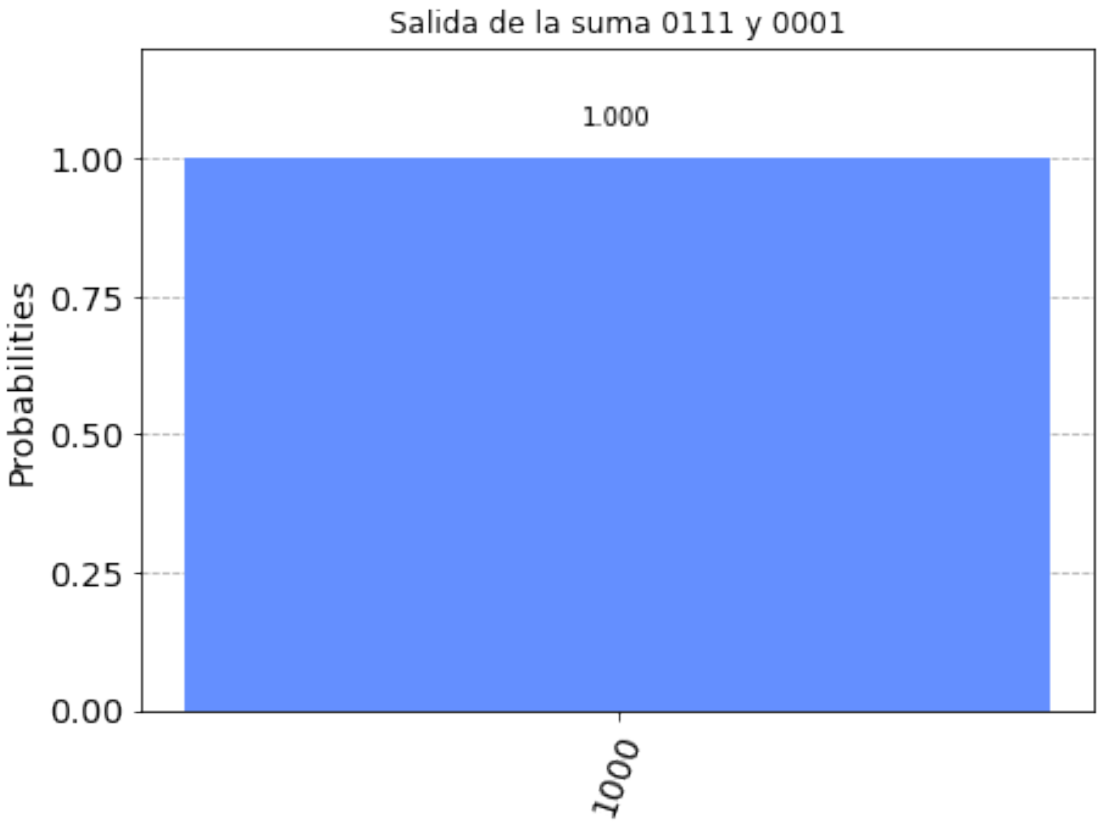
```
# Medimos los bits de manera que salgan en orden [q0,q1] y no al reves
qc.measure([0,1], [1,0])

# Simulamos el circuito 1000 veces y mostramos el resultado en un histograma
qasm_sim = Aer.get_backend('qasm_simulator')
qc = transpile(qc, qasm_sim)
qobj = assemble(qc)


result = qasm_sim.run(qc).result()

counts = result.get_counts(qc)
plot_histogram(counts, title='Salida de la suma 01 y 10')
```

[10]:



[11]:
```
qc = QuantumCircuit(8,4)

# Inicializamos a uno varias variables para realizar una suma que implique␣
 ↪tanto sumas de unos y ceros como de unos y unos
# Representan A = 0111 B = 0001
qc.x(1)
```

