TÉCNICO LISBOA

Quantum Computing

From Algorithms to Implementation

Gonçalo Barbosa Valentim

Thesis to obtain the Master of Science Degree in

Electrical and Computer's Engineering

Supervisor: Professor Leonel Augusto Pires Seabra de Sousa

Examination Committee

Chairperson: Professora Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Professor Leonel Augusto Pires Seabra de Sousa
Member of the Committee: Professor Yasser Rashid Revez Omar

October 2021

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Acknowledgements

The conclusion of this Thesis would not have been possible without the supervision of professor Leonel Augusto Pires Seabra de Sousa whom I would like to thank for guiding and helping me through this processes, correcting mistakes and making suggestions to improve the content of this work. I am also thankful to the faculty of Instituto Superior Técnico for providing me with knowledge to the best of their ability for the past five years. To my parents, a very special thank you for always doing their best to provide for me and allowing me to follow my dreams. A very big thank you to my girlfriend for always supporting me and keeping me motivated.

Abstract

As technology continues to evolve, classic computers have reached the point where the components can't get any smaller, leading to the development of alternative approaches like quantum computing. Quantum computers are a lot more powerful than classical ones, as unlike the classic bit, the quantum bit can be in more than two states allowing quantum computers to process several states at the same time. The aim of this Thesis is delving into the world of quantum computing by analysing ways to manipulate the state of qubits, and from there, build up to the implementation of quantum algorithms and its implementation in a quantum computer. The algorithm of choice for this Thesis is the Shor's algorithm for factoring large prime numbers, selected due to its current relevance, and in particular it uses the Quantum Fourier Transform (QFT), which is the key for many quantum algorithms. Algorithms are developed using Qiskit, an open-source software development kit founded by IBM Research, that runs on Python. To provide a better insight on the algorithms', a step by step design shows the parallelism between the mathematical equations behind the algorithms and the corresponding quantum gate. This Thesis also features a semi-classical implementation of Shor's algorithm that shows how classical computation can be used to improve quantum computing. Due to physical limitations of the quantum computers available for simulations, not all quantum circuits were tested on a real quantum device as it was originally intended.

Keywords

Quantum Computing; Quantum Fourier Transform; Shor's Algorithm; Qiskit.

Resumo

À medida que a tecnologia continua a evoluir, os computadores clássicos chegaram ao limite de compressão dos componentes, o que levou ao desenvolvimento de alternativas como computação quântica. Os computadores quânticos são consideravelmente mais potentes que os computadores clássicos; ao contrário do *bit* clássico, o bit quântico pode existir em mais do que dois estados, permitindo que os computadores quânticos processem múltiplos estados ao mesmo tempo. O objetivo desta Tese é explorar o mundo da computação quântica analisando formas de manipular o estado de *qubits*, e, partindo dessa análise, desenvolver e implementá-los em computadores quânticos. O algoritmo selecionado esta Tese foi o algoritmo de Shor para a factorização de grandes números primos. Este algoritmo foi selecionado devido à sua importância e relevância , e em particular, pelo uso da Transformada de Fourier Quântica, que é a chave de inúmeros outros algoritmos. Estes algoritmos são desenvolvidos com recurso a Qiskit, uma *kit* de desenvolvimento de software criado pela IBM, que corre em Python. Para proporcionar uma melhor análise destes algoritmos, estes serão implementados passo a passo recorrendo ao paralelismo entre as equações matemáticas que os formam e a correspondência em portas quânticas. Este Tese apresenta também uma versão semi-clássica do algoritmo de Shor, realçando como a computação clássica pode ser utilizada para complementar a computação quântica. Devido a limitações dos computadores quânticos disponíveis, nem todos os algoritmos puderam ser testados em sistemas físicos.

Palavras Chave

Computação quântica; Transformada de Fourier Quântica; Algoritmo de Shor; Qiskit.

Contents

List of Figures

Acronyms

Qubit Quantum bit

RSA Rivest-Shamir-Adleman

QFT Quantum Fourier Transform

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

MSB Most Significant Bit

LSB Least Significant Bit

Nomenclature

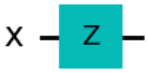
K	Kelvin
n_b	Number n is written in b basis
z^*	Complex conjugate of the complex number z
$\mathbb{C}^2$	Two-dimensional vector with complex entries
$\otimes$	Tensor product
$\|\rangle\,\|\rangle$	Abbreviated tensor product
$A^\dagger$	Adjoint of matrix A

Pauli-X Gate 

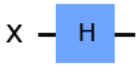
$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Pauli-Y Gate 

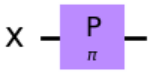
$$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Pauli-Z Gate 

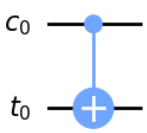
$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Hadamard Gate 

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

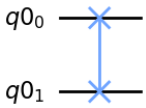
Phase Gate 

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

Controlled-NOT Gate 

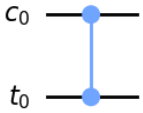
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Swap Gate



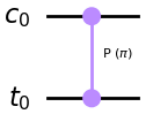
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Controlled-Z Gate



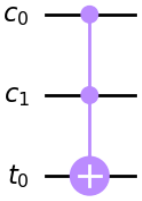
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Controlled-Phase Gate



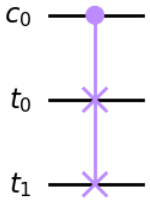
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{bmatrix}$$

Toffoli Gate



$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Fredkin Gate



$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

1

Introduction

Contents

1.1 Motivation

Since the early 20^{th} century when the atom was studied for the first time, quantum mechanics has been studied as a way to define Nature as atoms don't obey the traditional laws of physics. Quantum particles are capable of teleportation and even exist in two places at the same time - quantum entanglement.

Classical computation has evolved from computers with the size of a room to day to day common objects persons can carry on themselves, with chips containing millions of transistors as small as a fingernail. However, as technology continues to advance the more transistors are needed and unfortunately, they are already as small as they can get, what lead companies to start investing in other forms of computing approaches and technologies, like parallel computation and, more recently quantum computation [1].

