Nota 5: Aritmética basada en la *QFT*

El operador *QFT* ([1], Sec.7.3) transforma el estado básico $|y\rangle = |y_{n-1} y_{n-2} \ \dots y_0\rangle$ de un registro de n qubits en un estado de superposición

$$QFT|y\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{2\pi i}{N} yk} \, |k_{n-1} k_{n-2} \dots k_0\rangle, \, N = 2^n, \qquad (1)$$

en el cual k e y son los números naturales representados en numeración binaria por los estados básicos $|k\rangle = |k_{n-1} \, k_{n-2} \ \dots \ k_0\rangle$ e $|y\rangle$. Este estado de superposición contiene la misma información que el estado básico $|y\rangle$ inicial: aplicando a $QFT|y\rangle$ la transformada inversa se obtiene el estado básico $|y\rangle$. En el estado transformado (1), la información inicial (el estado básico $|y\rangle$) se ha trasladado a los N desfases $\frac{2\pi}{N} yk$ asociados a los N equiprobables estados básicos $|k\rangle = |k_{n-1} \, k_{n-2} \ \dots k_0\rangle$. Esta transformación, del dominio de los estados básicos, interpretados como números en base 2, al dominio de los desfases, sugiere un método de cálculo: transformar números en ángulos, ejecutar operaciones sobre ángulos, y volver al dominio de los números utilizando la transformada inversa. Para las operaciones sobre ángulos se supone que se dispone de puertas cuánticas que ejecutan rotaciones controladas.

1. Suma

En esta sección se describe el método propuesto en [2] para la suma.

1.1. Algoritmo

Considérense dos números naturales x e y, de n bits, representados en numeración binaria por los estados básicos

$$|x\rangle = \ |x_{n-1} \, x_{n-2} \ \dots x_0\rangle \text{ e } |y\rangle = |y_{n-1} y_{n-2} \ \dots y_0\rangle$$

de dos registros de n qubits. Defínase una operación CZ (CZ generalizada) que, al producto $|x\rangle\times|k\rangle$ de dos estados básicos, asocia el estado

$$CZ|x\rangle\times|k\rangle = e^{\frac{2\pi i}{N}xk}|x\rangle\times|k\rangle. \quad (2)$$

Obsérvese que si $n = 1$ y, por tanto, $N = 2$, entonces, según (2),

$$CZ|00\rangle = |00\rangle,\ CZ|01\rangle = |01\rangle,\ CZ|10\rangle = |10\rangle,\ CZ|11\rangle = e^{\pi i}|11\rangle = -|11\rangle.$$

Es la función ejecutada por la puerta CZ aplicada a un registro de dos qubits ([1], 4.80).

El algoritmo de suma se describe en la figura 1. Primero se calcula la trasformada $QFT|y\rangle$. Luego, se ejecuta la operación CZ aplicada al estado $|x\rangle\times QFT|y\rangle$, obteniéndose el estado

$$\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} e^{\frac{2\pi i}{N}xk} e^{\frac{2\pi i}{N}yk}\,|x\rangle\times|k\rangle = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} e^{\frac{2\pi i}{N}(x+y)k}\,|x\rangle\times|k\rangle. \quad (3)$$

De esta relación se deduce que

$$CZ|x\rangle\times QFT|y\rangle = |x\rangle\times QFT|x+y \bmod N\rangle, \quad (4)$$

con lo cual, utilizando la transformada inversa $IQFT = QFT^{+}$, se genera el estado

$$|x\rangle\times IQFT\cdot QFT|x+y \bmod N\rangle = |x\rangle\times|x+y \bmod N\rangle. \quad (5)$$