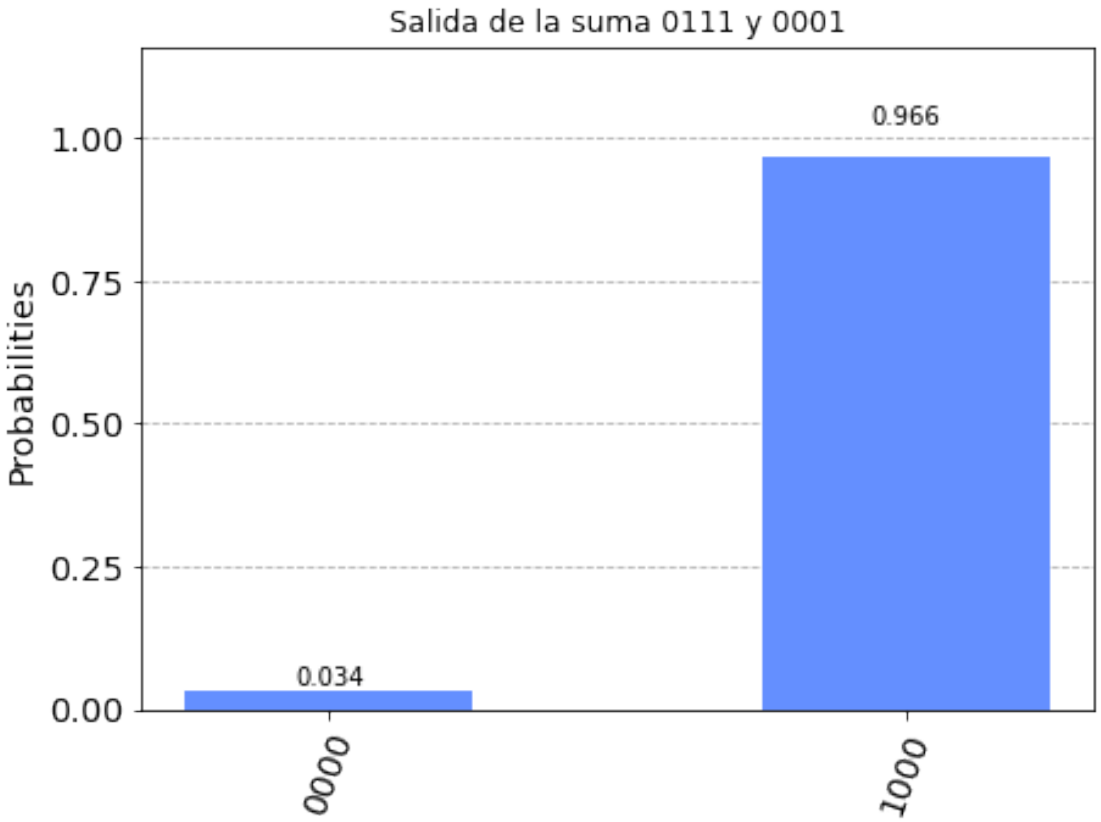
[11]:
```
qc.x(2)
qc.x(3)
qc.x(7)

# Aplicamos la QFT al circuito
QFT(qc, qbits = 4)

# Usamos un sumador de Draper
adder(qc, qbits = 4)

# Aplicamos la QFT inversa para devolver los valores
QFTI(qc, qbits = 4)

# Medimos los bits de manera que salgan en orden [q0,q1,q2,q3] y no al reves
qc.measure([0,1,2,3], [3,2,1,0])


# Simulamos el circuito 1000 veces y mostramos el resultado en un histograma
qasm_sim = Aer.get_backend('qasm_simulator')
qc = transpile(qc, qasm_sim)
qobj = assemble(qc)


result = qasm_sim.run(qc).result()

counts = result.get_counts(qc)
plot_histogram(counts, title='Salida de la suma 0111 y 0001')
```



[12]:
```
qc = QuantumCircuit(8,4)

# Inicializamos a uno varias variables para realizar una suma que implique␣
 ↪tanto sumas de unos y ceros como de unos y unos
# Representan A = 0111 B = 0001
qc.x(1)
qc.x(2)
qc.x(3)
qc.x(7)

# Aplicamos la AQFT al circuito con un maximo de dos rotaciones
AQFT(qc, qbits = 4)

# Usamos un sumador de Draper
adder(qc, qbits = 4)

# Aplicamos la QFT inversa para devolver los valores
QFTI(qc, qbits = 4)

# Medimos los bits de manera que salgan en orden [q0,q1,q2,q3] y no al reves
```

```
qc.measure([0,1,2,3], [3,2,1,0])


# Simulamos el circuito 1000 veces y mostramos el resultado en un histograma
qasm_sim = Aer.get_backend('qasm_simulator')
qc = transpile(qc, qasm_sim)
qobj = assemble(qc)


result = qasm_sim.run(qc).result()

counts = result.get_counts(qc)
plot_histogram(counts, title='Salida de la suma 0111 y 0001')
```

[12]:



1.2.5 Ejecuciones en backends reales

Ya que no tenemos disponibles backends reales con 8 qubits que no tengan menos de 16000 trabjaos encolados realizaremos la suma de 2 qubits. Para ello creamos dos circuitos uno con QFT y otro con AQFT de maximo numero de rotaciones 1 para que se diferencie con la QFT.

