Diseño de Unidades Funcionales Cuánticas: un Multiplicador Logarítmico

Antonio Valdivia de la Torre

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS.
FACULTAD DE INFORMÁTICA. UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de fin de grado del Grado en Doble Grado en Ingeniería Informática - Matemáticas, Universidad Complutense de Madrid

2018/2019

Tutores:

Alberto Antonio del Barrio García
Guillermo Botella Juan

Resumen

Los ordenadores cuánticos, aprovechando el sistema de representación cuántico de la información son capaces de realizar tareas inabordables para los ordenadores clásicos. En este texto se explicarán los conceptos y principios básicos de la computación cuántica, se hará una revisión del diseño de algunos de los circuitos aritméticos cuánticos (sumadores y multiplicadores) que existen en la actualidad; y se abordará el diseño de un nuevo multiplicador aproximado basado en el algoritmo de Mitchell.

Palabras clave

Computación cuántica, multiplicador, sumador, Mitchell.

Abstract

Quantum computers, taking advantage of the quantum-representation-of-information system, are able to perform tasks that can not be absorbed by classic computers. In this text we will explain the basic concepts and principles of quantum computing, we will make a review of the design of some of the quantum arithmetic circuits (adders and multipliers) that exist nowadays; and the design of a new approximate multiplier based on the Mitchell algorithm will be addressed.

Keywords

Quantum computing, multiplier, adder, Mitchell.

Índice general

Agradecimientos

Mi más sincero agradecimiento a Guillermo y Alberto, mis tutores, por haberme dado la oportunidad de participar en este proyecto. Oportunidad que me ha permitido acercarme al mundo de este nuevo paradigma de computación que es la computación cuántica y que claramente será parte importante del futuro de la computación en general.

A Daniel, mi compañero en el proyecto, por el gran trabajo realizado en las simulaciones y el proyecto en general y por haberme acompañado durante todo este viaje.

Por último dar las gracias a mi familia, a la inversión realizada en mi, a la confianza depositada y a todo el apoyo que me han dado durante estos años. Este trabajo ha sido posible gracias a ellos y no tendría sentido de otra forma.

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

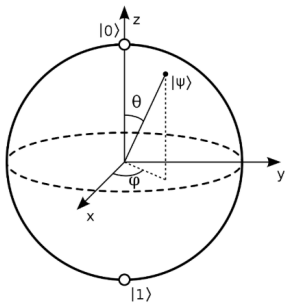


Figura 1.1: Esfera de Bloch

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 1.5, 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 qubit. 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 la fracción $\frac{1}{2^n}$ indica que se gira $\frac{\pi}{2^n}$ radianes 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.

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

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

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 1.1.

1.2. Múltiples qubits

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

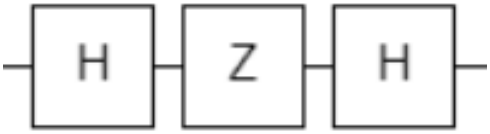


Figura 1.2: Circuito cuántico $H - Z - H$

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.

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

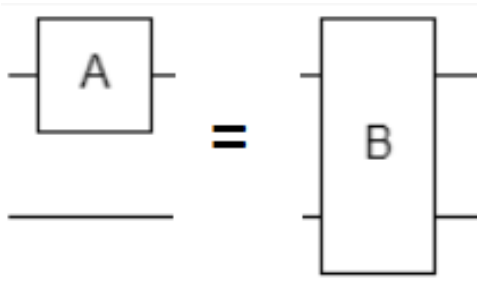


Figura 1.3: Circuito con puerta A en el primer qubit

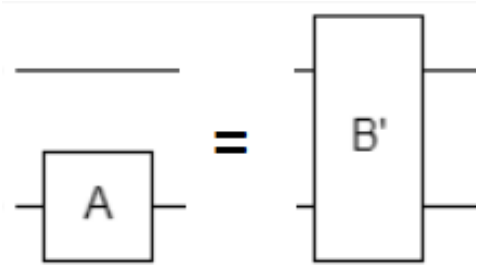


Figura 1.4: Circuito con puerta A en el segundo qubit

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 qubits y puertas de un qubit.

Se pueden ver algunas de las puertas cuánticas de múltiples qubits que utilizaremos en este texto así como la acción detallada que realizan en la tabla 1.1.

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 intercambia 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 la computación cuántica, 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

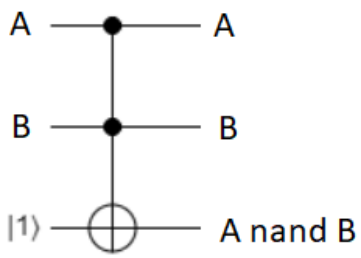


Figura 1.5: Implementación de una puerta $NAND$ a partir de la puerta de Toffoli

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 probarlo 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 framework 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 [qui]. 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 dispuestos para ello.

- Q# [mir]. 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.

2.3.1. Simuladores

Qiskit pone a disposición del desarrollador diversos simuladores con distintas utilidades. Destaca el simulador QasmSimulator, que permite hacer simulaciones tanto en el caso ideal, simulando condiciones ideales para la ejecución: tiempo de coherencia indefinido y nada de niveles de ruido; como con modelo de ruido, esto es pasando como parámetros niveles de decoherencia y de ruido para simular las condiciones de un computador físico real.

2.3.2. Máquinas reales

La API de IBMQ también nos permite lanzar nuestros circuitos a una máquina real situada en un host remoto.

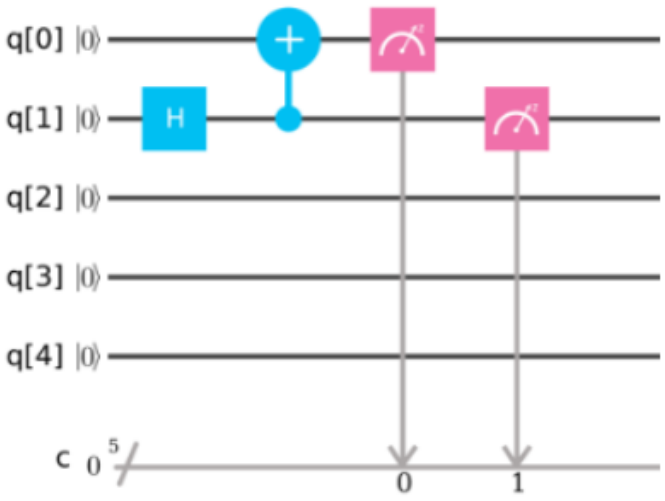
En este caso será suficiente con obtener solo el backend y no el modelo de ruido ya que vamos a lanzar el circuito contra la máquina real.

2.4. Tipos de algoritmos cuánticos

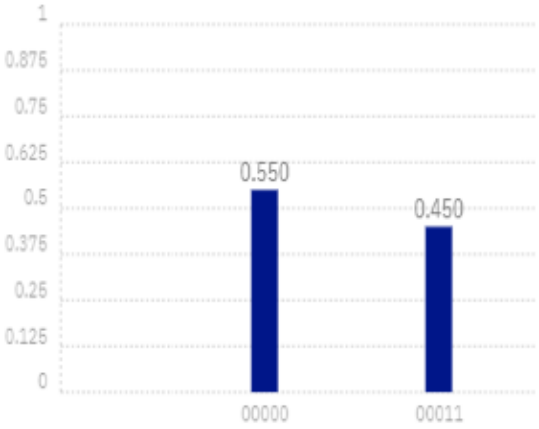
Es importante distinguir entre los conceptos de algoritmo probabilístico y el de sistema impreciso. Al igual que en el caso clásico, al diseñar un algoritmo cuántico podemos hablar de dos tipos, algoritmos deterministas y algoritmos probabilísticos. Los segundos son inherentemente indeterministas, sin embargo, debido a la complejidad del sistema físico que se 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 2.1a, 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 2.1b.

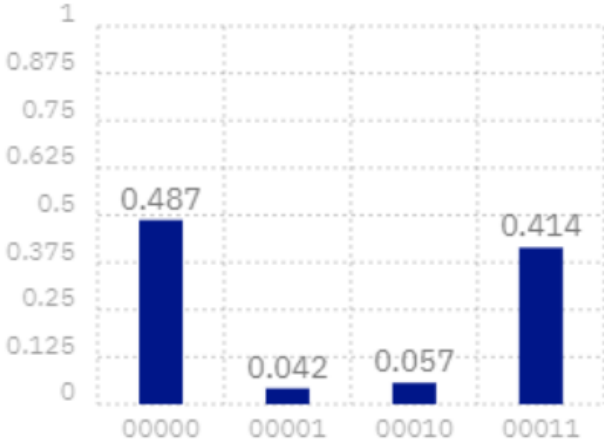
Sin embargo, si ejecutamos este mismo circuito en un computador real, observamos que



(a) Implementación en Qiskit



(b) Resultados en el caso ideal



(c) Resultados en el caso real

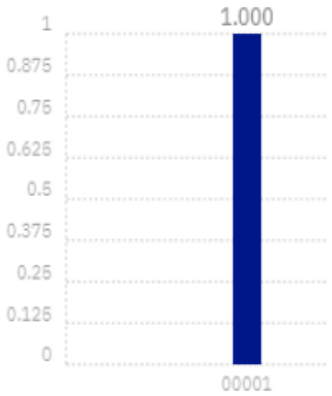
Figura 2.1: Circuito $H - CNOT$ y resultados

aparecen de forma marginal resultados teóricamente no observables (figura 2.1c). Esto se debe a las imprecisiones del ordenador real.

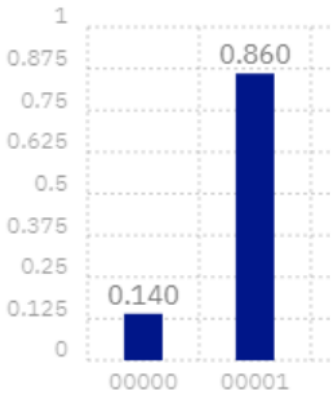
Un algoritmo probabilístico como el anterior se basa en realizar suficientes ejecuciones de algoritmo como para tener una probabilidad aceptable de que la conclusión que hemos sacado de los resultados sea la correcta. Por eso mismo son aceptables las pequeñas imprecisiones.

Por otra parte, el circuito $H - Z - H$ de la figura 2.2a, ya visto en en la sección 1.2, es claramente determinista, y de hecho, simulado sobre un computador cuántico ideal, el

(a) Implementación en Qiskit



(b) Resultados en el caso ideal



(c) Resultados en el caso real

Figura 2.2: Circuito $H - Z - H$ y resultados

resultado es totalmente determinista tal y como se observa en el gráfico de la figura 2.2b.

Sin embargo, si ejecutamos un circuito tan sencillo sobre un computador cuántico real, los resultados observados son los de la figura 2.2c.

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 representativos en algoritmos cuánticos deterministas, en el caso ideal, con una única eje-

cución 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. Hablar del coste de un algoritmo cuántico es hablar de cuatro parámetros [Sut19].

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 1.1 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. Transformada cuántica de Fourier (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

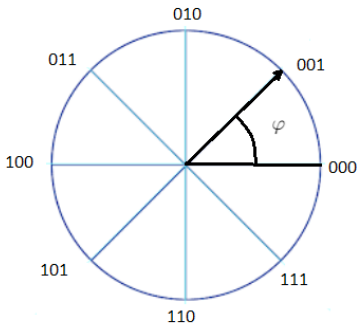


Figura 3.1: Representación del 1 en rotación de fase sobre una circunferencia

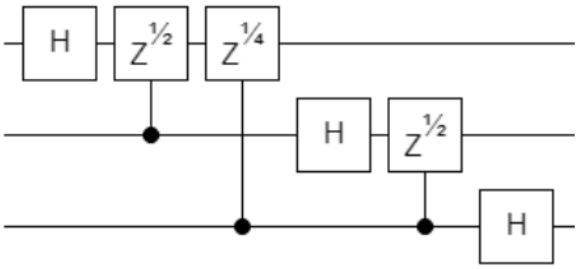


Figura 3.2: Circuito QFT de 3 qubits

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 qubit contiene información parcial sobre el estado cuántico 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).

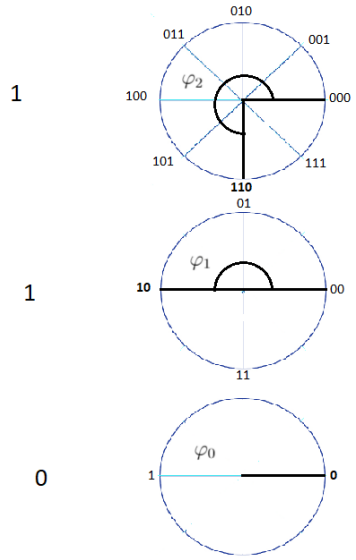


Figura 3.3: Representación del 110 en rotación de fase con 3 qubits

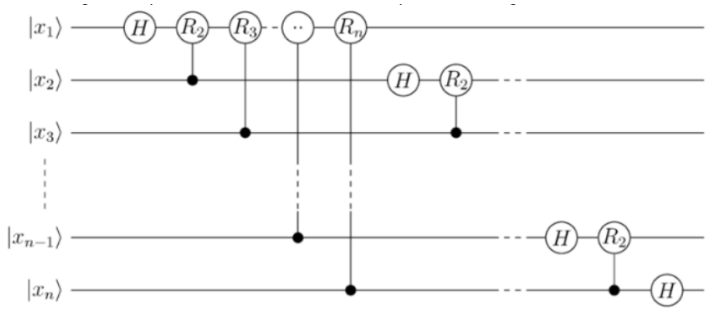


Figura 3.4: Circuito QFT de tamaño arbitrario

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 general, para construir la transformación inversa de cualquier circuito clásico, basta construir un circuito en el que se aplican las puertas inversas en orden inverso al del circuito

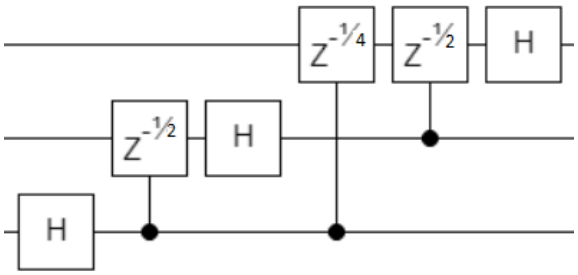


Figura 3.5: Circuito QFT^{-1} de tres qubits

original. Esto se hace 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. QFT 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.

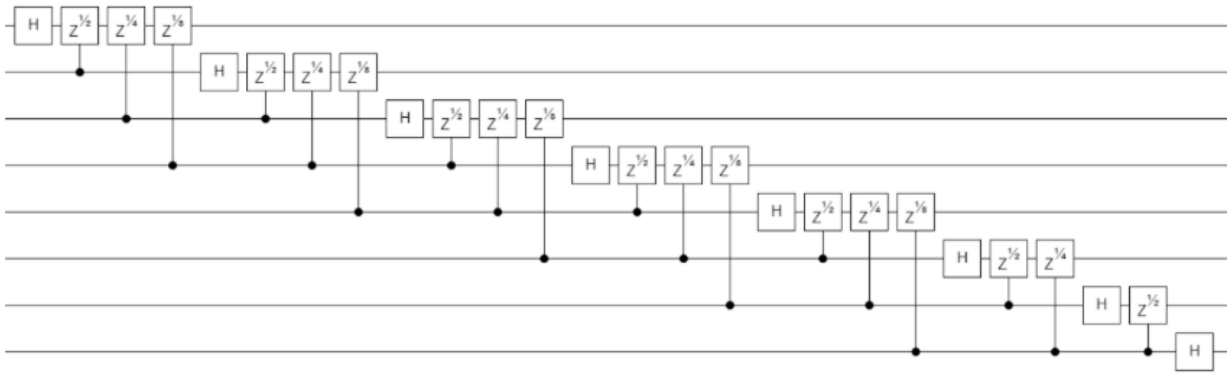


Figura 3.6: $AQFT$ 8 qubits

3.3. Sumadores

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

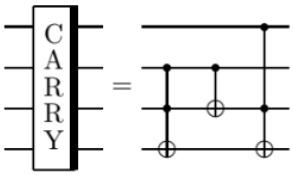
3.3.1. Sumador de Vedral

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 calcular 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.

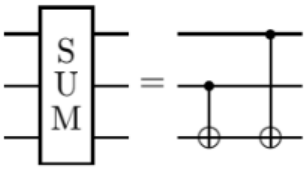
El coste de este circuito es de $8n$ en puertas y profundidad, $3n$ en qubits, n para cada operando, y n adicionales para el cálculo del acarreo, y de π en precisión de rotación.

3.3.2. Sumador de Cuccaro

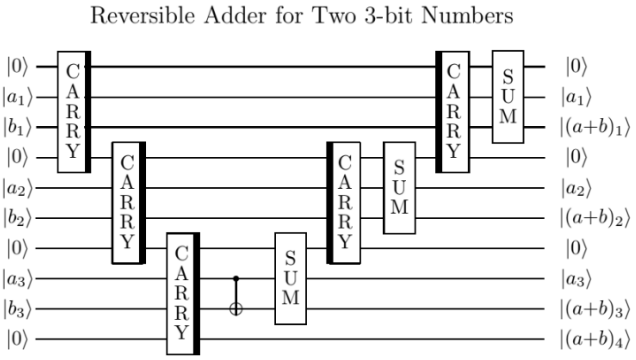
Este sumador fue presentado por Steven A. Cuccaro en [Cuc04] y se basa en los sumadores clásicos Ripple-Carry y está construido con majority-gates y UMA-gates. La función majority consiste en n entradas y una salida. La salida será $false$ si al menos $\frac{n}{2}$ entradas



(a) Puerta carry



(b) Puerta suma

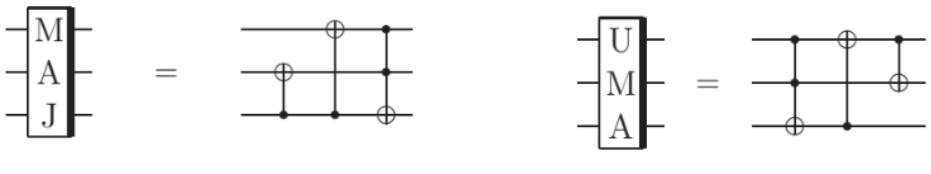


(c) Sumador

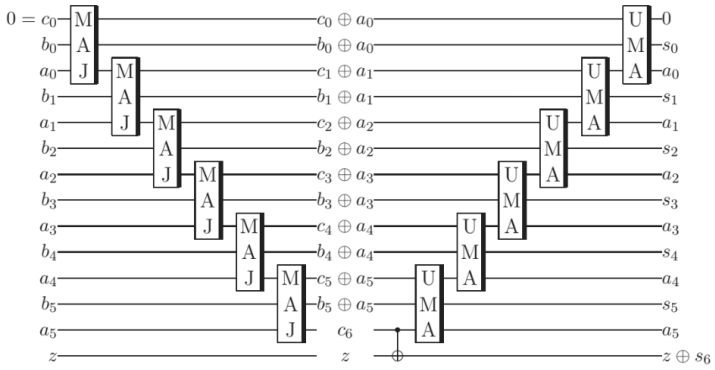
Figura 3.7: Sumador Vedral [VBE97]

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 biyectividad de todas las operaciones es posible restablecer 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 fue 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

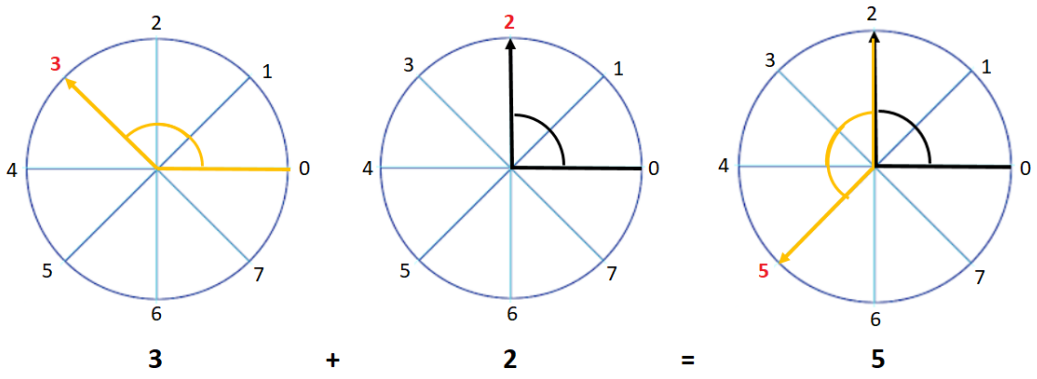
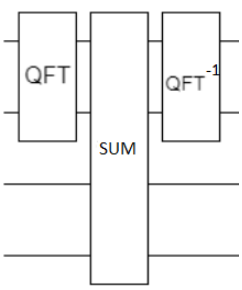
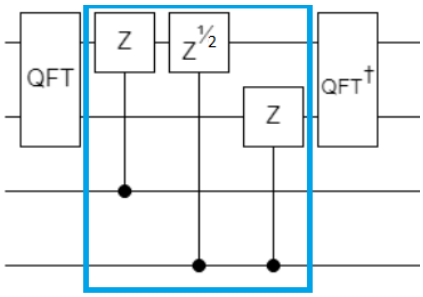


Figura 3.9: Suma de rotaciones de fase



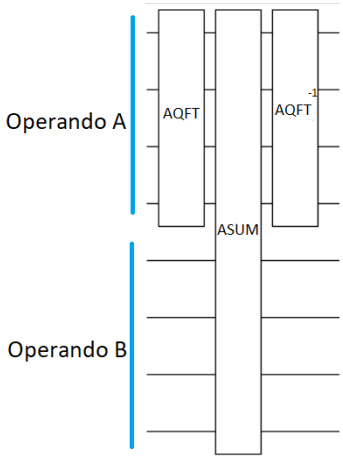
(a) Sumador basado en QFT



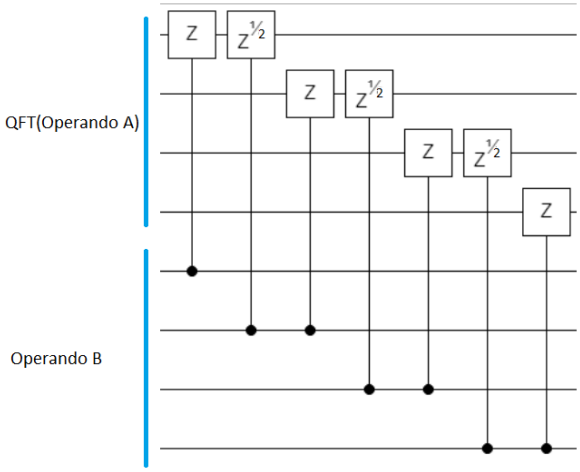
(b) Bloque SUM del sumador QFT

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 qubit menos significativo 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, y adicionalmente, una rotación de π radianes (que representa sumar $1 = 2^0$) en el qubit menos significativo del resultado para así tener la codificación parcial. La forma de que solo se hagan las rotaciones cuando los qubits del operando están a uno es utilizando puertas controladas. 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. Multiplicadores

En esta sección vamos a presentar una pequeña colección de multiplicadores para computadoras cuánticas.

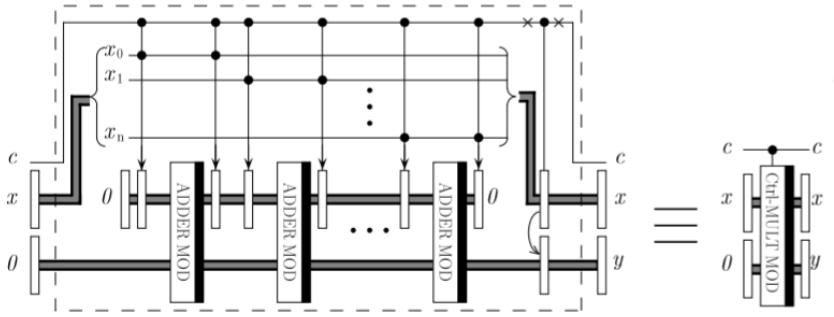


Figura 3.12: Multiplicador de Vedral [VBE97]

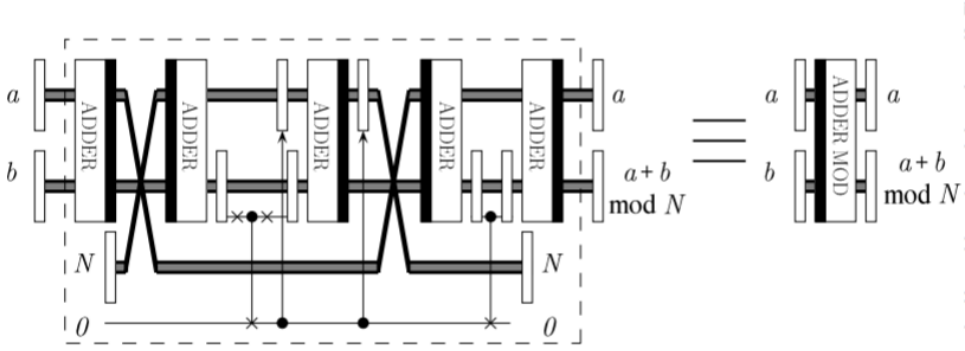


Figura 3.13: Sumador modular de Vedral [VBE97]

3.4.1. Multiplicador de Vedral

Uno de los primeros multiplicadores cuánticos fue introducido por Vedral en [VBE97]. Se trata de un multiplicador modular basado en sumas modulares sucesivas controladas por uno de los operandos. La figura 3.12 representa el esquema de este circuito, donde el módulo *MOD ADDER* es el sumador modular de la figura 3.13, que se basa a su vez en el sumador de la subsección 3.3.1. Cuando aparece con la barra negra a la izquierda es un restador, implementado mediante la inversión del sumador de 3.3.1.

Aunque en [VBE97] no se hace un análisis de coste, sí se hace en [Rui14]. Los costes son de $5n + 2$ en número qubits, $40n^2 - 10n$ en número de puertas y niveles de profundidad y de π en precisión de rotación.

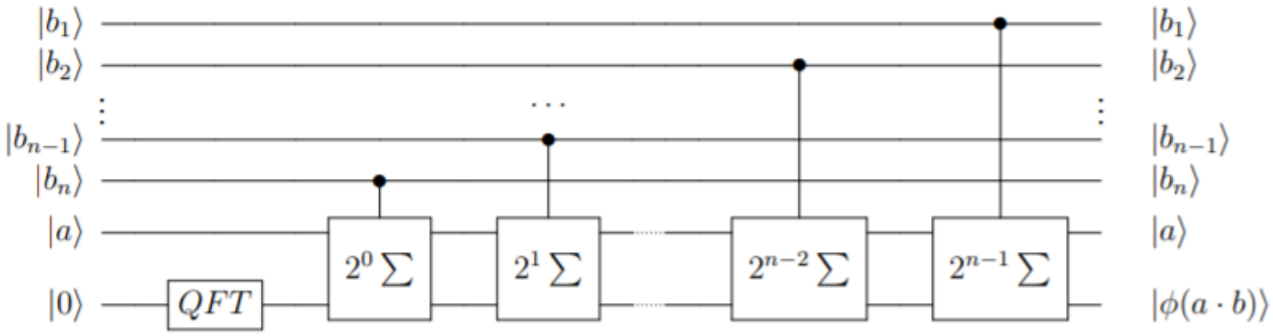


Figura 3.14: Multiplicador QFT [Rui14]

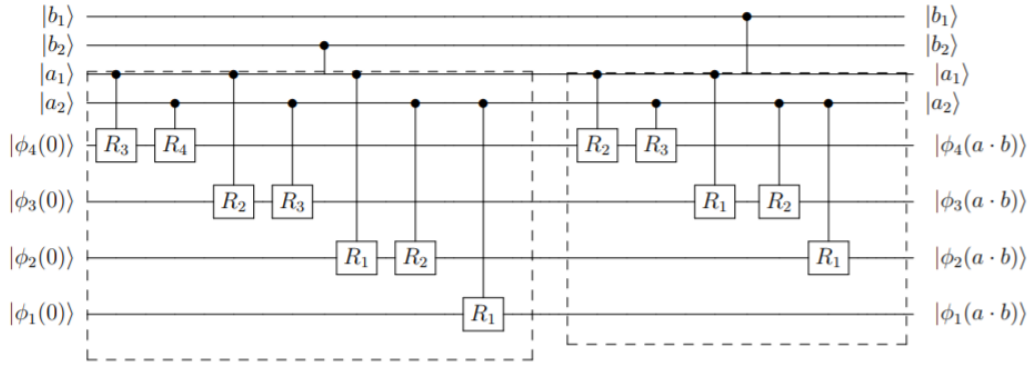


Figura 3.15: Multiplicador QFT de 2 qubits [Rui14]

3.4.2. Multiplicador basado en QFT

El siguiente circuito lo presentó L. Ruiz en [Rui14]. Se trata de un multiplicador basado en sumas sucesivas realizadas con sumador QFT. El esquema es el de la figura 3.14. Donde a y b son las entradas de tamaño n, y $|0\rangle$ son los $2n$ qubits en los que se escribirá el resultado.

En la figura 3.15 se observa un multiplicador QFT de 2 qubits. Las puertas R_k son puertas de rotación alrededor del eje Z de $\frac{\pi}{2^{k-1}}$ radianes, equivalentes a las puertas $Z^{\frac{1}{2^{k-1}}}$ que venimos tratando en el texto.

El coste de este circuito para multiplicar entradas de tamaño n es de $4n$ en número de qubits, n^3+5n^2+2n en número de puertas y niveles de profundidad, y de $\frac{\pi}{2^{n-1}}$ radianes en precisión de rotación.

3.5. Análisis de costes de unidades funcionales

	$Puertas$	$Profundidad$	$Ancho$	$Precisión$
$Sum.Vedral$ [VBE97]	$8n$	$8n$	$3n$	π
$Sum.Cuccaro$ [Cuc04]	$6n$	$6n$	$2n+2$	π
$Sum.QFT$ [Dra00]	$\frac{3(n^2+n)}{2}$	$\frac{3(n^2+n)}{2}$	$2n$	$\frac{\pi}{2^{n-1}}$
$Sum.AQFT$	$2n+3(n-1)logn$	$2n+3(n-1)logn$	$2n$	$\frac{\pi}{2^{[logn]}}$
$Mult.QFT$ [Rui14]	n^3+5n^2+2n	n^3+5n^2+2n	$4n$	$\frac{\pi}{2^{n-1}}$
$Mult.Vedral$ [VBE97]	$40n^2-10n$	$40n^2-10n$	$5n+2$	π

Cuadro 3.1: Tabla de costes de unidades funcionales

Capítulo 4

Multiplicador logarítmico

En este capítulo presentamos el diseño del multiplicador logarítmico fruto del trabajo de investigación realizado durante estos meses. 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 $log(A * B) = log(A) + log(B)$.

Dado un número natural $A > 0$, lo podemos escribir como en la ecuación 4.1. Donde llamaremos a k característica de A y a x mantisa de A.

$$A = 2^k(1 + x) \quad con \quad k \geq 0 \quad y \quad 0 \leq x < 1. \tag{4.1}$$

Obsérvese que no se puede representar el 0 con esta notación y por tanto no podremos multiplicar por cero. Una opción para solucionar este impedimento sería detectar que la entrada es nula y cablear la salida en estos casos.

Si tenemos dos números A y B en esta forma, y los multiplicamos, obtenemos la ecuación 4.2.

$$\begin{aligned} A &= 2^{k'}(1 + x'), \\ B &= 2^{k''}(1 + x''), \\ A * B &= 2^{k'+k''}(1 + x' + x'' + x'x''). \end{aligned} \tag{4.2}$$

Despreciando el término $x'x''$ tenemos que la multiplicación de $A * B$ se puede hacer a

partir de las sumas de la característica y la mantisa de A y B. Para que la representación sea consistente, si $x' + x'' \geq 1$, entonces debemos hacer que la característica del producto sea una unidad superior, y la mantisa pase a ser $x' + x'' - 1$. Quedando entonces la ecuación 4.3, para la multiplicación aproximada.

$$\begin{aligned} A * B &= 2^{k'+k''}(1 + (x' + x'')) && si \quad x' + x'' < 1, \\ A * B &= 2^{k'+k''+1}(1 + (x' + x'' - 1)) && si \quad x' + x'' \geq 1. \end{aligned} \tag{4.3}$$

El error relativo E cometido al despreciar el termino $x'x''$ es menor al $11,1\,\%$. Tal y como se describe en [Mit62].

La ejecución del algoritmo sobre entradas en codificación binaria requiere de extraer la característica y mantisa de las entradas, posteriormente hacer la suma y en caso de que la suma de las mantisas exceda la unidad, hacer el reajuste de la ecuación 4.3.

Dado un número en binario A, la característica k, será el índice, empezando en 0 y por la izquierda, del bit más significativo de A. Esto quiere decir, que para calcular la característica de A, podemos aplicar el algoritmo de detección del 1 más significativo conocido como Leading One Detector ($LOD()$). Y tendríamos que la característica de A sería como en la ecuación 4.4. Donde $bin(A)$ significa representación binaria de A y $LOD(X)$ significa el índice del 1 más significativo de X.

$$k = LOD(bin(A)). \tag{4.4}$$

El número de bits necesarios para codificar la característica de una entrada binaria de tamaño n es de $[logn]$ bits. Si utilizamos, y así lo haremos de ahora en adelante, entradas de tamaño potencia de 2, tendremos que si la entrada es de tamaño $n = 2^m$ bits, necesitaremos m bits para la codificación de la característica.

Por otro lado, la mantisa x de A, está formada por todos los bits a la derecha del 1 más significativo suponiendo que la coma decimal está justo a la izquierda del todo.

El número de bits necesarios para codificar la mantisa de una entrada binaria de tamaño n será claramente, $n-1$. Por tanto, dado un tamaño de entrada de $n = 2^m$ bits, necesitaremos

$m = logn$ bits para la característica, y $n - 1$ bits para la mantisa, lo que hace un total de $logn + n - 1$ bits para codificar la entrada.

En este sentido, dada una entrada de 8 bits, se requieren 10 bits para codificar mantisa y característica. Para $A = 00010110$, tendremos que la característica será $k = 100$ y la mantisa $x = 0.0110 = 0.011$. Si codificamos en los 3 bits más significativos el exponente, y a continuación la mantisa, en la que suprimiremos la coma, tendremos que la codificación logarítmica de A, $COD(A)$, será $COD(00010110) = 100|0110000$.

Usaremos la notación con | cuando queramos hacer explícita la distinción entre mantisa y característica. Obsérvese también, que la mantisa debe leerse (a partir de la barra) como 0.0110000.

Por otro lado, utilizaremos la notación $DEC()$ para denotar la inversa de la función $COD()$, de forma que dado un número en codificación logarítmica, devuelve el número correspondiente en notación binaria tradicional. Es importante tener en cuenta que las funciones $COD()$ y $DEC()$ dependen del tamaño de la entrada. Normalmente será el del contexto, pero utilizaremos subíndices ($COD_8()$ o $DEC_8()$) si queremos explicitar el tamaño de la entrada con la que trabajamos.

La función $DEC()$ debe realizar dos acciones: ajustar la mantisa, y colocar el 1 más significativo. Ajustar la mantisa consiste en mover la coma decimal tantas posiciones a la derecha como marque la característica y luego tener en cuenta que la coma debe quedar justo a la derecha del bit menos significativo (índice 0), por lo que puede ser necesario hacer un desplazamiento a la derecha. Para colocar el 1 más significativo basta colocar un 1 en la posición señalada por la característica.

Si tenemos dos números A y B codificados de forma logarítmica, la suma de los números nos dará el resultado del producto aproximado de la ecuación 4.3. Lo que es claro al observar que la mantisa de A se suma con la mantisa de B, y que en caso de superar la unidad, produce una llevada que se añade a la suma de las características de A y B.

En la ecuación 4.5 observamos como se haría la multiplicación aproximada para dos

entradas de ocho qubits en la que se comete un error del $6,5\,\%$.

$$
\begin{aligned}
&A = 00010110 = 22\\
&B = 00000111 = 7\\
\\
&COD(A) = 100|0110000\\
&COD(B) = 010|1100000\\
\\
&COD(A) + COD(B) = 111|0010000\\
\\
&A * B = 154\\
&\qquad = 10011010\\
&\qquad = DEC(COD(A * B))\\
&\qquad \approx DEC(COD(A) + COD(B))\\
&\qquad = 10010000\\
&\qquad = 144
\end{aligned}
\tag{4.5}
$$

Es importante notar que es posible que la suma de las codificaciones de lugar a acarreo. El acarreo no puede despreciarse pues es de vital importancia para que la función $DEC()$ funcione correctamente. Esto supone que para poder hacer la multiplicación de dos entradas de n bits, debemos aumentar el ancho de palabra que utilizan las funciones $COD()$ y $DEC()$ para asegurar que no hay acarreo (por ejemplo a $2n$). Una opción posible y por estudiar, sería utilizar un ancho de palabra distinto en las funciones $COD()$ y $DEC()$ para tratar de optimizar el ancho de palabra utilizado, lo que se traduciría en una mejora del número de qubits utilizados.

Una vez visto el algoritmo de Mitchell, el objetivo será crear los módulos que ejecuten las funciones $COD()$ y $DEC()$ para de esta forma, a partir de cualquiera de los sumadores visto antes, obtener un multiplicador. Si además, conseguimos que los módulos que realizan

las funciones de codificación y decodificación, tengan coste de orden menor o igual al de los sumadores, habríamos conseguido reducir la complejidad de la multiplicación a la de la suma. El coste a pagar, es el error cometido por la aproximación.

4.1. Codificador logarítmico cuántico

El modulo que se encargará de ejecutar la función $COD()$ constará de dos módulos: el primero se encargará de ejecutar el algoritmo LOD de detección del uno más significativo, y el segundo de ajustar la mantisa.

4.1.1. Módulo LOD

La figura 4.1 muestra el módulo cuántico que realiza el algoritmo LOD sobre una entrada de 4 qubits. Los qubits que corresponden a la entrada serán los cuatro superiores siendo el menos significativo el superior. El quinto qubit (el inferior), es un qubit adicional requerido por el algoritmo de codificación y comenzará en estado $|0\rangle$.

Dada una entrada, la salida del circuito será la posición del 1 más significativo de la entrada codificado en los qubits correspondientes a la característica (en el caso de la figura 4.1 los dos inferiores), y el resto de la entrada tal y como estaba salvo el uno más significativo que pasará a ser cero. La forma en la que se realiza, simula puertas OR anidadas que van comprobando si ya se había encontrado el 1 más significativo y si no comprueban si el qubit que toca analizar lo es y en tal caso lo codifican y ponen en estado $|0\rangle$ el qubit en el que se encuentra el 1 más significativo.

4.1.2. Módulo shifter de ajuste de mantisa

La forma en que se realizará el ajuste de la mantisa es mediante desplazamientos controlados por la característica que recordemos se encuentra en los qubits más significativos después de la aplicación del circuito LOD de la sección 4.1.1.

El circuito de la figura 4.2 realiza desplazamientos (circulares) controlados por la carac-

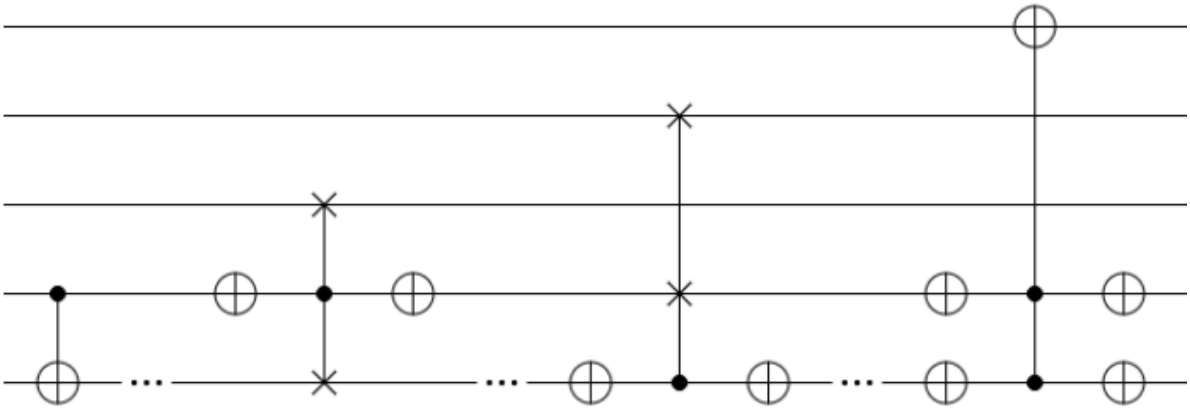


Figura 4.1: *LOD* de 4 qubits (Leading One Detector)

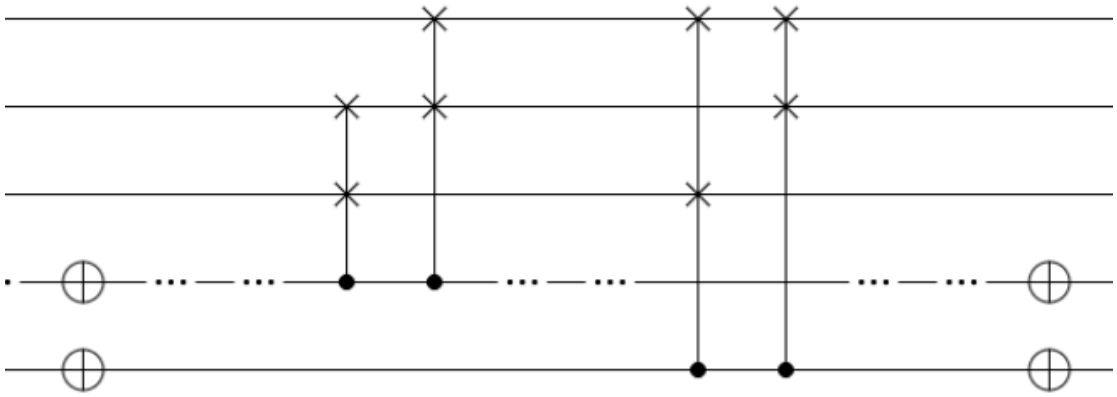


Figura 4.2: Modulo shifter de 4 qubits

terística de forma que si por ejemplo la característica codificada en los dos qubits inferiores es 00, entonces el se realiza un desplazamiento de una posición a la izquierda, y uno de dos posiciones más. Si fuese 01 se realizaría un desplazamiento de dos posiciones. Con 10 se realizaría un desplazamiento de una posición, y finalmente con 11 no se realizaría desplazamiento.

4.1.3. Codificador logarítmico $COD()$

En la figura 4.3 se observa el módulo $COD()$ completo resultado de concatenar el módulo *LOD* y el shifter. Además se han hecho algunas optimizaciones como eliminar las puertas

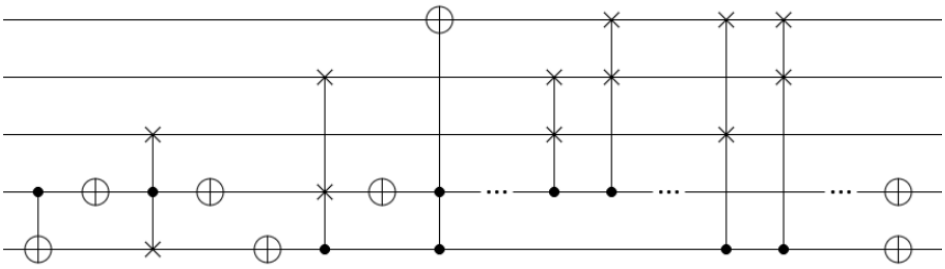


Figura 4.3: Módulo $COD()$ de 4 qubits

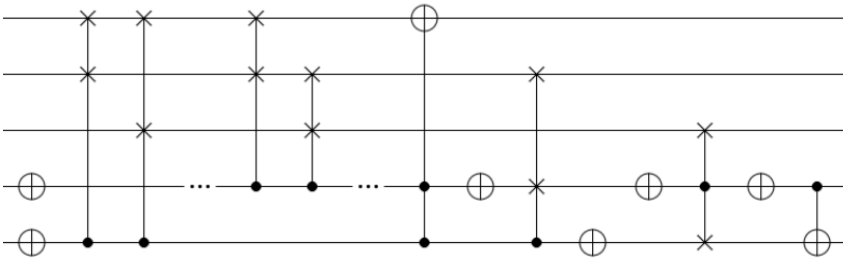


Figura 4.4: Módulo $DEC()$ de 4 qubits

X consecutivas.

4.2. Decodificador logarítmico cuántico

No entraremos en mucho detalle en la implementación del módulo que se encarga de realizar la función $DEC()$ puesto que todas las puertas utilizadas en el módulo $COD()$ tienen matriz característica unitaria y real, esto significa que la inversa es la propia matriz. La consecuencia de esto es que el módulo $DEC()$, que es la inversa del módulo $COD()$, se implementa colocando las puertas del módulo $COD()$, explicado en la sección 4.1, en orden inverso, tal y como se observa en el circuito de la figura 4.4.

4.3. Multiplicador logarítmico

Una vez introducidos los dos módulos principales, vamos a hablar sobre cual es el esquema del multiplicador completo así como de implementaciones generales para anchos de palabra mayores.

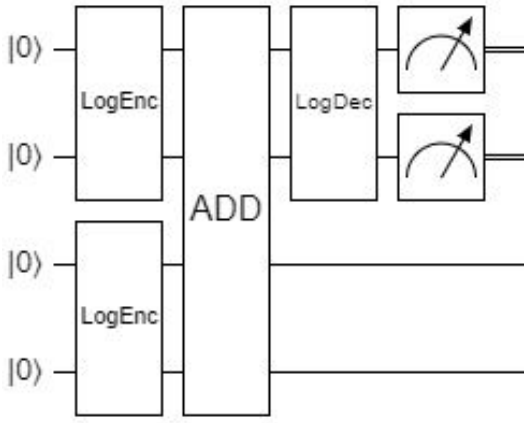


Figura 4.5: Esquema multiplicador logarítmico de 2 qubits

En primer lugar, dados los módulos de codificación y decodificación presentados en las secciones 4.1 y 4.2 respectivamente, y un sumador cualquiera, el esquema del multiplicador sería el de la figura 4.5.

4.3.1. Generalización del ancho de palabra

Debido a lo explicado en la sección 4.2 sobre la implementación del módulo $DEC()$ a partir del módulo $COD()$, solo desarrollaremos el módulo $COD()$.

En esta sección veremos que aparecerán puertas de Toffoli ($CCNOT$) y de Fredkin ($CSWAP$) generalizadas que requieren implementación a partir de puertas más pequeñas. También aparecerán puertas con anticontrol (puntos blancos), que representan que la puerta se activa solo cuando el qubit está a cero.

Generalización de puertas cuánticas

En primer lugar, la forma de implementar los puntos de anticontrol es encerrando un punto de control entre dos puertas X.

En [BBC95] se presentan varias formas de construir puertas con control generalizado a partir de puertas de control más pequeñas, la de la figura 4.6, es una forma de construirlas sin hacer uso de qubits auxiliares.

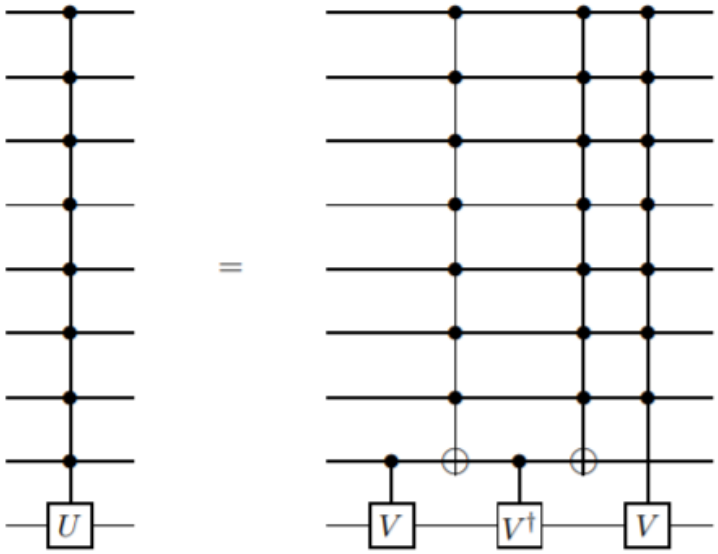


Figura 4.6: Puerta con control generalizado [BBC95]

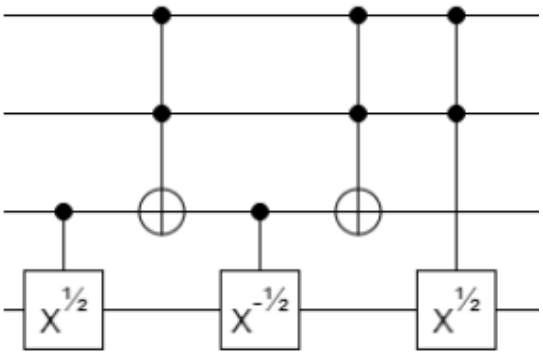
A partir de este esquema hemos construido la puerta de Toffoli de 4 qubits que aparece en la figura 4.7. Esquema que se podría seguir para construir puertas de Toffoli de tamaño arbitrario.

Por otro lado, la generalización de la puerta de Fredkin o $CSWAP$ a partir de la puerta de Toffoli de n qubits es la de la figura 4.8, donde los qubits $C1$ a Cn son los qubits de control, y los qubits $S1$ y $S2$ son los qubits de swap.

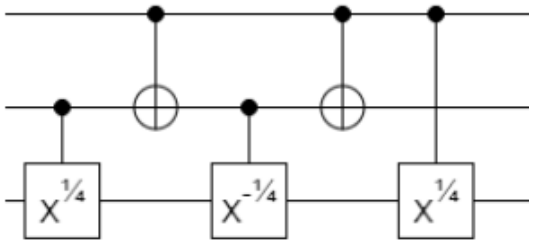
Generalización del módulo LOD

En la figura 4.9 representa un circuito LOD de 8 qubits generado de forma general. Como se explicó en la sección 4.1.1, este módulo se implementa a partir de la simulación de puertas OR. En la figura se requieren 8, una para cada qubit de entrada y se ven separadas por tres puntos.

En cada paso, empezando por el qubit más significativo de la entrada, debemos comprobar si ya se había encontrado el 1 más significativo, si es así no hacer nada, si no, comprobar si el qubit correspondiente contiene el estado 1 y en tal caso, codificar en la parte de la



(a) Esquema de Toffoli de orden 4



(b) Puerta $X^{\frac{1}{2}}$ con doble control

Figura 4.7: Puerta de Toffoli de orden 4

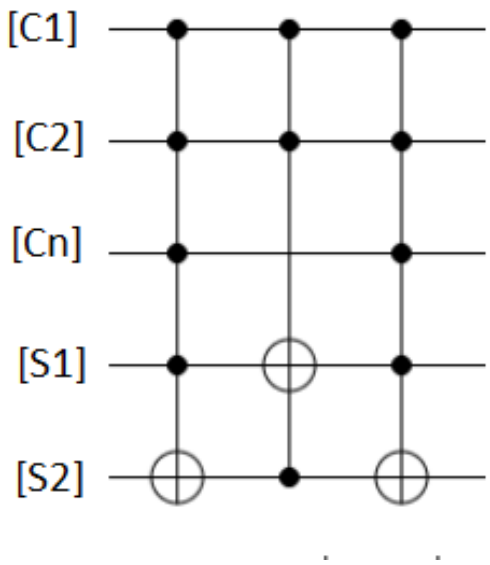


Figura 4.8: Puerta de Fredkin generalizada

característica (los tres qubits inferiores) y poner a cero el qubit. Recordemos que en la inicialización, tenemos la entrada en los ocho primeros qubits (índices 0-7) y los dos qubits adicionales 8 y 9 están en estado $|0\rangle$.

En el primer qubit (índice 7), si es 1, debemos poner un uno en cada uno de los qubits 8 y 9, si es 0 no debemos hacer nada, esto se hace mediante las puertas $CNOT$. Para el qubit 6, debemos comprobar que el qubit 7 no era el más significativo, esto se hace mediante el uso de puertas de anticontrol, que solo se activan si el qubit es nulo (no lo sería si el qubit 7

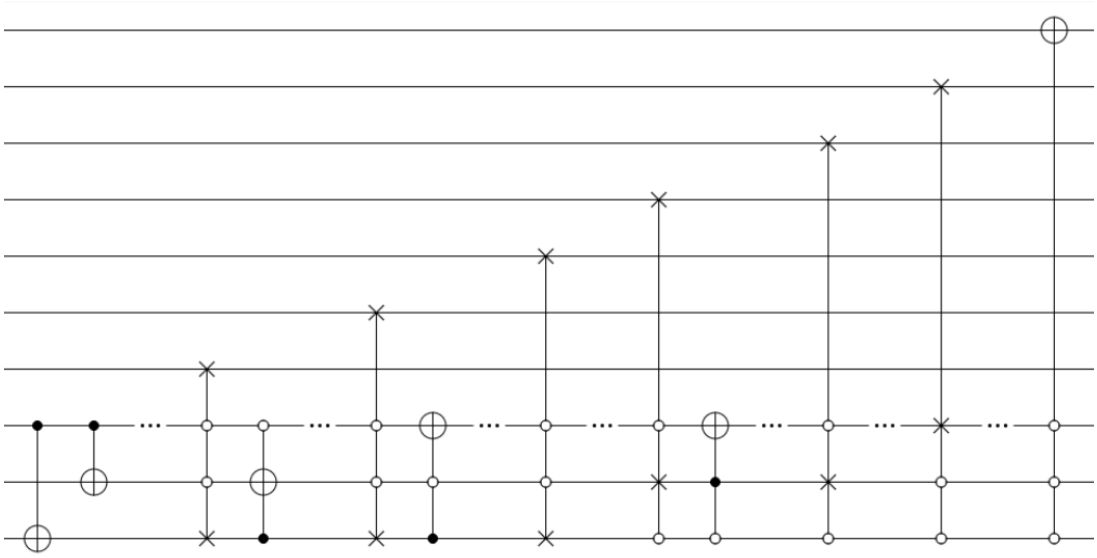


Figura 4.9: Circuito LOD generalizado

hubiese activado las puertas $CNOT$). En caso de que el qubit 6 fuese el más significativo, los qubits 7, 8 y 9 estarían a valor 0 y por tanto, la puerta de Fredkin de 4 qubits con anticontrol intercambiaría los qubits 6 y 9 dejando a 0 el qubit 6 como queríamos y codificando en la característica 100, la puerta de Toffoli siguiente, se encarga de comprobar que efectivamente la característica codificada tiene un 1 en el qubit 9 y un 0 en el 7 (no se activará si es 111) y de poner en tal caso un 1 en el qubit 8 dejando la codificación de la característica en 110, tal y como queríamos.

El caso general de los qubits intermedios es que dicho qubit será el más significativo si ninguno de los anteriores lo era, y por tanto los qubits de la característica están a cero, en este caso, se aplica la puerta de Fredkin generalizada para intercambiar el qubit que estamos tratando con el qubit que representaría el 1 más significativo de la característica, es decir, si estamos tratando el qubit 6, intercambiamos el qubit 6 y el 9, y ponemos anticontrol en el 7 y el 8; si estamos tratando el qubit 3, intercambiamos el qubit 3 y 8 y ponemos anticontrol en el 7 y el 9. Si resulta que el qubit tratado no está a uno, entonces la puerta de Fredkin generalizada solo intercambiará dos valores nulos y no se producirán cambios. Si si lo está, tendremos codificada en la característica el 1 más significativo del índice que

queremos representar, y habremos puesto un 0 en el qubit tratado. Si por ejemplo tratamos el qubit 5 y este es el más significativo, tendremos que la salida de la puerta de Fredkin tendrá 100 en la característica y un 0 en el qubit 5.

En esta situación, tendremos codificado el 1 más significativo de la característica, utilizando puertas de Toffoli generalizadas con anticontrol donde sea necesario podemos terminar de codificar la característica. Es importante que esta puerta solo se active si el qubit tratado es el más significativo. En este sentido, la puerta afectará al qubit que requiere que pongamos a 1 y tendrá un punto de control en el qubit más significativo de la característica (lo codificamos mediante la puerta de Fredkin), y puntos de anticontrol en todos los qubits restantes de la característica. En el caso del qubit 5, observamos que la puerta de Toffoli afecta al qubit 7, que sería el que hay que poner a 1 para tener codificado 101 en la característica; tiene un punto de control en el qubit 9, que es el codificado antes por la puerta de Fredkin; y un punto de anticontrol en el qubit 8, que garantiza que solo estaba a 1 el qubit más significativo de la característica. Tras su aplicación, si el qubit más significativo es el 5, la característica será 101.

La figura 4.10 muestra como se realizaría el tratamiento del qubit 5 en un circuito LOD para entradas de 32 qubits.

El tratamiento del qubit 0 es algo distinto pues si el 1 más significativo es el del qubit 0 (recordemos que el 0 no tiene representación logarítmica y por tanto no lo tratamos), entonces la característica debería codificar 000 y se debería poner dicho qubit a 0. Lo cual se hace mediante la puerta de Toffoli generalizada con anticontrol final que solo actúa si la característica codificada hasta ese momento es 000.