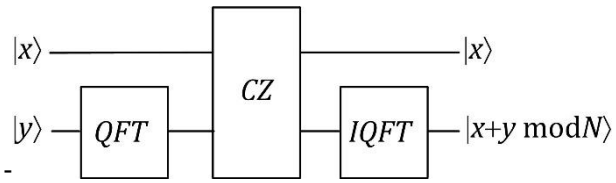

Figura 1 Cálculo de $x + y$ mod 2^n

1.2. Circuito

Los circuitos que ejecutan las operaciones QFT e $IQFT$ se describen en [1] (Fig.7.6 y 7.7). Teniendo en cuenta que $e^{2\pi i} = 1$, se deduce que

$$e^{\frac{2\pi i}{N}xk} = e^{2\pi i k_{n-1}(0.x_0)} \cdot e^{2\pi i k_{n-2}(0.x_1x_0)} \cdot \ldots \cdot e^{2\pi i k_0(0.x_{n-1}x_{n-2}\ldots x_0)}, \quad (6)$$

con lo cual

$$CZ|x\rangle\times|k\rangle = e^{2\pi i k_{n-1}(0.x_0)} \cdot e^{2\pi i k_{n-2}(0.x_1x_0)} \cdot \ldots \cdot e^{2\pi i k_0(0.x_{n-1}x_{n-2}\ldots x_0)}|x\rangle\times|k\rangle. \quad (7)$$

De forma equivalente, la operación anterior puede expresarse como sigue:

$$CZ|x\rangle\times|k\rangle =$$

$$|x\rangle\times(e^{2\pi i k_{n-1}(0.x_0)}|k_{n-1}\rangle)\times(e^{2\pi i k_{n-2}(0.x_1x_0)}|k_{n-2}\rangle)\times \ldots \times(e^{2\pi i k_0(0.x_{n-1}x_{n-2}\ldots x_0)}|k_0\rangle). \quad (8)$$

Cada factor de (8) puede expresarse en términos de rotaciones controladas. Por ejemplo

$$(e^{2\pi i k_0(0.x_{n-1}x_{n-2}\ldots x_0)}|k_0\rangle) = R_{2\pi(0.x_{n-1}x_{n-2}\ldots x_0)}|k_0\rangle =$$

$$R_{2\pi(0.0\ldots x_0)} \ldots R_{2\pi(0.0x_{n-2})}R_{2\pi(0.x_{n-1})}|k_0\rangle. \quad (9)$$

Queda por observar que

$$|x_{n-1}\rangle\times R_{2\pi(0.x_{n-1})}|k_0\rangle = CR_{2\pi(0.1)}|x_{n-1}\rangle\times|k_0\rangle = CR_{\pi}|x_{n-1}\rangle\times|k_0\rangle ,$$

$$|x_{n-2}\rangle\times R_{2\pi(0.0x_{n-2})}|k_0\rangle = CR_{2\pi(0.01)}|x_{n-2}\rangle\times|k_0\rangle = CR_{\pi/2}|x_{n-2}\rangle\times|k_0\rangle ,$$

…

$$|x_0\rangle\times R_{2\pi(0.0\ldots x_0)}|k_0\rangle = CR_{2\pi(0.0\ldots 1)}|x_0\rangle\times|k_0\rangle = CR_{\pi/2^{n-1}}|x_0\rangle\times|k_0\rangle . \quad (10)$$

A esta serie de rotaciones controladas corresponde la parte del circuito de la Fig.2 que modifica el estado del qubit k_0. Las conclusiones son similares para los otros qubits.

Obsérvese que, en la Fig.2, los qubits del registro $|k\rangle$ se numeran en orden inverso. De esta manera, la transformada *QFT*, previa a la transformada *CZ*, puede ejecutarse sin la permutación final, es decir, como en [1], Fig.7.6.

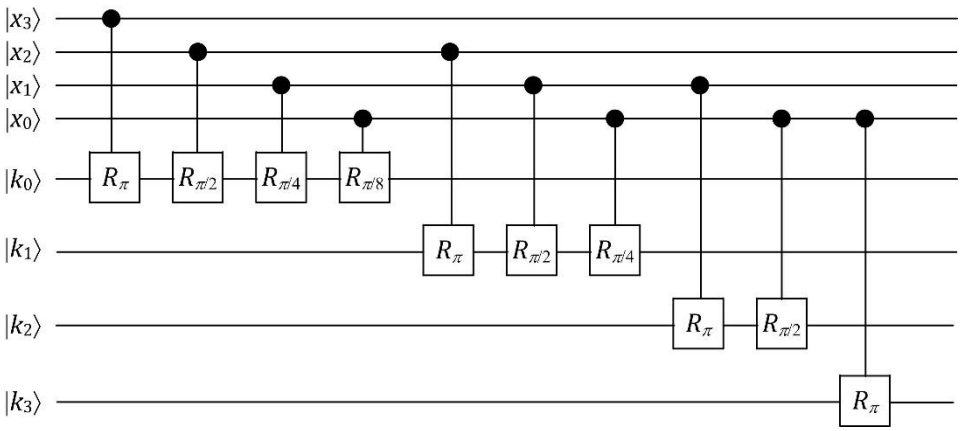

Figura 2 Circuito *CZ* (*n* = 4)

Ejemplo 1

Con *n* = 4, el sumador completo consta de los tres bloques de la Fig.3. El siguiente programa `nota5_1.py` lo describe y lo simula:

```
import cirq
###OPERANDOS: x = [x3, x2, x1, x0], y = [y3, y2, y1, y0]
x = [1,1,0,0]
y = [0,1,0,1]
x3,x2,x1,x0,y3,y2,y1,y0  = cirq.LineQubit.range(8)
adder = cirq.Circuit()
###INICIALIZACIÓN###
if y[0] == 1:
    adder.append(cirq.X(y3))
if y[1] == 1:
    adder.append(cirq.X(y2))
if y[2] == 1:
    adder.append(cirq.X(y1))
if y[3] == 1:
    adder.append(cirq.X(y0))
if x[0] == 1:
    adder.append(cirq.X(x3))
if x[1] == 1:
    adder.append(cirq.X(x2))
if x[2] == 1:
    adder.append(cirq.X(x1))
if x[3] == 1:
    adder.append(cirq.X(x0))
###QFT###
CRpi_2 = cirq.S.controlled()
CRpi_4 = cirq.T.controlled()
CRpi_8 = (cirq.T**0.5).controlled()
adder.append([cirq.H(y3),CRpi_2(y2,y3),CRpi_4(y1,y3),CRpi_8(y0,y3)])
adder.append([cirq.H(y2),CRpi_2(y1,y2),CRpi_4(y0,y2)])
adder.append([cirq.H(y1),CRpi_2(y0,y1)])
```

```
adder.append([cirq.H(y0)])
###CZ###
adder.append([cirq.CZ(x3,y3),CRpi_2(x2,y3),CRpi_4(x1,y3),
   CRpi_8(x0,y3)])
adder.append([cirq.CZ(x2,y2),CRpi_2(x1,y2),CRpi_4(x0,y2)])
adder.append([cirq.CZ(x1,y1),CRpi_2(x0,y1)])
adder.append([cirq.CZ(x0,y0)])
###IQFT###
CR_minus_pi_2 = (cirq.Z**(-0.5)).controlled()
CR_minus_pi_4 = (cirq.Z**(-0.25)).controlled()
CR_minus_pi_8 = (cirq.Z**(-0.125)).controlled()
adder.append([cirq.H(y0)])
adder.append([CR_minus_pi_2(y0,y1),cirq.H(y1)])
adder.append([CR_minus_pi_4(y0,y2),CR_minus_pi_2(y1,y2),cirq.H(y2)])
adder.append([CR_minus_pi_8(y0,y3),CR_minus_pi_4(y1,y3),
   CR_minus_pi_2(y2,y3),cirq.H(y3)])
###MEDICIÓN###
adder.append(cirq.measure(y3,y2,y1,y0,key = 'suma'))
un_simulador = cirq.Simulator()
resultado = un_simulador.run(adder, repetitions = 1)
print(resultado)
```

El resultado de la ejecución (x = 1100, y = 0101, $x+y$ mod 16 = 0001) es

```
suma=0, 0, 0, 1
```

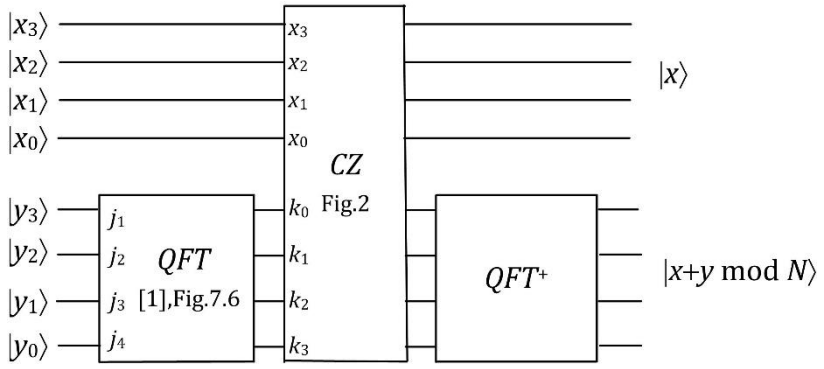

Figura 3 Sumador módulo 16

1.3. Suma con acarreo (no-modular)

Se trata de ejecutar la operación

$$|x_{n-1}\, x_{n-2}\ \ldots\ x_0\rangle\times|0\rangle\times|y_{n-1}\, y_{n-2}\ \ldots\ y_0\rangle \rightarrow |x_{n-1}\, x_{n-2}\ \ldots\ x_0\rangle\times|s_n\rangle\times|s_{n-1}\, s_{n-2}\ \ldots\ s_0\rangle, \qquad (11)$$

de tal manera que $s = x + y$, sin obviar el acarreo de salida s_n como se hace en la Sec.1. La solución es sencilla: se utiliza el método de la Sec.2 con

$$|x\rangle = |0\, x_{n-1}\, x_{n-2}\ \ldots x_0\rangle \text{ e } |y\rangle = |0\, y_{n-1} y_{n-2}\ \ldots y_0\rangle,$$

y se eliminan el qubit x_n , siempre en el estado $|0\rangle$, y la rotación condicional $CR_\pi|x_n\rangle\times|k_0\rangle$, que no se ejecuta nunca.

Ejemplo 2

El programa `nota5_2.py` describe y simula el circuito de la Fig.4.

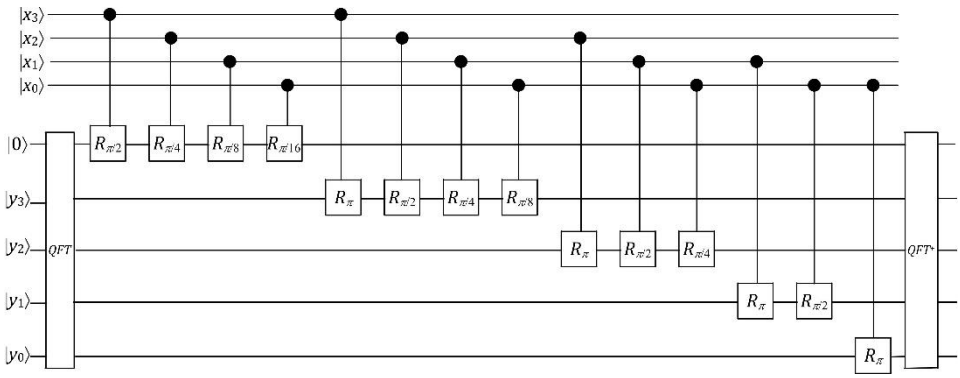

Figura 4 Sumador de 4 bits

1.4 Suma de *m* operandos

Considérense *m* números de *n* bits $X_1, X_2, \ldots, X_m$, representados por los estados básicos

$$|X_1\rangle = |X_{1\,n-1} X_{1\,n-2}\ \ldots X_{1\,0}\rangle, |X_2\rangle = |X_{2\,n-1} X_{2\,n-2}\ \ldots X_{2\,0}\rangle, \ldots, |X_m\rangle = |X_{m\,n-1} X_{m\,n-2}\ \ldots X_{m\,0}\rangle. \quad (12)$$

Genérese un circuito compuesto de *m* registros de *n* qubits, que genera la operación

$$|X_1\rangle\times|X_2\rangle\times\ldots\times|X_m\rangle \rightarrow |X_1\rangle\times|X_2\rangle\times\ldots\times|\, X_1+X_2+\ldots+X_m \bmod N\rangle. \quad (13)$$

Para ello se utiliza la relación (4) de forma recursiva, obteniéndose el circuito de la Fig.5.