Quantum computation is a lot more powerful than classical computation. Unlike the classical bit which can only be in a binary state ('0' or '1'), the quantum bit - qubit - can be in an infinite number of states besides those. This phenomena is denominated quantum superposition and allows quantum computers to process multiple states at the same time, while the classical computer can only compute them one at a time. Quantum supremacy, a theory that any and all limits set the classical computation can be surpassed by quantum computers, has been demonstrated by Google and a team of physicists from the University of California Santa Barbara [2].

Does this mean that once quantum computers develop further classical computation will become obsolete? No, this is unlikely to happen since the common user does not need or can use that much computational power. However, the launch of commercial quantum computers would have a huge impact, namely on current cryptography. Most encryption technology relies on the fact that, even knowing a way to, classical computers take a very long time to decipher keys, like the problem of finding the prime factors of a very large number which is the base for RSA encryption. Quantum computers however can perform this task efficiently, and when they get powerful enough to do it for very large numbers they will pose a threat to cybersecurity.

At the time of writing, many companies are already developing quantum computers, like IBM's Quantum One. Their goals are to isolate qubits in a controlled quantum state, which can be achieved, for instance, by cooling atoms to temperatures close to $0\,\mathrm{K}$. Another issues are that quantum computation is probabilistic and needs to be run many times, and quantum entanglement is not easily achievable making many computers to only have some qubits entangled, this means that there must also exist advanced compilers capable of mapping the computation qubits used in quantum gates to physical qubits that are entangled, which helps simulate a system where all the qubits are entangled. Even so, a team of physicists from Harvard-MIT developed a programmable quantum simulator composed of 256 qubits [3]. Therefore, since quantum computers already exist, and some are available for public use, the research object of this master thesis will be on developing and running an algorithm on a quantum computer.

1.2 Objectives

The aim of this Thesis is delving into the world of quantum computing given the growing rate and importance of quantum computation and information, and hopefully inspire other MSc and PhD students to continue researching in this area.

The present research aims to operate with quantum computation. Taking these points into account, them main objectives of this study are, in accordance with the order below:

- Analyse ways to manipulate the state of a single quantum bit;
- Analyse of how to use single qubit manipulation to build quantum circuits;
- Develop an algorithm;
- Simulate a system on a quantum simulator and on a real device.

1.3 Outline

This Thesis is organized into six prime chapters. This first chapter is an introduction to the objects of study containing the motivation behind the topics and the fixed objectives. The second chapter covers an overview on the history of quantum computation and the basic operations on quantum bits. It also provides information on how to use basic operations to build circuits. On the third chapter, the chosen algorithms and quantum computers are presented. Fourth chapter provides a circuit for the Quantum Fourier Transform and its application on the domain of quantum computing. On chapter five, the algorithm in study in this thesis is described along with it's quantum implementation. Lastly, the sixth chapter contains the conclusions of this study as well as suggestions for possible future work.

2

Theoretical Background

Contents

Created in the early 1920s, the theory of quantum mechanics, a mathematical framework or set of rules for the construction of physical theories, serves as base for the creation of quantum computation and quantum information with one of the goals of sharpening the human mind's intuition to the rules of quantum mechanics. For instance, in the 1980s, there was a theory of whether or not faster than light communication was possible, based on the possibility of cloning a quantum state. However, the impossibility of cloning quantum information - something easily done in a classical computer- gave origin to the *no-cloning theorem* [4].

Prior to the 1970s, quantum mechanics were mostly applied to try and control an enormous amount of quantum systems having no direct access to any of them, nevertheless, many techniques to control single quantum systems have been developed since then. This was a very important step to quantum computation and quantum information because by probing this new regime of Nature, new and unexpected phenomena were discovered, granting humans more control over single quantum systems which allowed them to extend it to a more complex one and discover applications to quantum computing and quantum information.

Even so, why is quantum computation needed when classical computation is already well developed and constantly evolving? Moore's Law describes that the number of transistors in a dense integrated circuit doubles roughly every two years and, as components get smaller and smaller quantum effects begin to interfere with their functioning. One possible solution for the problem posed by the eventual failure of Moore's Law is moving to a different computation paradigm, such as the one provided by quantum computation, and, even if a classical computers can be used to simulate a quantum computer, it is impossible to do so in an efficient manner. Thus, quantum computers offer an essential speed advantage over classical computers, and this speed advantage is so significant that many researchers believe that no conceivable amount of progress in classical computation would be able to overcome that gap.

One of the great early challenges of quantum computation and quantum information was the fact that the effects of realistic noise must be taken into account in evaluating the efficiency of a computational model. This issue was overcome by the development of a theory of quantum error-correcting codes and fault-tolerant quantum computation which in principle allows it to tolerate a finite amount of noise.

In 1985 David Deutsch attempted to define a computational device that would be capable of efficiently simulate an arbitrary physical system and because the laws of physics are quantum mechanic, Deutsch was led to define the conception of a quantum computer that enabled a challenge to the strong form of the Church-Turing thesis [5]. Deutsch considered the possibility of a quantum computer capable of efficiently solving computational problems, which have no efficient solution on a probabilistic Turing machine, and constructed an example suggesting that quantum computer might have more computational power than classical computers. This first step was then improved in the subsequent decade by

many others, leading to the demonstration, by Peter Shor in 1994, that two very important problems - the discrete logarithm problem and finding the prime factors of an integer -, could be efficiently solved on a quantum computer.

At about the same time, people were developing Richard Feynman idea, that there were essential difficulties in simulating quantum systems on classical computers, and that building a computer based on the principles of quantum mechanics would overcome them [6]. This lead several research teams to show that was possible to use quantum computers to efficiently simulate systems that have no known efficient simulation on a classical computer [7, 8].