Generalización del módulo shifter

Recordemos que el módulo shifter se encarga de ajustar la mantisa realizando para ello un desplazamiento circular en los $n-1$ qubits menos significativos de la entrada de tantas posiciones como marca $n-1-k$ siendo n el ancho de palabra y k la característica de la entrada que se ha codificado mediante la aplicación del módulo LOD. Como $n-1-k = k$,

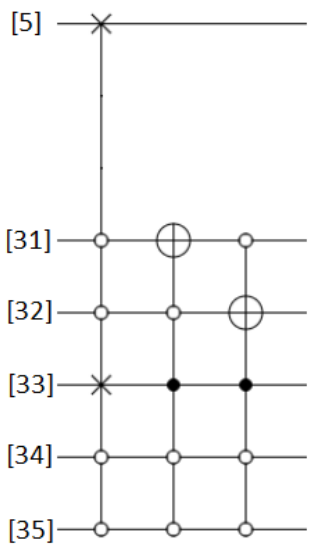
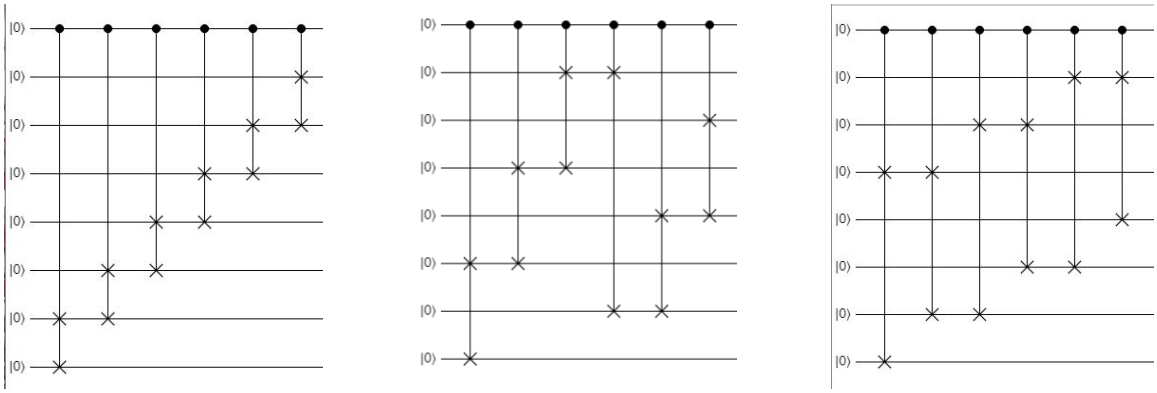


Figura 4.10: Tratamiento del qubit 5 en un LOD de 32 qubits



(a) Shifter de orden 1 (b) Shifter de orden 2 (c) Shifter de orden 4

Figura 4.11: Shifters para modulo shifter de 8 qubits

donde representa la negación lógica, tenemos que el módulo shifter, primero negará la característica, luego hará los desplazamientos controlados por la negación de k, y finalmente volverá a negar la característica para recuperar su valor original.

Necesitaremos desplazamientos de 2^m posiciones para implementar el módulo, de forma que si por ejemplo la característica en un codificador de $n = 8$ qubits es $k = 010$, entonces tenemos que $k = 101$, y que debemos hacer un desplazamiento de cinco posiciones, que se realizará a partir de una rotación de orden $2^0 = 1$ y una rotación de orden $2^2 = 4$.

Si tenemos una entrada de $n = 2^m$ qubits, necesitaremos shifter de ordenes 2^i para $1 \leq 2^i < m$.

La forma de generar un shifter que realice un desplazamiento hacia la izquierda (hacia abajo) de orden $m = 2^i$ es la siguiente. Utilizaremos puertas $CSWAP$ todas controladas por el mismo qubit, que no contemplaremos. Numeraremos el resto de qubits empezando por abajo desde 0 a $n = 2^j - 1$. Colocaremos iterativamente para $0 \leq i < n$ una puerta $CSWAP$ que intercambie los qubits $im\,\%n$ y $((i+1)m)\,\%n$. Es importante notar que esta forma de generar el shifter funciona solo porque m y n son primos relativos. En la figura 4.11 se observan los shifters de orden 1, 2 y 4 necesarios para realizar las rotaciones en un módulo $COD()$ de 8 qubits. El qubit de control es el de arriba.

Algoritmo de generalización del módulo $COD()$

A continuación explicaremos mediante pseudocódigo y lenguaje natural un algoritmo para la generación del módulo $COD()$ para ancho de palabra $n = 2^m$. Asumiremos que tenemos disponibles puertas de Toffoli y Fredkin de tamaño arbitrario y con anticontrol.

Listing 4.1: Algoritmo general para módulo $COD()$

```
# Anchura de la entrada = n
# Anchura del circuito = A = n + logn - 1
# Numeramos los qubits de 0 a A-1
# La entrada se encuentra en los qubits 0 - (n-1)
# El qubit menos significativo es el 0.
# [k] denota parte entera por abajo de k. [1.5] = 1

# Modulo LOD
    #Tratar qubit n-1
    Desde: i = n hasta i = A - 1
        Poner puerta CNOT(target = i ,control = n-1)

    #Tratar qubits desde (n-2) al 1
    Desde: i = n-2 hasta i = 1
        Poner puerta Fredkin(swap1 = i, swap2 = n-1+[logi],
        anticontrol = desde (n-1) hasta (A-1) distinto de n-1+[logi])

        # bin(k) devuelve una lista con la representacion binaria de k
        # empezando en el bit menos significativo de k

        lista L = bin(i)
```

```
            Desde j = 0 hasta j = longitud(L) - 2
                Si L(j) = 1
                    Poner puerta Toffoli(target = (n-1+j),
                    control = n-1+[logi], anticontrol =
                    desde (n-1) hasta (A-1)
                    distinto de (n-1+[logi]) y (n-1+j))

        #Tratar qubit 0
        Poner puerta Toffoli(target = 0,
        anticontrol = desde (n-1) hasta (A-1))

    # Modulo Shifter
        #Negar caracteristica
        Desde: i = n-1 hasta A - 1
            Poner puerta X(i)

        #Poner los shifters
        Desde: i = 0 hasta i = logn - 1
            Poner modulo shifter(desplazamiento = 2^i, control = n-1+i,
            target = desde (n-2) hasta 0)

        #Negar caracteristica
        Desde: i = n-1 hasta A - 1
            Poner puerta X(i)


    # Modulo auxiliar: shifter(desplazamiento, control, target)
        # Desplazamiento, cantidad de qubits que se desplazan
        # Control, qubit que controla el desplazamiento
        # Target, lista de qubits sobre los que se aplica el desplazamiento
        L = longitud(target)
        Desde: i = 0 hasta L - 2
            Poner puerta Fredkin(swap1 =
            target((i*desplazamiento)%L),
            swap2 = target(((i+1)*desplazamiento)%L),
            control = control)
```

4.4. Análisis de costes y complejidad

Para el estudio de complejidad, en primer lugar estudiaremos la complejidad de los módulos $COD()$ y $DEC()$. Como son la misma, nos centraremos en la del módulo $COD()$.

En primer lugar, el número de qubits necesarios para entradas de tamaño $n = 2^m$ es de $n + logn - 1 = n + m - 1$. Para analizar el coste en número de puertas analizaremos los módulos LOD y shifter por separado.

Para el módulo shifter (figura 4.11), necesitaremos dos barreras de $logn = m$ puertas X paralelas y un shifter de orden distinto por cada qubit correspondiente a la característica, lo que hace un total de $logn = m$ shifters. Cada shifter requiere $n - 2$ puertas $CSWAP$ por tanto el coste en puertas de este módulo será $2logn + (n-2)logn = nlogn$. El coste en profundidad será $2 + (n-2)logn$.

Por otro lado, calcular el coste en puertas del módulo LOD (figura 4.9) es más complicado debido a la presencia de las puertas generalizadas. En cualquier caso, separando el coste entre primer qubit, qubits intermedios y ultimo qubit.

Vemos que para entradas de tamaño $n = 2^m$ se requieren $logn - 1 = m - 1$ puertas $CNOT$ para el tratamiento del primer qubit.

Para el tratamiento de cada qubit intermedio se requiere una puerta de Fredkin de tamaño $logn + 1 = m + 1$ (cada una requiere 3 puertas de Toffoli de tamaño $logn = m$) con $logn - 1 = m - 1$ anticontroles y hasta $logn - 2 = m - 2$ puertas de Toffoli de tamaño $logn = m$ con $logn - 2 = m - 2$ anticontroles cada una.

Finalmente, para el tratamiento del último qubit se requiere una puerta de Toffoli de tamaño $logn + 1 = m + 1$ con $logn = m$ anticontroles.

Esto hace un total de $(4n-4)(log^2n - logn + 3) + 2logn$ puertas de coste 1, $(n-2)(logn+1)$ puertas de Toffoli de tamaño $logn$ y una puerta de Toffoli de tamaño $logn + 1$.

Es importante que en la optimización del circuito (eliminación de puertas X seguidas), se pierden gran parte de las puertas X necesarias para los anticontroles.

Debido a la complejidad del cálculo preciso de los costes en función de la entrada, haremos un cálculo explícito sobre entradas de 4 y 8 qubits que nos permitan comparar con otros circuitos. Es importante tener en cuenta, que a la hora de realizar la multiplicación, para poder tener entradas de n qubits, requerimos módulos $COD()$ y $DEC()$ de $2n$ qubits para evitar el desbordamiento.

4.4.1. Costes del multiplicador logarítmico de 2 qubits

Para utilizar entradas de 2 qubits, necesitamos módulos $COD()$ y $DEC()$ de 4 qubits. En la figura 4.3 podemos observar un módulo $COD()$ optimizado de 4 qubits. El coste son 5 qubits, 14 puertas, 12 niveles de profundidad y π en precisión de rotación.

Teniendo en cuenta que el coste del módulo $DEC()$ es el mismo que el descrito en el párrafo anterior y atendiendo a la figura 4.5, Requeriríamos 2 módulos $COD()$ paralelos, el sumador, y un módulo $DEC()$. Esto hace que el coste asociado a los módulos $COD()$ y $DEC()$ supongan un coste de 10 qubits de anchura, 42 puertas, 24 niveles de profundidad y π en precisión de rotación; a lo que habría que sumarle el coste del sumador.

Usando el sumador de Cuccaro de la subsección 3.3.2, los costes del sumador añadirían 30 puertas y niveles de profundidad, el ancho de palabra no se vería modificado pues no requerimos de carry de entrada ni salida. Y los costes totales serían 72 puertas, 54 niveles de profundidad, 10 qubits y π en precisión de rotación.

Usando el sumador QFT de la subsección 3.3.3, los costes del sumador añadirían 45 puertas y niveles profundidad y una precisión de rotación de $\frac{\pi}{16}$. Los costes totales serían de 87 puertas, 69 niveles de profundidad, 10 qubits y $\frac{\pi}{16}$ en precisión de rotación.

4.4.2. Costes del multiplicador logarítmico de 4 qubits

Para utilizar entradas de 4 qubits, necesitamos módulos $COD()$ y $DEC()$ de 8 qubits. En la figura 4.12 podemos observar un módulo $COD()$ optimizado de 8 qubits. El coste es de 33 puertas de coste 1, 6 puertas de Fredkin de 4 qubits, y una puerta de Toffoli de 4 qubits, lo que atendiendo a lo que se explicó en 4.3.1 supone un total de 204 puertas. En cuanto a profundidad, tenemos 28 niveles aportados por puertas de coste 1, y 171 niveles aportados por las puertas de Fredkin y la de Toffoli, lo que supone un total de 199 niveles de puertas. El coste en precisión de rotación viene marcado por las puertas necesarias para implementar la puerta de Toffoli de 4 qubits, que es de $\frac{\pi}{4}$.

Teniendo en cuenta que el coste del módulo $DEC()$ es el mismo que el descrito en el

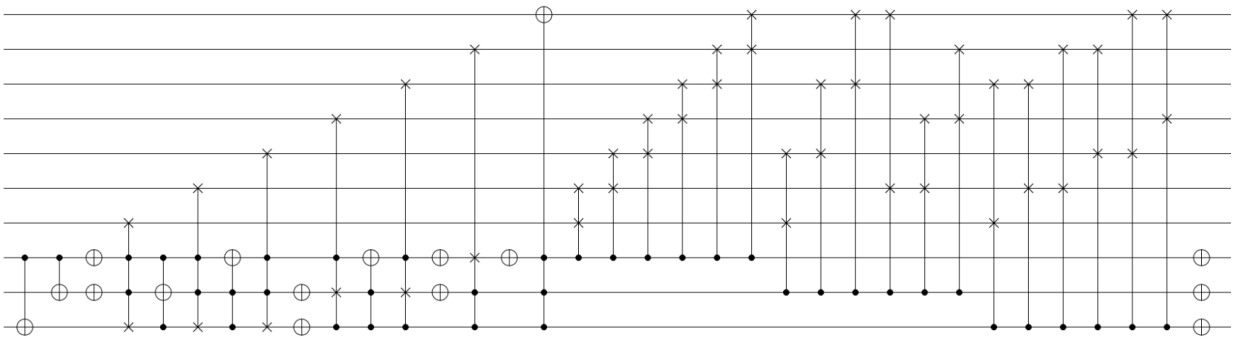


Figura 4.12: Módulo $COD()$ 8 qubits

párrafo anterior y atendiendo a la figura 4.5, Requeriríamos 2 módulos $COD()$ paralelos, el sumador, y un módulo $DEC()$. Esto hace que el coste asociado a los módulos $COD()$ y $DEC()$ supongan un coste de 20 qubits de anchura, 612 puertas, 398 niveles de profundidad y $\frac{\pi}{4}$ en precisión de rotación; a lo que habría que sumarle el coste del sumador.

Usando el sumador de Cuccaro de la subsección 3.3.2, los costes del sumador añadirían 60 puertas y niveles de profundidad, el ancho de palabra no se vería modificado pues no requerimos de carry de entrada ni salida. Y los costes totales serían 672 puertas, 458 niveles de profundidad, 20 qubits y $\frac{\pi}{4}$ en precisión de rotación.

Usando el sumador QFT de la subsección 3.10a , los costes del sumador añadirían 165 puertas y niveles profundidad y una precisión de rotación de $\frac{\pi}{512}$. Los costes totales serían de 777 puertas, 563 niveles de profundidad, 20 qubits y $\frac{\pi}{512}$ en precisión de rotación.

En este caso analizaremos también el caso del sumador $AQFT$ de la subsección 3.3.4 pues estamos ante un tamaño de puertas suficiente. Los costes añadidos por el sumador serían 118 puertas y niveles de rotación y $\frac{\pi}{16}$ en precisión de rotación. Los costes totales serían 730 puertas, 516 niveles de profundidad, 20 qubits y $\frac{\pi}{16}$ en precisión de rotación.

4.5. Comparación con otros multiplicadores

En la tabla 4.1 hacemos una comparación de los costes explícitos del multiplicador logarítmico de 4 qubits (4.4.1) y los multiplicadores de Vedral y QFT introducidos en la sección

3.4.

	$Puertas$	$Profundidad$	$Ancho$	$Precisión$
$Mul.Vedral$ [VBE97]	600	600	22	π
$Mul.QFT$ [Rui14]	152	152	16	$\frac{\pi}{8}$
$Mul.Log.Cuccaro$	672	458	20	$\frac{\pi}{4}$
$Mul.Log.QFT$	777	563	20	$\frac{\pi}{512}$
$Mul.Log.AQFT$	730	516	20	$\frac{\pi}{16}$

Cuadro 4.1: Tabla de costes de multiplicadores

Se observa que aunque el multiplicado logarítmico usando el sumador de Cuccaro podría llegar a competir con el multiplicador de Vedral, en ningún caso podríamos competir con el multiplicador basado en QFT. Esto se debe al bajo aprovechamiento del ancho de palabra del multiplicador logarítmico que utiliza módulos $COD()$ y $DEC()$ de tamaño el doble del ancho de palabra. En la sección 6.1 comentaremos una posible forma de aumentar la eficiencia del ancho de palabra para aumentar la eficiencia del multiplicador logarítmico.

Capítulo 5

Resultados de ejecución

En este capítulo presentamos los resultados de las ejecuciones de distintas entradas del codificador, el decodificador y el multiplicador logarítmico completo con distintos sumadores.

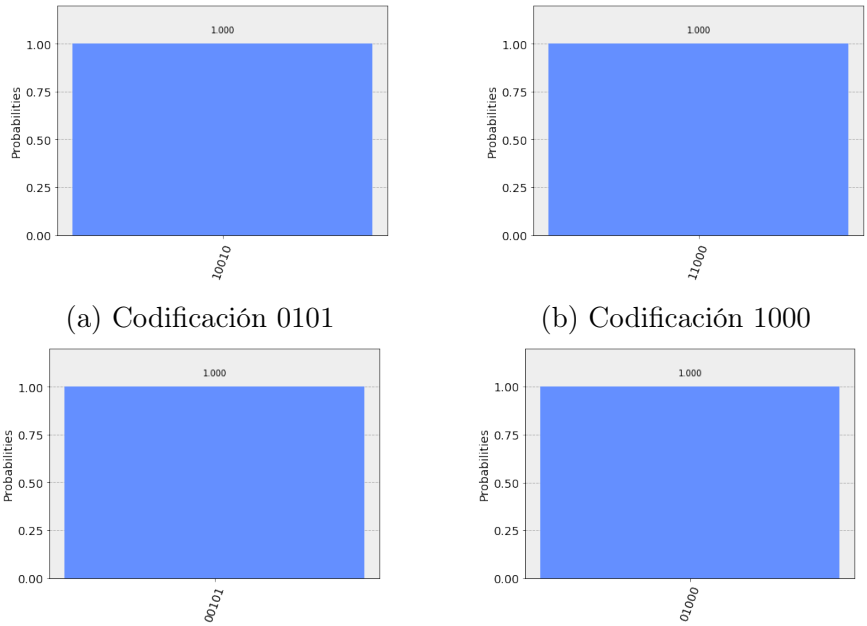
Todos los experimentos presentados a continuación se desarrollan en detalle en el Trabajo de Fin de Grado de Daniel David Abellán [Ser].

5.1. Resultados módulos $COD()$ y $DEC()$

Para cada experimento hemos empleado como entrada del codificador el 00101 y el 01000, y como entrada del decodificador 10010 = $COD(00101)$, cuya salida esperamos que sea 00101) y 11000 = $COD(01000)$, cuya salida esperamos que sea 01000. El qubit más significativo está a la izquierda en el texto y abajo en las gráficas. El ancho de palabra empleado es de 4 qubits por lo que para codificar cualquier valor serán necesarios un total de 5 qubits tal y como se indica en el capítulo 4, 2 qubits para la característica y 3 para la mantisa.

5.1.1. Simulación ideal

En la simulación en condiciones ideales podemos observar como efectivamente el circuito del módulo $COD()$ es determinista. Con la entrada 00101 en la figura 5.1a 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, que recordemos debe leerse como 0.010, por



(a) Codificación 0101

(b) Codificación 1000

(c) Decodificación 0101

(d) Decodificación 1000

Figura 5.1: Codificaciones y decodificaciones logarítmicas en condiciones ideales [Ser]

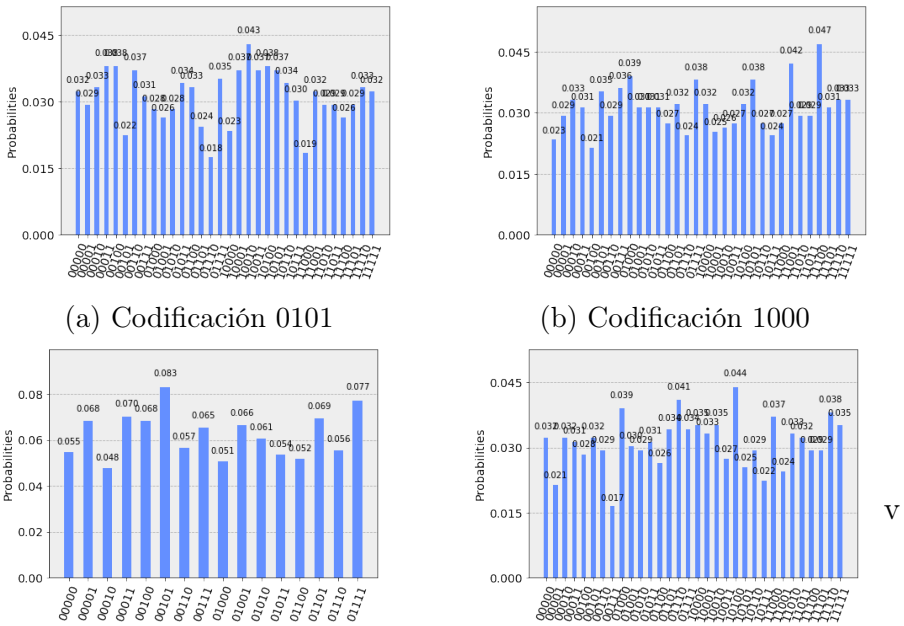
lo que el resultado es 10010.

Lo mismo sucede con la entrada 01000 donde el qubit a 1 más significativo está en la posición 3 de manera que los qubits de la característica deben tomar el valor 11, y los de la mantisa 000. El resultado final de la codificación es 11000 tal y como muestra la figura 5.1b.

Utilizando como entrada del módulo $DEC()$ la salida de los datos codificados podemos observar en las figuras 5.1c y 5.1d que la decodificación se hace correctamente y se obtiene de nuevo la entrada inicial. Si calculáramos la matriz unitaria de la concatenación de los módulos $COD()$ y $DEC()$ obtendríamos la matriz identidad de orden 5 en este caso.

5.1.2. Simulación con modelos de ruido

En este apartado hemos ejecutado los módulos $COD()$ y $DEC()$ en un simulador con modelo de ruido en un backend de 5 qubits. Como se puede observar en el conjunto de figuras 5.2 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



(a) Codificación 0101

(b) Codificación 1000

(c) Decodificación 0101

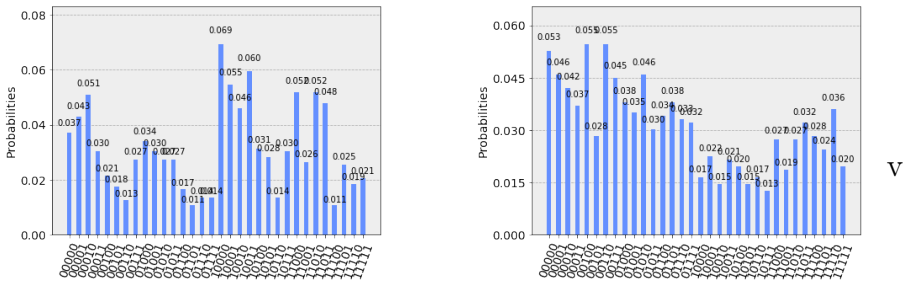
(d) Decodificación 1000

Figura 5.2: Codificaciones y decodificaciones logarítmicas en condiciones de ruido [Ser]

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 excluido 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 en comparación con 0.077 que tiene 01111 o 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 ante la presencia del ruido de una máquina real o un simulador con ruido, se observa una gran distorsión en los resultados.

5.1.3. Ejecución en máquina real

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 5.3



(a) Decodificación 0101 (b) Decodificación 1000

Figura 5.3: Codificaciones y decodificaciones logarítmicas en condiciones reales [Ser]

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 y imposible identificar el resultado correcto.

5.2. Resultados multiplicador logarítmico

Estudiado el comportamiento de los módulos $COD()$ y $DEC()$ hemos querido hacer lo mismo con el multiplicador al completo con distintos sumadores.

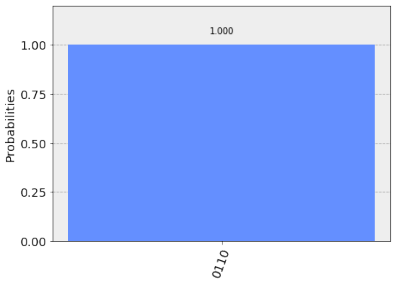
5.2.1. Simulación ideal

Para este experimento hemos empleado el simulador en condiciones ideales. A primera vista podemos observar en las gráficas de la figura 5.4 el correcto funcionamiento del multiplicador con los distintos sumadores.