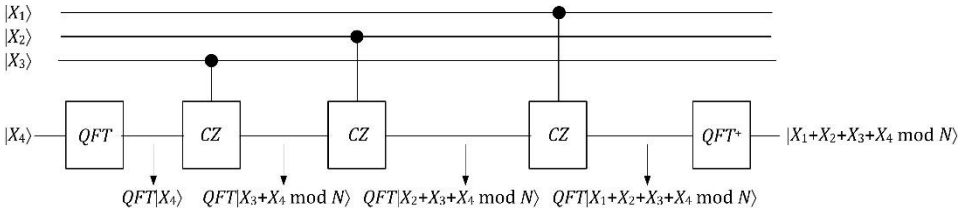

Figura 5 Sumador de 4 operandos

Ejemplo 3

El programa `nota5_3.py` describe y simula el circuito de la Fig.5.

2. Producto

Considérese un natural a. Si en el circuito de la Fig. 2 se sustituyen los ángulos

$$\pi,\ \pi/2,\ \pi/4,\ \ldots,\ \pi/2^{n-1},$$

por lo ángulos

$$(a \bmod 2)\pi,\ (a \bmod 4)\pi/2,\ (a \bmod 8)\pi/4,\ldots,\ (a \bmod 2^n)\pi/2^{n-1},$$

el circuito ejecuta la operación

$$a{\cdot}x + y \bmod 2^n. \tag{14}$$

Se deduce de las relaciones (2) y (3) sustituyendo x por $x{\cdot}a$, y observando que

$$R_{a\pi/2^{k-1}} = R_{(a \bmod 2^k)\,\pi/2^{k-1}}.$$

Ejemplo 4

El programa `nota5_4.py` describe y simula un circuito que ejecuta la operación (14) con $n = 4$.

El circuito del ejemplo 4 tiene dos registros de 4 qubits $|x\rangle$ e $|y\rangle$, y ejecuta la operación

$$|x\rangle\times|y\rangle \rightarrow |x\rangle\times|ax + y \bmod 16\rangle \tag{15}$$

siendo a un natural. Es un multiplicador por un valor constante. Se puede transformar en un multiplicador de dos operandos si se añade un registro $|a\rangle$ y se sustituye (15) por

$$|a\rangle\times|x\rangle\times|y\rangle \rightarrow |a\rangle\times|x\rangle\times|ax + y \bmod 16\rangle. \tag{16}$$

Supóngase que a se exprese con n bits:

$$a = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \ldots + a_1 2 + a_0.$$

Entonces,

$$(a \bmod 2)\pi = a_0\pi,\ (a \bmod 4)\pi/2 = a_1\pi + a_0\pi/2,\ \ldots,$$

$$(a \bmod 2^n)\pi/2^{n-1} = a_{n-1}\pi + \ldots + a_1\pi/2^{n-2} + a_0\pi/2^{n-1},$$

con lo cual

$$R_{(a \bmod 2)\pi} = R_{a_0\pi},\ R_{(a \bmod 4)\pi/2} = R_{a_1\pi} \cdot R_{a_0\pi/2},\ \ldots,$$

$$R_{(a \bmod 2^n)\pi/2^{n-1}} = R_{a_{n-1}\pi} \cdot \ldots \cdot R_{a_1\pi/2^{n-2}} \cdot R_{a_0\pi/2^{n-1}}. \tag{17}$$

De (16) y (17), con $n = 4$, se deduce el multiplicador de la figura 6 en el cual los bloques de tipo k ejecutan la operación $R_{\pi/2^k}$, $k = 0, 1, 2$ y 3.