Como observamos en los resultados podemos ver que la AQFT no reduce la cantidad

de aciertos, sino que en este caso los a aumentado hasta un 12% lo que nos demuestra que en la practica la AQFT, aunque tenga menos rotaciones controladas, no es una mala aproximación de la misma y da buenos resultados

[13]:
```
qc = QuantumCircuit(4, 2)

qc.x(1)
qc.x(2)
QFT(qc, qbits = 2)
adder(qc, qbits = 2)
QFTI(qc, qbits = 2)

qc.measure([0,1], [0, 1])
```

[13]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x2d07c17f4f0>
```

[14]:
```
qca = QuantumCircuit(4, 2)

qca.x(1)
qca.x(2)
AQFT(qca, qbits = 2, maxc = 1)
adder(qca, qbits = 2)
QFTI(qca, qbits = 2)

qca.measure([0,1], [0, 1])
```

[14]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x2d07c19bdc0>
```

[15]:
```
from qiskit import IBMQ
from qiskit.providers.ibmq import least_busy
IBMQ.
 ↪save_account('55dbb1b5e08af7c7c6ec803a2770842c2f2e9b16b557e0a3bf6e5025a41316a162a1655901146
 ↪overwrite=True)
shots = 1024

# Load local account information
IBMQ.load_account()
# Get the least busy backend
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x: x.configuration().
 ↪n_qubits >= 2
                                       and not x.configuration().simulator
                                       and x.status().operational==True))
print("least busy backend: ", backend)
# Run our circuit
t_qc = transpile(qc, backend, optimization_level=3)
qobj = assemble(t_qc)
job = backend.run(qobj)
```

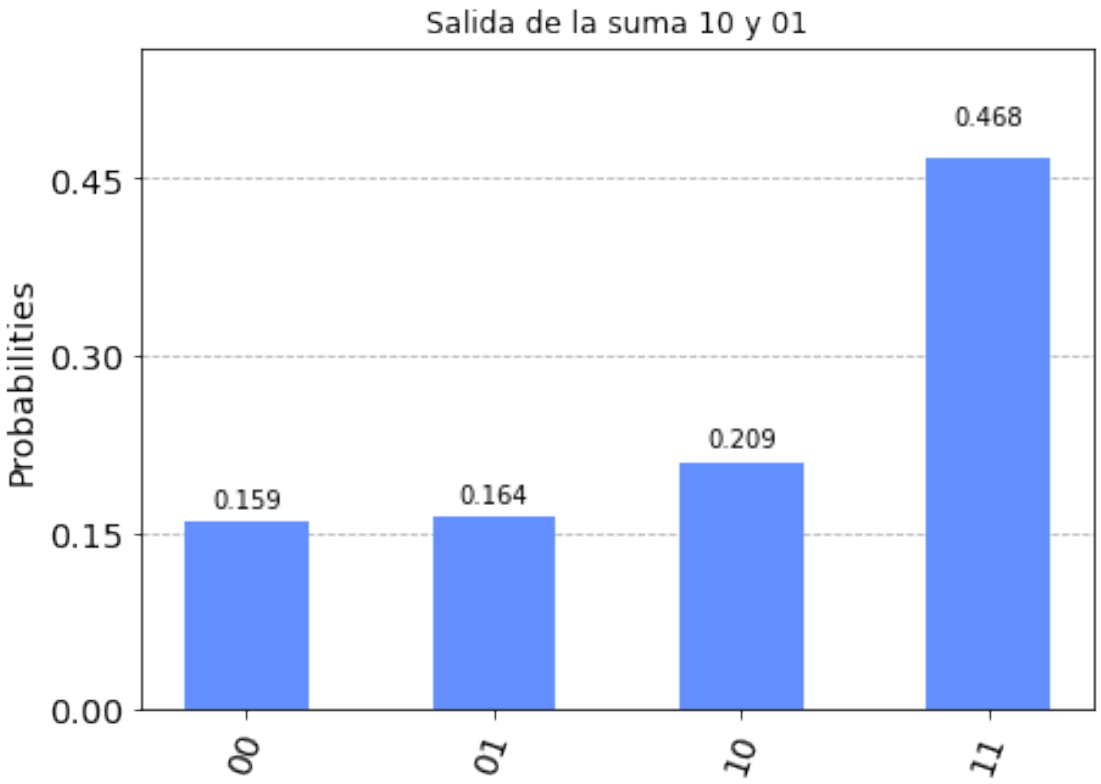
```
least busy backend:  ibmqx2
```

[16]:
```
# Monitoring our job
from qiskit.tools.monitor import job_monitor
job_monitor(job)
```

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

[17]:
```
# Plotting our result
result = job.result()
plot_histogram(result.get_counts(qc), title='Salida de la suma 10 y 01')
```

[17]:



[18]:
```
from qiskit import IBMQ
from qiskit.providers.ibmq import least_busy
shots = 1024

# Load local account information
IBMQ.load_account()
# Get the least busy backend
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x: x.configuration().
 ↪n_qubits >= 2
```

```
                                        and not x.configuration().simulator
                                        and x.status().operational==True))
print("least busy backend: ", backend)
# Run our circuit
t_qca = transpile(qca, backend, optimization_level=3)
qobj = assemble(t_qca)
job = backend.run(qobj)
```

```
least busy backend:  ibmqx2
```

[19]:
```
# Monitoring our job
from qiskit.tools.monitor import job_monitor
job_monitor(job)
```

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

[20]:
```
# Plotting our result
result = job.result()
plot_histogram(result.get_counts(qca), title='Salida de la suma 10 y 01')
```

[20]: