# Diseño y simulación de un procesador cuántico superconductor

Miguel Casanova
Departamento de Electrónica y Circuitos[1], Universidad Simón Bolívar

2018
September

[1]I am no longer a member of this department

# Índice general

# Índice de figuras

# Índice de cuadros

# Capítulo 4

# El simulador

El simulador se construyó utilizando la librería Qutip 4.2 de Python 3.6. Esta es una librería que incluye varias herramientas para realizar simulaciones de sistemas mecánico cuánticos, entre ellas, un solucionador de ecuaciones maestras. El funcionamiento básico del simulador desarrollado es el siguiente:

1. Leer estado inicial
2. Construir Hamiltoniano del sistema
3. Introducir Hamiltoniano y estado inicial en el solucionador de ecuaciones maestras.
4. Retornar solución

De esta manera se simulan las compuertas naturales de los transmones. Luego, a partir de estas se construyen todas las demás compuertas que se necesitaran para construir un conjunto de compuertas cuánticas con el cual poder ejecutar los algoritmos de Grover, Shor y PageRank.

Se simularon dos sistemas distintos, uno de cuatro qubits y otro de ocho qubits. El diseño original era el de cuatro qubits, con él se realizaron las simulaciones del algoritmo de Grover y del PageRank. Sin embargo, el algoritmo de Shor requiere de al menos ocho qubits para factorizar el número compuesto impar más pequeño: el número 15. Posteriormente también se realizo una generalización del simulador para poder trabajar con sistemas de n qubits.

El tipo de acoplamiento entre los qubits elegido es el acoplamiento de tipo bus. De esta manera trabajamos con un único resonador, el cual se

puede tracear. Esto reduce significativamente la dimensión del sistema a simular y nos permite tener más qubits. Además, de esta forma, la interación es más directa y basta con la compuerta iSWAP para construir cualquier otra compuerta multiqubits, el cual no sería el caso con qubits acoplados a distintos resonadores, pues se necesitarían compuertas de interacción entre resonadores.

## 4.1. Parámetros de los sistemas simulados

Se han elegido parámetros típicos de los sistemas de qubits[6].

1. Frecuencias de resonancia:
   - *a*) Resonador: 10GHz
   - *b*) Qubit 0: 5GHz
   - *c*) Qubit 1: 6GHz
   - *d*) Qubit 2: 7GHz
   - *e*) Qubit 3: 8GHz
   - *f*) *Qubit 4: 11GHz
   - *g*) *Qubit 5: 12GHz
   - *h*) *Qubit 6: 13GHz
   - *i*) *Qubit 7: 14GHz
2. Constante de acoplamiento: Todas iguales a 0,1$GHz$
3. Tasas de decaimiento: Todas iguales a 5e-6 (Unidades?)
4. Frecuencia de resonancia para iSWAP: 9GHz

*Sólo aplica para el caso del sistema de 8 qubits

## 4.2. Compuertas simples

Como se vio en el capítulo anterior, en los transmones se puede ejecutar de manera natural las compuertas Rx, Ry e iSWAP. Estas compuertas se implementan en el simulador y es a partir de ellas que se contruyen todas las demás.

$$H_{R_x} = -\frac{1}{2}\sum_i \Delta_{q_i}\sigma_{z_i} + \xi(t)\sigma_{x_{target}} \tag{4.1}$$

$$H_{R_y} = -\frac{1}{2}\sum_i \Delta_{q_i}\sigma_{z_i} + \xi(t)\sigma_{y_{target}} \tag{4.2}$$

$$H_{iSWAP} = \frac{g_1 g_2}{\Delta_{swap}}(\sigma_{+_1}\sigma_{-_2} + \sigma_{-_1}\sigma_{+_2}) \tag{4.3}$$

### 4.2.1. Rx y Ry

Estas compuertas se logran realizan un pulso gaussiano de microondas en fase (Rx) o en cuadratura (Ry). Se han elegido pulsos de 10ns de duración, truncados en $\pm 3\sigma$.

$$\xi(t) = A\Pi\left(\frac{t-\mu}{6\sigma}\right)\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.4}$$

Donde $\mu = 5ns$, $\sigma = \frac{5}{3}ns$, $\Pi(t)$ es la función rectangular, $A = \frac{\theta}{N}$ y $N = 0{,}9973$ es una constante de normalización. De esta manera se tiene el pulso gaussiano truncado deseado de 0ns a 10ns, cuya área bajo la curva sea igual al ángulo $\theta$ de la rotación.

### 4.2.2. iSWAP

Esta compuerta se logra aplicando un campo magnético tal que la frecuencia de resonancia de los dos qubits deseados se mueva a $\omega_{swap} = 9GHz$. Esta interacción se deja durante $\frac{\pi}{2J}$, donde $J = \left|\frac{g_1 g_2}{\Delta_{swap}}\right|$ y $\Delta_{swap} = \omega_{swap} - \omega_r$. Esto es, esta interacción se deja por 25ns.

Si se desea realizar la compuerta $\sqrt{iSWAP}$, se debe dejar la misma interacción por sólo 12.5ns, que es la mitad del tiempo.

## 4.3. Compuertas compuestas

Las compuertas anteriores forman un conjunto universal de compuertas cuánticas. A partir de secuencias de rotaciones en X e Y se puede formar cualquier rotación sobre cualquier eje de la esfera de Bloch, es decir, se puede realizar cualquier compuerta de un qubit. Con esto y cualquier compuerta de entrelazamiento, en nuestro caso $\sqrt{iSWAP}$, se tiene un conjunto universal de compuertas cuánticas y se puede realizar cualquier otra compuerta a partir de ellas.

### 4.3.1. X

Como tenemos $Rx(\theta)$, basta con hacer $\theta = \pi$ para realizar X, módulo una fase global de $-i$.

$$X = \begin{pmatrix} \cos\left(\frac{\pi}{2}\right) & -i\sin\left(\frac{\pi}{2}\right) \\ -i\sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{pmatrix} = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix} = -i \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{4.5}$$

### 4.3.2. Y

Como tenemos $Ry(\theta)$, basta con hacer $\theta = \pi$ para realizar Y, módulo una fase global de $-i$.

$$Y = \begin{pmatrix} \cos\left(\frac{\pi}{2}\right) & -\sin\left(\frac{\pi}{2}\right) \\ \sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = -i \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{4.6}$$

### 4.3.3. Rz

Esta compuerta se realiza aplicando una transformación a Rx tal que el eje de rotación se rote y coincida con el eje Z. Es decir, el eje X se rota $\pi/2$ alrededor de Y:

$$\begin{aligned} Rz(\theta) &= Ry(\frac{-\pi}{2})Rx(\theta)Ry(\frac{\pi}{2}) \\ &= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \\ &= \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} \end{aligned} \tag{4.7}$$

### 4.3.4. Z

Ahora, con Rz, se puede realizar Z haciendo $\theta = \pi$, módulo una fase global de $-i$.

$$Z = \begin{pmatrix} e^{-i\frac{\pi}{2}} & 0 \\ 0 & e^{-i\frac{\pi}{2}} \end{pmatrix} = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} = -i \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{4.8}$$

### 4.3.5. H

Esta compuerta transforma la base X en la base Z y se realiza con $Ry(\pi/2)$ seguido de X.

$$H = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{4.9}$$

Sólo que en nuestro caso, X también agrega una fase global de $-i$.

### 4.3.6. CNOT

Esta compuerta se realizó siguiendo el esquema de Schuch y Siewert [7]. De esta manera se logra la compuerta CNOT, módulo una fase global de $\frac{-1-i}{\sqrt{2}}$.

### 4.3.7. SWAP

Esta compuerta se realiza con una secuencia de CNOTs.

### 4.3.8. Compuertas condicionales generales

Barenco et al [1] demostraron que con la compuerta CNOT y compuertas de un qubit se puede realizar cualquier compuerta condicional bipartita de la siguiente manera:

$$CU = IdxCCNOTIdxBCNOTIdxA \tag{4.10}$$

Donde $CXBXA = U$ y $CBA = \mathbb{1}$.

Siguiendo este esquema se construyó: CRy, CRz y CH.