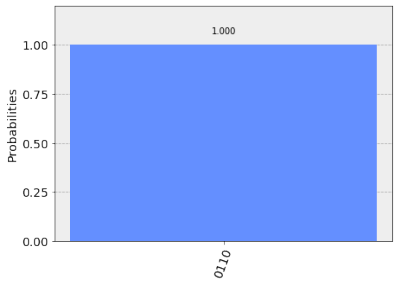
Por otro lado, en la figura 5.5 (realizado con sumador QFT) observamos como la salida de la multiplicación $3*3$ es $8 = 1000$, lo que está dentro del error cometido por la aproximación.

5.2.2. Simulación con modelos de ruido

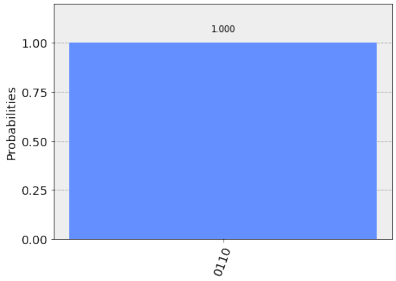
En este experimento hemos añadido al simulador el modelo de ruido del backend *'ibmq_16_melbourne'* de 14 qubits ya que necesitamos al menos 10 qubits para ejecutar el circuito.



(a) Con sumador QFT



(b) Con sumador de Cuccaro



(c) Con sumador $AQFT$

Figura 5.4: Multiplicador logarítmico de 2 qubits en condiciones ideales $(2 * 3)$ [Ser]

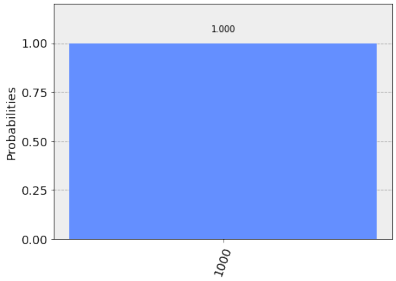
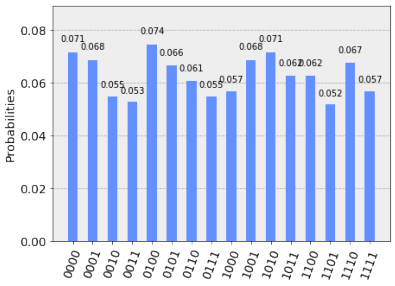
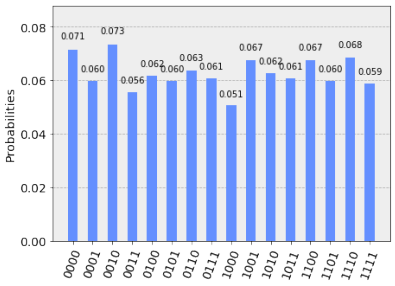


Figura 5.5: Aproximación del multiplicador logarítmico $(3 * 3 = 8)$ [Ser]

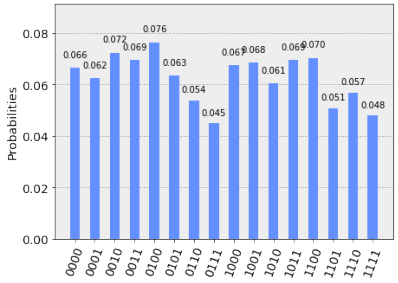
En la figura 5.6 podemos observar que ambos resultados son muy difusos y no es posible diferenciar un resultado del resto de valores debido a la gran influencia del ruido. Queda patente que la complejidad en profundidad y precisión del circuito afectan notablemente a los resultados obtenidos teniendo en cuenta que los resultados individuales del codificador vistos en el apartado 5.1.2 ya son muy difusos cabría esperar que los resultados del multiplicador completo con varios módulos similares también lo sean.



(a) Con sumador QFT



(b) Con sumador de Cuccaro



(c) Con sumador $AQFT$

Figura 5.6: Multiplicador logarítmico de 2 qubits en condiciones de ruido $(2 * 3)$ [Ser]

5.2.3. Ejecución en máquina real

En este experimento hemos empleado el backend *'ibmq_16_melbourne'* de 14 qubits para lanzar el circuito propuesto. Dados los resultados previstos por la simulación con el modelo de ruido y a los costes de hacer ejecuciones en la máquina real solo hemos ejecutado un experimento.

Como podemos ver en la figura 5.7 el resultado es el esperado y ya simulado en el apartado 5.2.2. 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.

Este problema de distorsión en los resultados debido a los niveles de ruido y decoherencia del computador real no son exclusivos de este diseño sino que de hecho es la tónica general en cualquier experimento en máquina real con un número de puertas y unos niveles de profundidad relativamente altos.

Para reflejar la forma en que la decoherencia y el ruido afectan a cualquier circuito, adjuntamos los resultados del sumador QFT de Draper (subsección 3.3.3) presentados en

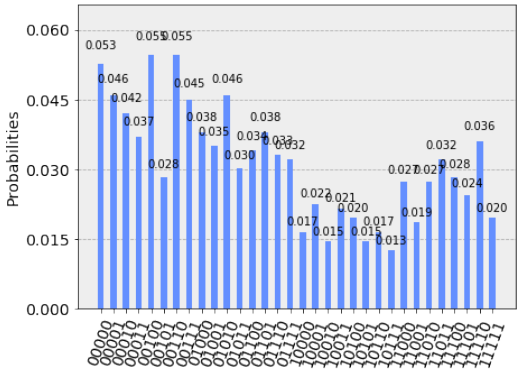
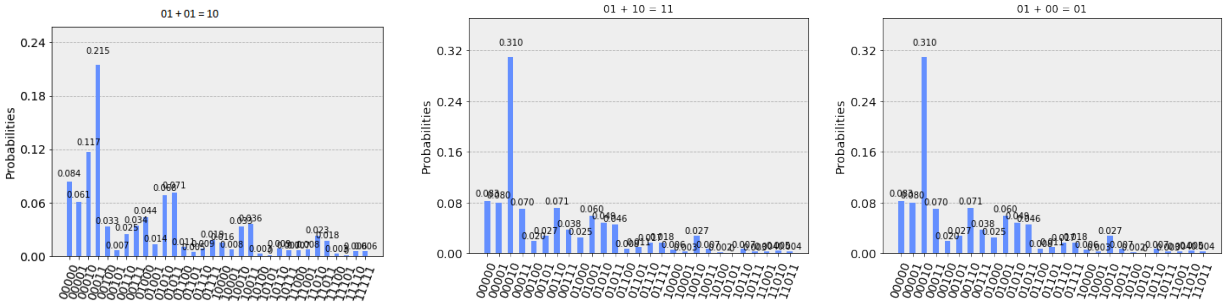


Figura 5.7: Multiplicador logarítmico de 2 qubits en condiciones reales $(2 * 3 = 6)$ [Ser]



(a) 01 + 01 (b) 01 + 10 (c) 01 + 00

Figura 5.8: Suma de 2 qubit [SdlTB+19]

[SdlTB+19].

La figura 5.8 muestra los resultados de la ejecución de la suma de varias entradas en el sumador QFT de dos qubits. El backend utilizado es *IBM Q 5 Tenerife*.

Este experimento refleja como con un sumador de tan solo 2 qubits ya se ve una gran distorsión en los resultados, aunque aún es posible distinguir claramente el resultado correcto.

Capítulo 6

Conclusiones

En este proyecto hemos hecho un estudio profundo de todo aquello que rodea al ámbito de la computación cuántica con el objetivo final de llegar al diseño del multiplicador logarítmico presentado en el capítulo 4.

En el capítulo 1 hicimos una primera aproximación a la computación cuántica tratando el concepto del qubit, las operaciones que se pueden realizar con uno o varios qubits y toda la matemática detrás de este nuevo sistema de manejo de la información. Esto es la base para la comprensión del resto del texto y quizás el capítulo más complejo de este trabajo.

Para comprender un poco más la naturaleza del trabajo en computación cuántica, en el capítulo 2 hacemos un acercamiento a las herramientas disponibles para trabajar con computación cuántica y también explicamos las partes y tipos de un circuito cuántico. También explicamos la forma en que se hacen los estudios de complejidad de los circuitos o algoritmos cuánticos y explicamos la forma en que se han hecho los cálculos de costes y complejidad en este texto dado que aún no hay en la comunidad un método estandarizado para el estudio de los costes y la complejidad en computación cuántica.

Nuestro método se basa en los parámetros de anchura, número de puertas, niveles de profundidad y precisión de rotación; conceptos que se explican en profundidad en la sección 2.5.

El capítulo 3 presentamos una colección de circuitos aritméticos con la intención de tener una colección de sumadores para poder utilizar en el diseño del multiplicador logarítmico,

y una colección de algunos de los multiplicadores diseñados hasta el momento para poder comparar los resultados obtenidos en nuestro diseño.

La parte más importante de este proyecto está en el capítulo 4, donde abordamos la adaptación del algoritmo de Mitchell para la multiplicación aproximada a la computación cuántica. El objetivo es obtener un método por el que reducir la multiplicación a la suma y obtener un algoritmo de multiplicación eficiente aunque aproximado.

Los resultados del diseño son buenos en lo referente a costes, especialmente teniendo en cuenta que aún cabe mucha optimización en el diseño del multiplicador. Sin embargo, como se puede observar en la sección 5.1.3, la ejecución en máquina física aún esta lejos de ser útil, pues al igual que pasa con el resto de circuitos de cierta complejidad, los niveles de ruido y la decoherencia causan estragos en la distribución de los resultados que, lejos de ser determinista, a penas permite identificar el resultado correcto.

6.1. Lineas futuras

Como vimos al final del capítulo 4 en la sección 4.5, el principal problema del multiplicador logarítmico es el bajo aprovechamiento del ancho de palabra debido al problema que suponía el acarreo (4). Una opción para solucionar esto y aumentar enormemente la eficiencia del multiplicador sería utilizar todo el ancho de palabra n para la entrada en el módulo $COD()$, utilizar un sumador con acarreo y utilizar utilizar un módulo $DEC()$ de tamaño $2n$.

El esquema utilizado sería parecido al de la figura 6.1. Las optimizaciones que observaríamos serían muy interesantes pues cabe esperar una notable mejora respecto del tamaño de entrada del número de qubits utilizados, el número de puertas, los niveles de profundidad e incluso de la precisión de rotación.

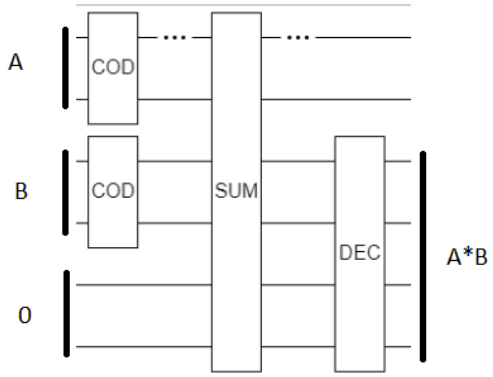


Figura 6.1: Diseño de multiplicador logarítmico optimizado

Bibliografía

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

[BBC95] A. Barenco, C.H. Bennett, and R. Cleve. Elementary gates for quantum computation. *Phys. Rev. Letters*, pages 21–22, March 1995.

[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. https://github.com/quantumlib/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.

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

[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.

[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.

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

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

[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.

[qui] Quirk. https://github.com/Strilanc/Quirk.

[Rui14] L. Ruiz. *Operaciones Aritméticas Básicas con Circuitos Cuánticos*. PhD thesis, Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid, September 2014.

[SdlTB+19] Daniel David Abellán Serrano, Antonio Valdivia de la Torre, Alberto A. Del Barrio, Guillermo Botella, and Ginés Carrascal. Simulating and executing cir-

cuits 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.

[Ser] Daniel David Abellán Serrano. Implementación en qiskit de unidades funcionales cuánticas: Un multiplicador logarítmico.

[Sut19] Sutor B. Gaining a Quantum Advantage. https://www.ibm.com/blogs/research/2018/10/quantum-advantage-2/, 2019. [Online; accessed 25-February-2019].

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