To culminate these development, in 1995, Ben Schumacher defined the quantum bit, commonly known as qubit, as a tangible physical resource [9].

2.1 The Qubit

The quantum bit, or qubit, is an abstract mathematical object analogous to the bit and, just like the bit, it can be realized as an actual physical system. However, being treated as abstract entities allow the creation of general theories independent of a specific system for realization.

Just like classical bits have states, 0 or 1, quantum bits also have the states $|0\rangle$ and $|1\rangle$ which are represented using the Dirac notation, also known as Bra-ket notation. This notation uses bras('$\langle|$') and kets('$|\rangle$') to represent vectors. Taking $a, b \in \mathbb{C}^2$ as example, it is possible to write their representation using the Dirac notation,

$$\begin{aligned} |a\rangle &= \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \\ \langle b| &= \begin{pmatrix} b_1^* & b_2^* \end{pmatrix}, \\ \langle b|a\rangle &= a_1 b_1^* + a_2 b_2^*, \\ |a\rangle\langle b| &= \begin{pmatrix} a_1 b_1^* & a_1 b_2^* \\ a_2 b_1^* & a_2 b_2^* \end{pmatrix} \\ &. \end{aligned} \tag{2.1}$$

However, the main difference between qubits and classical bits is that qubits can be in other states other than $|0\rangle$ or $|1\rangle$ and, it is also possible to form linear combinations of states, which are often defined as a superposition of states. Even so, the states $|0\rangle$ and $|1\rangle$ are known as the computational basis states and a state not in any of these two is in a superposition of both, which can be written as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.2}$$

where the numbers α and β are complex numbers. Quantum mechanics however, define that when measuring a qubit the only possible outcomes are the computational basis states, and as such, when

measuring a state in superposition the only possible results are either $|0\rangle$, with probability $|\alpha|^2$, or $|1\rangle$, with probability $|\beta|^2$, which ultimately implies that $|\alpha|^2 + |\beta|^2 = 1$, since the probabilities must add to one. When measuring a qubit, it is only possible to measure the difference in phase between $|0\rangle$ and $|1\rangle$ and not the global phase of the qubit. This means that, instead of using two complex numbers, α and β can be real numbers if a term representing the relative phase between $|0\rangle$ and $|1\rangle$ is added to the definition of the state, resulting in

$$|\psi\rangle = \alpha\,|0\rangle + e^{i\varphi}\beta\,|1\rangle\,, \qquad \alpha, \beta, \varphi \in \mathbb{R}. \tag{2.3}$$

Since α and β are now real numbers, $|\alpha|^2 + |\beta|^2 = 1$ can be simplified to $\sqrt{\alpha^2 + \beta^2} = 1$. Using the trigonometric identity $\sqrt{\sin^2 x + \cos^2 x} = 1$ on this equality allows α and β to be described by a single variable θ,

$$\alpha = \cos\frac{\theta}{2}, \quad \beta = \sin\frac{\theta}{2} \qquad \theta \in [0, \pi]. \tag{2.4}$$

Replacing α and β from equation 2.3 with these new values results in

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle\,. \tag{2.5}$$

This equation defines a point on a three-dimensional unit sphere called Bloch sphere, represented in figure 2.1. The Bloch sphere is used to represent a qubit's state however there is no generalization of it for multiple qubits.

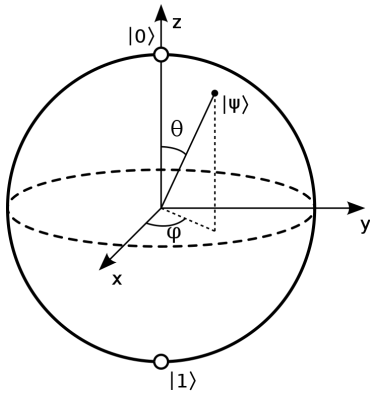


Figure 2.1: Bloch sphere [10]

Looking at figure 2.1 it is noticeable that the poles on the z-axis correspond to the basis states $|0\rangle$ and $|1\rangle$. All the six poles from the Bloch sphere have their own representation as state,

$$
\begin{aligned}
z\text{-axis} &\rightarrow \begin{cases} |0\rangle & \theta = 0, \varphi \text{ arbitrary} \\ |1\rangle & \theta = \pi, \varphi \text{ arbitrary} \end{cases}, \\
x\text{-axis} &\rightarrow \begin{cases} |+\rangle & \theta = \frac{\pi}{2}, \varphi = 0 \\ |-\rangle & \theta = \frac{\pi}{2}, \varphi = \pi \end{cases}, \\
y\text{-axis} &\rightarrow \begin{cases} |+i\rangle & \theta = \frac{\pi}{2}, \varphi = \frac{\pi}{2} \\ |-i\rangle & \theta = \frac{\pi}{2}, \varphi = \frac{3\pi}{2} \end{cases}.
\end{aligned}
\tag{2.6}
$$

It is also important to note that, when measuring a qubit in superposition the result will always be $|0\rangle$ or $|1\rangle$, and doing so will also cause the qubit to change the state, collapsing the superposition into the measured state.

Even if the Bloch sphere can only be used to represent one qubit, just like in classical computation if there are more than one qubits the number of states is also 2^n (n being the number of qubits) so, for a system with two qubits, the four possible states are $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. Also, a pair of qubits can also be in superposition described as,

$$
|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle\,. \tag{2.7}
$$

Measuring this state will result in $|x\rangle$ (x=00, 01, 10 or 11) with probability $|\alpha_x|^2$ where, just like for the single qubit case, the sum of the probabilities equals one.

2.2 Basic Gates

Classical computation consists in applying changes to the state of classical bits using logic gates. Quantum computation also consists on manipulating the state of qubits to manipulate and transport information and, to do this, it makes use of quantum gates.

2.2.1 Single Qubit Gates

A single qubit gate is a quantum gate that when applied to a qubit affects it's state in a predicted way. Like the classical single bit gate NOT, that changes the binary value of the input, there exists a quantum gate NOT that affects a qubit in a analogous way. However, to study the behaviour of a quantum gate, one must always look at the superposition state (equation 2.2). It describes all possible states a qubit can have, for instance $|0\rangle$ is just a superposition state where $\alpha = 1$ and $\beta = 0$. So, consider the function of the NOT gate, it makes a state that was 0 go to 1 and vice versa, this can also be thought as, when measuring the state after applying a NOT gate, the probability of measuring 1 or 0 swapped. Thinking about this behaviour in quantum computation, it means that, the quantum equivalent of a NOT gate, would force

$$\alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \beta\,|0\rangle + \alpha\,|1\rangle\,, \tag{2.8}$$

which in the Bloch sphere corresponds to making a rotation of π around the x-axis.

This quantum version of the not gate is one of the Pauli gates, a set of single qubit gates that apply a rotation of π around each of the axis, and whose name come from each of those axis. This means that there are three Pauli gates, the Pauli-X gate, the Pauli-Y and the Pauli-Z gates. Each of these gates can be described by a 2×2 matrix.

$$\begin{aligned} \text{Pauli-X} = \text{X} &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ \text{Pauli-Y} = \text{Y} &= \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \\ \text{Pauli-Z} = \text{Z} &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \end{aligned} \tag{2.9}$$

When applied to equation 2.2 these matrices define the transformations in equation 2.10.

$$\begin{aligned} \text{Pauli-X} &: \alpha\,|0\rangle + \beta\,|1\rangle \xrightarrow{X} \beta\,|0\rangle + \alpha\,|1\rangle \\ \text{Pauli-Y} &: \alpha\,|0\rangle + \beta\,|1\rangle \xrightarrow{Y} i\beta\,|0\rangle - i\alpha\,|1\rangle \\ \text{Pauli-Z} &: \alpha\,|0\rangle + \beta\,|1\rangle \xrightarrow{Z} \alpha\,|0\rangle - \beta\,|1\rangle \end{aligned} \tag{2.10}$$

The states corresponding to applying each Pauli gate to an uniform superposition state ($\alpha = \beta = \frac{1}{\sqrt{2}}$) are represented in the Bloch spheres on figure 2.2.

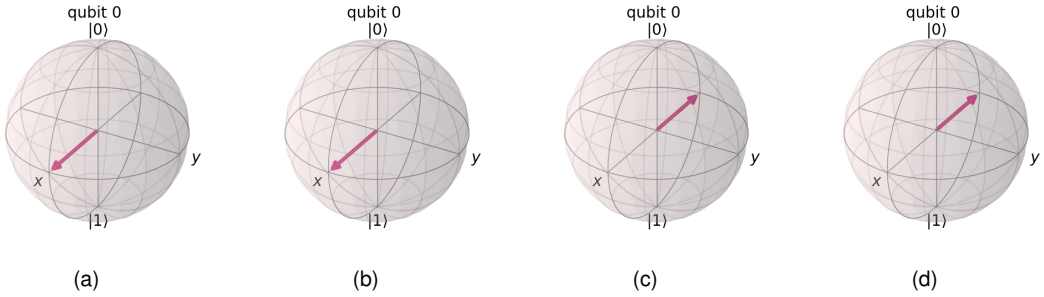
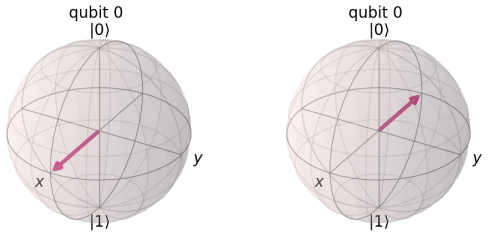


Figure 2.2: Bloch sphere representation of states produced by the Pauli gates.(a) Initial state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$;(b) State after rotation over x-axis $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$;(c) State after rotation over y-axis $\frac{1}{\sqrt{2}}(i\,|0\rangle - i\,|1\rangle)$;(d) State after rotation over z-axis $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

To simulate the results from figure 2.2 it was necessary to have a qubit in superposition. However, in quantum computation the initial state of the qubits used in simulation is $|0\rangle$, which implies the existence of a gate capable of transforming the basis state $|0\rangle$ into the uniform superposition state $|+\rangle$. This gate is called the Hadamard gate and it is defined by

$$\text{Hadamard} = \text{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.11}$$

From equation 2.11 one can easily see that applying an Hadamard gate to the basis states $|0\rangle$ and $|1\rangle$ will result in the states $|+\rangle$ and $|-\rangle$, respectively. That can also be shown through the Bloch spheres in figure 2.3.



(a) (b)

Figure 2.3: Bloch sphere representation of states produced by the Hadamard gate.(a) Hadamard applied to $|0\rangle$ produce $|+\rangle$;(b) Hadamard applied to $|1\rangle$ produce $|-\rangle$.

Besides the already referred gates there are two more single qubits gates worth mentioning, the Identity gate and the Phase gate, defined in equation 2.12

$$\begin{aligned} \text{Identity} = \text{I} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\ \text{Phase} = \text{P}(\varphi) &= \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}. \end{aligned} \tag{2.12}$$

The Identity gate, does not impose any change to the state of the qubit, analogous to the hardware specific operation NOP. The Phase gate, has a parameter φ corresponding to the rotation around the z-axis the gate applies. The Phase gate can also be referred to as S-gate or $\sqrt{Z}$-gate if $\varphi = \frac{\pi}{2}$, or as T-gate or $\sqrt[4]{Z}$ if $\varphi = \frac{\pi}{4}$.

