Nota 8: Divisor

En esta nota se describe un circuito cuántico que materializa un algoritmo de división entera: dados dos números naturales x e y, se calculan dos números naturales q, el cociente, y r, el resto, que satisfacen las relaciones

$$x = q \cdot y + r,\ r < y. \tag{1}$$

1. Algoritmo

Se utiliza el algoritmo de división sin restauración (*non-restoring algorithm*) que, en su versión original, ejecuta las siguientes operaciones:

considérense un número natural $y > 0$, un número entero x que pertenece al intervalo

$$-y \leq x < y, \tag{2}$$

y un número natural $p > 0$. El algoritmo calcula el valor de dos números enteros q, el cociente, y r, el resto, que satisfacen las relaciones siguientes:

$$x \cdot 2^p = q \cdot y + r,\ -y \leq r < y,\ \text{sign}(r) = \text{sign}(q) = \text{sign}(x). \tag{3}$$

De esta última relación se deduce que

$$x/y = q \cdot 2^{-p} + r \cdot 2^{-p}/y,\ -2^{-p} \leq r \cdot 2^{-p}/y < 2^{-p}. \tag{4}$$

Por tanto, q es una aproximación del cociente x/y con una precisión de p bits fraccionarios.

El algoritmo es el siguiente (ver por ejemplo [2], algorithm 6.6):

Algoritmo 1.1 ($x \cdot 2^p = q \cdot y + r$, non-restoring algorithm)

```
r = x
for i in range(p):
        if r < 0:
                qi = 0
                r = 2*r + y
        else:
                qi = 1
```

```
                r = 2*r - y
q0 = 1 - q0
qp = 1
if q0 = 0 and r < 0:
        r = r + y
        q = q - 1
elif  q0 = 1 and r ≥ 0:
        r = r - y
        q = q + 1
```

El resultado es un número fraccionario $q_0.\ q_1\ q_2\ \ldots\ q_p$ donde q_0 es el bit de signo. La instrucción *if* … *elif* … final corrige el resultado si el resto r y el dividendo x no tienen el mismo signo.

Ejemplo 1

Con $x = 13$, $y = 47$, $p = 6$, se ejecutan las operaciones siguientes:

$r = 13$,

$r = 26 - 47 = -21$, $q_0 = 1$,

$r = -42 + 47 = 5$, $q_1 = 0$,

$r = 10 - 47 = -37$, $q_2 = 1$,

$r = -74 + 47 = -27$, $q_3 = 0$,

$r = -54 + 47 = -7$, $q_4 = 0$,

$r = -14 + 47 = 33$, $q_5 = 0$,

$q_0 = 0$, $q_6 = 1$

El resultado es

$q = 0010001 = 17$, $r = 33$.

Se comprueba que

$13.2^6 = 17 \cdot 47 + 33$, con $-47 \leq 33 < 47$.

2. División entera

Considérense dos números naturales x e y que pertenecen a los intervalos

$$x < 2^n,\ 0 < y < 2^n. \tag{5}$$

Para calcular el cociente q y el resto r definidos por (1), se ejecuta el algoritmo 1.1 con $p = n$ e y sustituido por $Y = y{\cdot}2^n$. De esta manera, $Y \geq 2^n$, y, según (5), $x < 2^n \leq Y$. El algoritmo 1.1 genera un cociente q y un resto R que satisfacen (3), es decir,

$$x{\cdot}2^n = q{\cdot}Y + R,\ 0 \leq R < Y,$$

con lo cual, multiplicando por 2^{-n},

$$x = q{\cdot}y + r,\ r = R{\cdot}2^{-n},\ 0 \leq r < y.$$

Ya que Y es divisible por 2^n y que x es no-negativo, se puede reescribir el algoritmo 1.1 como sigue:

```
r = x - Y/2
for i in range(1,n):
        if r < 0:
                qi = 0
                r = r + Y/2**(i+1)
        else:
                qi = 1
                r = r - Y/2**(i+1)
qn = 1
if r < 0:
        r = r + y
        q = q - 1
```