Barenco et al. también presentan un método para agregar más qubits de control a una compuerta condicional:

CIRCUITO

De esta manera se construyó: CCRy, CCCRy, CCRz y CCCRz. También se contruyó parcialmente de esta manera la compuerta de Toffoli, CCCNOT y CCCCNOT, sin embargo, debido a la fase global que queda al contruir X, Y y Z a partir de Rx, Ry y Rz, hace falta una componente adicional para poder construir estas compuertas. Esto ocurre porque la fase global también queda condicionada y deja de ser global. Para ilustrar mejor este detalle, tomemos como ejemplo el caso de Toffoli. Siguiendo el esquema anterior se llega a:

$$Toffoli' = e^{\pi/8} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -i \\ 0 & 0 & 0 & 0 & 0 & 0 & -i & 0 \end{pmatrix} \tag{4.11}$$

En lugar de:

$$Toffoli = e^{\phi} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.12}$$

Barenco et al. proponen métodos alternativos para lograr aproximaciones de esta compuerta, pero todas ellas introducen fases locales (sólo que de $\pi$ en lugar de $-\pi/2$), lo cual convierte a la compuerta en una completamente distinta. Es por esto que he desarrollado la compuerta de fase condicional y un método para eliminar la fase local que introduce la compuerta $Toffoli'$.

### 4.3.9. CP

Rz y la compuerta de cambio de fase P son completamente equivalentes cuando actuan como compuertas de un qubit, pues la única diferencia entre ellas es una fase global. Sin embargo, cuando se condicionan, dejan de ser equivalentes.

$$P_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} = e^{i\theta/2} Rz(\theta) \tag{4.13}$$

$$CRz(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-i\theta/2} & 0 \\ 0 & 0 & 0 & e^{i\theta/2} \end{pmatrix} \tag{4.14}$$

$$CP_\theta = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix} \neq e^{i\theta/2} CRz(\theta) \tag{4.15}$$

Se puede construir un sistema de ecuaciones linealmente independientes con las fases introducidas por $Rz(\theta_1)\otimes\mathbb{1}$, $\mathbb{1}\otimes Rz(\theta_2)$, $CRz_1(\theta_3)$ y $CRz_0(\theta_4)$, donde $CRz_0$ y $CRz_1$ son la compuerta CRz con el qubit de la partición 1 y el qubit de la partición 0 como target, respectivamente.

$$CRz_0(\theta_4)CRz_1(\theta_3)(\mathbb{1}\otimes Rz(\theta_2))(Rz(\theta_1)\otimes\mathbb{1}) = \begin{pmatrix} e^{i(-\theta_1-\theta_2)/2} & 0 & 0 & 0 \\ 0 & e^{i(-\theta_1+\theta_2-\theta_4)/2} & 0 & 0 \\ 0 & 0 & e^{i(\theta_1-\theta_2-\theta_3)/2} & 0 \\ 0 & 0 & 0 & e^{i(\theta_1+\theta_2+\theta_3} \end{pmatrix} \tag{4.16}$$

Donde se quiere que:

$$(-\theta_1-\theta_2)/2 = \phi \tag{4.17}$$
$$(-\theta_1+\theta_2-\theta_4)/2 = \phi \tag{4.18}$$
$$(\theta_1-\theta_2-\theta_3)/2 = \phi \tag{4.19}$$
$$(\theta_1+\theta_2+\theta_3+\theta_4)/2 = \phi+\theta \tag{4.20}$$
$$\tag{4.21}$$

Esto se logra tomando:

$$\theta_1 = \theta/4 \tag{4.22}$$
$$\theta_2 = \theta/4 \tag{4.23}$$
$$\theta_3 = \theta/2 \tag{4.24}$$
$$\theta_4 = \theta/2 \tag{4.25}$$
$$\tag{4.26}$$

$$CRz_0(\theta_4)CRz_1(\theta_3)(\mathbb{1}\otimes Rz(\theta_2))(Rz(\theta_1)\otimes\mathbb{1}) = \begin{pmatrix} e^{i(-\theta)/4} & 0 & 0 & 0 \\ 0 & e^{i(-\theta)/4} & 0 & 0 \\ 0 & 0 & e^{i(-\theta)/4} & 0 \\ 0 & 0 & 0 & e^{i3\theta/4} \end{pmatrix} = e^{-\theta/4} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.27}$$

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
from scipy.stats import norm
from qutip import *

def gaussianpulse(x,ts,tf):
    s = (tf-ts)/6
    m = (ts+tf)/2
    return (np.heaviside(x-m+3*s,1)-np.heaviside(x-m-3*s,1)) \
            *norm.pdf(x, loc = m, scale = s)/0.997300204

def squarepulse(x,ts,tf):
    s = (tf-ts)/6
    m = (ts+tf)/2
    return (np.heaviside(x-m+3*s,1)-np.heaviside(x-m-3*s,1))/(6*s)

def plot_drive_expect(res,args):
    tlist = res.times

    if args == 0:
        fig, axes = plt.subplots(1, 1, sharex=True, figsize=(12,4))

        axes.plot(tlist, np.real(expect(qop('n',0), res.states)), \
                    'b', linewidth=2, label="qubit 0")
        axes.plot(tlist, np.real(expect(qop('n',1), res.states)), \
                    'g', linewidth=2, label="qubit 1")
        axes.plot(tlist, np.real(expect(qop('n',2), res.states)), \
                    'c', linewidth=2, label="qubit 2")
        axes.plot(tlist, np.real(expect(qop('n',3), res.states)), \
                    'm', linewidth=2, label="qubit 3")
        axes.set_ylim(0, 1)

        axes.set_xlabel("Time (ns)", fontsize=16)
        axes.set_ylabel("Occupation probability", fontsize=16)
        axes.legend()

    else:
        fig, axes = plt.subplots(2, 1, sharex=True, figsize=(12,8))
```

```
        axes[0].plot(tlist, np.array(list(ksi_t(tlist,args))) / (2*np.pi), \
                        'b', linewidth=2, label="drive envelope")
        axes[0].set_ylabel("Energy (GHz)", fontsize=16)
        axes[0].legend()

        axes[1].plot(tlist, np.real(expect(qop('n',0), res.states)), 'b', \
                        linewidth=2, label="qubit 0")
        axes[1].plot(tlist, np.real(expect(qop('n',1), res.states)), 'g', \
                        linewidth=2, label="qubit 1")
        axes[1].plot(tlist, np.real(expect(qop('n',2), res.states)), 'c', \
                        linewidth=2, label="qubit 2")
        axes[1].plot(tlist, np.real(expect(qop('n',3), res.states)), 'm', \
                        linewidth=2, label="qubit 3")
        axes[1].set_ylim(0, 1)

        axes[1].set_xlabel("Time (ns)", fontsize=16)
        axes[1].set_ylabel("Occupation probability", fontsize=16)
        axes[1].legend()

    fig.tight_layout()

# Parametros del sistema

N = 50

wr = 10.0 * 2 * np.pi
wq = np.array([5.0 * 2 * np.pi, 6.0 * 2 * np.pi, 7.0 * 2 * np.pi, \
               8.0 * 2 * np.pi])
wq_swap = 9 * 2 * np.pi

g = np.array([0.1 * 2*np.pi, 0.1 * 2*np.pi, 0.1 * 2*np.pi, 0.1 * 2*np.pi])

D = wq - wr
D_swap = wq_swap - wr

chi = g**2 / abs(wr-wq)

kappa = 0.001
gamma = np.array([5e-6, 5e-6, 5e-6, 5e-6])
```

```
# cavity operators
a = destroy(N)
# a = tensor(destroy(N), qeye(2), qeye(2), qeye(2), qeye(2))
n = a.dag() * a
Id_r = qeye(N)

def qop_part(operator, target):
    if target == 0:
        qop_dict = {'sm' : destroy(2), 'sp' : (destroy(2)).dag(),
                    'sx' : sigmax(), 'sy' : sigmay(), 'sz' : sigmaz(),
                    'n' : (destroy(2)).dag() * destroy(2)}
        return qop_dict[operator]
    else:
        return qeye(2)

def qop(operator, target):
    return tensor(qop_part(operator, target-0), qop_part(operator, \
                    target-1), qop_part(operator, target-2), \
                    qop_part(operator, target-3))

#c_ops = [np.sqrt(gamma[0]) * qop('sm', 0), np.sqrt(gamma[1]) * \
            qop('sm', 1), np.sqrt(gamma[2]) * qop('sm', 2), \
            np.sqrt(gamma[3]) * qop('sm', 3)]
c_ops = []

def ksi_t(t, args):
    return args['A'] * gaussianpulse(t,args['ts'],args['tf'])

def ksi_tm(t, args):
    return args['A'] * gaussianpulse(t,args['ts'],args['tf']) * \
            np.exp(-1j*args['w']*(t-args['ts']))

def ksi_tp(t, args):
    return args['A'] * gaussianpulse(t,args['ts'],args['tf']) * \
            np.exp(1j*args['w']*(t-args['ts']))

def ksiS_t(t, args):
    return args['A'] * squarepulse(t,args['ts'],args['tf'])
```

```
def ksiS_tm(t, args):
    return args['A'] * np.exp(-1j*args['w']*(t-args['ts']))

def ksiS_tp(t, args):
    return args['A'] * np.exp(1j*args['w']*(t-args['ts']))

def Rx(psi0, target, theta):
    tlist = np.linspace(0, 10, 200)

    wd = wq[target]

    Dr = wr-wd
    Dq = wq-wd

    Hsyst = 0
    for i in range(4):
        Hsyst = Hsyst - Dq[i]*qop('sz',i)/2

    H_t = [[qop('sx',target)/2, ksi_t], Hsyst]

    args = {'A' : theta, 'ts' : 0, 'tf' : 10, 'w' : wq[target]}
    res = mesolve(H_t, psi0, tlist, c_ops, [], args = args)

    # plot_drive_expect(res,args)

    return res

def Ry(psi0, target, theta):
    tlist = np.linspace(0, 10, 200)

    wd = wq[target]

    Dr = wr-wd
    Dq = wq-wd

    Hsyst = 0
    for i in range(4):
        Hsyst = Hsyst - Dq[i]*qop('sz',i)/2

    H_t = [[qop('sy',target)/2, ksi_t], Hsyst]
```

```
    args = {'A' : theta, 'ts' : 0, 'tf' : 10, 'w' : wq[target]}
    res = mesolve(H_t, psi0, tlist, c_ops, [], args = args)

    # plot_drive_expect(res,args)

    return res

def Rz(psi0, target, theta):
    res = Ry(psi0, target, np.pi/2)
    res = Rx(res.states[-1], target, theta)
    return Ry(res.states[-1], target, -np.pi/2)

def X(psi0, target):
    return Rx(psi0, target, np.pi)

def Y(psi0, target):
    return Ry(psi0, target, np.pi)

def Z(psi0, target, theta):
    return Rz(psi0, target, np.pi)

def H(psi0, target):
    res = Ry(psi0, target, np.pi/2)
    return X(res.states[-1], target)

def sqrtiSWAP(psi0, target1, target2):
    wqt1 = wq[target1]
    wq[target1] = wq_swap

    wqt2 = wq[target2]
    wq[target2] = wq_swap

    D = wq - wr

    J = np.abs(g[target1] * g[target2] * (D[target1] + D[target2]) / \
        (D[target1] * D[target2]))/2

    tf = np.pi/(4*J)
    tlist = np.linspace(0, tf, 250)
```

```
    Hsyst = g[target1]*g[target2] * (qop('sp',target1)*qop('sm',target2) \
        + qop('sm',target1)*qop('sp',target2)) / (D_swap)

    res = mesolve(Hsyst, psi0, tlist, c_ops, [])

    wq[target1] = wqt1
    wq[target2] = wqt2
    D = wq - wr

    args = {'A' : 0, 'ts' : 0, 'tf' : tf, 'w' : wq[target1]}
    # plot_drive_expect(res,args)

    return res

def iSWAP(psi0, target1, target2):
    wqt1 = wq[target1]
    wq[target1] = wq_swap

    wqt2 = wq[target2]
    wq[target2] = wq_swap

    D = wq - wr

    J = np.abs(g[target1] * g[target2] * (D[target1] + D[target2]) / \
        (D[target1] * D[target2]))/2

    tf = np.pi/(2*J)
    tlist = np.linspace(0, tf, 500)

    Hsyst = g[target1]*g[target2] * (qop('sp',target1)*qop('sm',target2) \
        + qop('sm',target1)*qop('sp',target2)) / (D_swap)

    res = mesolve(Hsyst, psi0, tlist, c_ops, [])

    wq[target1] = wqt1
    wq[target2] = wqt2
    D = wq - wr

    args = {'A' : 0, 'ts' : 0, 'tf' : tf, 'w' : wq[target1]}
```

```
    # plot_drive_expect(res,args)

    return res

def CNOT(psi0, control, target):
    res = H(psi0, target)
    res = Rz(res.states[-1], target, -np.pi/2)
    res = Rz(res.states[-1], control, -np.pi/2)
    res = iSWAP(res.states[-1], control, target)
    res = H(res.states[-1], control)
    res = iSWAP(res.states[-1], control, target)
    res = Rx(res.states[-1], target, np.pi/2)
    res = iSWAP(res.states[-1], control, target)
    res = Rx(res.states[-1], control, np.pi/2)
    res = iSWAP(res.states[-1], control, target)
    return Rx(res.states[-1], target, np.pi/2)

def CRy(psi0, control, target, theta):
    res = Ry(psi0,target,theta/2)
    res = CNOT(res.states[-1],control,target)
    res = Ry(res.states[-1],target,-theta/2)
    return CNOT(res.states[-1],control,target)

def CRz(psi0, control, target, theta):
    res = Rz(psi0,target,theta/2)
    res = CNOT(res.states[-1],control,target)
    res = Rz(res.states[-1],target,-theta/2)
    return CNOT(res.states[-1],control,target)

def SWAP(psi0, target1, target2):
    res = CNOT(psi0, target1, target2)
    res = CNOT(res.states[-1], target2, target1)
    return CNOT(res.states[-1], target1, target2)

def CH(psi0, control, target):
    res = Ry(psi0, target, np.pi/4)
    res = CNOT(res.states[-1], control, target)
    return Ry(psi0, target, -np.pi/4)

def CP(psi0, control, target, theta, b = 0b11):
```

```
    if b == 0b00:
        res = Rz(psi0, control, -3*theta/4)
        res = Rz(res.states[-1], target, -3*theta/4)
        res = CRz(res.states[-1], control, target, theta/2)
        res = CRz(res.states[-1], target, control, theta/2)

    elif b == 0b01:
        res = Rz(psi0, control, -3*theta/4)
        res = Rz(res.states[-1], target, 5*theta/4)
        res = CRz(res.states[-1], control, target, -3*theta/2)
        res = CRz(res.states[-1], target, control, theta/2)

    elif b == 0b10:
        res = Rz(psi0, control, theta/4)
        res = Rz(res.states[-1], target, theta/4)
        res = CRz(res.states[-1], control, target, -3*theta/2)
        res = CRz(res.states[-1], target, control, theta/2)

    elif b == 0b11:
        res = Rz(psi0, control, theta/4)
        res = Rz(res.states[-1], target, theta/4)
        res = CRz(res.states[-1], control, target, theta/2)
        res = CRz(res.states[-1], target, control, theta/2)

    return res

def Toffoli(psi0, control1, control2, target):
    res = H(psi0, target)
    res = CRz(res.states[-1], control2, target, -np.pi/2)
    res = CNOT(res.states[-1], control1, control2)
    res = CRz(res.states[-1], control2, target, np.pi/2)
    res = CNOT(res.states[-1], control1, control2)
    res = CRz(res.states[-1], control1, target, -np.pi/2)
    res = H(res.states[-1], target)
    return CP(res.states[-1], control1, control2, -np.pi/2, b = 0b11)

def CCRz(psi0, control1, control2, target, theta):
    res = CRz(psi0, control2, target, theta/2)
    res = CNOT(res.states[-1], control1, control2)
    res = CRz(res.states[-1], control2, target, -theta/2)
```

```
    res = CNOT(res.states[-1], control1, control2)
    return CRz(res.states[-1], control1, target, theta/2)

def CCRy(psi0, control1, control2, target, theta):
    res = CRy(psi0, control2, target, theta/2)
    res = CNOT(res.states[-1], control1, control2)
    res = CRy(res.states[-1], control2, target, -theta/2)
    res = CNOT(res.states[-1], control1, control2)
    return CRy(res.states[-1], control1, target, theta/2)

def CCP(psi0, control1, control2, target, theta, b = 0b11):
    res = CP(psi0, control2, target, theta/2, b = b)
    if b == 0b00 or b == 0b01:
        res = X(res.states[-1], control1)
    res = CNOT(res.states[-1], control1, control2)
    if b == 0b00 or b == 0b01:
        res = X(res.states[-1], control1)
    res = CP(res.states[-1], control2, target, -theta/2, b = b)
    if b == 0b00 or b == 0b01:
        res = X(res.states[-1], control1)
    res = CNOT(res.states[-1], control1, control2)
    if b == 0b00 or b == 0b01:
        res = X(res.states[-1], control1)
    return CP(res.states[-1], control1, target, theta/2, b = b)

def CCNOT(psi0, control1, control2, target):
    return Toffoli(psi0, control1, control2, target)

def Z(psi0, target):
    res = Ry(psi0, target, np.pi)
    return Rx(res.states[-1], target, -np.pi)

def mZ(psi0, target):
    res = Ry(psi0, target, np.pi)
    return Rx(res.states[-1], target, np.pi)

def CCCNOT(psi0, control1, control2, control3, target):
    res = H(psi0, target)
    res = CCRz(res.states[-1], control2, control3, target, -np.pi/2)
    res = CNOT(res.states[-1], control1, control2)
```

```
    res = CCRz(res.states[-1], control2, control3, target, np.pi/2)
    res = CNOT(res.states[-1], control1, control2)
    res = CCRz(res.states[-1], control1, control3, target, -np.pi/2)
    res = H(res.states[-1], target)
    res = CP(res.states[-1], control2, control3, -np.pi/4)
    res = CNOT(res.states[-1], control1, control2)
    res = CP(res.states[-1], control2, control3, np.pi/4)
    res = CNOT(res.states[-1], control1, control2)
    return CP(res.states[-1], control1, control3, -np.pi/4)

def CCCRy(psi0, control1, control2, control3, target, theta):
    res = CRy(psi0, control3, target, theta/2)
    res = CCNOT(res.states[-1], control1, control2, control3)
    res = CRy(res.states[-1], control3, target, -theta/2)
    res = CCNOT(res.states[-1], control1, control2, control3)
    return CCRy(res.states[-1], control1, control2, target, theta/2)

def CCCRz(psi0, control1, control2, control3, target, theta):
    res = CRz(psi0, control3, target, theta/2)
    res = CCNOT(res.states[-1], control1, control2, control3)
    res = CRz(res.states[-1], control3, target, -theta/2)
    res = CCNOT(res.states[-1], control1, control2, control3)
    return CCRz(res.states[-1], control1, control2, target, theta/2)

def CCCP(psi0, control1, control2, control3, target, theta, b = 0b11):
    res = CP(psi0, control3, target, theta/2, b = b)
    if b == 0b00 or b == 0b01:
        res = X(res.states[-1], control1)
        res = X(res.states[-1], control2)
    res = CCNOT(res.states[-1], control1, control2, control3)
    if b == 0b00 or b == 0b01:
        res = X(res.states[-1], control1)
        res = X(res.states[-1], control2)
    res = CP(res.states[-1], control3, target, -theta/2, b = b)
    if b == 0b00 or b == 0b01:
        res = X(res.states[-1], control1)
        res = X(res.states[-1], control2)
    res = CCNOT(res.states[-1], control1, control2, control3)
    if b == 0b00 or b == 0b01:
        res = X(res.states[-1], control1)
```

```
    res = X(res.states[-1], control2)
return CCP(res.states[-1], control1, control2, target, theta/2, b = b)
```

# Bibliografía


[1] Adriano Barenco, Charles H. Bennet, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, Jhon A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 1995.

[2] Sttiwuer Díaz-Solórzano. Esquemas de medidas. *QIC*, 2014.

[3] Rudolf Gross and Achim Marx. Applied superconductivity: Josephson effect and superconducting electronics. *Walther-Meißner-Institut*, 2005.

[4] A. P. Drozdov, M. I. Eremets, I. A. Troyan, V. Ksenofontov, and S. I. Shylin. Conventional superconductivity at 203 kelvin at high pressures in the sulfur hydride system. *Nature*, 525:73–76, 2015.

[5] G. Wendin. Quantum information processing with superconducting circuits: a review. *IOP Science*, 2017.

[6] Alexandre Blais, Jay Gambetta, A. Wallraff, D. I. Schuster, S. M. Girvin, M. H. Devoret, , and R. J. Schoelkopf. Quantum-information processing with circuit quantum electrodynamics. *Physical Review A*, 2007.

[7] Norbert Schuch and Jens Siewert. Natural two-qubit gate for quantum computation using the xy interaction. *Physical Review A*, 2003.