Parámetros del sistema

Se utiliza un resonador con frecuencia de resonancia de 10GHz y cuatro transmones con frecuencias de resonancia de 5, 6, 7 y 8GHz. Se fijan las constantes de acoplamiento entre el resonador y los transmones en 100MHz.

La constante de decaimiento de los qubits se fija en 510^{-6} para tener una fidelidad de al menos 90% en todas las compuertas definidas en esta librería. Con una tasa de decaimiento en el orden de 10^{-5} la compuerta CCCNOT tiene una fidelidad de 49%, por lo que se necesita una tasa de decaimiento menor o qubits de ancilla para realizar corrección de errores. Para no aumentar el tamaño del sistema y consumir más memoria en las simulaciones, se ha elegido usar una tasa de decaimiento menor.

Mathematica:

```
\[Omega]r = 2 \[Pi] 10.0;
Subscript[\[Omega]q, i_] := 2 \[Pi] {5.0, 6.0, 7.0, 8.0}[[i + 1]];
\[Omega]qswap = 2 \[Pi] 9.0;
Subscript[g, i_] := 2 \[Pi] {0.1, 0.1, 0.1, 0.1}[[i + 1]];
Subscript[\[CapitalDelta], i_] := Subscript[\[Omega]q, i] - \[Omega]r;
```

Python:

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

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

D = wq - wr
D_swap = wq_swap - wr

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

kappa = 0.0001
gamma = np.array([5e-6, 5e-6, 5e-6, 5e-6])
```

Pulsos

El pulso gaussiano se trunca en $\pm 3\sigma$ y se reescala por 1/0.997300204 para normalizarlo.

Mathematica:

```
GaussianPulse[x_, ts_,
   tf_] := (UnitStep[x - \[Mu] + 3 \[Sigma]] -
```

```
      UnitStep[x - \[Mu] - 3 \[Sigma]]) PDF[
      NormalDistribution[\[Mu], \[Sigma]], x]/
     0.997300204 /. {\[Mu] -> (tf + ts)/2, \[Sigma] -> (tf - ts)/6};
```

Python:

```
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
```

El pulso rectangular se divide entre su duración para normalizarlo también.

Mathematica:

```
SquarePulse[x_, ts_,
   tf_] := (UnitStep[x - \[Mu] + 3 \[Sigma]] -
      UnitStep[x - \[Mu] - 3 \[Sigma]])/(6 \[Sigma]) /. {\[Mu] -> (
     tf + ts)/2, \[Sigma] -> (tf - ts)/6};
```

Python:

```
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)
```

Compuertas simples

Con los transmones se pueden realizar naturalmente rotaciones en X e Y y la compuerta de entrelazamiento $\sqrt{iSWAP}$.

- Compuerta Rx:

Aplicando un pulso cuya componente en cuadratura en el Hamiltoniano efectivo tenga la forma de un pulso gaussiano de energía θ se logra una rotatición de θ en X.

Mathematica:

```
Rx[target_, \[Theta]_] :=
  Module[{H := -1/
      2 Sum[(Subscript[\[Omega]q, i] - Subscript[\[Omega]q,
          target]) Subscript[\[Sigma]z, i], {i, 0, 3}]},
   H = H +
     1/2 \[Theta] GaussianPulse[t, 0, 10] Subscript[\[Sigma]x, target];
   MatrixExp[-I NIntegrate[H, {t, 0, 10}]]
   ];
```

Python:

```
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)

    return res
```

- Compuerta Ry

Aplicando un pulso cuya componente en fase en el Hamiltoniano efectivo tenga la forma de un pulso gaussiano de energía θ se logra una rotatición de θ en Y.

Mathematica:

```
Ry[target_, \[Theta]_] :=
  Module[{H := -1/
      2 Sum[(Subscript[\[Omega]q, i] - Subscript[\[Omega]q,
          target]) Subscript[\[Sigma]z, i], {i, 0, 3}]},
   H = H +
     1/2 \[Theta] GaussianPulse[t, 0, 10] Subscript[\[Sigma]y, target];
   MatrixExp[-I NIntegrate[H, {t, 0, 10}]]
   ];
```

Python:

```
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)

    return res
```

- Compuerta $\sqrt{iSWAP}$

Colocando dos qubits en resonancia se logra que estos interactuen. Manteniendo esta interacción durante $\frac{pi\Delta_s}{4g_1g2} = 12.5ns$, se logra la compuerta $\sqrt{iSWAP}$.

Mathematica:

```
sqrtiSWAP[target1_, target2_] :=
  Module[{H = -1/
       2 Sum[Subscript[\[Omega]q, i] Subscript[\[Sigma]z, i], {i, 0,
         3}] + Sum[(Subscript[g, i]
           Subscript[g, j]/Subscript[\[CapitalDelta],
           i]) (Subscript[\[Sigma]p, i] .Subscript[\[Sigma]m, j] +
           Subscript[\[Sigma]m, i] .Subscript[\[Sigma]p, j])/2, {i, 0,
         3}, {j, 0, 3}],
    J = 0, \[CapitalDelta]swap = \[Omega]qswap - \[Omega]r},
   H = H +
     1/2 (Subscript[\[Omega]q, target1] Subscript[\[Sigma]z,
         target1] +
        Subscript[\[Omega]q, target2] Subscript[\[Sigma]z, target2]) -
      Subscript[g, target1] Subscript[g,
      target2] (Subscript[\[CapitalDelta], target1] +
         Subscript[\[CapitalDelta],
         target2])/(Subscript[\[CapitalDelta], target1]
          Subscript[\[CapitalDelta],
         target2]) (Subscript[\[Sigma]p,
          target1] .Subscript[\[Sigma]m, target2] +
         Subscript[\[Sigma]m, target1] .Subscript[\[Sigma]p,
          target2])/2;
   H = H -
     1/2 \[Omega]qswap (Subscript[\[Sigma]z, target1] +
        Subscript[\[Sigma]z, target2]) +
     Subscript[g, target1] Subscript[g,
      target2] (Subscript[\[Sigma]p, target1] .Subscript[\[Sigma]m,
          target2] +
         Subscript[\[Sigma]m, target1] .Subscript[\[Sigma]p,
          target2])/\[CapitalDelta]swap;
   H = Subscript[g, target1] Subscript[g,
     target2] (Subscript[\[Sigma]p, target1] .Subscript[\[Sigma]m,
```

```
      target2] +
     Subscript[\[Sigma]m, target1] .Subscript[\[Sigma]p,
      target2])/\[CapitalDelta]swap;
  J = Abs[
    Subscript[g, target1] Subscript[g, target2] /\[CapitalDelta]swap];
  MatrixExp[-I H \[Pi]/(4 J)]
  ];
```

Python:

```
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

    return res
```

- Compuerta iSWAP:

Igual que $\sqrt{iSWAP}$, pero la interacción se deja el doble de tiempo.

Mathematica:

```
iSWAP[target1_, target2_] :=
 Module[{H = -1/
      2 Sum[Subscript[\[Omega]q, i] Subscript[\[Sigma]z, i], {i, 0,
        3}] + Sum[(Subscript[g, i]
          Subscript[g, j]/Subscript[\[CapitalDelta],
          i]) (Subscript[\[Sigma]p, i] .Subscript[\[Sigma]m, j] +
          Subscript[\[Sigma]m, i] .Subscript[\[Sigma]p, j])/2, {i, 0,
```

```
        3}, {j, 0, 3}],
     J = 0, \[CapitalDelta]swap = \[Omega]qswap - \[Omega]r},
    H = H +
      1/2 (Subscript[\[Omega]q, target1] Subscript[\[Sigma]z,
          target1] +
         Subscript[\[Omega]q, target2] Subscript[\[Sigma]z, target2]) -
       Subscript[g, target1] Subscript[g,
       target2] (Subscript[\[CapitalDelta], target1] +
          Subscript[\[CapitalDelta],
          target2])/(Subscript[\[CapitalDelta], target1]
           Subscript[\[CapitalDelta],
          target2]) (Subscript[\[Sigma]p,
           target1] .Subscript[\[Sigma]m, target2] +
          Subscript[\[Sigma]m, target1] .Subscript[\[Sigma]p,
           target2])/2;
    H = H -
      1/2 \[Omega]qswap (Subscript[\[Sigma]z, target1] +
         Subscript[\[Sigma]z, target2]) +
      Subscript[g, target1] Subscript[g,
       target2] (Subscript[\[Sigma]p, target1] .Subscript[\[Sigma]m,
           target2] +
          Subscript[\[Sigma]m, target1] .Subscript[\[Sigma]p,
           target2])/\[CapitalDelta]swap;
    H = Subscript[g, target1] Subscript[g,
      target2] (Subscript[\[Sigma]p, target1] .Subscript[\[Sigma]m,
          target2] +
         Subscript[\[Sigma]m, target1] .Subscript[\[Sigma]p,
          target2])/\[CapitalDelta]swap;
    J = Abs[
      Subscript[g, target1] Subscript[g, target2] /\[CapitalDelta]swap];
    MatrixExp[-I H \[Pi]/(2 J)]
    ];
```

Python:

```
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

    return res
```

Compuertas compuestas

A partir de las compuertas simples se pueden construir otras.

- Compuerta X:

Esta se realiza con una rotación de π en X.

Mathematica:

```
X[target_] := Rx[target, \[Pi]];
```

Python:

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

- Compuerta Y:

Esta compuerta se realiza con una rotación de π en Y.

Mathematica:

```
Y[target_] := Ry[target, \[Pi]];
```

Python:

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

- Compuerta Rz:

Éstas se logran con una secuencia de rotaciones en X e Y.

Mathematica:

```
Rz[target_, \[Theta]_] :=
  Ry[target, -\[Pi]/2].Rx[target, \[Theta]].Ry[target, \[Pi]/2];
```

Python:

```
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)
```

- Compuerta Z:

Esta compuerta se realiza con una rotacion de π en Z.

Mathematica:

```
Z[target_] := Rz[target, \[Pi]];
```

Python:

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

- Compuerta de Hadamard:

Esta se logra con una secuencia de rotaciones en X e Y.

Mathematica:

```
H[target_] := X[target].Ry[target, \[Pi]/2];
```

Python:

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

- Compuerta CNOT:

Esta compuerta se construye con una secuencia de rotaciones en X y Z y de iSWAPs siguiento el esquema presentado por Schuch y Siewert en Phys. Rev. A 67, 032301, 2003.

Mathematica:

```
CNOT[control_, target_] :=
  Rx[target, \[Pi]/2].iSWAP[control, target].Rx[
    control, \[Pi]/2].iSWAP[control, target].Rx[
    target, \[Pi]/2].iSWAP[control, target].H[control].iSWAP[control,
    target].Rz[control, -\[Pi]/2].Rz[target, -\[Pi]/2].H[target];
```

Python:

```
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)
```

- Compuerta CRy:

Esta compuerta se realizó siguiendo el esquema de controlización presentado por Barenco et al. en Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CRy[control_, target_, \[Theta]_] :=
  CNOT[control, target].Ry[target, -\[Theta]/2].CNOT[control,
    target].Ry[target, \[Theta]/2];
```

Python:

```
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)
```

- Compuerta CRz:

Barenco et al. Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CRz[control_, target_, \[Theta]_] :=
  CNOT[control, target].Rz[target, -\[Theta]/2].CNOT[control,
    target].Rz[target, \[Theta]/2];
```

Python:

```
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)
```

- Compuerta SWAP:

Schuch, Siewert. Phys. Rev. A 67, 032301, 2003

Mathematica:

```
SWAP[target1_, target2_] :=
  CNOT[target1, target2].CNOT[target2, target1].CNOT[target1,
    target2];
```

Python:

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

- Compuerta CH:

Modificación de CRy

Mathematica:

```
CH[control_, target_] :=
  Ry[target, -\[Pi]/4].CNOT[control, target].Ry[target, \[Pi]/4];
```

Python:

```
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)
```

- Compuerta CP:

Explicación

Mathematica:

```
CP00[control_, target_, \[Theta]_] :=
  CRz[target, control, \[Theta]/2].CRz[control,
    target, \[Theta]/2].Rz[target, -3 \[Theta]/4].Rz[
    control, -3 \[Theta]/4];

CP01[control_, target_, \[Theta]_] :=
  CRz[target, control, \[Theta]/2].CRz[control,
    target, -3 \[Theta]/2].Rz[target, 5 \[Theta]/4].Rz[
    control, -3 \[Theta]/4];

CP10[control_, target_, \[Theta]_] :=
  CRz[target, control, \[Theta]/2].CRz[control,
    target, -3 \[Theta]/2].Rz[target, \[Theta]/4].Rz[
    control, \[Theta]/4];

CP11[control_, target_, \[Theta]_] :=
  CRz[target, control, \[Theta]/2].CRz[control,
    target, \[Theta]/2].Rz[target, \[Theta]/4].Rz[
    control, \[Theta]/4];
```

Python:

```
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
```

- Compuerta Toffoli (CCNOT):

Barenco et al. Phys. Rev. A 52, 3457, 1995. + CP

Mathematica:

```
Toffoli[control1_, control2_, target_] :=
  CP11[control1, control2, -\[Pi]/2].H[target].CRz[control1,
    target, -\[Pi]/2].CNOT[control1, control2].CRz[control2,
    target, \[Pi]/2].CNOT[control1, control2].CRz[control2,
    target, -\[Pi]/2].H[target];
```

Python:

```
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)
```

- Compuerta CCRz:

Barenco et al. Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CCRz[control1_, control2_, target_, \[Theta]_] :=
  CRz[control1, target, \[Theta]/2].CNOT[control1, control2].CRz[
    control2, target, -\[Theta]/2].CNOT[control1, control2].CRz[
    control2, target, \[Theta]/2];
```

Python:

```
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)
```

- Compuerta CCRy:

Barenco et al. Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CCRy[control1_, control2_, target_, \[Theta]_] :=
  CRy[control1, target, \[Theta]/2].CNOT[control1, control2].CRy[
    control2, target, -\[Theta]/2].CNOT[control1, control2].CRy[
    control2, target, \[Theta]/2];
```

Python:

```
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)
```

- Compuerta CCP:

Barenco et al. Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CCP00[control1_, control2_, target_, \[Theta]_] :=
  CP00[control1, target, \[Theta]/2].CNOT[control1, control2].CP00[
    control2, target, -\[Theta]/2].CNOT[control1, control2].CP00[
    control2, target, \[Theta]/2];
```

```
CCP01[control1_, control2_, target_, \[Theta]_] :=
  CP01[control1, target, \[Theta]/2].CNOT[control1, control2].CP01[
    control2, target, -\[Theta]/2].CNOT[control1, control2].CP01[
    control2, target, \[Theta]/2];
```

```
CCP10[control1_, control2_, target_, \[Theta]_] :=
  CP10[control1, target, \[Theta]/2].CNOT[control1, control2].CP10[
    control2, target, -\[Theta]/2].CNOT[control1, control2].CP10[
    control2, target, \[Theta]/2];
```

```
CCP11[control1_, control2_, target_, \[Theta]_] :=
  CP11[control1, target, \[Theta]/2].CNOT[control1, control2].CP11[
    control2, target, -\[Theta]/2].CNOT[control1, control2].CP11[
    control2, target, \[Theta]/2];
```

Python:

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

- Compuerta mZ:

Explicación

Mathematica:

```
mZ[target_] := X[target].Y[target];
```

Python:

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

- Compuerta CCCNOT:

Barenco et al. Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CCCNOT[control1_, control2_, control3_, target_] :=
  CP11[control1, control3, -\[Pi]/4].CNOT[control1, control2].CP11[
    control2, control3, \[Pi]/4].CNOT[control1, control2].CP11[
    control2, control3, -\[Pi]/4].H[target].CCRz[control1, control3,
    target, -\[Pi]/2].CNOT[control1, control2].CCRz[control2,
    control3, target, \[Pi]/2].CNOT[control1, control2].CCRz[control2,
     control3, target, -\[Pi]/2].H[target];
```

Python:

```
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)
```

- Compuerta CCCRy:

Barenco et al. Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CCCRy[control1_, control2_, control3_, target_, \[Theta]_] :=
  CCRy[control1, control2, target, \[Theta]/2].CCNOT[control1,
    control2, control3].CRy[control3, target, -\[Theta]/2].CCNOT[
    control1, control2, control3].CRy[control3, target, \[Theta]/2];
```

Python:

```
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)
```

- Compuerta CCCRz:

Barenco et al. Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CCCRz[control1_, control2_, control3_, target_, \[Theta]_] :=
  CCRz[control1, control2, target, \[Theta]/2].CCNOT[control1,
    control2, control3].CRz[control3, target, -\[Theta]/2].CCNOT[
    control1, control2, control3].CRz[control3, target, \[Theta]/2];
```

Python:

```
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)
```

- Compuerta CCCP:

Barenco et al. Phys. Rev. A 52, 3457, 1995.

Mathematica:

```
CCCP00[control1_, control2_, control3_, target_, \[Theta]_] :=
  CCP00[control1, control2, target, \[Theta]/2].CCNOT[control1,
    control2, control3].CP00[control3, target, -\[Theta]/2].CCNOT[
    control1, control2, control3].CP00[control3, target, \[Theta]/2];
```

```
CCCP01[control1_, control2_, control3_, target_, \[Theta]_] :=
  CCP01[control1, control2, target, \[Theta]/2].CCNOT[control1,
    control2, control3].CP01[control3, target, -\[Theta]/2].CCNOT[
    control1, control2, control3].CP01[control3, target, \[Theta]/2];
```

```
CCCP10[control1_, control2_, control3_, target_, \[Theta]_] :=
  CCP10[control1, control2, target, \[Theta]/2].CCNOT[control1,
    control2, control3].CP10[control3, target, -\[Theta]/2].CCNOT[
    control1, control2, control3].CP10[control3, target, \[Theta]/2];
```

```
CCCP11[control1_, control2_, control3_, target_, \[Theta]_] :=
  CCP11[control1, control2, target, \[Theta]/2].CCNOT[control1,
    control2, control3].CP11[control3, target, -\[Theta]/2].CCNOT[
    control1, control2, control3].CP11[control3, target, \[Theta]/2];
```

Python:

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

- Compuerta :

Explicación

Mathematica:

Python:

- Compuerta :

Explicación

Mathematica:

Python: