IMPLEMENTACIÓN CON QISKIT DE UNIDADES FUNCIONALES CUÁNTICAS: UN MULTIPLICADOR LOGARÍTMICO

DANIEL DAVID ABELLÁN SERRANO

GRADO EN INGENIERÍA INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTESNE DE MADRID

Trabajo Fin Grado en Ingeniería Informática

Fecha
31-05-2019

Director/es y/o colaborador:

Alberto A. Del Barrio
Guillermo Botella

Resumen en castellano

La computación cuántica ha dado un salto enorme desde el desarrollo teórico a la realidad gracias a las nuevas tecnologías que han permitido crear computadoras cuánticas reales por eso es que en este trabajo se ponen a prueba los conceptos básicos de la computación cuántica así como varias unidades funcionales básicas a partir de las cuales se ha desarrollado e implementado un multiplicador logarítmico. En el presente texto se pueden encontrar las implementaciones de estas unidades funcionales así como los resultados obtenidos y análisis tras su ejecución en diversos simuladores y máquinas reales.

Palabras clave

Computación cuántica, computación clásica, bit, qubit, Qiskit, sumador, multiplicador, multiplicador logarítmico, QFT (Quantum Fourier Transform), Equivalencia Cuántica-Clásica, simuladores, algoritmos cuánticos, complejidad computacional.

Abstract

Quantum computing has taken a huge leap from theoretical development to reality thanks to the new technologies that have allowed real quantum computers to be created. That is why in this work the basic concepts of quantum computing as well as several functional units are tested from which a logarithmic multiplier has been developed and implemented. In the present text you can find the implementations of these functional units as well as the results obtained and analysis after their execution in different simulators and real machines.

Keywords

Cuantum computing, classic computing, qubit, bit, Qiskit, adder, multiplier, algoritmic multiplier, QFT (Quantum Fourier Transform), Quantum-Classical Equivalence, simulators, cuantum algorithms, computational complexity.

Índice general

Agradecimientos

A Alberto y Guillermo por darme la oportunidad de entrar en el universo cuántico y encontrar un nuevo camino. A Antonio por toda su ayuda durante el curso para que este proyecto floreciera.

Dedicatoria

A mi abuela Angelita y mi madre María de los Ángeles por aguanterme un año más en casa y permitirme finalizar la titulación. A Aitor por bajarme del cielo a la tierra. A Bárbara por apoyarme tantos años. A Rafa, Alejandro, Verónica, Ainhoa, Pablo, Miguel, Clara, Paco, Adrianes, Daniel y Sergio por animarme a seguir cada viernes. A Manuel por sus consejos tan peculiares. Y por útimo a mi sobrina Zoe y mi hermana Rosa.

Capítulo 1

Principios básicos sobre computación cuántica

En este capítulo introduciremos los principios básicos sobre computación cuántica necesarios para comprender los temas abordados más adelante en este texto.

1.1. El qubit

Cuando hablamos de computación cuántica, el concepto base y el primero que debemos abordar es el concepto de qubit [C+18]. Un qubit es la unidad de almacenamiento de información mínima de la computación cuántica. Recibe ese nombre por paralelismo con el bit clásico. Un bit clásico puede estar en dos posibles estados, cero y uno, que representan la diferencia de potencial entre dos puntos determinados, es máxima (uno) o es mínima (cero). Sin embargo, un qubit puede estar en infinitos estados que se corresponden con un vector unitario en un espacio de Hilbert, complejo por definición, de dimensión dos.

Para visualizar cual es el estado de un qubit se suele pensar en un vector dentro de la esfera unitaria tridimensional, lo que recibe el nombre de representación en la esfera de Bloch, que podemos visualizar en la figura 1.1. A la hora de trabajar con un qubit, podemos operar sobre él mediante rotaciones en la esfera dando lugar a otros estados. Estas rotaciones quedan representadas por matrices unitarias, complejas por definición, de orden dos.

Esto pone de manifiesto la enorme, de hecho, infinita, cantidad de estados en que puede

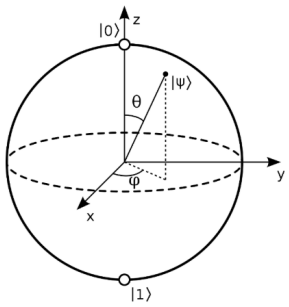


Figura 1.1: Esfera de Bloch

estar un qubit. Sin embargo, a la hora de intentar visualizar o consultar el estado de un qubit, acción a la que de ahora en adelante nos referiremos como realizar una medición, ocurren dos cosas que determinan la forma en que se trabaja en computación cuántica.

En primer lugar, aunque los estados posibles son infinitos, solo dos estados son observables, el vector $(1, 0)$ del espacio de Hilbert que se corresponde con el vector que apunta al polo norte de la esfera y que identificamos con el $|0\rangle$; y el vector $(0, 1)$ que se corresponde con el vector que apunta al polo sur de la esfera e identificamos con el $|1\rangle$ como se muestra en la ecuación 1.1. La notación ket $|\rangle$ se utiliza para distinguir los estados cuánticos de los clásicos. Esto sugiere la inmediata pregunta de qué ocurre cuando medimos un qubit en otro estado. En tal caso, se dice que el qubit está en un estado de superposición puesto que se tratará de una combinación lineal, posiblemente compleja, de los dos vectores anteriores. Al realizar la medición observaremos el estado $|0\rangle$ o el $|1\rangle$ según una distribución de probabilidad que depende del estado del qubit.

$$|0\rangle \Longleftrightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle \Longleftrightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{1.1}$$

Supongamos un estado cualquiera α. Como los estados $|0\rangle$ y $|1\rangle$ son una base del espacio de estados, α se podrá expresar como combinación lineal de los dos estados básicos tal y como vemos en la ecuación 1.2.

$$\alpha = a(1, 0) + b(0, 1) \; con \; |a|^2 + |b|^2 = 1. \tag{1.2}$$

En esta situación al realizar una medición observaremos el estado $|0\rangle$ con probabilidad

$|a|^2$ y $|1\rangle$ con probabilidad $|b|^2$. Por ejemplo, un estado de superposición sobre el ecuador de la esfera de Bloch como el que se observa en la ecuación 1.3.

$$\alpha = \frac{1}{\sqrt{2}}(1,1) = \frac{1}{\sqrt{2}}(1,0) + \frac{1}{\sqrt{2}}(0,1). \tag{1.3}$$

Al realizar una medición sobre el estado α observaremos el estado $|0\rangle$ con probabilidad $\frac{1}{2}$ y el estado $|1\rangle$ con probabilidad también $\frac{1}{2}$.

En segundo lugar, al medir el estado de un qubit destruimos el estado y por tanto no podemos realizar más operaciones sobre él, ni recuperar el estado para realizar otra medición o acción sobre dicho qubit.

Todo esto hace que a la hora de diseñar circuitos y algoritmos que esperamos que corran en computadores cuánticos haya que olvidar momentáneamente la forma de trabajar clásica y hacer un cambio de perspectiva.

A continuación haremos un resumen de como se opera con esta unidad de información. Todo esto se desarrolla en detalle en [HPKM07, Hir12, NC11].

1.1.1. Operaciones sobre un qubit

Como hemos mencionado, operar con un qubit consiste en ejercer sobre él una rotación dentro de la esfera de Bloch. Cada una de estas rotaciones se corresponde con una matriz unitaria de orden dos. Una matriz unitaria es una matriz compleja que cumple la propiedad de que su inversa es su traspuesta conjugada.

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{1.4}$$

Sea la matriz A como en la ecuación 1.4. Entonces como vemos en la ecuación , el operador que define A transforma $|0\rangle$ en $|1\rangle$ y viceversa. Es la matriz que define el operador NOT cuántico que realiza una rotación de 180º respecto del eje X en la esfera de Bloch.

$$\begin{aligned} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \end{aligned} \tag{1.5}$$

Cada matriz unitaria define una puerta cuántica de un bit. Una de las puertas más importantes es la puerta de Hadamard o puerta H (ecuación 1.6), que transforma el estado $(1,0)$ en el estado $\frac{1}{\sqrt{2}}(1,1)$ y el $(0,1)$ en el $\frac{1}{\sqrt{2}}(1,-1)$, ambos sobre el ecuador de la esfera de Bloch. Creando así un estado de superposición entre los estados $|0\rangle$ y $|1\rangle$. También es especialmente relevante la puerta de rotación de fase o puerta R_φ (ecuación 1.7), que realiza una rotación de φ radianes sobre el eje Z. A las puertas que realizan rotaciones sobre los distintos ejes también se les suele llamar puertas $X^{\frac{1}{2^n}}$, $Y^{\frac{1}{2^n}}$ y $Z^{\frac{1}{2^n}}$ donde $\frac{1}{2^n}$ indica la porción de vuelta que se gira alrededor del eje dado. Por ejemplo, la puerta Z realiza una rotación de media vuelta alrededor del eje Z y la puerta $X^{\frac{1}{2}}$ realiza una rotación de un cuarto de vuelta alrededor del eje X.

$$H \Longleftrightarrow \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \tag{1.6}$$

$$R_\phi \Longleftrightarrow \begin{pmatrix}1 & 0\\ 0 & e^{i\varphi}\end{pmatrix}. \tag{1.7}$$

Debido a su naturaleza, los operadores cuánticos son invertibles. Esto quiere decir que no podemos colapsar la información de dos qubits en uno solo pues perderíamos información y la operación no sería reversible. Dada una puerta cuántica y la matriz unitaria que la define, la puerta inversa será la definida por la inversa de la matriz, que recordemos coincide con su traspuesta conjugada. Además, si enlazamos varias puertas en un circuito, este será equivalente a una única puerta definida por el producto de las matrices que definen las puertas individuales que enlazamos. Lo cual ocurre en el circuito de la imagen 1.2, que es equivalente a la puerta not como vemos en ecuación 1.8.

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & e^{i\pi}\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}. \tag{1.8}$$

Se pueden ver algunas de las puertas cuánticas de un qubit que utilizaremos en este texto así como la acción detallada que realizan en la tabla 4.1.

1.2. Múltiples qubits

Toda la matemática y base de esta sección se desarrolla en [HPKM07, Hir12, NC11].

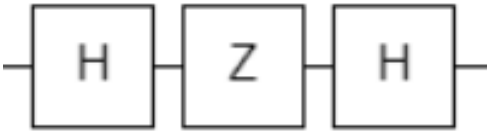


Figura 1.2: Circuito cuántico $H - Z - H$

Cuando trabajamos con múltiples qubits, digamos n, hay que tener en cuenta que entre todos forman un sistema cuántico en el que cada estado se corresponde con un vector unitario de un espacio de Hilbert de dimensión 2^n. Esto demuestra la gran potencia de la computación cuántica. Sin embargo, al igual que pasaba con un qubit, solo los estados correspondientes con los vectores de la base canónica $e_i = (0, 0 \ldots, 0, 1, 0 \ldots, 0)$ son observables en una medición. Hay que tener en cuenta que el vector e_i se corresponde con el estado $|bin(i)\rangle$ donde $bin(i)$ es la representación binaria de i.

Esto es, un sistema con dos qubits, se corresponde con un espacio de Hilbert de dimensión cuatro en el que los estados observables son los que se muestran en la ecuación 1.9.

$$\begin{pmatrix}1\\0\\0\\0\end{pmatrix} \Longleftrightarrow |00\rangle, \quad \begin{pmatrix}0\\1\\0\\0\end{pmatrix} \Longleftrightarrow |01\rangle, \qquad \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \Longleftrightarrow |10\rangle, \quad \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \Longleftrightarrow |11\rangle. \tag{1.9}$$

De forma análoga a lo que ocurre con un sistema de un solo qubit, cualquier otro estado α del espacio se puede expresar como una combinación lineal de los vectores e_i que forman la base canónica (ecuación 1.10).

$$\alpha = \sum_{i=0}^{2^n-1} a_i e_i \quad con \quad \sum_{i=0}^{2^n-1} |a_i|^2 = 1. \tag{1.10}$$

Además, la probabilidad $p_\alpha(e_i)$ de observar en una medición del estado α el estado e_i es $|a_i|^2$. El ejemplo de la ecuación 1.11, en un sistema de dos qubits ayuda a visualizar la

situación.

$$\alpha = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\\0\\0\end{pmatrix} + \frac{1}{2}\begin{pmatrix}0\\1\\0\\0\end{pmatrix} + \frac{1}{2}\begin{pmatrix}0\\0\\0\\1\end{pmatrix} = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|11\rangle, \tag{1.11}$$
$$p_\alpha(|00\rangle) = \frac{1}{2} \quad p_\alpha(|01\rangle) = \frac{1}{4} \quad p_\alpha(|10\rangle) = 0 \quad p_\alpha(|11\rangle) = \frac{1}{4}.$$

1.2.1. Operaciones sobre múltiples qubits

Igual que ocurría en un sistema con un qubit, cada operación sobre un sistema de múltiples qubits se describe mediante una matriz unitaria, en este caso de orden 2^n. Cada una de estas matrices es en realidad un cambio de base, es decir, para definir un operador o puerta que actúa sobre n qubits, en realidad solo es necesario saber cuál queremos que sea la imagen de cada uno de los vectores e_i de la base canónica, la matriz formada por los vectores $im(e_i)$, imagen de cada e_i, será la matriz que defina la puerta. Hemos de recordar que esa matriz debe ser un cambio de base unitario. Es decir, los vectores $im(e_i)$ deben ser un conjunto de vectores linealmente independiente para garantizar que la matriz sea de cambio de base, y además deben tener norma uno.

Supongamos que queremos definir una puerta sobre un sistema de dos qubits que actúe como se muestra en la ecuación 1.12.

$$\begin{aligned}|00\rangle &\longrightarrow |00\rangle,\\ |01\rangle &\longrightarrow |01\rangle,\\ |10\rangle &\longrightarrow |11\rangle,\\ |11\rangle &\longrightarrow |10\rangle.\end{aligned} \tag{1.12}$$

Una vez definida la imagen de los vectores e_0, e_1, e_2 y e_3, al ser estos una base del espacio, queda definida la imagen de cualquier otro vector, y la matriz que define nuestro operador será como en la ecuación 1.13.

$$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}. \tag{1.13}$$

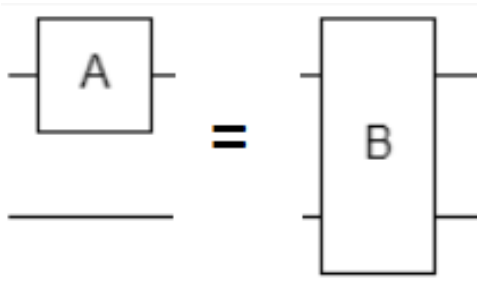


Figura 1.3: Circuito con puerta A en el primer qubit

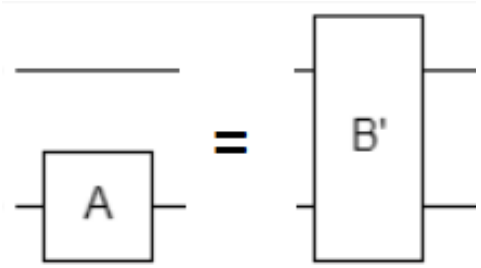


Figura 1.4: Circuito con puerta A en el segundo qubit

Esta es la puerta que llamamos $CNOT$ o not controlada. que invierte el segundo qubit si el primero está a uno.

En el caso general, la matriz M_P que define una puerta P que manda el vector e_i a $Im_P(e_i)$ será la de la ecuación 1.14.

$$M_P = (Im_P(e_0) \quad Im_P(e_1) \quad Im_P(e_2) \quad ... \quad Im_P(e_{2^n-1})) \tag{1.14}$$

Si aplicamos a un sistema de dos qubits una puerta A de un qubit con matriz M_A (ecuación 1.15) como el descrito en las figuras 1.3 y, las matrices M_B y $M_{B'}$ de orden cuatro que describen la operación realizada sobre el sistema de dos qubits serán, dependiendo de a qué qubit apliquemos la operación, serán respectivamente las de las ecuaciones 1.16 y 1.17.

$$M_A = \begin{pmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \end{pmatrix}, \tag{1.15}$$

$$M_B = \begin{pmatrix} a_{00}I & a_{10}I \\ a_{01}I & a_{11}I \end{pmatrix}, \tag{1.16}$$

$$M_{B'} = \begin{pmatrix} M_A & 0 \\ 0 & M_A \end{pmatrix}. \tag{1.17}$$

Esto quiere decir que determinadas puertas de múltiples qubits se pueden describir como puertas de tamaño menor, de hecho, de igual forma que en computación clásica la puerta $NAND$ o el conjunto de puertas AND, OR, NOT se denominan conjuntos de puertas universales pues permiten la implementación de cualquier circuito, todos los circuitos cuánticos se pueden implementar a partir de la puerta $CNOT$ de dos bits y puertas de un bit.

Se pueden ver algunas de las puertas cuánticas de múltimples qubits que utilizaremos en este texto así como la acción detallada que realizan en la tabla 4.1.

Puerta	Descripción
X	Rotación de π radianes alrededor del eje X
Y	Rotación de π radianes alrededor del eje Y
Z	Rotación de π radianes alrededor del eje Z
H	Mapea el eje Z en el X y viceversa
$X^{\frac{1}{2^n}}$	Rotación de $\frac{\pi}{2^n}$ radianes alrededor del eje X
$Y^{\frac{1}{2^n}}$	Rotación de $\frac{\pi}{2^n}$ radianes alrededor del eje Y
$Z^{\frac{1}{2^n}}$	Rotación de $\frac{\pi}{2^n}$ radianes alrededor del eje Z
$X^{\frac{-1}{2^n}}$	Rotación de $\frac{-\pi}{2^n}$ radianes alrededor del eje X
$Y^{\frac{-1}{2^n}}$	Rotación de $\frac{-\pi}{2^n}$ radianes alrededor del eje Y
$Z^{\frac{-1}{2^n}}$	Rotación de $\frac{-\pi}{2^n}$ radianes alrededor del eje Z
U	Esta puerta generaliza todas las anteriores que son casos particulares.
$CNOT$	Si el primer qubit es 1, aplica la puerta X al segundo.
$SWAP$	Intercambia el estado de los dos qubits a los que afecta.
CP	Puerta P controlada. Si el primer qubit es 1, aplica la puerta P al segundo.
$CCNOT$	Si el primer y segundo qubit son 1, aplica la puerta X al tercero.
$CSWAP$	Si el primer qubit es 1 interccambia los estados de los qubits segundo y tercero.

Cuadro 1.1: Tabla de puertas cuánticas

1.3. Equivalencia Cuántica-Clásica

Hemos hablado de la potencia de quantum computing, pero aún no hemos visto que realmente es al menos igual de potente que la computación clásica, de hecho es más potente. Para ello vamos a ver que se puede implementar un conjunto universal clásico mediante puertas cuánticas.

Lo primero que hay que tener en cuenta es que los operadores cuánticos deben ser reversibles, es decir, no podemos implementar una puerta dos a uno, aunque sí una dos a dos o tres a tres que se comporte en una de sus salidas como la puerta que nos interesa.

Con este fin presentamos la puerta de Toffoli o $CCNOT$ de tres qubits, una de las puertas más importantes. La puerta de Toffoli se comporta ante entradas clásicas, es decir, estados observables, según describe la tabla 1.2, y tiene la matriz de la ecuación 1.18.

INPUT			OUTPUT		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	0

Cuadro 1.2: Tabla de verdad de la puerta de Toffoli

$$M_T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.18}$$

Cuando la entrada menos significativa es uno, esta puerta se comporta como la puerta $NAND$ en esa salida (imagen 1.5), que tal y como hemos comentado, es en sí misma un

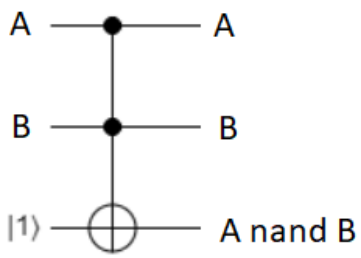


Figura 1.5: Implementación de una puerta $NAND$ a partir de la puerta de Toffoli

conjunto clásico universal, y por tanto cualquier circuito clásico podría implementarse como circuito cuántico solo a partir de esta puerta.

Esto podría darnos la idea de implementar cualquier circuito clásico existente mediante su equivalencia cuántica esperando una ganancia de algún tipo solo por ser cuántico. Aunque ciertamente es posible implementar cualquier circuito clásico mediante su traducción a puertas $NAND$, esto no suele ser una buena idea pues no estaríamos teniendo en cuenta la potencia de la representación cuántica de la información y posiblemente caeríamos en un circuito con muchas más puertas de las necesarias, y desde luego no podríamos reducir el orden de complejidad del algoritmo clásico. Por no hablar de que cabe la posibilidad de que el circuito, aunque correcto, sea tan largo que no tenga utilidad más allá de correrlo en un simulador, al menos con la tecnología actual.

Más adelante veremos ejemplos de comparativas entre circuitos cuánticos realizados a partir de la traducción directa del circuito clásico y una alternativa aprovechando la naturaleza de la representación cuántica de la información.

Capítulo 2

Introducción a los circuitos cuánticos

Aunque se suele hablar de programas o algoritmos cuánticos, en realidad actualmente es más correcto hablar de circuitos cuánticos. Aunque en teoría, por la equivalencia entre computación cuántica y clásica contada en la sección 1.3, sería posible contar con un procesador cuántico, aún nos encontramos técnicamente muy lejos de algo así.

Se puede decir que actualmente trabajar con computación cuántica es en realidad diseñar circuitos cuánticos. Un diseño a muy bajo nivel más parecido a la programación hardware, como VHDL, que a los lenguajes de alto nivel conocidos por todos.

Un circuito cuántico tiene tres partes, en primer lugar la preparación de la entrada, es decir, llevar cada qubit a un estado determinado que represente la entrada. Generalmente el estado inicial suele ser $|00...0\rangle$ y después mediante puertas NOT se invierten los qubit que así lo requieran.

Una vez tenemos la entrada adecuada se procede a operar con los qubits aplicando los operadores o puertas. Es importante tener en cuenta que durante toda esta fase no se puede consultar el estado de los qubits.

Finalmente se procede a realizar las mediciones sobre los qubits. Al realizar las mediciones destruimos el estado cuántico del qubit y observamos uno de los dos estados observables $|0\rangle$ o $|1\rangle$. Este resultado se almacena en un registro clásico como 0 o 1 respectivamente.

2.1. Computadores cuánticos reales

La tecnología que se utiliza actualmente para representar el modelo de computación cuántico consiste en manipular partículas microscópicas mediante la aplicación de microondas. Se trata de un sistema muy complejo y en el que la precisión es crucial. Más adelante evidenciaremos como una mínima imprecisión en una rotación, podría hacer que el resultado de un cálculo cambiara completamente.

Además, hay que conseguir mantener el sistema estable para que los estados de los qubits sean fijos y fiables, y hay que evitar interacciones no deseadas entre los qubits y el exterior, y también entre unos qubits y otros.

Esto pone de manifiesto la complejidad tecnológica del sistema y por qué de momento muchos de los resultados que se obtienen son teóricos o sólo pueden ejecutarse sobre simuladores.

Sin embargo, la tecnología en este campo avanza a pasos agigantados, y aunque los circuitos que diseñamos hoy puede que sólo funcionen en un simulador o en un computador con muy pocos qubits, se espera que en un futuro cercano sean útiles en computadores cuánticos reales.

2.2. Frameworks de diseño de circuitos cuánticos

En esta sección presentamos algunos de los frameworks y herramientas disponibles para el desarrollo de programas en computadoras cuánticas NISQ.

-  Quiskit [AAea19]. Es un framwork open-source en python para el desarrollo de programas sobre computadoras cuánticas que permite utilizarlas como coprocesadores en la nube. Dispone de un set completo para construir programas cuánticos, una colección de simuladores y la posibilidad de ejecutar los circuitos en computadoras reales a través de la librería de IBM.
-  Quirk [Se]. Es una aplicación web que proporciona una GUI para construir y

simular circuitos cuánticos de tamaño reducido. El sistema de edición "drag and drop"permite incluir cualquier puerta lógica o displays intermedios para depurar el circuito. La simulación se hace en tiempo real a la par que se edita el circuito mostrándose el resultado en todo momento en los displays situados al final del

- Q# [Mic]. Es un lenguaje de programación desarrollado por Microsoft que permite programar computadoras cuánticas y utilizarlas como coprocesador en la nube.
- Cirq [Cir]. Es un framework open-source desarrollado por Google para el desarrollo de software en computadoras cuánticas NISQ.

2.3. Qiskit

Qiskit es un framework de código abierto para la computación cuántica. Proporciona herramientas para crear y manipular circuitos cuánticos y ejecutarlos en prototipos de dispositivos cuánticos reales y simuladores. Sigue el modelo de circuito para la computación cuántica universal, y puede usarse para cualquier hardware cuántico que siga este modelo.

Fue fundado por IBM Research para permitir el desarrollo de software para su servicio de computación cuántica en la nube. Las contribuciones también son realizadas por patrocinadores externos, típicamente de instituciones académicas.

Qiskit será el framework que utilizaremos en este texto para la implementación de los circuitos con los que realizaremos pruebas y experimentos pues permite automatizar la representación de circuitos cuánticos gráficamente, y también proporciona diversas formas de visualizar los resultados obtenidos incluyendo gráficas varias.

Además Qiskit permite lanzar los circuitos diseñados contra distintos backends incluyendo simuladores, en los que se pueden configurar niveles de ruido y precisión de los sistemas; y también contra los diversos computadores cuánticos reales que tiene IBM disponibles para el público en la nube.

Podemos ver un sencillo ejemplo para crear un circuito con Quiskit en el listing 2.1 cuyo

resultado se puede ver en la figura 2.3a.

Listing 2.1: Ejemplo de creación de un circuito sencillo con Quiskit

```
#Imports
import numpy as np
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit import Aer, execute
from qiskit.tools.visualization import plot_histogram
from qiskit.providers.aer import QasmSimulator

# Circuit building
qr = QuantumRegister(2, 'qr')
cr = ClassicalRegister(2, 'cr')
circ = QuantumCircuit(qr, cr)
circ.h(qr[0])
circ.cx(qr[0], qr[1])
circ.measure(qr, cr

# Execution on simulator
simulator = Aer.get_backend('qasm_simulator')
result = execute(circ, simulator).result()
counts = result.get_counts(circ)
plot_histogram(counts, title='Bell-State counts')
```

En la tabla 4.1 vimos las puertas utilizadas en los circuitos hasta el momento que también implementa Qiskit. Todas las puertas presentadas son casos particulares de la puerta U a partir de la cual se implementan el resto. Las máquinas reales de IBM solo tienen implementadas las puertas $U2$, $U3$ (Puerta U con dos o tres parámetros) y CX a partir de las cuales se construyen los casos particulares. Un ejemplo son las rotaciones sobre los ejes X, Y y Z. Se pueden conseguir pasando una matriz de Pauli a la puerta U para que se comporte como cada una de ellas. Otra puerta interesante es la CCX o puerta de $Toffoli$ que es una negación doblemente controlada. Esta puerta se forma a partir de una combinación de CX, H, T y T^t tal y como podemos ver en la figura 2.1 [Gee]. Estos son factores muy a tener en cuenta a la hora de medir con precisión la complejidad en número de puertas de los circuitos cuánticos ya que las puertas multi-qubit realemnte están compuestas por varias de un solo qubit o dos qubits que computan los casos parciales.

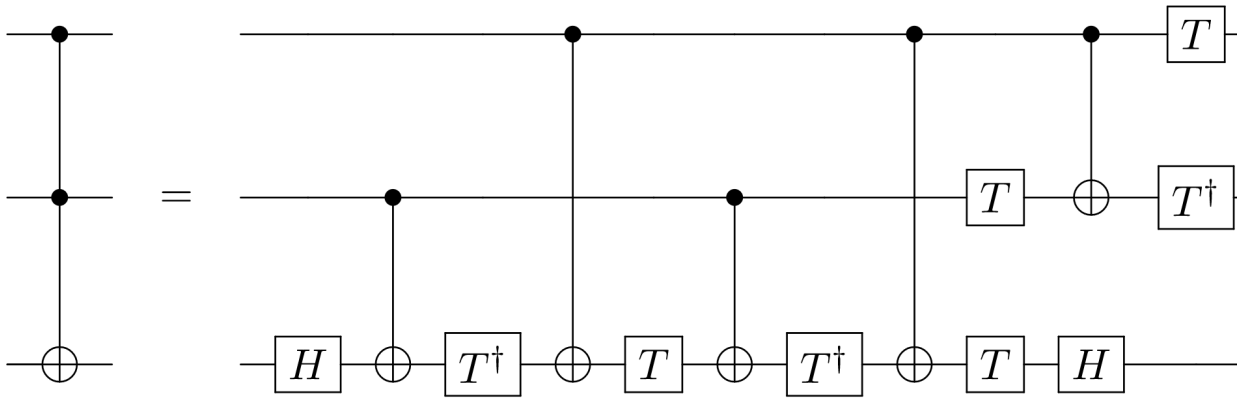


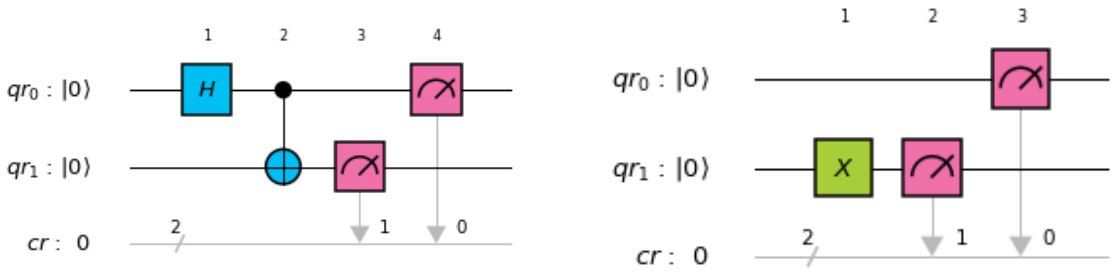
Figura 2.1: Toffoli gate

2.3.1. Simuladores

En esta sección presentamos los diferentes simuladores que Qiskit pone a disposición del desarrollador con una prueba de cada uno en dos circuitos sencillos representados en las figuras 2.2a y 2.2b. Cada vez que se lanza un circuito a un backend (simulador o real) se ejecuta varias veces (1000 por defecto) para poder obtener una distribución estadística con los resultados arrojados por todas las ejecuciones. En las gráficas de barras presentadas a lo largo de texto cada barra representa un estado observable al que se ha llegado tras la ejecución del circuito. El número encima de la barra representa la cantidad de veces que se ha llegado a ese resultado mediante su probabilidad (en la figura 2.3 vemos que 00 ha salido com probabilidad 0.527 y 11 con probabilidad 0.473). En la parte inferior está el estado observable codificado en bits clásicos (la operación de medición vuelca el resultado sobre un registro clásico). En los circuitos implementados para este texto el bit más significativo es el que está más abajo en las imágenes tal y como podemos ver en la figura 2.2b que representa el dos en binario (10).

Ideal (QasmSimulator)

Este simulador recrea el comportamiento de una computadora cuántica y es la herramienta más utilizada para simular circuitos en condiciones ideales. Es perfecto para probar y validar circuitos antes de llevarlos a ejecutar en una máquina real 2.3.2. Para poder utili-



(a) Bell's state

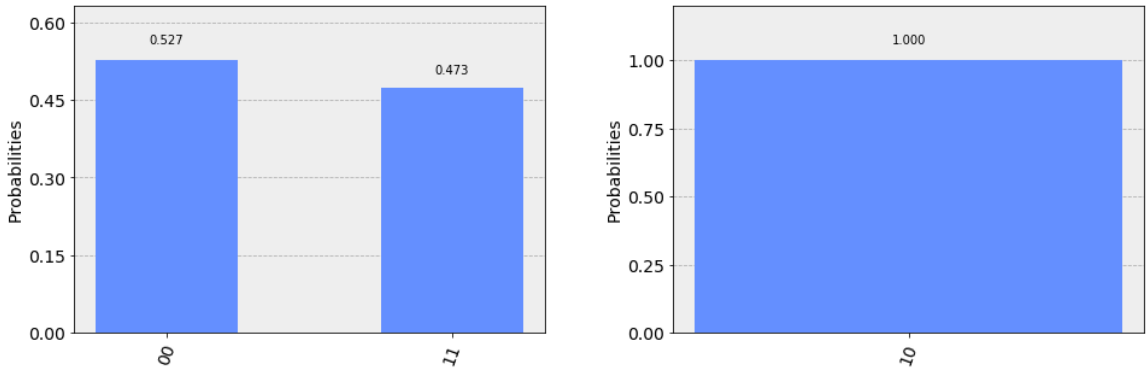
(b) Dos en binario

Figura 2.2: Circuitos de muestra

zarlo se necesita obtener el objeto del backend y un circuito ya creado como se muestra en el ejemplo del listing 2.1. Podemos observar como un circuito tan simple como escribir el número dos en binario devuelve siempre el mismo resultado 2.3b al ser un circuito determinista. Tras ejecutar un circuito simple como es el de Bell's state o entrelazamiento cuántico entre dos qubits 2.3a podemos observar el comportamiento de un circuito indeterminista y la distribución de los distintos estados observables que ha arrojado la ejecución. Por supuesto el simulador no devuelve la gráfica ya construída sino una colección con los resultados de cada ejecución. Es importante señalar que en los circuitos presentados solo medimos los qubits que contienen la salida y no el resto ya que no aportan nada al resultado final y pueden llegar a complicar mucho el procesado de la colección de resultados obtenida como veremos en las siguientes secciones.

Modelo de ruido

Este simulador es el mismo que el de la sección anterior junto con un modelo de ruido que nos permite simular una computadora cuántica en condiciones reales. Para obtener este modelo de ruido hay que acudir a la API de IBMQ [IBM19b] para que nos proporcione en tiempo real un modelo de ruido de sus máquinas. Como podemos ver en el ejemplo 2.2 es necesario un token especial que se obtiene a través del portal de IBM Quantum Experience [IBM19a] (es necesario la creación de una cuenta) y que utilizamos para recuperar el modelo de alguna de las máquinas disponibles. Las máquinas disponibles son *'ibmqx4'* y



(a) Bell's state

(b) Dos en binario

Figura 2.3: Simulación ideal

'ibmq_16_melbourne' de 5 y 14 qubits respectivamente. Para poder utilizar este modelo de máquina real es necesario pasarlo junto con el simulador para que los aplique durante la ejecución del circuito. El circuito elegido es un sumador basado en QFT que está encapsulado en una librería ya que no nos hace falta conocer los detalles para el desarrollo de las características de los simuladores o backend reales. A continuación exponemos las tres componentes principales del modelo de ruido:

1) *Coupling map* 2.4. Este grafo define la manera en la que están distribuídos los qubits así como los enlaces entre ellos. Este factor añade una restricción a la hora de diseñar circuitos ya que habrá casos en los que dos o más qubits no puedan interactuar entre sí siendo imposible el entrelazamiento de todos los qubits del modelo como es el caso de los qubits 0 y 8 en la máquina *'ibmq_16_melbourne'*. Un ejemplo donde esta restricción aplica es una QFT de 8 o más qubits en el backend *'ibmq_16_melbourne'*.

2) *Basis gates*. Son las puertas de cálculo básicas implementadas en las máquinas reales que conforman un conjunto universal. Las puertas implementadas en este caso son $U2$, $U3$ y CX. Todas las puertas vistas en la tabla 4.1 se pueden implementar como casos particulares de $U2$ y $U3$ (las rotaciones de un qubit de X, Y y Z son casos particulares de U2) o como una combinación de $U2$, $U3$ y CX como por ejemplo la puerta de Toffoli o CCX. No hay que olvidarse de la puerta de medición que pese a no estar incluída en este conjunto (no

sirve para calcular) también es importante tenerla presente como veremos más adelante.

3) *Noise model*. Contiene el factor de ruido de cada puerta en cada qubit (incluída la puerta de medición). Debido a este factor de ruído los resultados obtenidos tras la ejecución de los circuitos se verán afectados distorsionándolos tal y como se puede ver en la figura 2.5b donde un circuito determinista está arrojando cuatro resultados diferentes. Por supuesto en este caso es fácil detectar una moda en el resultado que nos permite identificar el estado observable que contiene la salida correcta. Como veremos en el capítulo 5 con circuitos más complejos los resultados quedan más afectados siendo imposible determinar la salida correcta. Sin embargo hay otro factor a tener en cuenta que puede distorsionar más aun los resultados y son los *qubits dummy* o qubits vacíos. Como vimos en la sección 1 todos los qubits conforman un espacio $n-dimensional$ de Hilbert complejo cuyas operaciones se definen mediante el producto tensor de manera que todos los qubits están 'unidos' y unos se afectan a otros. Por lo tanto es importante tratar correctamente los *qubits dummy* para que no afecten al resultado tal y como vemos en la figura 2.5c donde hemos incluído dos *qubits dummy* en los cuales no se realiza operación alguna pero debido al ruído afectan al resto generando un resultado más distorsionado. Vemos que las dos modas (0010 y 1010) contienen el resultado correcto del circuito que es $'10'$ sin embargo el qubit más significativo se ha visto afectado por el ruído poniéndose a $'1'$ en uno de los casos a pesar de no operar sobre él. La manera de tratar estos qubits dummy consiste en no medirlos nunca de manera que los resultados queden unidos solo en los qubits útiles que si medimos. Cada vez que se ejecuta un circuito con modelo de ruido o en una máquina real todos los qubits están activos aunque nuestro circuito no los use de manera que hay que tenerlos siempre en cuenta. En todos los circuitos del texto (salvo el ejemplo de la figura 2.5c) se han tratado los qubits dummy en las ejecuciones con modelo de ruído y en máquinas reales.

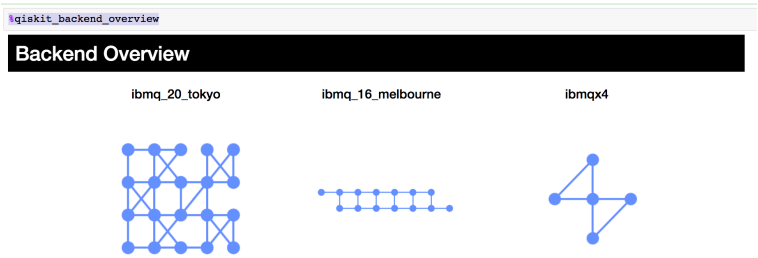
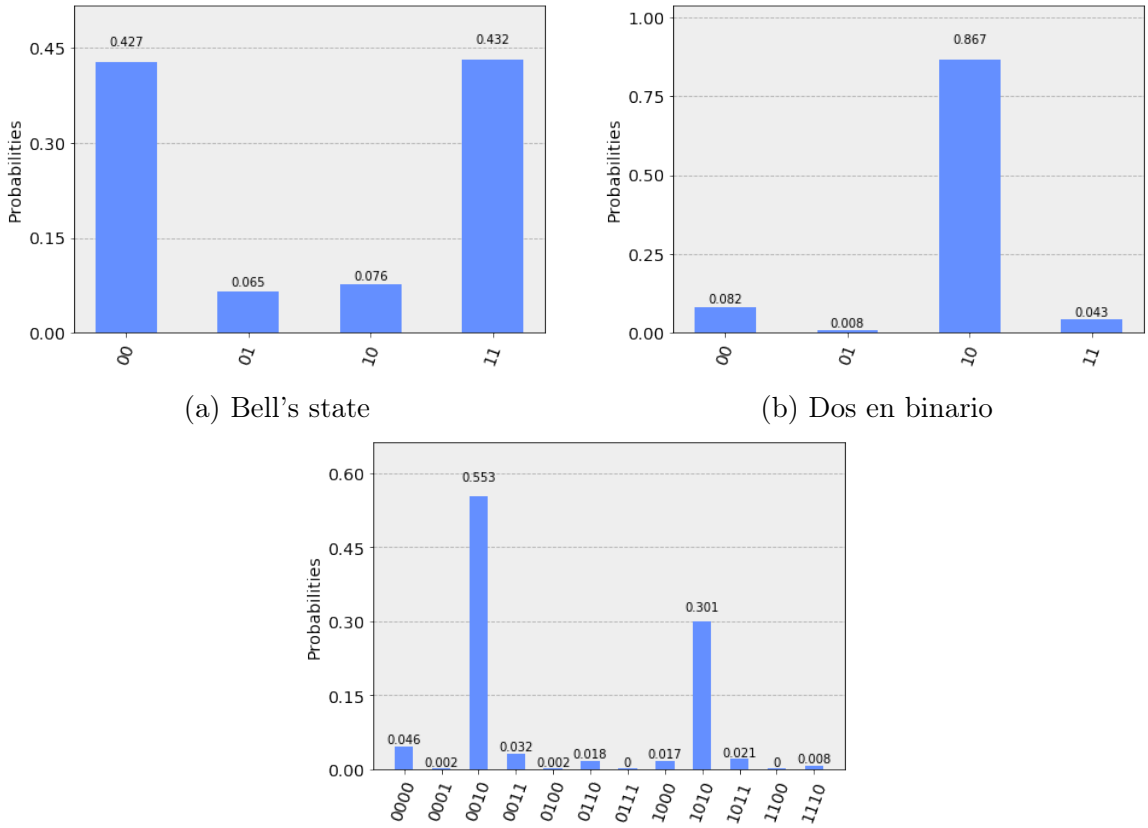


Figura 2.4: Coupling maps

Listing 2.2: Ejemplo de creación de un circuito con modelo de ruido

```
#Imports
    # Qiskit tools
    from qiskit.providers.aer import noise
    from qiskit import execute, Aer, IBMQ
    from qiskit.tools.visualization import plot_histogram

    # My circuit
    from aritmetica import sumadorQFT

# Circuit building
circuit = circuit = sumadorQFT.getAdderCircuit(1, 1, 2)
counts = qsamExp.qsamExperiment(circuit)
plot_histogram(counts)

# Model extraction
IBMQ.enable_account('token')

device16 = IBMQ.get_backend('ibmqx4')
properties = device16.properties()

    # Model properties
    coupling_map = device16.configuration().coupling_map
    noise_model = noise.device.basic_device_noise_model(properties)
    basis_gates = noise_model.basis_gates

# Execution on simulator
simQasm = Aer.get_backend('qasm_simulator')
result = execute(circuit,
                 simQasm,
                 noise_model=noise_model,
                 coupling_map=coupling_map,
                 basis_gates=basis_gates).result()
counts = result.get_counts(circuit)
plot_histogram(counts)
```



(a) Bell's state

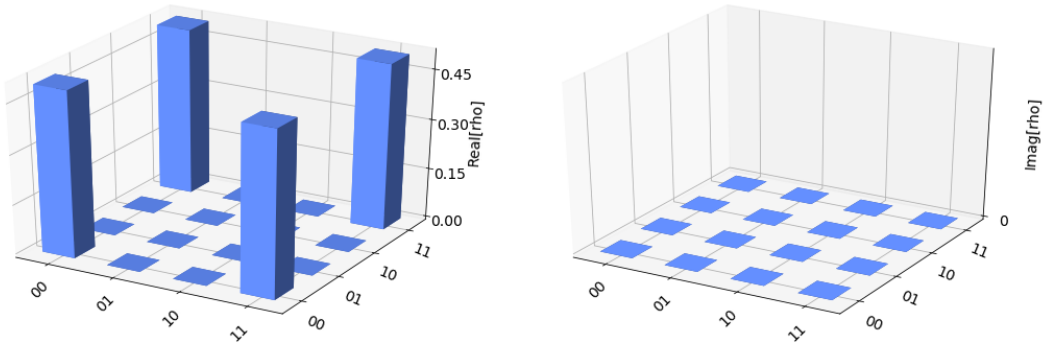
(b) Dos en binario

(c) Dos en binario con qubit dummy

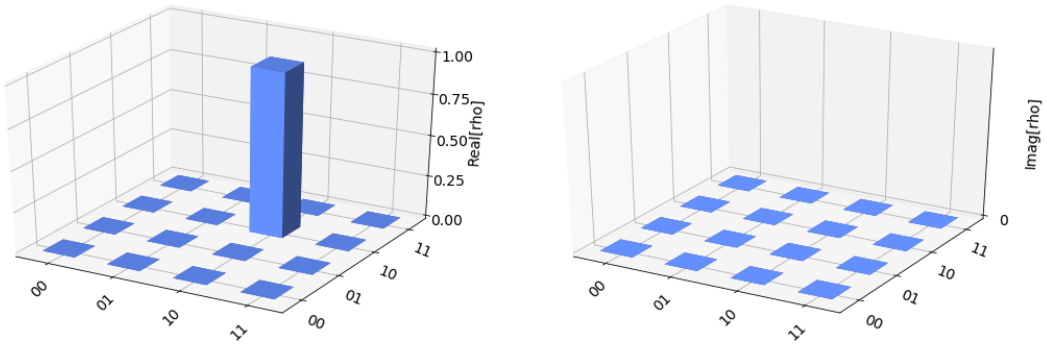
Figura 2.5: Simulación con modelo de ruido

Statevector

Este simulador lanza una sola vez el circuito que recibe y muestra el estado vectorial final de la ejecución dividido en la parte real y la compleja. Para lanzar este simulador auxiliar se procede exactamente igual que con el simulador ideal 2.3.1 solo que indicando que queremos el *'statevector_simulator'* tal y como podemos observar en el código 2.3. Además es necesaria otra manera de imprimir las grñaficas ya que ahora no obtenemos una colección de resultados si no del estado final de la ejecución. Con los circuitos de prueba observamos en las figuras 2.6a y 2.6b que solo influyen en la parte real. Este simulador auxiliar se emplea para verificar los circuitos al ofrecer una visión conjunta de todas las posibles salidas tanto observables como no observables.



(a) Bell's state



(b) Dos en binario

Figura 2.6: Simulación statevector

Listing 2.3: Ejemplo de ejecución de un circuito en un simulador Statevector

```
#Imports
    # Qiskit tools
    from qiskit import execute, IBMQ
    from qiskit.tools.visualization import plot_histogram
    # My circuit
    from aritmetica import sumadorQFT

# Circuit building
circuit = circuit = sumadorQFT.getAdderCircuit(1, 1, 2)
counts = qsamExp.qsamExperiment(circuit)
plot_histogram(counts)

# Backend extraction
IBMQ.enable_account('token')
simulator = IBMQ.get_backend('statevector_simulator')

# Execution on real backend
result = execute(circuit, simulator).result()
statevector = result.get_statevector(circ)
plot_state_city(statevector, title='Bell state')
```

Unitary matrix

Esta herramienta se encarga de calcular la matriz representativa del circuito. Para ejecutar este simulador hay que seguir los mismos pasos que para el simulador ideal solo que indicando que queremos el simulador auxiliar *'unitary_simulator'*. Este simulador nos devolverá una matriz de números complejos en forma de array de arrays 2.7. Su trabajo consiste en realizar el producto tensor de todas las matrices representativas de cada puerta a lo largo del circuito. Debido a las propiedades del producto tensor esta operación toma mucho esfuerzo en tiempo y recursos para llevarse a cabo pues tiene un coste exponencial con respecto al número de qubits del circuito pero puede ser muy útil como herramienta de precálculo ya que invertir cierto tiempo en calcular la matriz representativa de un circuito nos permite codificarlo en una sola puerta y aprovechar al máximo las propiedades de las computadoras cuánticas. Para que esto fuera posible también haría falta una máquina real que pueda codificar cualquier puerta a través de su matriz representativa de cualquier tamaño y precisión de rotación, por el momento no existe tal máquina. En las figuras 2.1 y 2.2 podemos observar las matrices representativas de los circuitos Bell's state y un dos respectivamente. Aunque las matrices tengan valores enteros o reales no hay que olvidar que estamos trabajando en el dominio de los números complejos. En estos casos las componentes complejas son todas 0 y por eso se omiten.

$$\begin{pmatrix} 0.7 & 0.7 & 0 & 0 \\ 0 & 0 & 0.7 & 0.7 \\ 0 & 0 & 0.7 & -0.7 \\ 0.7 & -0.7 & 0 & 0 \end{pmatrix} \quad (2.1)$$

(a) Matriz de Bell's state

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.2)$$

(b) Matriz de dos en binario

Figura 2.7: Matrices unitarias

Listing 2.4: Ejemplo de cálculo de la matrix unitaria de un circuito

```
#Imports
    # Qiskit tools
    from qiskit import execute, IBMQ
    from qiskit.tools.visualization import plot_histogram

    # My circuit
    from aritmetica import sumadorQFT

# Circuit building
circuit = circuit = sumadorQFT.getAdderCircuit(1, 1, 2)
counts = qsamExp.qsamExperiment(circuit)
plot_histogram(counts)

# Backend extraction
IBMQ.enable_account('token')
simulator = IBMQ.get_backend('unitary_simulator')

# Execution on real backend
result = execute(circuit, simulator).result()
unitary = result.get_unitary(circuit)
print("Circuit unitary:\n", unitary)
```

2.3.2. Máquinas reales

La API de IBMQ también nos permite lanzar nuestros circuitos a una máquina real situada en un host remoto. En este caso será suficiente con obtener solo el backend y no recuperar todo el modelo de ruido ya que no vamos a lanzar el circuito sobre un simulador si no a la máquina real. Podemos ver en el código de ejemplo 2.5 que todo queda más simplificado. En cambio se añade un tiempo de espera por la respuesta del backend ya que la tarea debe esperar su turno de ejecución. No es posible ejecutar continuadamente circuitos en la máquina real pues hay un sistema de créditos que limita la cuota de acceso ya que

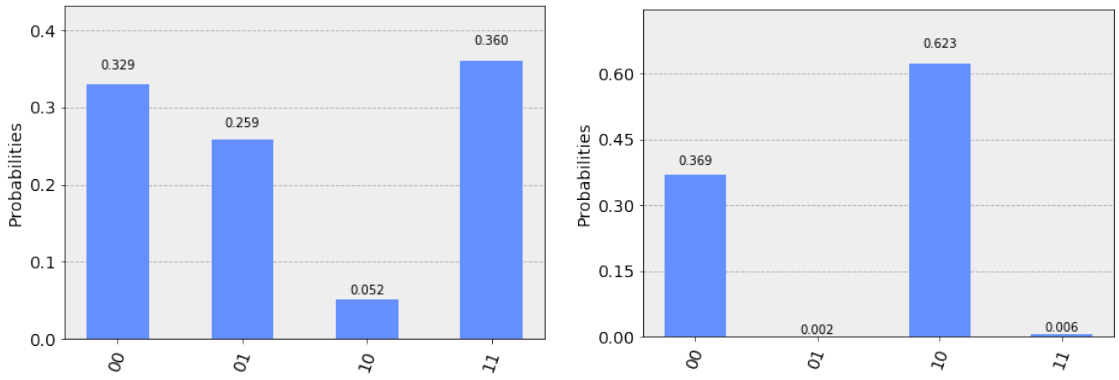
se trata de máquinas abiertas al público con todo el tráfico consecuente de ello. En este caso las restricciones de ruido, puertas implementadas y grafo de distribución de los qubits también aplican al igual que el simulador con modelo de ruido 2.3.1. Cabría esperar que los resultados de la simulación con ruido y los resultados arrojados por la máquina real fueran parecidos pero esto no tiene por qué ser así tal y como puede verse en las figuras 2.5a y 2.8a ya que la máquina real se descalibra con el paso del tiempo y varían los fatores de ruído. Además de las restricciones indicadas en la sección 2.3.1 hay que añadir que las máquinas reales son recursos remotos a los cuales se accede a través de la red con las consecuencias que ello implica.

Listing 2.5: Ejemplo de creación de un circuito con modelo de ruido

```
#Imports
    # Qiskit tools
    from qiskit import execute, IBMQ
    from qiskit.tools.visualization import plot_histogram

    # My circuit
    from aritmetica import sumadorQFT

# Circuit building
circuit = circuit = sumadorQFT.getAdderCircuit(1, 1, 2)
counts = qsamExp.qsamExperiment(circuit)
plot_histogram(counts)

# Backend extraction
IBMQ.enable_account('token')
device16 = IBMQ.get_backend('ibmqx4')

# Execution on real backend
result = execute(circuit, device16).result()
counts = result.get_counts(circuit)
plot_histogram(counts)
```

2.4. Tipos de algoritmos cuánticos

Es importante distinguir entre los conceptos de algoritmo probabilístico y el de sistema impreciso. Al igual que en el caso clásico, al diseñar un algoritmo cuántico podemos hablar de dos tipos, algoritmos deterministas y algoritmos probabilísticos. Los segundos son inherentemente indeterministas, sin embargo, debido a la complejidad del sistema físico que se



(a) Bell's state

(b) Dos en binario

Figura 2.8: Ejecución en una máquina real

utiliza en los computadores reales y la imprecisión que esto conlleva, los algoritmos cuánticos deterministas, también producen resultados indeterministas, aunque sería más correcto decir imprecisos. Ejemplificaremos inmediatamente esta situación.

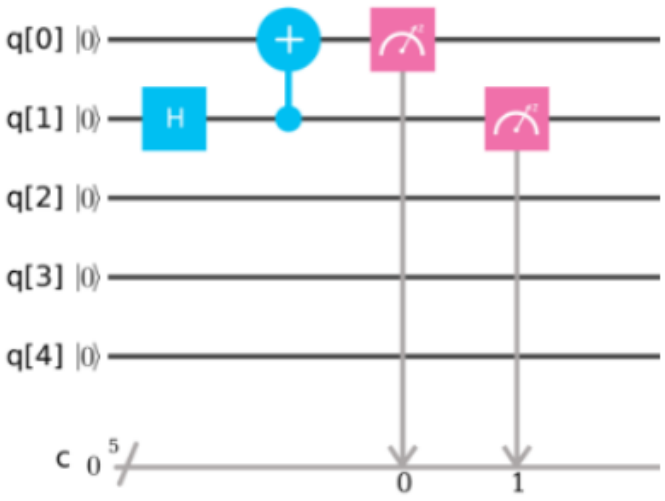
El circuito de la figura 2.9a, implementado en Qiskit, consta únicamente de una puerta Hadamard, una $CNOT$ y dos mediciones. En el caso de computador cuántico ideal, se entiende por esto sin imprecisiones, los resultados son probabilísticos, pues sólo son observables los estados $|00\rangle$ y $|11\rangle$, cada uno con una probabilidad próxima a 0.5, tal y como se observa en el gráfico de la figura 2.9b.

Sin embargo, si ejecutamos este mismo circuito en un computador real, observamos que aparecen de forma marginal resultados teóricamente no observables (figura 2.9c). Esto se debe a las imprecisiones del ordenador real.

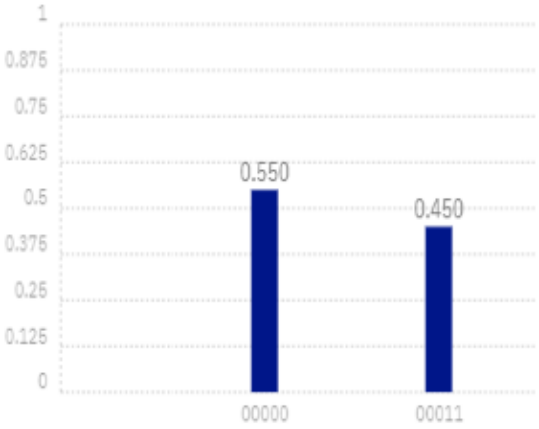
Un algoritmo probabilístico como el anterior se basa en realizar suficientes ejecuciones de algoritmo como para tener una probabilidad aceptable de que la conclusión que hemos sacado de los resultados sea la correcta. Por eso mismo son aceptables las pequeñas imprecisiones.

Por otra parte, el circuito $H - Z - H$ de la figura 2.10a, ya visto en en la sección 1.2, es claramente determinista, y de hecho, simulado sobre un computador cuántico ideal, el resultado es totalmente determinista tal y como se observa en el gráfico de la figura 2.10b.

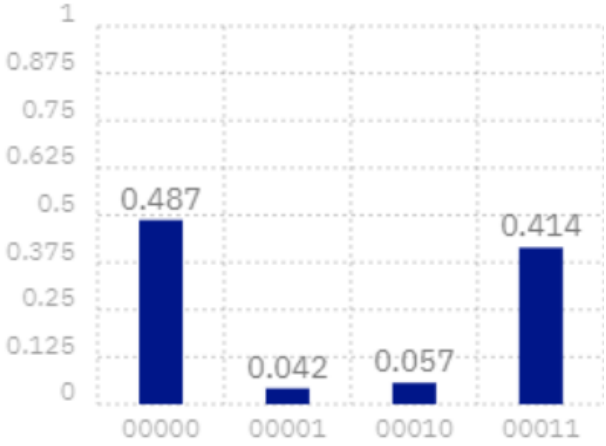
Sin embargo, si ejecutamos un circuito tan sencillo sobre un computador cuántico real,



(a) Implementación en Qiskit



(b) Resultados en el caso ideal



(c) Resultados en el caso real

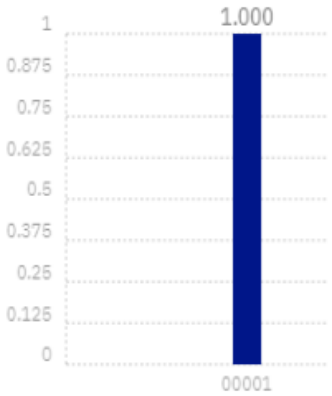
Figura 2.9: Circuito $H - CNOT$ y resultados

los resultados observados son los de la figura 2.10c.

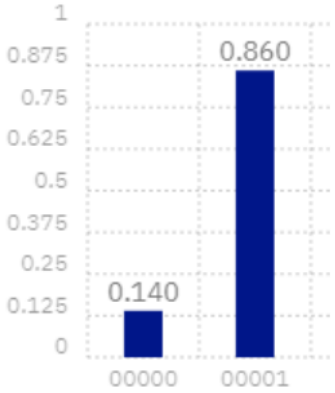
Esto pone de manifiesto que a pesar de realizar un experimento determinista, debido a las imprecisiones del sistema, es necesario realizarlo varias veces para poder extraer conclusiones fiables sobre el resultado. Esto no significa que el algoritmo en sí sea probabilista y es importante evitar esta confusión.

Se espera que con el avance de la tecnología disminuyan las imprecisiones de los sistemas de computación y con ellas el número de ejecuciones necesarias para obtener resultados

(a) Implementación en Qiskit



(b) Resultados en el caso ideal



(c) Resultados en el caso real

Figura 2.10: Circuito $H - Z - H$ y resultados

representativos en algoritmos cuánticos deterministas, en el caso ideal, con una única ejecución sería suficiente; sin embargo, el número de ejecuciones de un algoritmo probabilista seguirá requiriendo un número alto de repeticiones.

2.5. Eficiencia y complejidad

Al principio de este texto se mencionó que algunos algoritmos cuánticos habían conseguido reducir el coste de varios algoritmos clásicos, pero, ¿a qué nos referimos en realidad con este coste? Por un lado hay que encontrar una forma de comparar algoritmos clásicos con cuánticos, para lo que normalmente se puede recurrir a la profundidad del circuito, es

decir, al camino crítico de ambos circuitos. Por otro lado, hay que encontrar una forma de comparar algoritmos cuánticos entre si. En este sentido, debido a la inmadurez del campo, aún no hay una metodología consolidada para ello, por eso, lo más importante es utilizar una métrica consistente y objetiva en nuestras comparaciones que no desprecie puertas y que permita extraer conclusiones. Para ello tomamos como ejemplo las métricas vistas en [Cuc04, Dra00] para confeccionar la que empleamos en este estudio. Hablar del coste de un algoritmo cuántico es hablar de cuatro parámetros.

En primer lugar el número de qubits necesarios o anchura del circuito. En muchos algoritmos son necesarios qubits adicionales para la ejecución del algoritmo. Esto es que para operar sobre una entrada de n qubits, es posible que se requieran $n + m$ qubits. Reducir el número de qubits necesarios para resolver un problema claramente aumenta la eficiencia del algoritmo.

Por otro lado está el número de puerta cuánticas necesarias. Cuanto menor sea el número de puertas en función del tamaño de la entrada mejor. Este parámetro sin embargo presenta una problemática adicional y es la forma en que se cuentan las puertas. En algunos textos se cuentan puertas de hasta tres qubits, en otros solo se pueden utilizar puertas de uno y dos qubits. Una de las medidas más estandarizadas es la de contabilizar la equivalencia del circuito a analizar solo utilizando puertas de un qubit y puertas $CNOT$.

En nuestro caso, hemos decidido contabilizar las puertas de uno y dos qubits, así como las puerta de Toffoli y Fredkin ($CSWAP$) de tres qubits, tan presentes en las implementaciones; y asumir que el resto de puertas deben ser implementadas a partir de puertas más simples. En la tabla 4.1 se ve una muestra de las puertas que contabilizaremos a este efecto.

Otro aspecto a tener en cuenta es la profundidad del circuito, el camino más largo hasta la medición. Esto es especialmente importante a la hora de lanzar un circuito contra un computador real pues los tiempos de estabilidad actuales son reducidos y cuanto más largo sea el circuito más probable es que el resultado que obtengamos sea producto de una imprecisión, interferencia etc y por tanto mayor será el número de ejecuciones necesarias

para poder observar tendencias estadísticas en el resultado. En el caso peor, si el circuito es demasiado largo, los resultados que observaremos serán totalmente aleatorios e inútiles.

Finalmente, el último parámetro de coste es la precisión de rotación requerida. Esto es de vital importancia a la hora de ejecutar el circuito en una máquina física, a mayor precisión requerida, mayor probabilidad de que una imprecisión invalide un resultado.

Nº de qubits	Contabiliza el número de qubits utilizados.
Nº de puertas	Número de puertas de uno y dos qubits y puertas de Toffoli utilizadas.
Profundidad	Cuenta las puertas del camino más largo hasta la medición.
Precisión de rotación	Se corresponde con la rotación más pequeña que debe hacer el circuito.

Cuadro 2.1: Tabla de parámetros de complejidad

Con el fin de ejemplificar un estudio de complejidad sobre un circuito, estudiaremos los cuatro parámetros mencionados a partir del circuito de la figura 3.2 de la sección 3.1.

Podemos observar que tiene un coste de 6 en puertas y profundidad, de 3 en número de qubits o anchura y de $\frac{\pi}{4}$ en precisión de rotación.

Este tipo de estudios se suelen hacer en función del tamaño de los operandos de entrada. En la sección 3.1 veremos en detalle el coste para entradas de tamaño arbitrario de este circuito.

Capítulo 3

Circuitos cuánticos aritméticos

En este capítulo vamos a presentar una pequeña colección de los circuitos cuánticos más relevantes. Para describirlos mostraremos su estructura interna y su complejidad tal y como se describe en la sección 2.5.

3.1. Quantum Fourier Transform (QFT)

La transformada cuántica de Fourier[Cop94], en adelante QFT, es la equivalente cuántica de la transformada rápida de Fourier [Mic94]. Se trata de una transformación lineal que codifica una entrada en binario (generalmente representando un número) en rotaciones de fase. Esto es, codifica el número como un desplazamiento de fase en la circunferencia ecuador de la esfera de Bloch. La figura 3.1 representa la representación en rotación de fase, siendo φ la fase, del 001 en una entrada de tres qubits.

Esto puede sugerir la idea de que teniendo puertas de precisión suficiente podríamos codificar números enteros de tamaño arbitrario en un único qubit y es cierto, el problema viene en que no podríamos decodificar para obtener un resultado legible, recordemos que los únicos estados observables son $|0\rangle$ y $|1\rangle$.

Por eso, el circuito de QFT en realidad requiere n qubits para codificar un número de tamaño hasta 2^n. Supongamos la entrada en binario de longitud n, entonces la QFT codifica el número completo en el qubit más significativo, los $n-1$ qubits menos significativos en el segundo qubit más significativo, y así hasta el qubit menos significativo que sólo se codifica

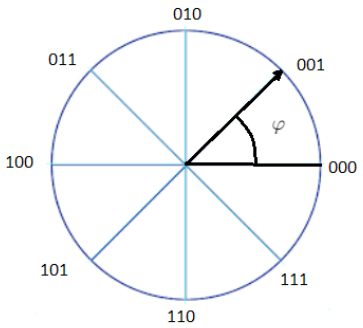


Figura 3.1: Representación del 1 en rotación de fase sobre una circunferencia

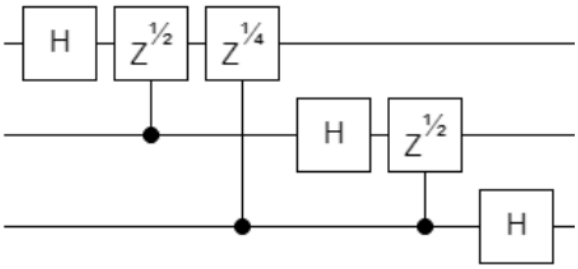


Figura 3.2: Circuito QFT de 3 qubits

a sí mismo.

El circuito de la figura 3.2 realiza la QFT de tres qubits. Supongamos que queremos codificar la entrada $|110\rangle$. Entonces en el qubit superior se codificará en rotación de fase el $|110\rangle$, en el del medio $|10\rangle$ y el inferior $|0\rangle$ tal y como se muestra en la figura 3.3.

Esta representación nos permite operar sobre todo el estado cuántico haciendo rotaciones sobre el eje Z, pero debido a que cada qbuit contiene información parcial sobre el estado cuantico codificado, los cambios se deben aplicar en todos los qubits para no perder información. Todas las operaciones realizadas en una computadora cuántica son unitarias y reversibles, tal y como se explica en el capítulo 1, y una vez realizados los cálculos es necesario aplicar la transformación inversa, QFT^{-1}, para que la información salga del estado de superposición y se convierta en un estado observable (o al menos tener una alta probabilidad de ello).

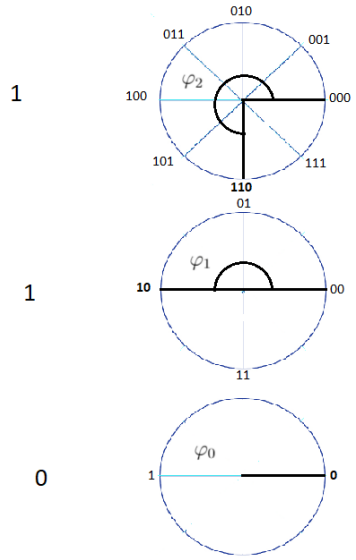


Figura 3.3: Representación del 110 en rotación de fase con 3 qubits

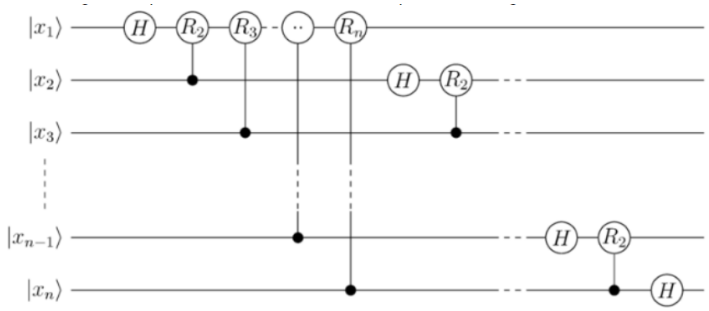


Figura 3.4: Circuito QFT de tamaño arbitrario

El coste de este circuito en una computadora cuántica es, siendo n el tamaño de la entrada, $\frac{n^2+n}{2}$ en número de puertas y profundidad, n en qubits, y $\frac{\pi}{2^{n-1}}$ en precisión de rotación. En la figura 3.4 se muestra un circuito QFT de tamaño n arbitrario.

3.1.1. QFT^{-1}

Quedaría hablar de la transformación inversa, QFT^{-1}, que permite transformar un registro codificado en rotaciones de fase en una entrada binaria clásica.

En genaral, para construir la transformación inversa de cualquier circuito clásico, basta construir un circuito en el que se aplican las puertas inversas en orden inverso. Esto se hace

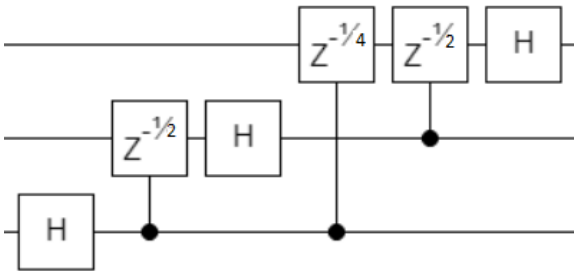


Figura 3.5: Circuito QFT^{-1} de tres qubits

evidente si tenemos en cuenta lo mencionado en la sección 1.1.1 acerca de como interaccionan las matrices de las puertas al concatenarlas.

En este sentido, la QFT no es distinta. En la figura 3.5 se observa el circuito inverso del presentado en la figura 3.2.

3.2. Transformada cuántica de Fourier aproximada ($AQFT$)

En la sección 3.1 veíamos como a medida que aumenta el tamaño de la entrada, realizar la QFT requiere puertas de rotación de precisión el doble. Esto puede darnos la acertada idea de realizar una QFT eliminando las rotaciones más pequeñas obteniendo así un resultado aproximado. De hecho, ante la presencia de determinados niveles de ruido, se ha demostrado que la QFT aproximada, $AQFT$ [BEST96] en adelante, puede ser más precisa que la QFT completa tal y como se menciona en [BEST96].

La siguiente pregunta es hasta qué nivel de puertas es razonable llegar. Esto depende de los niveles de decoherencia y ruido del computador físico, pero los valores óptimos están alrededor de $logn$ puertas de rotación [BEST96]. Esto además reduce el orden de profundidad y puertas del circuito de $\frac{n^2+n}{2}$ a $n+(n-1)logn$ y el de precisión de rotación de $\frac{\pi}{2^{n-1}}$ a $\frac{\pi}{2^{[logn]}}$.

El circuito de la figura 3.6 realiza la $AQFT$ a una entrada de ocho qubits manteniendo un máximo de $3 = logn$ niveles de puertas de rotación.

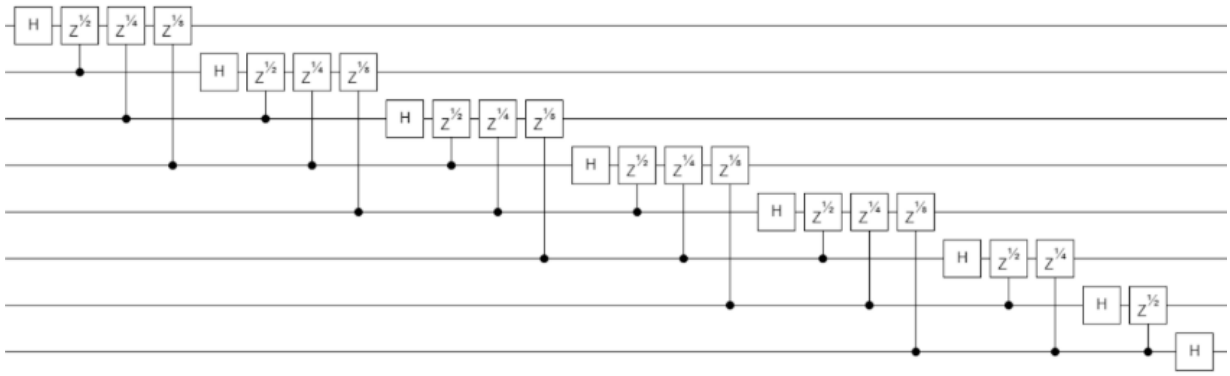


Figura 3.6: $AQFT$ 8 qubits

3.3. Sumadores

En esta sección presentamos algunos de los diferentes sumadores existentes para computadoras cuánticas.

3.3.1. Sumador de Vedral

Este sumador fue presentado por Vedral en [VBE97] y representa la forma clásica de sumar dos números basada en la llevada o carry. Para ello son necesarias dos unidades básicas de cómputo. La puerta carry de la figura 3.7a se encarga de clacular el carry $i-esimo$ a partir de los qubits a_i, b_i y c_{i-1} mediante puertas $CNOT$ y puertas de Toffoli o $CCNOT$. La puerta suma de la figura 3.7b computa la suma $i-esima$ entre a_i y b_i mediante dos puertas $CNOT$. El circuito se compone de dos partes diferenciadas. La primera se encarga de calcular la llevada y la segunda la suma.

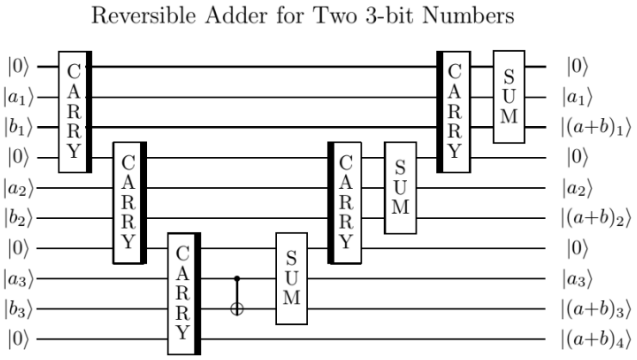
El coste de este circuito es de $8n$ en puertas y profundidad, $3n$ en qubits, n qubits para cada operando y n qubits para el carry, y de π en precisión de rotación.

3.3.2. Sumador de Cuccaro

Este sumador fue presentado por Steven A. Cuccaro en [Cuc04] y se basa en los sumadores clásicos Ripple-Carry y está construído con majority-gates y UMA-gates. La función majority consiste en n entradas y una salida. La salida será $false$ si al menos $\frac{n}{2}$ entradas

(a) Puerta carry

(b) Puerta suma

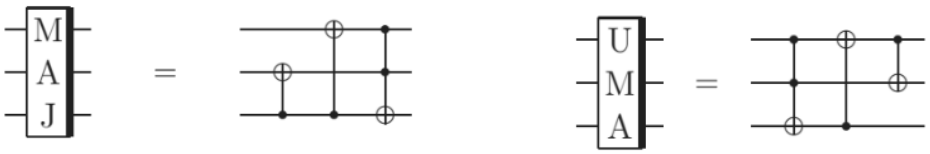


(c) Sumador

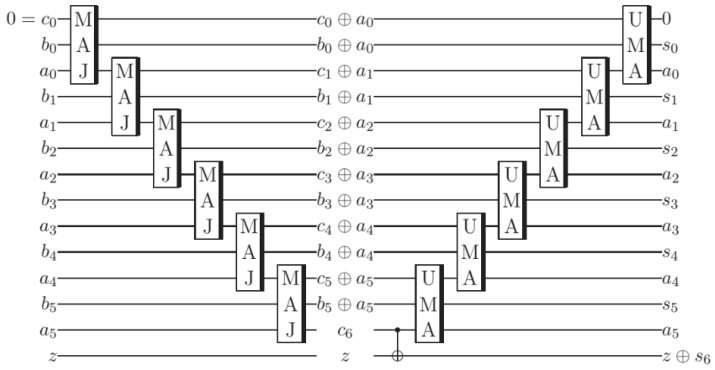
Figura 3.7: Sumador Vedral [VBE97]

tienen el valor $false$ y $true$ en otro caso. Debido a que todas las operaciones en una computadora cuántica tienen que ser biyectivas las majority gates cuánticas (figura 3.8a tendrán tres qubits de salida aunque solo uno de ellos portará la solución. Este sumador requiere $2n$ qubits más dos qubits adicionales para el carry de entrada y el de salida. La idea es calcular el carry con las pueras majority, sin qubits adicionales, y situarlo en el qubit adicional final. Debido a la biyección de todas las operaciones es posible reestablecer los qubits afectados por el cálculo del carry y así sumarlos posteriormente para obtener la suma y el carry finales. Este proceso se lleva a cabo con la puerta UMA de la figura 3.8b, que encapsula la inversa de la puerta Majority y la suma de dos operandos.

El coste de este circuito es de $6n$ en puertas y profundidad, $2n+2$ en qubits, un registro para cada operando y dos qubits adicionales para el carry de entrada y salida, y de π en precisión de rotación.



(a) Majority gate (b) UMA gate



(c) Sumador

Figura 3.8: Sumador de Cuccaro [Cuc04]

3.3.3. Sumador basado en QFT

El siguiente sumador fué introducido por Thomas G. Draper en [Dra00]. Aprovecha la potencia de la representación cuántica de la información para realizar la suma. Es el que queda representado en la figura 3.10a y consta de tres módulos: QFT, SUM, QFT^{-1}. Los módulos QFT y QFT^{-1} se desarrollaron en la sección 3.1 por lo que omitiremos su explicación. Este sumador no requiere qubits adicionales y requiere $2n$ qubits de ancho de palabra para albergar ambos operandos. Además hay que mencionar que es un sumador modular.

El resultado de la suma se almacenará en uno de los registros operando. Para hacer la suma primero es necesario realizar la QFT en uno de los registros, en concreto el que albergará el resultado. Una vez realizada la QFT la suma se realiza mediante rotaciones controladas por el segundo operando.

Cuando tenemos un operando codificado en rotaciones de fase como se explica en la sección 3.1, realizar una suma consiste en realizar una rotación sobre esa fase tal y como se

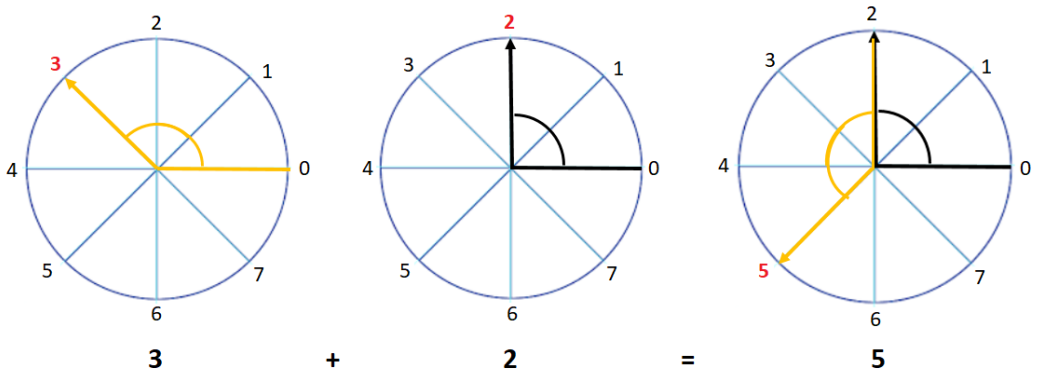
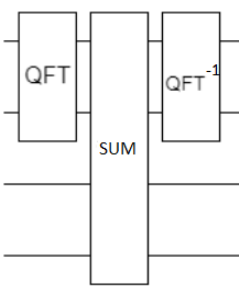
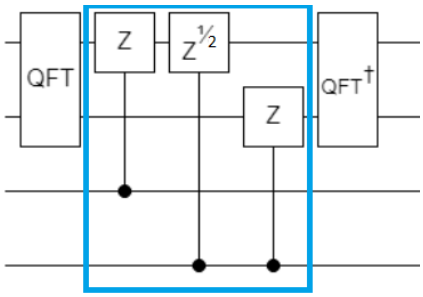


Figura 3.9: Suma de rotaciones de fase



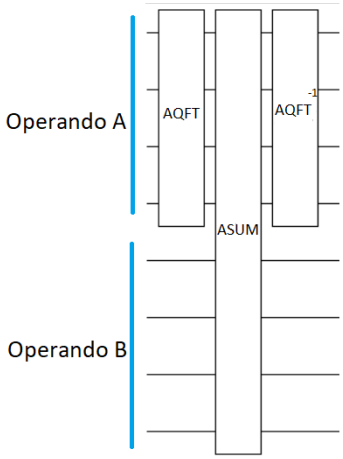
(a) Sumador basado en QFT



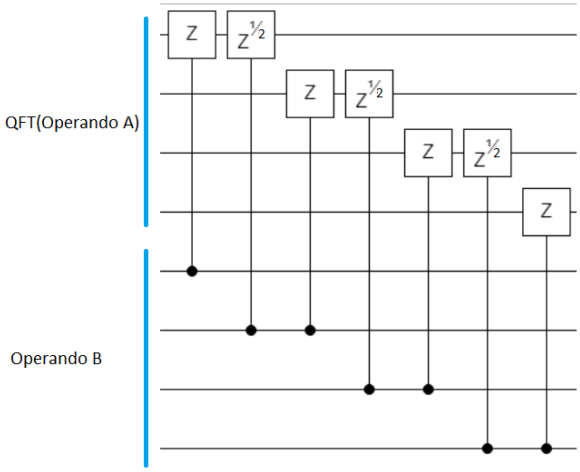
(b) Bloque SUM del sumador QFT

observa en la figura 3.9. Hay que tener en cuenta, que para poder hacer la decodificación correctamente, hay que hacer las sumas parciales en todos los qubits, al igual que ocurría con la QFT.

El valor de la rotación ($\pi, \frac{\pi}{2}, \frac{\pi}{4}...$) queda determinado por el peso de los qubit operando y resultado afectados. Es decir, si el qubit más significativo del registro operando está a uno, este realizará una rotación de π radianes (que representa sumar $2 = 2^1$) en el qubit más significativo del resultado. Por otro lado, si el siguiente qubit menos significatico del operando está a uno (y estamos trabajando con operandos de dos qubits), este realizará una rotación de $\frac{\pi}{2}$ radianes (que representa sumar $1 = 2^0$) en el qubit más significativo del resultado para así tener la codificación parcial. Este proceso se repite hasta codificar en el qubit $i - esimo$ todos los qubits $k - esimos, 0 <= k <= i$. Las rotaciones solo se producen cuando los qubits del operando están a uno es utilizano puertas controladas (CX en este caso). Este módulo queda representado en la figura 3.10b.



(a) Sumador $AQFT$



(b) Bloque $ASUM$

El coste de este circuito $\frac{3(n^2+n)}{2}$ en puertas y profundidad, $2n$ en qubits (un registro para cada operando), y de $\frac{\pi}{2^{n-1}}$ en precisión de rotación.

3.3.4. Sumador basado en $AQFT$

Este tipo de sumador maneja los mismos conceptos que el sumador basado en QFT de la subsección 3.3.3 pero utilizando la transformada cuántica de Fourier aproximada ($AQFT$) mencionada en la sección 3.2. En este sentido, el esquema del sumador es el de la figura 3.11a.

El bloque $ASUM$ que realiza la suma aproximada, debe tener el mismo número de niveles de puertas de rotación que la $AQFT$ aplicada. En la figura se observa el bloque $ASUM$ de un sumador basado en $AQFT$ de 4 qubits y con $logn = 2$ niveles de puertas de rotación.

Teniendo en cuenta solo $logn$ niveles de puertas de rotación, los costes presentados en la sección 3.3.3 se reducirían a $2n + 3(n-1)logn$ en puertas y profundidad y a $\frac{\pi}{2^{[logn]}}$ en precisión de rotación.

3.4. Multiplicador basado en QFT

Este multiplicador está basado en la QFT explicada en la sección 3.1. Su estructura es muy similar a la del sumador QFT de la sección 3.3.3. Su principal diferencia reside

en que uno de los operandos no está en el registro cuántico ya que determina el factor de las rotaciones a la hora de operar. Para multiplicar $a * b$ situamos a por duplicado en dos registros cuánticos los sumarlos $b-1$ veces. Para sumarlos $b-1$ veces empleamos el mismo esquema que en el sumador QFT salvo que el valor de cada rotación se multiplica por $b-1$ simulando $b-1$ sumas consecutivas de a. Por supuesto hay que mencionar que es posible construir un multiplicador basado en la $AQFT$ de la misma manera que lo hemos hecho con la QFT. Los costes de este circuito son los mismos que los del sumador basado en QFT.

Capítulo 4

Comparativa de costes

	$Puertas$	$Profundidad$	$Ancho$	$Precisión$
$Sum.Vedral$ [VBE97]	$8n$	$8n$	$3n$	π
$Sum.Cuccaro$ [Cuc04]	$6n$	$6n$	$2n+2$	π
$Sum.QFT$ [Dra00]	$\frac{3(n^2+n)}{2}$	$\frac{3(n^2+n)}{2}$	$2n$	$\pm\frac{\pi}{2^{n-1}}$
$Sum.AQFT$	$2n+3(n-1)logn$	$2n+3(n-1)logn$	$2n$	$\pm\frac{\pi}{2^{[logn]}}$
$Mult.QFT$	$\frac{3(n^2+n)}{2}$	$\frac{3(n^2+n)}{2}$	$2n$	$\pm\frac{\pi}{2^{n-1}}$
$Enc/DecLog.n=2^m$	$(*_2)$	$(*_1)$	$n+logn-1$	$\pm\frac{\pi}{4}$ $(*_3)$
$Mult.Log.$	$(*_1)$	$(*_1)$	$(*_1)$	$(*_1)$

Cuadro 4.1: Tabla de complejidad de los circuitos propuestos

$(*_1)$ La complejidad del multiplicador logarítmico requiere una mención especial ya que no siempre es la misma. Como vimos en el esquema 5.9 este multiplicador acepta cualquier módulo sumador mientras que los módulos de codificación y decodificación son siempre los mismos. Esta característica implica que los módulos de codificación y decodificación son los que determinan la cota inferior de costes y el sumador empleado la cota superior.

$(*_2)$ El número de puerta viene determinado por el módulo LOD y el módulo shifter. Sus costes respectivos son $nlogn$ para el shifter y $(4n-4)(log^2n-logn+3)+2logn+(n-2)(logn+1)+1$. La suma de todos estos costes es el número de puertas necesarias para construir este circuito.

$(*_3)$ Este coste se debe a que las puertas de Toffoli de dimensión n se construyen a partir de puertas CX, H, T y $T^{-}1$. Las puertas T y $T^{-}1$ son caso particulares de la puerta $U1$ con

una rotación de $\frac{\pi}{4}$ y $\frac{-\pi}{4}$ respectivamente las cuales marcan la cota superior en la complejidad de precisión.

Capítulo 5

Implementación y análisis de resultados

En este capítulo se muestran y discuten los resultados obtenidos tras ejecutar los circuitos planteados en las secciones anteriores (*QFT* 3.1, sumadores 3.3 y multiplicador logarítmico [dlT]) en un simulador ideal, en un simulador con modelos de ruido y en una máquina real mediante el framework Qiskit 2.3 y la API de IBMQ. Para no redundar en el resto de secciones indicamos en el listing 5.1 el esquema general que siguen todos los circuitos presentados. Los registros auxiliares o clásicos no tienen por qué tener ancho de palabra *nq*, puede ser el que se necesite en cada caso.

Listing 5.1: Esquema general de un circuito

```
# Length of the circuit
nq = k

# Quantum register creation
    # Operand registers
    qrA = QuantumRegister(nq, 'qA')
    qrB = QuantumRegister(nq, 'qB')
    ...
    qrZ = QuantumRegister(nq, 'qZ')

    # Ancilla registers
    qrAux1 = QuantumRegister(nq, 'qAux1')
    ...
    qrAuxN = QuantumRegister(nq, 'qAuxN')

# Classical registers
crA = ClassicalRegister(nq, 'cA')
...
crZ = ClassicalRegister(nq, 'cZ')
```

```
# Add the registers tod the circuit instance
circuit.add_register(qrA)
...
circuit.add_register(qrZ)

# Assing initial values to the operands and ancillas
    qubitInit(op1, nq, qrA, circuit)
    ...
    qubitInit(opN, nq, qrZ, circuit)
    qubitInit(aux1, nq, qrAux1, circuit)
    ...
    qubitInit(auxN, nq, qrAuxN, circuit)

# oncatenate the operarion
someOperation.add(circuit, qrA, ... , qrZ, nq)

# Measure the qubits that contains the final result
for i in 0 to nq:
    measure.add(qrK, crK)
```

5.1. Sumadores

En esta sección presentamos una comparativa de los diferentes sumadores presentados en la sección 3.3. Los sumadores son el de Vedral 3.3.1, Cuccaro 3.3.2, Drapper basado en *QFT* 3.3.3 y el basado en *AQFT* 3.3.4. Además proporcionamos el pseucocódigo necesario para la implementación de los circuitos. En todas las sumas intentaremos sumar $1 + 1$.

5.1.1. Pseudocódigo para la creación de los sumadores

Drapper QFT 3.3.3

Listing 5.2: Esquema sumador QFT

```
qrA: operand 1
qrB: operand 2
nq: register length, nq >= 2
circuit: circuit instance
# Note that an ancilla qubit is needed for carry so nq+1 length must be pased to
# the QFT, QFTT and ADD builders in order to include it in the computation.
# This must be considered at registers creation time too.

# Adder general scheme
AdderCircuit(circuit, qrA, qrB, nq){
    # QFT
    doQFT(circuit, qrA, nq+1)
```

```
    # Add
    doAdd(circuit, qrA, qrB, nq+1)
    # QFT-1
    doQFTT(circuit, qrA, nq+1)
}

doQFT(circuit, qrA, nq){
    for c in nq-1 down to 0:
        circuit.h(qr[c])
        e = 1
        for d in c-1 down to 0:
            circuit.crz(pi/(2**e), qr[d], qr[c])
            e+=1
}

doQFTT(circuit, qr, nq){
    for c in 0 to nq:
        circuit.h(qr[c])
        e = 1
        for d in c+1 to nq:
            circuit.crz(-pi/(2**e), qr[c], qr[d])
            e+=1
}

doAdd(circuit, qr1, qr2, nq){
    for c in 0 to nq:
        e = 0
        for d in c to nq:
            circuit.crz(pi/(2**e), qr2[c], qr1[d])
            e+=1
}
```

AQFT 3.3.4

Listing 5.3: Esquema sumador AQFT

```
qrA: operand 1
qrB: operand 2
nq: register length, nq >= 2
pr: AQFT precision
circuit: circuit instance
# Note that an ancilla qubit is needed for carry so nq+1 length must be pased
# to the QFT, QFTT and ADD builders in order to include it in the computation.
# This must be considered at registers creation time too.
# Adder general scheme
AdderCircuit(circuit, qrA, qrB, nq, pr){
    # AQFT
    doAQFT(circuit, qrA, nq+1)
    # Add
```

```
    doAdd(circuit, qrA, qrB, nq+1)
    # AQFT-1
    doAQFTT(circuit, qrA, nq+1)
}

doAQFT(circuit, qrA, nq, pr){
    for c in nq-1 down to 0:
        circuit.h(qr[c])
        e = 1
        for d in c-1 down to 0:
            if e > pr:
                break
            circuit.crz(pi/(2**e), qr[d], qr[c])
            e+=1
}

doAQFTT(circuit, qr, nq, pr){
    for c in 0 to nq:
        circuit.h(qr[c])
        e = 1
        for d in c+1 to nq:
            if e > pr:
                break
            circuit.crz(-pi/(2**e), qr[c], qr[d])
            e+=1
}

doAdd(circuit, qr1, qr2, nq, pr){
    for c in 0 to nq:
        e = 0
        for d in c to nq:
            if e > pr:
                break
            circuit.crz(pi/(2**e), qr2[c], qr1[d])
            e+=1
}
```

Vedral 3.3.1

Listing 5.4: Esquema sumador Vedral

```
qrA: operand 1
qrB: operand 2
qrZ: carry register
nq: register length, nq >= 4
circuit: circuit instance

# General scheme of Vedral adder
AdderCircuit(circuit, qrA, qrB, qrZ, nq){
    # Compute carry
```

```
    for i in 0 to nq:
        doCarry(circuit, qrZ[i], qrA[i], qrB[i], qrZ[i+1])

    # CX
    circuit.cx(qrA[nq-1], qrB[nq-1])
    # Compute add
        # Most significant qbit
        doAdd(circuit, qrZ[nq-1], qrA[nq-1], qrB[nq-1])

        # Rest of qubits
        for i in nq-2 down to 0:
            doCarry(circuit, qrZ[i], qrA[i], qrB[i], qrZ[i+1])
            doAdd(circuit, qrZ[i], qrA[i], qrB[i])
}

# Partial calculation of the ith carry
doCarry(circuit, qr1, qr2, qr3, qr4){
    circuit.ccx(qr2, qr3, qr4)
    circuit.cx(qr2, qr3)
    circuit.ccx(qr1, qr3, qr4)
}

# Partial calculation of the ith add
doAdd(circuit, qr1, qr2, qr3) {
    circuit.cx(qr2, qr3)
    circuit.cx(qr1, qr3)
}
```

Cuccaro 3.3.2

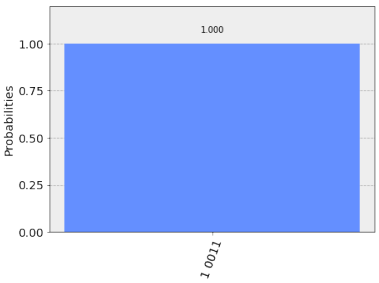
Listing 5.5: Esquema sumador Cuccaro

```
qrA: operand 1
qrB: operand 2
qrZ: carry register
nq: register length, nq >= 2
circuit: circuit instance

AdderCircuit(circuit, qrA, qrB, qrZ, nq){

    for i in range 1 to nq:
        circuit.cx(qrA[i], qrB[i])

    circuit.cx(qrA[1], qrZ[0])
    circuit.ccx(qrA[0], qrB[0], qrZ[0])
    circuit.cx(qrA[2], qrA[1])
    circuit.ccx(qrZ[0], qrB[1], qrA[1])
    circuit.cx(qrA[3], qrA[2])

    for i in 2 to nq-2:
```

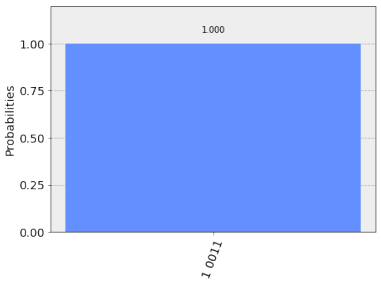
```
        circuit.ccx(qrA[i-1], qrB[i], qrA[i])
        circuit.cx(qrA[i+2], qrA[i+1])

    circuit.ccx(qrA[nq-3], qrB[nq-2], qrA[nq-2])
    circuit.cx(qrA[nq-1], qrZ[1])
    circuit.ccx(qrA[nq-2], qrB[nq-1], qrZ[1])

    for i in 1 to nq-1:
        circuit.x(qrB[i])

    circuit.cx(qrZ[0], qrB[1])

    for i in 2 to nq:
        circuit.cx(qrA[i-1], qrB[i])

    circuit.ccx(qrA[nq-3], qrB[nq-2], qrA[nq-2])

    for i in nq-3 down to 1:
        circuit.ccx(qrA[i-1], qrB[i], qrA[i])
        circuit.cx(qrA[i+2], qrA[i+1])
        circuit.x(qrB[i+1])

    circuit.ccx(qrZ[0], qrB[1], qrA[1])
    circuit.cx(qrA[3], qrA[2])
    circuit.x(qrB[2])

    circuit.ccx(qrA[0], qrB[0], qrZ[0])
    circuit.cx(qrA[2], qrA[1])
    circuit.x(qrB[1])
    circuit.cx(qrA[1], qrZ[0])

    for i in 0 to nq:
        circuit.cx(qrA[i], qrB[i])
}
```

5.1.2. Simulación ideal

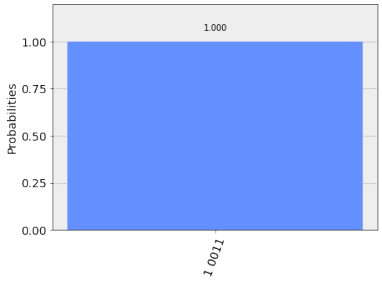
En este caso hemos simulado los sumadores con un ancho de palabra de cuatro qubits más uno del acarreo salvo el basado en AQFT por el cual hemos optado por incrementar el número de qubits a ocho qubits para apreciar mejor las propiedades de la AQFT. Recordemos que en las gráficas el bit más significativo es el que se encuentra abajo. Como podemos observar tras ejecutar los circuitos en el simulador en condiciones ideales los resultados de la figura 5.1 son completamente deterministas, es decir siempre devuelven el mismo estado observable en cada ejecución salvo el sumador basado en AQFT que tiene naturaleza indeterminista y el



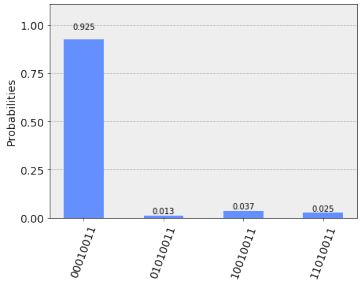
(a) Vedral 3.3.1



(b) Steven A. Cuccaro 3.3.2



(c) *QFT* 3.3.3



(d) *AQFT* 3.3.4

Figura 5.1: Sumadores en simulador ideal (7 + 12)

resultado contiene un pequeño factor de error al perder información debido a las propiedades de la AQFT 3.2. A cambio de perder un poco de información hemos podido ejecutar un circuito con menos complejidad en número de puertas y una precisión menor en las rotaciones lo cual nos da ventajas para casos en los que el ancho de palabra sea arbitrariamente grande y no sea necesaria una precisión exacta en los resultados. En definitiva se reduce la complejidad del circuito a cambio de un pequeño factor de imprecisión.

5.1.3. Simulación con modelos de ruido

Para este experimento hemos modificado los parámetros de ancho de palabra de los circuitos debido a las restricciones que impone el modelo real que hemos aplicado un modelo de ruido de una máquina real al simulador de manera que los resultados obtenidos son totalmente dispares en los diferentes circuitos y completamente distintos a los resultados de la simulación en condiciones ideales.

1) En el sumador de Vedral hemos optado por un ancho de palabra de 2 qubits más el

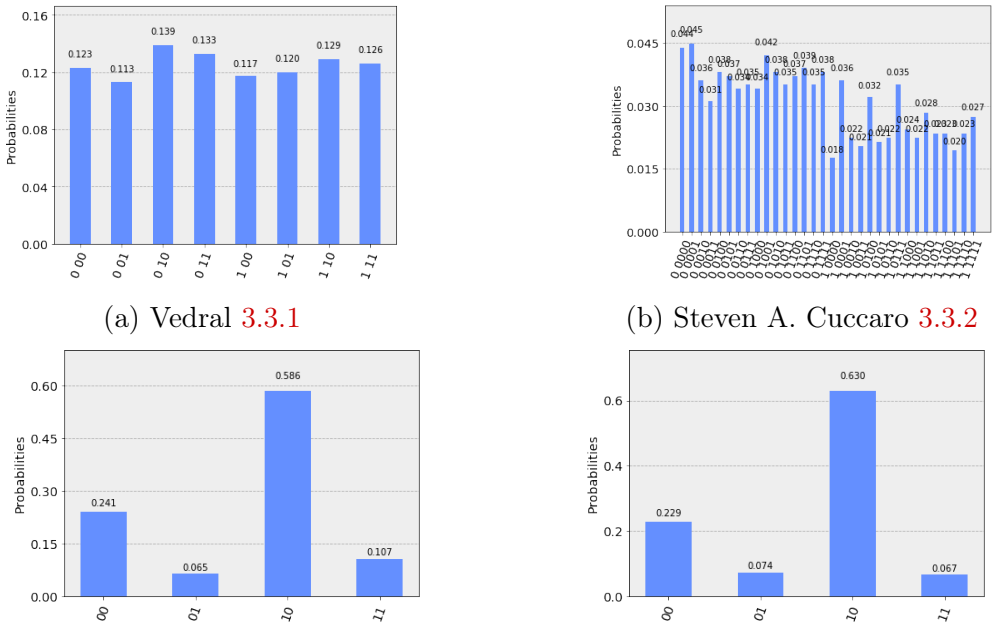
acarreo por lo tanto hay un total de $3 * 2 = 6$ qubits de ancho de palabra así que hemos empleado el backend *'ibmq_16_melbourne'* de 14 qubits. Podemos ver en la figura 5.2a como los resultados no muestran una moda clara que pueda indicarnos la solución de la operación. Aunque el circuito solo necesita 6 qubits los 8 restantes influyen en el resultado a pesar de no ser utilizados ya que están presentes en el backend induciendo ruido al resto de qubits.

2) En el caso del sumador Steven A. Cucaro hemos optado por un ancho de palabra de 4 qubits más el acarreo por lo que hacen un total de $2 * 4 + 2 = 10$ qubits. De nuevo hemos empleado el simulador *'ibmq_16_melbourne'* de 14 qubits. Y una vez más el resultado obtenido en la figura 5.2b presenta una distribución uniforme en la que no destaca ningún estado observable sobre el resto. Los qubits sobrantes y la alta complejidad del ancho de palabra del circuito vuelven a distorsionar por completo el resultado.

3)Para los sumadores QFT 5.2c y $AQFT$ 5.2d hemos optado por un ancho de palabra de 2 qubits debido a las restricciones que muestra el backend *'ibmq_16_melbourne'* de 14 qubits en su grafo de distribución de los qubits que podemos ver en el centro de la figura 2.4. Por lo tanto hemos empleado el backend *'ibmqx4'* de 5 qubits. En estos dos casos destacan las modas de los resultados correctos debido a que el número de qubits empleados en total (incluídos los dummy) es inferior. Si adaptáramos estos circuitos al backend *'ibmq_16_melbourne'* de 14 qubits los resultados serían igual de difusos que en los sumadores de Vedral y Cuccaro y más teniendo en cuenta que su complejidad en número de puertas, precisión y profundidad es mayor tal y como indicamos en el capítulo 4 de comparativa de costes.

5.1.4. Ejecución real

Para la ejecución en la máquina real tenemos las mismas restricciones que en la simulación con modelo de ruido por lo tanto hemos empleado los mismos circuitos que en la sección anterior 5.1.3.



(a) Vedral 3.3.1

(b) Steven A. Cuccaro 3.3.2

(c) *QFT* 3.3.3

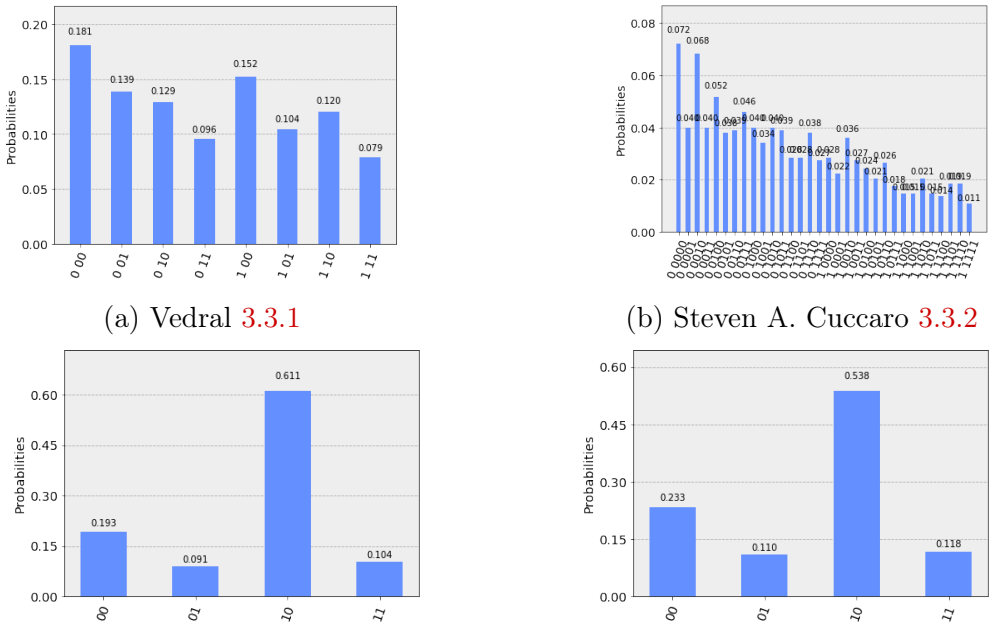
(d) *AQFT* 3.3.4

Figura 5.2: Sumadores en simulador con modelos de ruido (1 + 1)

1) El sumador de Vedral presenta una distribución homogénea sin una moda que indique el resultado de la suma debido de nuevo a los 8 dummy qubits y a su ancho de palabra de 6 qubits.

2) En el sumador de Cuccaro observamos de nuevo una distribución de los resultados totalmente difusa sin una moda que destaque sobre el resto. Los motivos de este resultado son los mismos que en la simulación con modelo de ruido: los qubits dummy y una complejidad de 6 qubits en el ancho de palabra.

3) Los sumadores QFT en cambio si presentan una moda clara con el resultado de la suma debido a que se han ejecutado en una computadora de solo 5 qubits la cual tiene un factor de error mucho más reducido que la de 14 qubits. Los resultados también son fieles a los obtenido en la simulación con modelos de ruido.



(a) Vedral 3.3.1

(b) Steven A. Cuccaro 3.3.2

(c) *QFT* 3.3.3

(d) *AQFT* 3.3.4

Figura 5.3: Sumadores en computadora real (1 + 1)

5.2. Multiplicador logarítmico

En estas sección presentamos los resultados de las ejecuciones en distintas condiciones del codificador logarítmico, el decodificador logarítmico y del multiplicador logarítmico planteados en el TFG [dlT]. En resumen la idea explotada es adaptar el algoritmo de Mitchell [Mit62, KDHB18, KBO+19] a las puertas cuánticas para reducir la multiplicación a una suma y obtener un resultado aproximado. La propiedad matemática que explotaremos para ello es $log(A*B) = log(A)+log(B)$. Para ello se codifica un número en su forma logarítmica en base a la ecuación 5.2 donde llamaremos a k característica de A y a x mantisa de A. A modo de ejemplo la codificación quedaría tal y como se muestra en la ecuación 5.1

$$A = 00010110 = 22,$$

$$B = 00000111 = 7.$$

$$COD(A) = 100|0110000,$$

$$COD(B) = 010|1100000.$$

$$COD(A) + COD(B) = 111|0010000. \tag{5.1}$$

$$\begin{aligned} A * B &= 154 \\ &= 10011010 \\ &= DEC(COD(A * B)) \\ &\approx DEC(COD(A) + COD(B)) \\ &= 10010000 \\ &= 144. \end{aligned}$$

$$A = 2^k(1 + x) \quad con \quad k \geq 0 \quad y \quad 0 \leq x < 1. \tag{5.2}$$

5.2.1. Pseudocódigo para la creación del codificador y el decodificador logarítmicos

Para crear el decodificador basta con aprovechar la propiedad de reversibilidad e invertir el circuito del codificador. Además las inversas de las puertas empledas son ellas mismas.

Listing 5.6: Esquema codificador logarítmico

```
encode4Qbits(circuit, qr){
    # LOD
    circuit.cx(qr[3], qr[4])
    circuit.x(qr[3])
    circuit.cswap(qr[3], qr[2], qr[4])
    circuit.x(qr[3])
    circuit.x(qr[4])
    circuit.cswap(qr[4], qr[1], qr[3])
```

```
    circuit.x(qr[3])
    circuit.ccx(qr[4], qr[3], qr[0])

    # Shifter
    circuit.cswap(qr[3], qr[2], qr[1])
    circuit.cswap(qr[3], qr[1], qr[0])
    circuit.cswap(qr[4], qr[2], qr[0])
    circuit.cswap(qr[4], qr[1], qr[0])

    # Restore characteristic
    circuit.x(qr[4])
    circuit.x(qr[3])
}
```

Listing 5.7: Esquema multiplicador logarítmico

```
logMultiplier2Qbits(mult1, mult2, circuit, qrA, qrB) {
    # Encode
    encode4Qbits(circuit, qrA)
    encode4Qbits(circuit, qrB)

    # Adder
    someAdder.get(circuit, qrA, qrB)

    # Decode
    decode4Qbits(circuit, qrA)
}
```

5.2.2. Codificador/decodificador logarítmico

En este apartado presentamos los resultados de los experimentos con el codificador logarítmico.

Este diseño se compone de dos módulos principales:

1) El LOD (Leading One Detector). Se encarga de detectar el qubit a 1 más significativo, obtener su posición y escribirlo en la zona de la caracaterística 5.4.

2) Shifter. Su tarea consiste en desplazar la mantisa hacia la izquierda controlado por la característica 5.5.

Para cada supuesto hemos empleado como entrada del codificador el 5 (00101) y el 8 (01000), y como entrada del decodificador el 18 (10010, que nos lelvará de vuelta a 00101) y el 24 (11000, que nos lelvará de vuelta a 01000). Nota: el bit más significativo está a la izquierda en el texto y abajo en las gráficas. El ancho de palabra empleado es de 4 qubits

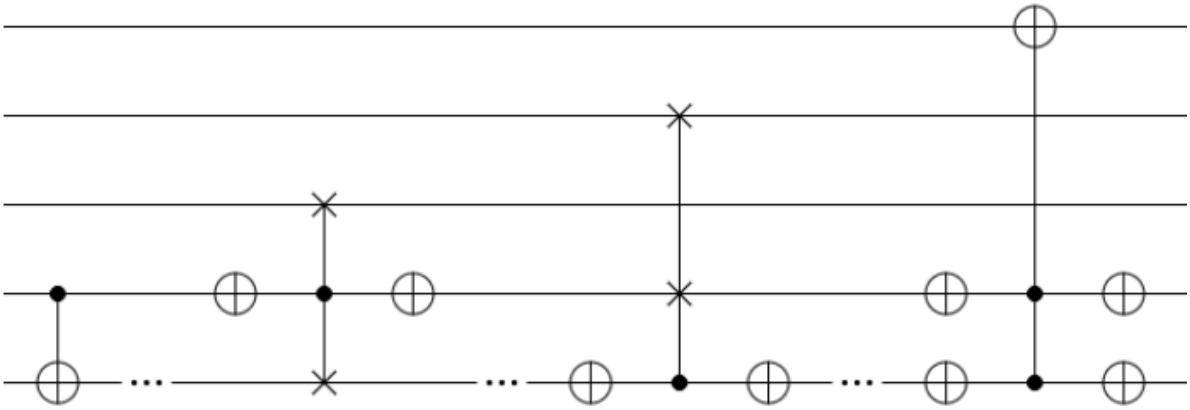


Figura 5.4: *LOD* de 4 qubits (Leading One Detector)

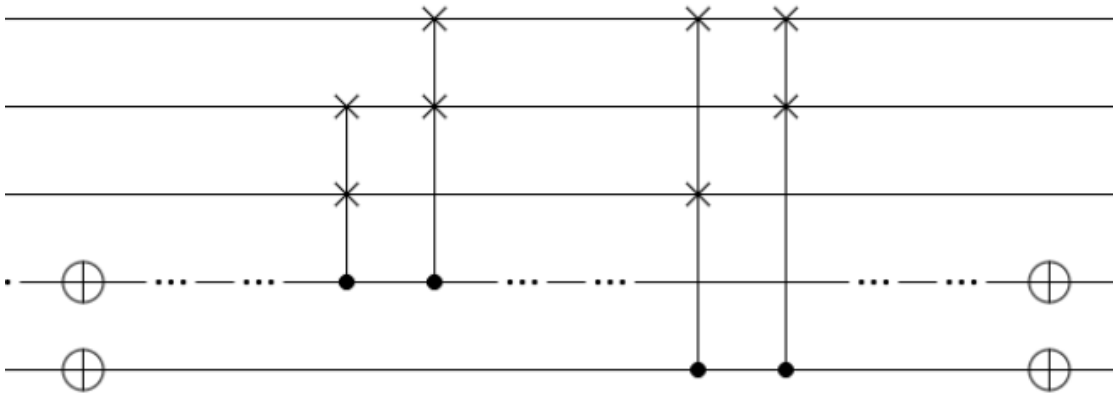
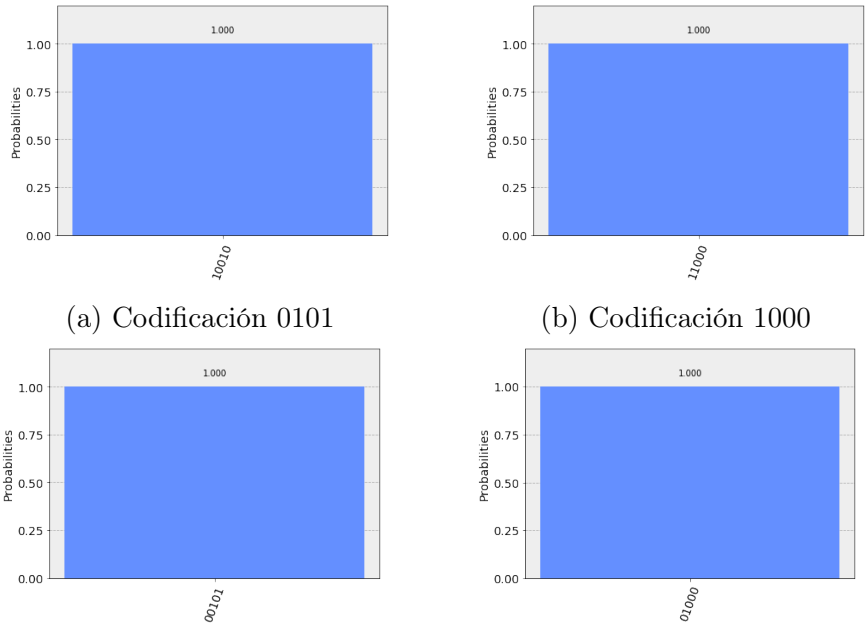


Figura 5.5: Modulo shifter de 4 qubits

por lo que para codificar cualquier valor serán necesarios un total de 5 qubits tal y como se indica en el diseño [dlT], 2 qubits para la característica y 3 para la mantisa.

Simulación ideal

En la simulación en condiciones ideales podemos observar como efectivamente el circuito del codificador es determinista. Con la entrada 00101 en la figura 5.6a observamos como los qubits de la característica están a $'10'$ ya que el qubit más significativo a 1 de 00101 está en la posición 2. La mantisa tiene el valor 010 al desplazar habia la izquierda una posición 001 por lo tanto queda como resultado de la codificación $'10|010'$. Lo mismo sucede con la entrada



(a) Codificación 0101

(b) Codificación 1000

(c) Decodificación 0101

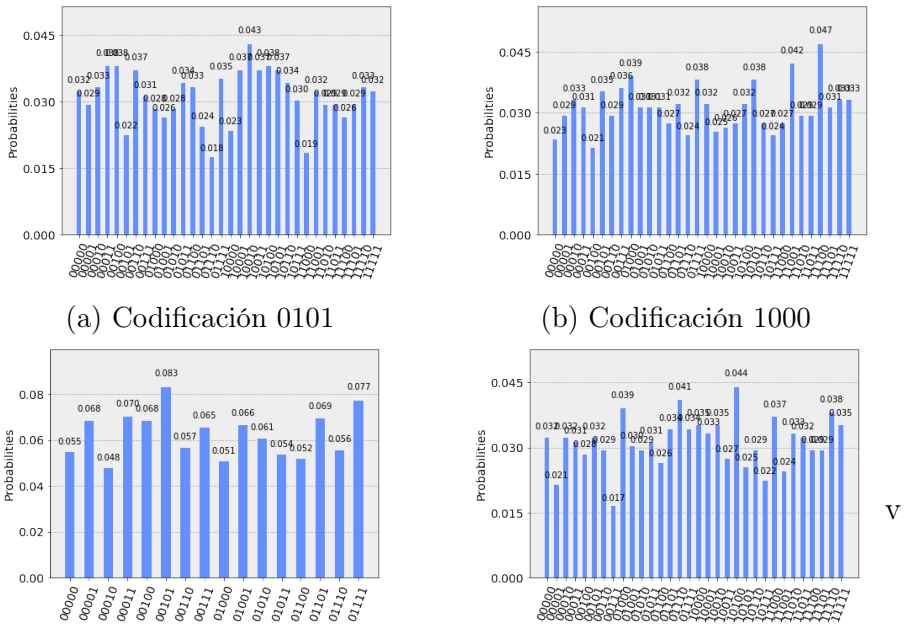
(d) Decodificación 1000

Figura 5.6: Codificaciones y decodificaciones logarítmicas en condiciones ideales

01000 donde el qubit a 1 más significativo está en la posición 3 de manera que los qubits de la característica valen $'11'$, la mantisa es 000 y no es necesario ningún desplazamiento quedando como resultadode la codificación $'11|000'$ tal y como muestra la figura 5.6b. En cuanto al decodificador le hemos pasado como entrada los números codificados y podemos observar en las figuras 5.6c y 5.6d que la decodificación se hace correctamente cumpliéndose la propiedad de reversibilidad de los circuitos en computadoras cuánticas. Si calculáramos la matriz unitaria de la concatenación del codificador y el decodificador obtendríamos la matriz identidad que en este caso es de dimensión $5x5$.

Simulación con modelos de ruido

En este apartado hemos ejecutado el codificador y el decodificador en un simulador con un modelo de ruido en un backend de 5 qubits. En este caso no se aplica ningún tipo de restricción al diseño ya que no hay dummy qubits y el grafo de distribución de los qubits (el situado a la derecha en la figura 2.4) es compatible con el circuito. Hemos obtenido el



(a) Codificación 0101

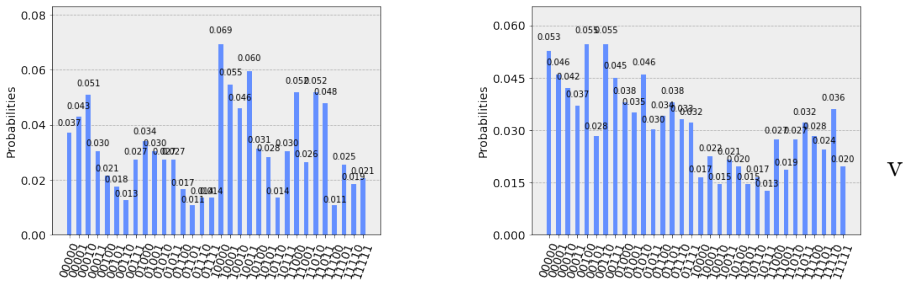
(b) Codificación 1000

(c) Decodificación 0101

(d) Decodificación 1000

Figura 5.7: Codificaciones y decodificaciones logarítmicas en condiciones de ruido

modelo de ruido del backend de 5 qubits *'ibmqx4'*. Como se puede observar en el conjunto de figuras 5.7 los resultados arrojados muestran distribuciones uniformes sin que destaquen los resultados esperados. En el caso de la entrada 00101 su valor codificado 10010 tiene una probabilidad de 0.043 que a pesar de ser la más alta no destaca mucho sobre el resto de valores como por ejemplo 10100 que tiene una probabilidad de 0.038. En el caso del decodificador sitúa la salida en los 4 qubits menos significativos y el quinto nos sobra de manera que lo hemos excluído de la medición para mitigar su impacto. Aun así para la entrada 10010 el valor de salida 00101 apenas tiene una probabilidad de 0.083 para un 0.077 que tiene 01111 ó 0.069 de 01101 de manera que no destaca lo suficiente sobre el resto como para identificarlo como respuesta correcta. Por lo tanto sabemos que los diseños e implementaciones del codificador y el decodificador son correctos pero no funcionan muy bien en condiciones de ruido ya que los resultados sufren una gran distorsión.



(a) Decodificación 0101 (b) Decodificación 1000

Figura 5.8: Codificaciones y decodificaciones logarítmicas en condiciones reales

Ejecución real

Para este experimento no se aplica ningún tipo de restricción al circuito al igual que en el experimento de simulación con modelo de ruido. Hemos empleado el backend real de 5 qubits *'ibmqx4'*. Debido al tiempo de espera necesario para la respuesta del backend real hemos reducido el experimento a solo un dato de entrada, el 5. Como podemos observar en la figura 5.8 los resultados vuelven a ser difusos al igual que en la simulación con el modelo de ruido. Aunque la distribución de los resultados no es tan uniforme resulta imposible identificar correctamente el resultado correcto.

5.2.3. Esquema completo

En esta sección exponemos la implementación y los resultados de la ejecución del diseño de un multiplicador logarítmico planteado en el TFG *"Diseño de Unidades Funcionales Cuánticas: Un Multiplicador Logarítmico"* [dlT] cuyo esquema podemos ver en la figura 5.9.

El circuito se compone de dos codificadores logarítmicos, un sumador basado en QFT y un decodificador logarítmico. Como podemos observar en el esquema solo hemos medido los qubits que contienen el resultado de la multiplicación ignorando el resto ya que no son necesarios. El multiplicador implementado multiplica un número de 2 qubits de ancho de palabra por lo que son necesarios 4 qubits para albergar la solución. El codificador y decodificador empleados son los mismos que en la sección 5.2.2 que codifican y decodifican valores de 4 qubits. Por lo tanto para este multiplicador de 2 qubits serán necesarios un total

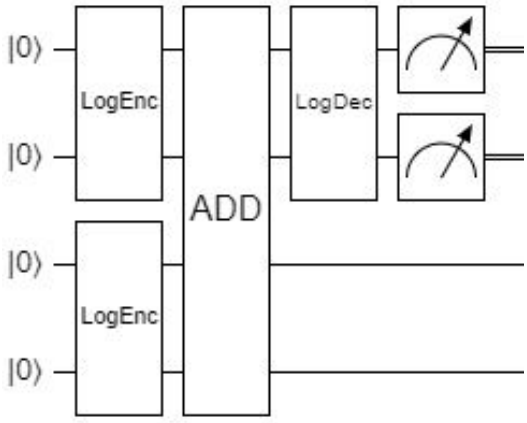
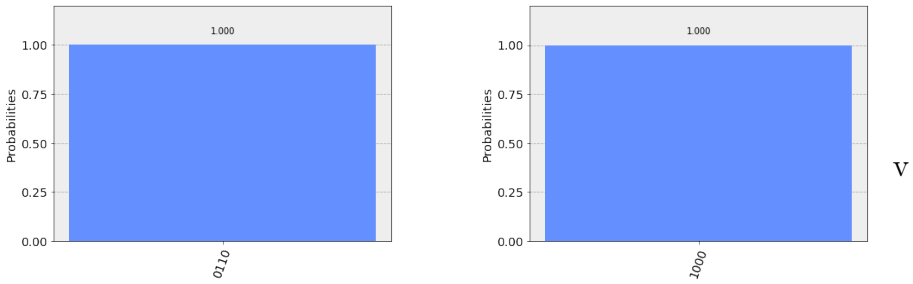


Figura 5.9: Esquema multiplicador logarítmico de 2 qubits

de 10 qubits ya que las codificaciones de cada operando precisan de 5 qubits. El sumador que hemos elegido es el basado en QFT de la sección 3.3.3 ya que no requiere ningún qubit adicional y es completamente in place. Es posible incluir cualquier otro sumador como veremos más adelante. Las operaciones que hemos realizado son $2 * 3 = 6$ y $3 * 3 = 8$. La segunda multiplicación, a pesar de ser errónea es el resultado que arroja el multiplicador logarítmico debido a su factor de error (más detalles en [dlT]). Si hacemos el cálculo manualmente siguiendo las indicaciones de la sección 5.2 observamos que 00011 codificado es 01100. Si sumamos 01100 + 01100 bit a bit el resultado es 11000 que nos indica que el bit más significativo es el de la posición 3 y que su mantisa es 0 por lo tanto nos queda el valor 01000 al decodificar que es 8 en decimal.

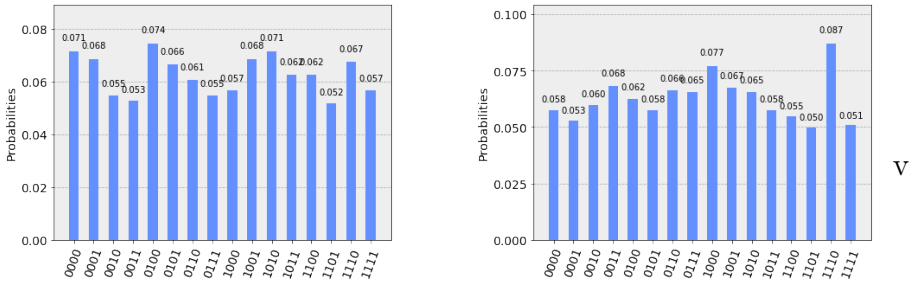
Simulación ideal

Para este experimento hemos empleado el simulador en condiciones ideales. A primera vista podemos observar en las gráficas de la figura 5.10 que el multiplicador logarítmico propuesto es un circuito determinista ya que arrojan el mismo resultado en todas las ejecuciones. En la figura 5.10a observamos como la salida de la multiplicación $2 * 3$ es 0110 ó 6 en decimal de manera que el cálculo es correcto. En el caso de la operación $3 * 3$ 5.10b el resultado es 01000 que es 8 en decimal por lo tanto el circuito ha devuelto el resultado



(a) 2 * 3 = 6 (b) 3 * 3 = 8

Figura 5.10: Multiplicador logarítmico de 2 qubits en condiciones ideales



(a) 2 * 3 = 6 (b) 3 * 3 = 8

Figura 5.11: Multiplicador logarítmico de 2 qubits en condiciones de ruido

esperado tras el cálculo manual realizado previamente.

Simulación con modelos de ruido

En este experimento hemos añadido al simulador el modelo de ruido del backend *'ibmq_16_melbourne'* de 14 qubits ya que necesitamos al menos 10 qubits para ejecutar el circuito. No se aplica ninguna restricción para este circuito aunque habrá 4 qubits dummy que influirán en el resultado final. En la figura 5.13 podemos observar que ambos resultados son muy difusos y no es posible diferenciar un resultado del resto de valores debido a la gran influencia del ruido. Queda patente que la complejidad en profundidad y precisión del circuito afectan notablemente a los resultados obtenidos. Teniendo en cuenta que los resultados individuales del codificador vistos en la sección 5.2.2 ya son muy difusos cabría esperar que los resultados del multiplicador completo con varios módulos similares tamién lo sean.

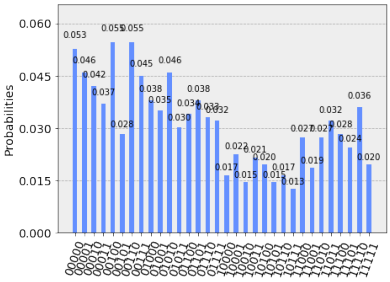


Figura 5.12: Multiplicador logarítmico de 2 qubits en condiciones reales, 2 * 3 = 6

Ejecución real

En este experimento hemos empleado el backend *'ibmq_16_melbourne'* de 14 qubits para lanzar el circuito propuesto. En esta ocasión tampoco hay restricciones de ningún tipo para el circuito salvo los 4 qubits dummy que afectarán al resto de qubits induciendo ruido. Debido a los resultados previstos por la simulación con el modelo de ruido solo hemos ejecutado una de las dos operaciones propuestas. Como podemos ver en la figura 5.12 el resultado es el esperado 5.2.3. Claramente no es posible identificar la respuesta correcta ya que el valor 00110 solo presenta una probabilidad de 0.055 que a pesar de ser de las más altas no es la única.

5.2.4. Multiplicador logarítmico con otros sumadores

En esta sección exponemos los resultados que arrojan varias versiones del multiplicador logarítmico que difieren en el sumador empleado entre los módulos de codificación y decodificación. Excluímos el sumador QFT ya que los resultados para este sumador ya han sido expuestos en la sección 5.2.3. Los sumadores que vamos a emplear son el basado en AQFT 3.10a, el de Vedral 3.3.1 y el de Cuccaro 3.3.2. Solo simularemos los circuitos en condiciones ideales ya que como hemos visto en la sección 5.2.2 el codificador logarítmico por sí solo arroja resultados demasiado difusos por lo que cabe esperar que todos los módulos en su conjunto también arrojen los mismos resultados. Además el objetivo de esta sección es demostrar el funcionamiento conceptual del multiplicador sea cual sea el sumador utilizado.

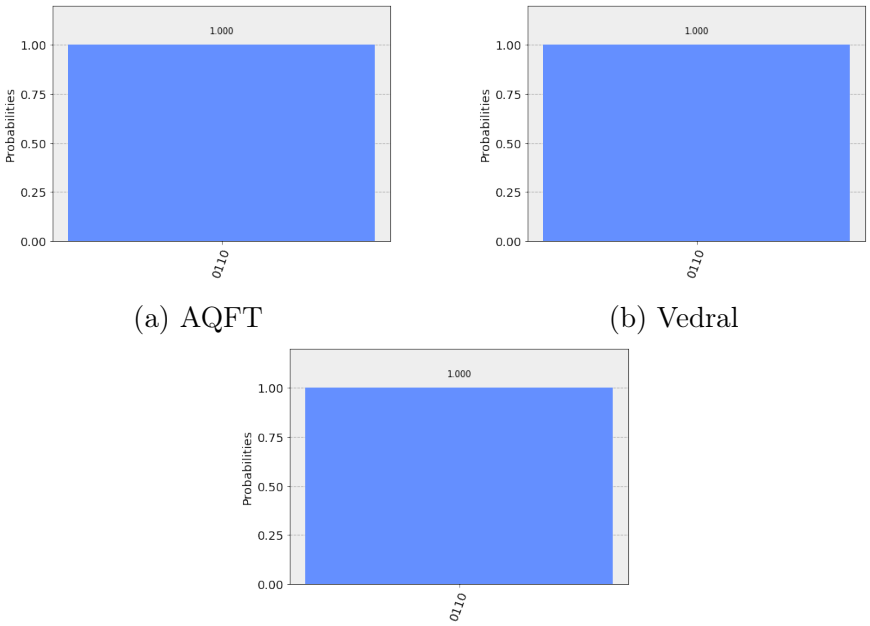
La operación que vamos a analizar es $2 * 3 = 6$ ya que sabemos que este caso devuelve una respuesta correcta al multiplicar.

AQFT

En esta sección hemos empleado el sumador basado en AQFT para probar el multiplicador logarítmico. Al tratarse de un circuito indeterminista el sumador 'contagia' esta propiedad al resto del circuito de manera que el resultado obtenido puede no ser exacto pero en este caso el resultado es exacto tal y como indica la figura 5.13a, 0110. Aun así hay que tener en cuenta la combinación del factor de error del sumador basado en AQFT y el factor de error del codificador/decodificador logarítmico a la hora de aplicar este sumador. Para anchos de palabra mayores los resultados si son variados como vimos por ejemplo en la figura 5.1d que muestra el resultado de una suma con el sumador AQFT por sí solo. No hemos llevado a cabo el experimento con ancho de palabra mayor debido a que el codificador empleado sería 10 qubits de ancho siendo necesarios 20 en total y no hay disponible un backend de 20 qubits para probarlo. Además redundaría en la exposición del concepto de multiplicador logarítmico del cual ya hemos visto un ejemplo de imprecisión al multiplicar $3 * 3 = 8$ en 5.10b.

Vedral y Cuccaro

En este experimento hemos empleado los sumadores de Vedral y Cuccaro para realizar la multiplicación logarítmica. En el caso del sumador de Vedral se necesitan un total de 15 qubits por lo que no disponemos de un backend con los suficientes qubits para ejecutarlos con un modelo de ruido o en real. El backend con más qubits de que disponemos es el *'ibmq_16_melbourne'* que tiene 14 qubits. Como podemos observar en las figuras 5.13b y 5.13c el resultado obtenido al utilizar estos dos sumadores es el correcto habiendo obtenido 0110 que es 6 en decimal. Un apunte a tener en cuenta a la hora de concatenar estos circuitos son los qubits ancilla necesarios para calcular el acarreo. Debido a que realmente estamos multiplicado dos operando de 2 qubits cada uno la suma más grande que puede darse es la



(a) AQFT

(b) Vedral

(c) Cuccaro

Figura 5.13: Multiplicador logarítmico de 2 qubits con diferentes sumadores

que representa la operación $3 * 3$, $01100 + 01100$ cuyo resultado hemos visto en la sección 5.2.3 que es 11000, 8 en decimal de manera que el acarreo para cualquiera de las posibles sumas de 2 qubits siempre va a ser 0 así que los registros que llevan el acarreo simplemente se pueden obviar a la hora de conectar con la entrada del codificador.

Capítulo 6

Conclusiones

Tras este estudio realizado sobre los conceptos básicos de la computación cuántica y varias unidades funcionales la conclusión más general es que aun queda mucho camino que recorrer sobre todo en el desarrollo de un hardware que sea capaz de ejecutar los circuitos planteados con un factor de error que permita hacerlos operativos y se puedan utilizar para construir nuevos sistemas más complejos. Un ejemplo de ello es el multiplicador logarítmico en el cual hemos visto que cada uno de sus módulos por sí solos son inoperativos en una máquina real al arrojar siempre resultados muy difusos que no permiten rescatar la salida correcta. Sin embargo las herramientas vistas como Qiskit permiten que poco a poco la computación cuántica pase del ámbito teórico matemático al práctico acercándose a la informática y permitiendo experimentar de manera muy sencilla con todos estos conceptos. Dada la naturaleza de las computadoras cuánticas las unidades funcionales planteadas carecen de utilidad real al no explotar al completo las cualidades del paradigma y sobre todo al tener ya máquinas clásicas que realizan esas tareas de manera muy eficaz (salvo la QFT). Sin embargo a la hora de plantear una introducción o un transvase desde la computación clasica a la computación cuántica en el ámbito académico las unidades funcionales planteadas son esenciales ya que contamos con sus equivalentes en computación clásica y su comportamiento se conoce previamente. Esto es crucial para comprender correctamente todos los conceptos básicos de la computación cuántica ya que actualmente la barrera de conocimiento es considerable por el alto nivel de conceptos matemáticos necesarios. Por otro

lado el mundo que se abre en el ámbito de la computación será una revolución por la gran capacidad de cálculo que tendrán esatas computadoras. Buena prueba de ello es por ejemplo la posibilidad de encapsular un circuito entero en una sola operación como vimos la posibilidad de calcular la matriz unitaria representativa de un circuito al completo. Por último indicar que los resultados presentados se basan en la investigación publicada en [SdlTB+19] previa a este trabajo.

6.1. Futuro

1) Los siguientes pasos a seguir pueden ser aplicar las unidades funcionales vistas a las nuevas actualizaciones de Qiskit que se vayan publicando en el futuro que permitan aprovechar mejor las cualidades de sus simuladores y computadoras reales.

2) Como hemos visto simular un circuito cuántico puede ser una tarea muy tediosa para una computadora clásica de manera que volcar todos estos procesos a una GPU puede ser muy interesante para así poder simular circuitos más complejos en tiempos razonables.

3) Otro camino a seguir podría ser el de diseñar e implementar las unidades funcionales planteadas como generalización en el espacio de Hilbert complejo en vez de los números clásicos enteros que hemos empleado.

4) Otra continuación es aprovechar que cada qubit se comporta como una partícula elemental de manera que es posible utilizarlos para simular sistemas naturales como átomos o moléculas pudiendo así liberar a las computadoras clásicas de estas tareas tan tediosas que una computadora cuántica puede hacer aprovechando sus característican.

Bibliografía

[AAea19] Gadi Aleksandrowicz, Thomas Alexander, and Panagiotis Barkoutsos et al. Qiskit: An open-source framework for quantum computing, 2019.

[BEST96] A. Barenco, A. Ekert, K. Suominen, and P. Törmä. Approximate quantum fourier transform and decoherence. *Phys. Rev. Letters*, pages 139–146, july 1996.

[C+18] Patrick J. Coles et al. Quantum algorithm implementations for beginners. *CoRR*, abs/1804.03719, 2018.

[Cir] Cirq.

[Cop94] Coppersmith D. An approximate fourier transform useful in quantum factoring. https://arxiv.org/abs/quant-ph/0201067, 1994.

[Cuc04] Cuccaro S.A. and others. A new quantum ripple-carry addition circuit. https://arxiv.org/abs/quant-ph/0410184, 2004.

[dlT] Antonio Valdivia de la Torre. Diseño de unidades funcionales cuánticas: Un multiplicador logarítmico.

[Dra00] Draper T.G. Addition on a quantum computer. https://arxiv.org/abs/quant-ph/0008033, 2000.

[Gee] Geek3. Toffoli gate, own workcreated in latex using q-circuit. https://commons.wikimedia.org/w/index.php?curid=75723375.

[Hir12] Mika Hirvensalo. Mathematics for quantum information processing. In *Handbook of Natural Computing*, pages 1381–1412. 2012.

[HPKM07] Mika Hirvensalo, Raymond Laflamme Phillip Kaye, and Michele Mosca. An introduction to quantum computing, oxford university press (2007) ISBN 019857049x. *Computer Science Review*, 1(1):73–76, 2007.

[IBM19a] IBM. IBM Q Experience. https://quantumexperience.ng.bluemix.net/qx/experience, 2019. [Online; accessed 25-February-2019].

[IBM19b] IBM. IBM Quantum devices & simulators - IBM Q. https://www.research.ibm.com/ibm-q/technology/devices/, 2019. [Online; accessed 25-February-2019].

[KBO+19] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida, and N. Bagherzadeh. Efficient mitchell's approximate log multipliers for convolutional neural networks. *IEEE Transactions on Computers*, 68(5):660–675, May 2019.

[KDHB18] M. S. Kim, A. A. Del Barrio, R. Hermida, and N. Bagherzadeh. Low-power implementation of mitchell's approximate logarithmic multiplication for convolutional neural networks. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 617–622, Jan 2018.

[Mic] Microsoft. Microsoft q#. https://www.microsoft.com/en-us/quantum.

[Mic94] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1 edition, 1994.

[Mit62] J. N. Mitchell. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers*, EC-11(4):512–517, Aug 1962.

[NC11] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 10th edition, 2011.

[SdlTB+19] Daniel David Abellán Serrano, Antonio Valdivia de la Torre, Alberto A. Del Barrio, Guillermo Botella, and Ginés Carrascal. Simulating and executing circuits employing the cuantum computing paradigm. In *Proceedings of the 51th Computer Simulation Conference, SummerSim 2019, Berlin, Germany, July 22-24, 2019*, page (in press), 2019.

[Se] qw3rtman Strilanc and ebraminio. Quirk. https://github.com/Strilanc/Quirk.

[VBE97] V. Vedral, A. Barenco, and A. Ekert. Quantum networks for elementary arithmetic operations. *Phys. Rev. Letters*, pages 147–153, july 1997.