Although every gate has its own symbol and definition, all of them can be generically defined by

$$U(\theta, \varphi, \gamma) = \begin{bmatrix} \cos\frac{\theta}{2} & -e^{i\gamma}\sin\frac{\theta}{2} \\ e^{i\varphi}\sin\frac{\theta}{2} & e^{i(\varphi+\gamma)}\cos\frac{\theta}{2} \end{bmatrix}, \tag{2.13}$$

like the Hadamard and S-gate.

$$\begin{aligned} U(\frac{\pi}{2}, 0, \pi) &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H, \\ U(0, 0, \frac{\pi}{2}) &= \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{bmatrix} = S. \end{aligned} \tag{2.14}$$

2.2.2 Multiple Qubit Gates

Having control over the state of one qubit is good, however, to do proper computation, qubits need to interact with each other, just like in the basic logic gates from classical computation (AND, OR, XOR, NAND, NOR). To accomplish this, in 1995 Chris Monroe [11], demonstrates the operation of a two qubit "controlled-not" gate know as C-NOT. The matrix describing this gate's functionality is given by

$$\text{C-NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.15}$$

The C-NOT gate acts on two qubits, a control qubit c and a target qubit t. If the control qubit is $|0\rangle$, the target qubit state remains unchanged however, if the control qubit is $|1\rangle$, a Pauli-X gate is applied to the target qubit. This interaction can be mapped by

$$|c\rangle\,|t\rangle \xrightarrow{C-NOT} |c\rangle\,|c \oplus t\rangle\,. \tag{2.16}$$

Summarized, the C-NOT gate stores in the target qubit the XOR of both qubits. This raises the possibility of one being able to realize a quantum implementation of the classical logic gates. However, all unitary quantum gates are always invertible and as such can always be inverted by another quantum gate, contrary to the classic logic gates that are non-invertible. This means that it is impossible to build quantum gates that replicate classical logic gates the same way the Pauli-X gate replicates the classical NOT.

2.3 Quantum Circuits

Having defined some basic gates, the next step is to use them to build a quantum circuit, like the one in the diagram of figure 2.4. Quantum circuit diagrams are analysed from left to right, each line represents the evolution in time of a different qubit, represented top to bottom. A quantum register is a set of qubits used to represent a variable and, each individual qubit of the register is named in the diagram next to the correspondent line, as is the case of the variables $|t\rangle$ and $|c\rangle$ represented in figure 2.4. This qubits are numerated according to the size of the register and the qubit corresponding to the LSB is the one with index '0'. So, reading the value stored in a quantum register is done bottom to top.

Some well-known quantum circuits, composed mostly by basic gates and C-NOTs, are commonly treated as complex quantum gates. Since they are built using C-NOTs, the gates will be described using two quantum registers, the control register $|c\rangle$ and the target register $|t\rangle$.

Analyzing the circuit from figure 2.4, which is composed by three C-NOT gates, an equation describing the behaviour of the circuit can be written. Naming the C-NOTs, C-NOT$_1$, C-NOT$_2$ and C-NOT$_3$, left

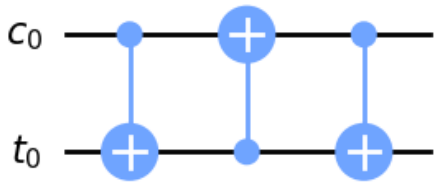


Figure 2.4: Quantum circuit diagram to implement a swap gate.

to right the function applied by the circuit is given by

$$\begin{aligned} &|c\rangle\,|t\rangle \xrightarrow{C-NOT_1} |c\rangle\,|c\oplus t\rangle \\ &|c\rangle\,|c\oplus t\rangle \xrightarrow{C-NOT_2} |c\oplus(c\oplus t)\rangle\,|c\oplus t\rangle = |t\rangle\,|c\oplus t\rangle \\ &|t\rangle\,|c\oplus t\rangle \xrightarrow{C-NOT_3} |t\rangle\,|(c\oplus t)\oplus t\rangle = |t\rangle\,|c\rangle\,. \end{aligned} \tag{2.17}$$

This shows that this circuit swaps the values from the registers and as expected is known as the swap gate.

Another complex gate that can be built using C-NOTs and basic gates is the controlled-Z gate, which, as the name expresses, is a controlled version of the Pauli-Z gate. This gate operates on the fact that applying a Z gate to $|+\rangle$ has the same effect of applying an X gate to $|0\rangle$, it flips the qubit. Since the Hadamard gate takes $|0\rangle$ to $|+\rangle$, applying an Hadamard gate to the qubit, followed by a Pauli-X gate and another Hadamard produces the same effect has a Pauli-Z gate, and so, to make a controlled version of the Z gate it is enough to replace the Pauli-X gate by a C-NOT gate. The circuit describing this gate is represented on figure 2.5.

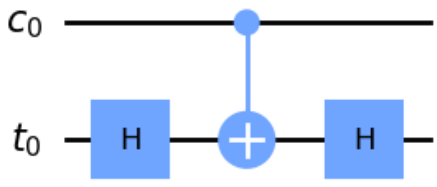


Figure 2.5: Quantum circuit diagram to implement a controlled Z gate.

The equality described can be proven by analyzing the circuit of figure 2.5, assuming $|c=1\rangle$ and $|+\rangle$

$$
\begin{aligned}
&|1\rangle |+\rangle \xrightarrow{H_1} |1\rangle |0\rangle \\
&|1\rangle |0\rangle \xrightarrow{C-NOT} |1\rangle |0 \oplus 1\rangle = |1\rangle |1\rangle \\
&|1\rangle |1\rangle \xrightarrow{H} |1\rangle |-\rangle ,
\end{aligned}
\tag{2.18}
$$

which shows that this circuit takes the target register from $|+\rangle$ to $|-\rangle$, applying the same changes as the Pauli-Z gate. It is also important to notice that when $|c = 0\rangle$ the C-NOT gate does not make any change leaving the circuit as $HH = I$.

Another gate that can be built in a similar way is the controlled Phase gate. As it can be seen in figure 2.1, an arbitrary state $|\psi\rangle$ can be represented in the Bloch sphere by the amplitude of the angles θ and φ and, in the same way the quantum gates can also be described by the transformations to those angles. For instance, the C-NOT gate adds π to both θ and φ, and the Phase gate only adds the desired phase to φ leaving θ unchanged. Using this representation, is possible to analyze the circuit from figure 2.6 and prove that it is in fact, a controlled Phase gate.

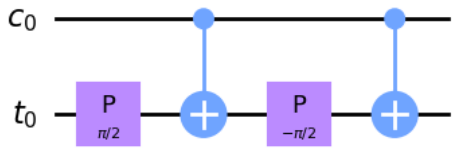


Figure 2.6: Quantum circuit diagram to implement a controlled Phase gate gate.

Using the $|\psi\rangle = |(\theta, \psi)\rangle$ notation to represent a state in the Bloch sphere, the circuit behaviour can be describe, for $|\psi\rangle = |+\rangle = \left|(\frac{\pi}{2}, 0)\right\rangle$, as

$$
\begin{aligned}
&|1\rangle \left|(\frac{\pi}{2}, 0)\right\rangle \xrightarrow{P_{\varphi/2}} |1\rangle \left|(\frac{\pi}{2}, \frac{\varphi}{2})\right\rangle \\
&|1\rangle \left|(\frac{\pi}{2}, \frac{\varphi}{2})\right\rangle \xrightarrow{C-NOT_1} |1\rangle \left|(\frac{\pi}{2}, -\frac{\varphi}{2})\right\rangle \\
&|1\rangle \left|(\frac{\pi}{2}, -\frac{\varphi}{2})\right\rangle \xrightarrow{P_{-\varphi/2}} |1\rangle \left|(\frac{\pi}{2}, -\varphi)\right\rangle \\
&|1\rangle \left|(\frac{\pi}{2}, -\varphi)\right\rangle \xrightarrow{C-NOT_2} |1\rangle \left|(\frac{\pi}{2}, \varphi)\right\rangle .
\end{aligned}
\tag{2.19}
$$

For the case where the control qubit is $|0\rangle$ the C-NOTs do nothing and the Phase gates apply the inverse of one another and as such apply the identity. This behaviour leads to the conclusion that the circuit from figure 2.6 is indeed a controlled P gate.

Just as the C-NOT is a controlled X gate, there is a gate called the Toffoli gate that is essentially

a doubly-controlled X gate, also known as CC-NOT. As expected, this gate functions exactly like the C-NOT gate, however, instead of requiring one control qubit set to $|1\rangle$ to flip the target qubit, it requires both control qubits to be set. A circuit to implement the Toffoli gate is represented in figure 2.7, an image taken from [12], where the behaviour of the circuit is analysed.

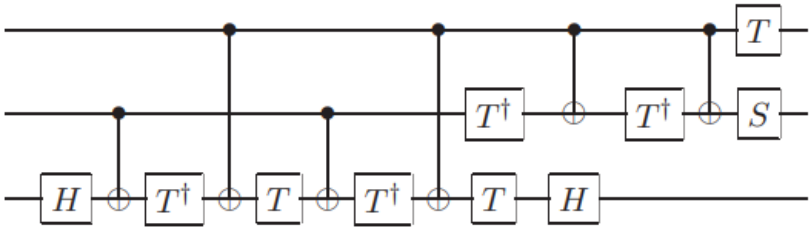


Figure 2.7: Quantum circuit diagram to implement a Toffoli gate. Taken from [12].

Most of the circuits described in this section, are simply constructions that apply a controlled version of a basic gate. However, the circuit in figure 2.8, is a controlled version of a quantum circuit, the controlled Swap gate, also known as the Fredkin gate. Comparing the circuit in figure 2.4 with the one in figure 2.8, the only difference between the two is that, for the Fredkin gate, the second C-NOT is controlled. This means that, when the control bit is $|c = 1\rangle$ the circuit behaves like a normal swap gate, swapping $|t_0\rangle$ and $|t_1\rangle$. When the control is $|c = 0\rangle$, the Toffoli gate does nothing leaving the circuit as two sequential C-NOT gates that, since they have the same control and target qubit, which is equivalent to applying the Identity gate.

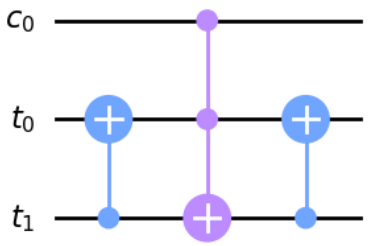


Figure 2.8: Quantum circuit diagram to implement a Fredkin gate.

3

Quantum Computing

Contents

As aforementioned, quantum computing will be implemented and tested tested on both real quantum devices and high-end quantum simulators. This chapter provides information about the algorithms and the quantum computer provider chosen as well as the reasoning behind those choices.

3.1 Algorithms

The choice of the algorithm that this Thesis is focused on was made taking into account the current importance of the algorithm and its relevance for possible studies. One of the most important quantum algorithms is Shor's algorithm for factoring large numbers [13]. Number factorization as always taken interest by investigators. The inability to efficiently do number factorization is one of the backbones of modern cryptography, as such, the quantum implementation of Shor's algorithm as been object of intensive research. An added bonus of this algorithm is that it uses the Quantum Fourier Transform (QFT) which is the key for many quantum algorithms. This way, instead of studying only the implementation of a complex Shor's Algorithm, this thesis will also focus on a simpler yet very relevant quantum circuit, the QFT.

3.1.1 Quantum Fourier Transform

The discrete Fourier transform (DFT) is a mathematical transformation that converts a finite sequence of equally-spaced samples into a function of frequency. This is very useful as it allows for problems with no trivial solution in the time domain to be studied in the frequency domain, where a result could be easier to achieve. The DFT can, for instance, be used to examine information encoded in the frequency of a signal by calculating it's spectrum or to determine a system's frequency response facilitating the analysis of the system in the frequency domain. Efficient algorithms to implement this transformation to sequences are denominated Fast Fourier Transforms (FFT). The quantum Fourier transform (QFT) is an efficient quantum algorithm analogous to the FFT, meaning that it applies the DFT on the quantum mechanical amplitudes of a quantum state. Given the nature of the qubit, the QFT provides a way to estimate phase, which only by itself is not something extremely valuable but when applied to other algorithms can be used to solve very important problems, such as factorization and finding the number of solutions to a search problem.

