

## Nota 1: Circuitos combinacionales

En [1], Sec.6.1.1, se define una función  $U_f(n)$  que, a cualquier función binaria  $f$  de  $n$  variables binarias, asocia un operador unitario definido por una matriz  $(2^{n+1}) \times (2^{n+1})$ . El circuito cuántico sintetizado a partir de ese operador materializa la función  $f$ . A continuación, se define una función  $U_f(n, m)$  que, a cualquier conjunto de  $m$  funciones binarias  $f_0, f_1, \dots, f_{m-1}$  de  $n$  variables binarias  $x_0, x_1, \dots, x_{n-1}$ , asocia un operador unitario definido por una matriz  $(2^{n+m}) \times (2^{n+m})$ . El circuito cuántico sintetizado a partir de ese operador materializa las funciones  $f_j$ . Consta de  $n+m$  qubits

$$q_0, q_1, \dots, q_{n-1}, q_n, q_{n+1}, \dots, q_{n+m-1}.$$

El operador unitario  $U_f$ , aplicado a un estado básico, ejecuta la siguiente operación

$$|x_0 \dots x_{n-1} y_0 \dots y_{m-1}\rangle \xrightarrow{U_f} |x_0 \dots x_{n-1} y_0 \oplus f_0(x_0, \dots, x_{n-1}) \dots y_{m-1} \oplus f_{m-1}(x_0, \dots, x_{n-1})\rangle.$$

Es un operador unitario. En particular, obsérvese que  $U_f U_f = I$ . La matriz correspondiente se define como sigue:

$U_f(i, j) = 1$  si, y solo si, (en numeración binaria)

$$i = i_0 \dots i_{n-1} i_n \dots i_{n+m-1}, j = i_0 \dots i_{n-1} i_n \oplus f_0(i_0, \dots, i_{n-1}) \dots i_{n+m-1} \oplus f_{m-1}(i_0, \dots, i_{n-1}).$$

### Ejemplo 1

El programa siguiente (`nota1_1.py`) describe el operador  $U_f$  sobre 4 qubits  $x_0, x_1, y_0, y_1$ , que corresponde a las funciones definidas por la tabla 1.

$x_0$	$x_1$	$y_0$	$y_1$
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	1

**Tabla 1** Funciones de conmutación

```

n = 2
m = 2
truth_table = [[1,0],
                [1,1],
                [0,0],
                [0,1]]

def ToBin(a,p):
    int_a = a
    bin_a = []
    for i in range(p):
        r = int_a%2
        bin_a = [r] + bin_a
        int_a = int(int_a/2)
    return bin_a

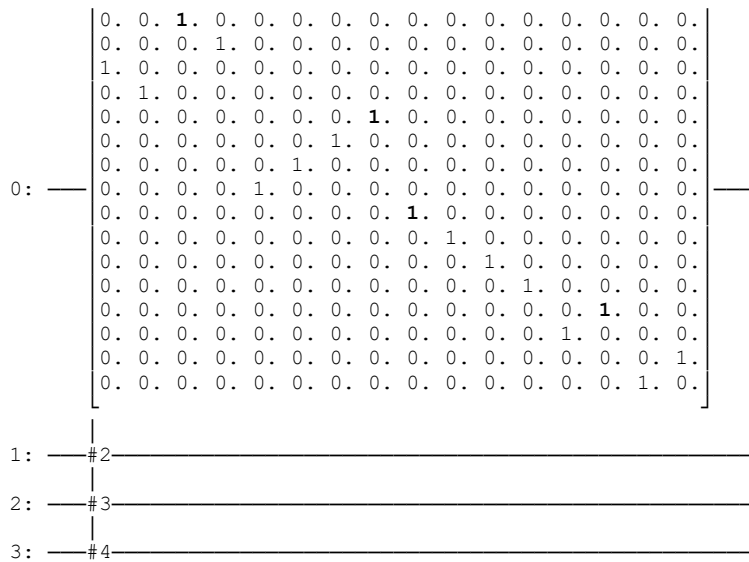
def Uf(n,m):
    N = 2**(n+m)
    matrix = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            bin_i = ToBin(i,n+m)
            bin_j = ToBin(j,n+m)
            matrix[i][j] = 1
            if (bin_j[:n] != bin_i[:n]):
                matrix[i][j] = 0
            for k in range(m):
                if bin_j[n+k] != (bin_i[n+k] + F(bin_i[:n],k))%2:
                    matrix[i][j] = 0
        return matrix

def F(x,y):
    acc = 0
    for i in range(n):
        acc = acc + x[i]*(2**(n-i-1))
    return truth_table[acc][y]

import cirq
U = cirq.MatrixGate(Uf(n,m))
x0,x1,y0,y1 = cirq.LineQubit.range(4)
F = cirq.Circuit()
F.append([U(x0,x1,y0,y1)])
print(F)

```

**Resultado de la ejecución**



Se observa que este circuito ejecuta las siguientes operaciones:

**0000→ 0010,**  
**0100→ 0111,**  
**1000→ 1000,**  
**1100→ 1101.**

Son las operaciones que corresponden a la tabla 1.

## Referencia

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