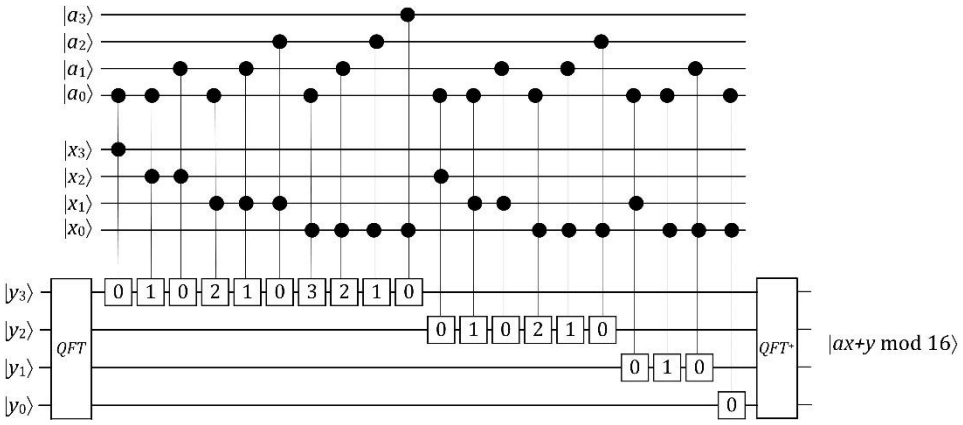

Figura 6 Multiplicador de 4 bits

Ejemplo 5

El programa `nota5_5.py` describe y simula el circuito de la Fig.6.

3. Rotaciones controladas

Los operadores utilizados en los circuitos descritos en esta nota son rotaciones de tipo CR_φ y CCR_φ . Para la síntesis del operador CR_φ, se puede utilizar el método de la Sec.6.2.2.3 de [1]. Primero comprúebese que

$$XR_{-\varphi/2}X\,R_{-\varphi/2}X = e^{-i\varphi/2}I, \tag{18}$$

con lo cual

$$R_\varphi XR_{-\varphi/2}X\,R_{-\varphi/2} = e^{-i\varphi/2}R_\varphi\,. \tag{19}$$

Por tanto, la operación CR_φ es ejecutada por el circuito de la Fig.6.12 de [1] con $A = R_\varphi$, $B = C = R_{-\varphi/2}$, y $R_a = R_{\varphi/2}$.

Para la síntesis del operador CCR_φ, se sustituyen, en el circuito anterior, los operadores CX por operadores CCX (Toffoli), y el operador $R_{\varphi/2}$ por un operador $CR_{\varphi/2}$. Para este último se puede utilizar el mismo método que antes, obteniéndose finalmente el circuito de la figura 7.

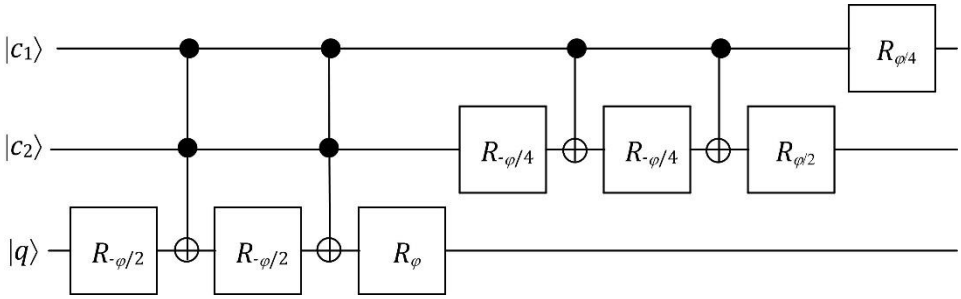

Figura 7 Rotación R_φ controlada por dos qubits

Ejemplo 6

El programa siguiente (`nota5_6.py`) describe y simula el circuito de la Fig.7 con $|q\rangle = H|0\rangle = |+\rangle$ y $\varphi = \pi/16$. Recuérdese ([1], 5.2) que $R_\varphi = Z^{\varphi/\pi}$, con lo cual $R_{\pi/16} = Z^{1/16}$.

```
import cirq
import numpy as np
phi_div_pi = 1/16
c1,c2,q = cirq.LineQubit.range(3)
ccr = cirq.Circuit()
control = [1,1]
ccr.append([cirq.H(q)])
if control[0] == 1:
    ccr.append(cirq.X(c1))
if control[1] == 1:
    ccr.append(cirq.X(c2))
ccr.append([(cirq.Z**(-phi_div_pi/2))(q),cirq.CCX(c1,c2,q),
   (cirq.Z**(-phi_div_pi/2))(q), cirq.CCX(c1,c2,q),
   (cirq.Z**phi_div_pi)(q)])
ccr.append([(cirq.Z**(-phi_div_pi/4))(c2),cirq.CX(c1,c2),
   (cirq.Z**(-phi_div_pi/4))(c2),
    cirq.CX(c1,c2),(cirq.Z**(phi_div_pi/2))(c2)])
ccr.append((cirq.Z**(1/64))(c1))
un_simulador = cirq.Simulator()
estado_final = un_simulador.simulate(ccr)
print(ccr)
print(estado_final)
```

Con $c_1 = c_2 = 1$ el resultado es

```
qubits: (cirq.LineQubit(0), cirq.LineQubit(1), cirq.LineQubit(2))
output vector: 0.707|110⟩ + (0.694+0.138j)|111⟩
```

y con $c_1 = 1$ y $c_2 = 0$ el resultado es

```
qubits: (cirq.LineQubit(0), cirq.LineQubit(1), cirq.LineQubit(2))
output vector: 0.707|100⟩ + 0.707|101⟩
```

Para comprobar el resultado, calcúlese $\frac{e^{i\pi/16}}{\sqrt{2}}$:

```
test = (np.exp(1.j*np.pi/16))/(2**0.5)
print("test =", test)
```

Resultado:

```
test = (0.6935199226610737+0.13794968964147147j)
```

4. Comentarios

El algoritmo de suma descrito en la Sec.1 consiste en

- transformar el estado básico $|y\rangle$, de un registro *RegY* que representa uno de los operandos de la suma, en un estado de superposición $QFT|y\rangle$,
- ejecutar rotaciones de los qubits de ese registro *RegY*, bajo el control de los qubits de otro registro *RegX*, cuyo estado inicial $|x\rangle$ es un estado básico que representa el segundo operando,
- aplicar la transformada inversa *IQFT* al registro *RegY*.

Tanto en las transformadas *QFT* e *IQFT*, como en las operaciones *CZ*, las rotaciones se ejecutan con ángulos iguales a $\pi/2^t$, $0 \leq t < n$. Por lo tanto, la eficiencia de este tipo de circuito dependerá de la posibilidad de implementar operadores $R_{\pi/2^t}$, con valores de 2^t que pueden ser muy grandes. Otra dificultad de implementación es la distancia entre los qubits sometidos a operaciones controladas y los qubits que las controlan (hasta una distancia de $n = 4$ en el circuito de la Fig.2).

En el caso de la multiplicación (Fig.6), la parte superior del circuito es equivalente a una matriz de puertas *AND* (corresponden a los puntos) que calcula todos los productos binarios $a_i x_j$. La parte inferior es equivalente a un sumador de n+1 operandos (Sec.1.4) que calcula

$$(y_{n-1}2^{n-1} + \ldots + y_1 2 + y_0) + (a_{n-1}x_0 2^{n-1} + \ldots + a_1 x_0 2 + a_0 x_0) +$$

$$(a_{n-1}x_1 2^{n-1} + \ldots + a_1 x_1 2 + a_0 x_1)2 + \ldots + (a_{n-1}x_{n-1}2^{n-1} + \ldots + a_1 x_{n-1} 2 + a_0 x_{n-1})\, 2^{n-1},$$

mod N. Es el clásico algoritmo *Shift and Add*. Desde luego, este circuito plantea los mismos problemas que el sumador: se necesitan operadores $R_{\pi/2^t}$, con grandes valores de 2^t, y la distancia entre los qubits sometidos a operaciones

controladas y los qubits que las controlan es larga. Tal vez una solución más tradicional, utilizando también una matriz de puertas *AND*, pero ejecutando las sumas de otra manera, podría ser más eficiente (dos ejemplos de sumadores se describen en la Nota 4). El circuito incluiría solamente puertas *CX* y *CCX*.

Referencias

[1] J.P.Deschamps, Computación Cuántica, Marcombo, Barcelona, 2023.

[2] T. G. Draper, Addition on a quantum computer, arXiv:quant-ph/0008033v1, 2000.

[3] L.Ruiz-Perez and J.C.Garcia-Escartin, Quantum arithmetic with the Quantum Fourier Transform https://arxiv.org/pdf/1411.5949.pdf, 2017.