3.1.2 Shor's Algorithm

In 1995, Peter Shor described a way to solve in polynomial time two problems thought to be unfeasible on a classical computer [13]. At that time, the best factoring algorithm was the number field sieve [14], requires $\exp\left\{\theta(n^{\frac{1}{3}}\log^{\frac{2}{3}}n)\right\}$ operations to factor an n-bit integer. This algorithm scales exponentially with

the size of the number being factored and so, as numbers get bigger, it quickly becomes impossible to factor them, which makes factoring considered an intractable problem for classical computation. Shor's Algorithm however, can factor an n-bit integer with polynomial O($n^2 \log n \log\log n$) number of operations representing an exponential speed-up over it's classical analogous. This is an important achievement providing evidence of superior processing power of quantum computers compared to classical ones. Modern cryptography, like the RSA public-key cryptosystem, which relies on being impossible to factor the product of two large prime numbers, serve to prove from a practical standpoint the importance of Shor's Algorithm.

3.2 Quantum computers

In this Thesis is considered that, although Shor's algorithm can factor big numbers, in practice today we are talking about integers with 2048 bits, the size of keys adopted by RSA based cryptosystems. The necessary resources to process such big numbers are not available physically in a quantum computer. However, they can be simulated in high-end classical computers specifically designed to simulate quantum circuits, referred to as quantum simulators.

The quantum computers and quantum simulators used in this Thesis to run circuits are part of IBM Quantum Experience [15], a cloud application that allows users to program real quantum computers. The first quantum computer presented by IBM was the IBM Q System One [16] which served as base for the development of the other available quantum computers. System One was released with a twenty-seven-qubit Falcon processor, which was later surpassed by the Hummingbird processor. This last processor that provides sixty five qubits was unveiled by IBM alongside their roadmap for technology scaling, where their goal of surpassing one thousand qubits by the year of 2023 was established [17]. IBM also provides a framework that allows users to run quantum circuits on their quantum computers. This framework comes in the form of Qiskit [18], an open-source software development kit that provides tools for the design and simulation of quantum programs founded by IBM Research that runs on Python. Qiskit works like a descriptive programming language. The user starts by defining how many qubits are present in the circuit, then he can apply a quantum gate to each qubit. The Python code in figure 3.1 uses Qiskit to describe a two qubit circuit: an Hadamard gate is applied to the first qubit and a Pauli-X gate to the second, then a C-NOT gate controlled on the first qubit acts on the second. This circuit schematic can be visualized using Qiskit's draw method. This method performs the graphical representation of the circuit's diagram using matplotlib -a Python library for creating static, animated, and interactive visualizations [19] -, as can be visualized in figure 3.2. Circuits built this way can then be simulated on the user's own device using Qiskit's Aer simulators or on IBM's quantum computers and quantum simulators. Running quantum simulation on a personal device can be ill advised given the

amount resources necessary to perform such task.

```
"""Define size of the circuit"""
qr_1 = QuantumRegister(1)
qr_2 = QuantumRegister(1)
qc = QuantumCircuit(qr_1, qr_2)
"""Apply gates to the qubits"""
qc.h(qr_1)
qc.x(qr_2)
qc.cx(qr_2, qr_1)
```

Figure 3.1: Code to implement a simple circuit using Qiskit.

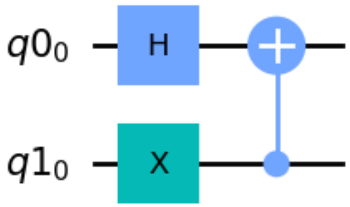


Figure 3.2: Diagram of the circuit created using code for figure 3.1.

There are twenty-one quantum systems made available by IBM [20] ranging from one (Canary processor) to sixty-five (Hummingbird processor) available qubits. However, only eight of these are available for ordinary users. Of the eight available systems, one is composed of only one qubit while the other seven are composed by five qubits. For these quantum computers, the quantum volume - a single number metric that quantifies the largest random circuit of equal width and depth that the computer successfully implements [21] -, is one for the single qubit system and eight, sixteen or thirty-two for the remaining. Despite the amount of available qubits for simulations being low, IBM Q also provides five high-performance simulators for prototyping quantum circuits and algorithms, and exploring their performance under realistic device noise models [22] Simulators of different types: a five thousand qubit Clifford simulator that only allows the use of Clifford group operations [23]; a one hundred qubit Matrix Product State simulator which is more efficient for states with low entanglement [24]; a sixty-three qubit extended Clifford simulator [25]; a thirty-two qubit Schrödinger wave function simulator that computes the wave fucntion of the state vector as instructions are applied; and a thirty-two qubit general-purpose simulator that choses the simulation method based on the input circuit. All these systems can be freely accessed from Qiskit using the IBM Quantum API provided the access token generated at [15].

4

Quantum Fourier Transform

Contents

The QFT is a basis transformation algorithm that acts as the anchor for many quantum algorithms by facilitating the transition from the computational basis to the Fourier basis while also allowing phase estimation. This quantum algorithm is an efficient implementation of the discrete Fourier transform in a quantum computer.

On this chapter, the quantum version of the Fourier transform, as well as the inverse Fourier transform, will be deduced, implemented and finally validated by being run on IBM's quantum computers and quantum simulators

4.1 Quantum Fourier Transform

The DFT takes an input vector of complex numbers (x) and outputs a transformed vector of complex numbers of the same length (y).