Ejemplo 2

Con x = 47, y = 13, n = 6, con lo cual $Y = 13{\cdot}2^6 = 832$. se ejecutan las operaciones siguientes:

r = 47 – 832 = -369,

r = -369 + 208 = -161, $q_1 = 0$,

r = -161 + 104 = -57, $q_2 = 0$,

r = -57 + 52 = -5, $q_3 = 0$,

r = -5 + 26 = 21, $q_4 = 0$,

r = 21 - 13 = 8, $q_5 = 1$,

$q_6 = 1$

El resultado es

$q = 000011 = 3$, $r = 8$.

Se comprueba que

$47 = 3 \cdot 13 + 8$, con $0 \leq 8 < 13$.

3. Circuitos

Véase primero cómo se ejecuta el cálculo, etapa por etapa, con $n = 4$, sin la corrección final. Llámense r_0, r_1, r_2, r_3 y r_4 los sucesivos valores de r. Inicialmente,

$$r_0 = x = 0\ 0\ 0\ 0\ x_3\ x_2\ x_1\ x_0,\ Y/2 = y \cdot 2^3 = y_3\ y_2\ y_1\ y_0\ 0\ 0\ 0.$$

Obsérvese primero que $0 \leq r_0 < Y$, con lo cual $r_1 = r_0 - Y/2$ pertenece al intervalo

$$-Y/2 \leq r_1 < Y/2.$$

Luego, si $0 \leq r_1 < Y/2$, entonces $r_2 = r_1 - Y/4$ y

$$-Y/4 \leq r_2 < Y/4, \tag{6}$$

y si $-Y/2 \leq r_1 < 0$, entonces $r_2 = r_1 + Y/4$, con lo cual se cumple (6) también. Por inducción se demuestra que

$$-Y/2^i \leq r_i < Y/2^i,\ i = 0, 1, 2, 3, 4. \tag{7}$$

Ya que $Y/2$ se expresa con 7 bits (en general, con $2n$-1 bits), $Y/2^i$ se expresa con 8-i bits (en general, con $2n$-i bits), y los números del intervalo (7) con 9-i bits (en general, con $2n$+1-i bits). En este caso ($n = 4$), r_1 se expresa con 8 bits, r_2 con 7 bits, r_3 con 6 bits, y r_4 con 5 bits.

$$
\begin{array}{ccccccccc}
 & 0 & 0 & 0 & 0 & x_3 & x_2 & x_1 & x_0 \\
- & 0 & y_3 & y_2 & y_1 & y_0 & 0 & 0 & 0 \\
\hline
 & r_{14} & r_{13} & r_{12} & r_{11} & r_{10} & x_2 & x_1 & x_0 \\
+/- & & 0 & y_3 & y_2 & y_1 & y_0 & 0 & 0 \\
\hline
 & & r_{24} & r_{23} & r_{22} & r_{21} & r_{20} & x_1 & x_0 \\
+/- & & & 0 & y_3 & y_2 & y_1 & y_0 & 0 \\
\hline
 & & & r_{34} & r_{33} & r_{32} & r_{31} & r_{30} & x_0 \\
+/- & & & & 0 & y_3 & y_2 & y_1 & y_0 \\
\hline
 & & & & r_{44} & r_{43} & r_{42} & r_{41} & r_{40}
\end{array}
$$

Los coeficientes r_{14}, r_{24}, r_{34} y r_{44} son los bits de signo de los sucesivos valores de r, con lo cual

$$q = (1\text{-}r_{14})\ (1\text{-}r_{24})\ (1\text{-}r_{34})\ 1,\ r = r_{44}\, r_{43}\, r_{42}\, r_{41}\, r_{40}.$$

Si $r_{44} = 1$, una corrección final es necesaria.

El circuito correspondiente se muestra en la Fig.1. Consta de 4 (en general n) sumadores-restadores y de puertas X.

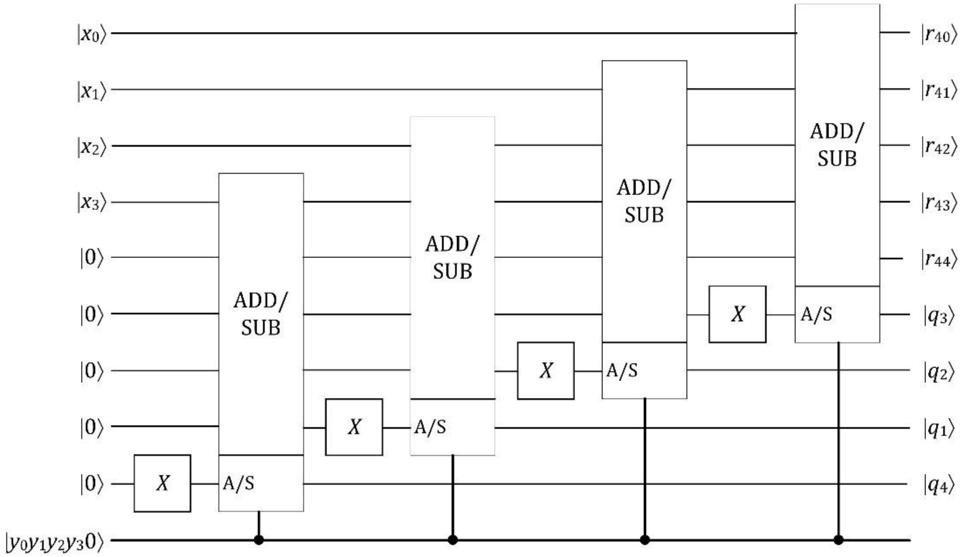

Figura 1 Divisor cuántico (n = 4)

En la Fig.2 se muestra un sumador-restador sintetizado con un sumador, por ejemplo el de la Sec.2 de la Nota 4, y con puertas CX. Se basa en la propiedad siguiente:

$$\overline{\bar{a} + b} = a - b,$$

donde el operador ¯ aplicado a un número binario sustituye cada bit por su complemento. Desde luego, cualquier sumador podría utilizarse, por ejemplo el que se describe en [3].

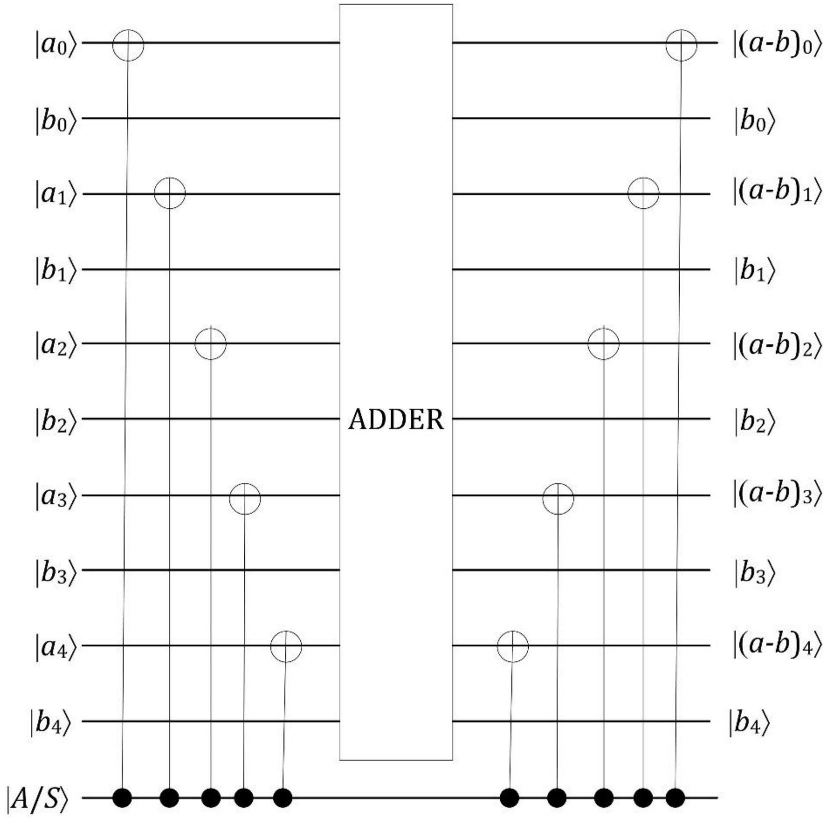

Figura 2 Sumador-restador (n = 4)

En la Fig.3 se muestra el circuito de corrección que corresponde a la instrucción *if* final del algoritmo 1.2. Consta principalmente de un sumador condicional. Si r_{44} = 1, se ejecuta la suma. En caso contrario no se ejecuta ninguna operación. En [4] se describe un sumador condicional. En esta nota, a efectos de simulación, se utiliza el método *controlled*() aplicado a un sumador convencional.

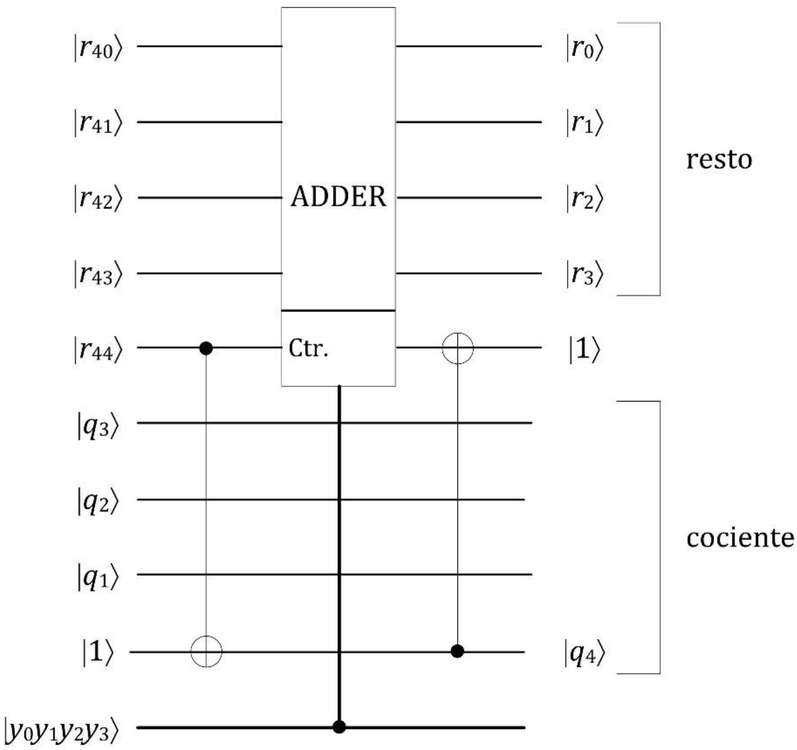

Figura 1 Circuito de corrección (n = 4)

Ejemplo 3

El programa `nota8_1.py` describe el circuito completo. Se definen sumadores de 4 y 5 qubits basados en el circuito de la Sec.2 de la Nota 4:

```
class ADDER5(cirq.Gate):
...

class ADDER4(cirq.Gate):
...
```

El sumador-restador se define con el sumador de 5 qubits y con puertas *CX*:

```
Adder5 = ADDER5()
class ADDSUB(cirq.Gate):
    def __init__(self):
        super(ADDSUB, self)
    def _num_qubits_(self):
```

```
        return 12
    def _decompose_(self, qubits):
        C0,A0,A1,A2,A3,A4,B0,B1,B2,B3,B4,AS = qubits
        Yield  [cirq.CX(AS,A0),cirq.CX(AS,A1),cirq.CX(AS,A2),
            cirq.CX(AS,A3),cirq.CX(AS,A4)]
        yield Adder5(C0,A0,A1,A2,A3,A4,B0,B1,B2,B3,B4)
        yield [cirq.CX(AS,A0),cirq.CX(AS,A1),cirq.CX(AS,A2),
            cirq.CX(AS,A3),cirq.CX(AS,A4)]
    def _circuit_diagram_info_(self, args):
        return ["ADDSUB"] * self.num_qubits()
AddSub = ADDSUB()
```

El sumador controlado se define con el método `controlled()` aplicado a un sumador de cuatro bits:

```
Adder4 = ADDER4()
ContAdder = Adder4.controlled()
```

Queda por describir y simular los circuitos de las figuras 1 y 3:

```
#######OPERANDOS##########
x = [0,1,1,1]
y = [0,0,1,0]
#######CIRCUITO##########
reg = cirq.LineQubit.range(15)
DIV = cirq.Circuit()
for i in range(4):
    if x[i] == 1:
        DIV.append(cirq.X(reg[i]))
    if y[i] == 1:
        DIV.append(cirq.X(reg[i+9]))
DIV.append(cirq.X(reg[8]))
DIV.append(AddSub(reg[14],reg[3],reg[4],reg[5],reg[6],reg[7],reg[9],
    reg[10],reg[11],reg[12],reg[13],reg[8]))
DIV.append(cirq.X(reg[7]))
DIV.append(AddSub(reg[14],reg[2],reg[3],reg[4],reg[5],reg[6],reg[9],
    reg[10],reg[11],reg[12],reg[13],reg[7]))
DIV.append(cirq.X(reg[6]))
DIV.append(AddSub(reg[14],reg[1],reg[2],reg[3],reg[4],reg[5],reg[9],
    reg[10],reg[11],reg[12],reg[13],reg[6]))
DIV.append(cirq.X(reg[5]))
DIV.append(AddSub(reg[14],reg[0],reg[1],reg[2],reg[3],reg[4],reg[9],
    reg[10],reg[11],reg[12],reg[13],reg[5]))
DIV.append(cirq.CX(reg[4],reg[8]))
DIV.append(ContAdder(reg[4],reg[14],reg[0],reg[1],reg[2],reg[3],
    reg[9],reg[10],reg[11],reg[12]))
DIV.append(cirq.CX(reg[8],reg[4]))
DIV.append(cirq.measure(reg[3],reg[2],reg[1],reg[0]))
DIV.append(cirq.measure(reg[7],reg[6],reg[5],reg[8]))
########SIMULACIÓN##########
simulador = cirq.Simulator()
resultado = simulador.run(DIV, repetitions=1)
#print(DIV)
print(resultado)
```

Con x = 1101 e y = 0101, es decir, x = 13 e y = 5, el resultado es

```
q(3),q(2),q(1),q(0)=0, 0, 1, 1
q(7),q(6),q(5),q(8)=0, 0, 1, 0
```

es decir, r = 0011 = 3 y q = 0010 = 2. Efectivamente, 13 = 2·5 + 3.

Referencias

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

[2] J.P. Deschamps, G.J.A.Bioul, and G.D.Sutter, Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems, Wiley, New York, 2006, ISBN 978-0471-68783-2

[3] H. Thapliyal and N. Ranganathan, "Design of efficient reversible logic based binary and bcd adder circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 9, no. 3, p. 17, 2013

[4] E. Muñoz-Coreas and H. Thapliyal, "T-count Optimized Design of Quantum Integer Multiplication," *ArXiv e-prints*, Jun. 2017. Available: https://arxiv.org/abs/1706.05113