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k/N}. \tag{4.1}$$

The QFT is the quantum implementation of equation (4.1); instead of complex numbers it operates on quantum states. An arbitrary quantum state $\sum_{i=0}^{N-1} x_i |i\rangle$ is mapped to the state $\sum_{i=0}^{N-1} y_i |i\rangle$ which when applied to equation 4.1 can be written as

$$|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x y/N} |y\rangle . \tag{4.2}$$

It is important to note that the quantum state $|x\rangle$ in equation 4.2 should be written in a n-base notation representative of the quantum system. In this case, the quantum states are represented with a binary notation with each qubit corresponding to a single bit, $x = x_1 x_2 ... x_n$. Given the relation between a decimal and a binary number,

$$(y)_{10} = 2^{n-1}(y_1)_2 + 2^{n-2}(y_2)_2 + ... + 2^0(y_n)_2 \equiv (y)_{10} = \sum_{k=1}^{n} (y_k)_2 2^{n-k}. \tag{4.3}$$

The DFT transforms a vector of complex numbers into another vector of the same size, typically a vector is composed by N equidistant samples of some arbitrary function. When representing the quantum states using binary notation, the maximum number of representable states is given by 2^n where n is the total number of qubits in the system. By replacing this value in equation 4.1,

$$|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i x y/2^n} |y\rangle \tag{4.4}$$

and rewriting equation 4.3 to get

$$(y/2^n)_{10} = \sum_{k=1}^{n} (y_k)_2/2^k, \tag{4.5}$$

it's possible to derive an expression for the mapping of the QFT in the binary case by combining equations 4.4 and 4.5. Let QFT($|x\rangle$) be the quantum Fourier transform of the state $|x\rangle$ where $x = x_n...x_2x_1$, this transformation can be described by:

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x \sum_{k=1}^{n} y_k/2^k} |y\rangle . \tag{4.6}$$

Equation 4.6 can be further simplified through algebraic manipulation (appendix A) to the expression considered by Nielsen, Michael A., and Isaac Chuang as the *definition* of QFT in [12],

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \left[(|0\rangle + e^{2\pi i 0.x_1} |1\rangle) \otimes (|0\rangle + e^{2\pi i 0.x_2 x_1} |1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i 0.x_n ... x_2 x_1} |1\rangle) \right]. \tag{4.7}$$

The first term of equation 4.7 corresponds to the state of the most significant bit of $|x\rangle$ and the last term to the least significant one.

4.2 Circuit Design

To ensure that the QFT is unitary the circuit should be designed using only unitary gates. A simplified expression of the function to implement allows a better view of the necessary gates. Looking at 4.7 it's noticeable that every qubit suffers the same modifications differing only in the value of the phase so, it is easier to determine the circuit by analysing a single qubit and then replicating the process to the other ones. Isolating the last qubit from 4.7 results in

$$\frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.x_n ... x_2 x_1} |1\rangle), \tag{4.8}$$

where the $\frac{1}{\sqrt{2}}$ in 4.8 comes from the factor $\frac{1}{\sqrt{N}}$ and $N = 2^n$. This equation can be further simplified by decomposing the binary fractional notation in the exponent, originating,

$$\frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i (\frac{x_1}{2^n} + \frac{x_2}{2^{n-1}} + ... + \frac{x_n}{2^1})} |1\rangle). \tag{4.9}$$

This simplified expression is very similar to the implemented by the Hadamard Gate, differing only in the phase. The phase is depended on the other qubits, so a circuit to implement this function can be achieved by utilizing Hadamard Gates (equation 4.10) and controlled rotations with different amplitudes (equation 4.11). The layout of such circuit is shown in figure 4.1.

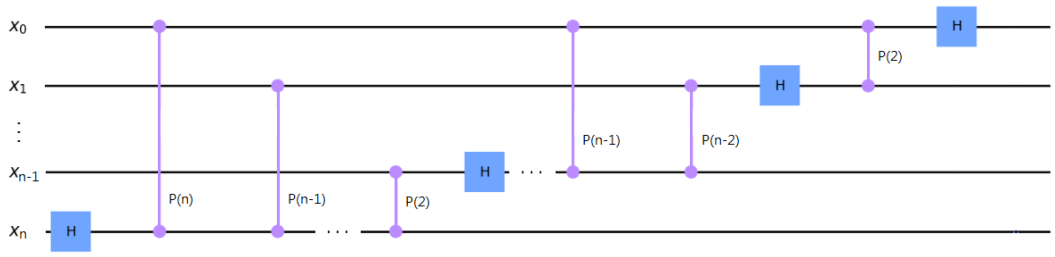


Figure 4.1: Circuit to implement QFT.

$$H\left|x_k\right\rangle = \frac{1}{\sqrt{2}}(\left|0\right\rangle + e^{\pi i x_k}\left|1\right\rangle) \tag{4.10}$$

$$P(k) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^k}} \end{pmatrix} \tag{4.11}$$

For the last qubit, after applying the Hadamard gate, it is transformed into

$$\frac{1}{\sqrt{2}}(\left|0\right\rangle + e^{\frac{2\pi i x_n}{2}}\left|1\right\rangle). \tag{4.12}$$

After that, qubit the last qubit suffers $n-1$ rotations controlled by the other qubits, the amplitude of these rotations depends on the index difference between the target and the control qubit. The transformation applied by all the rotations to the state produced in 4.12 makes the state

$$\frac{1}{\sqrt{2}}(\left|0\right\rangle + e^{\frac{2\pi i}{2^n}x_1 + \frac{2\pi i}{2^{n-1}}x_2 + \ldots + \frac{2\pi i}{2}x_n}\left|1\right\rangle) = \frac{1}{\sqrt{2}}(\left|0\right\rangle + e^{2\pi i 0.x_n \ldots x_2 x_1}\left|1\right\rangle). \tag{4.13}$$

The correctness of the circuit can be confirmed by comparing the final state of the last qubit (MSB) described in 4.13 with the definition of the QFT (equation 4.7). It can be seen, that the final state of the last first qubit doesn't match the target state but instead matches the intended final state of the first qubit (LSB). This can be easily fixed by swapping the qubits at the last stage of the circuit. Adding these swaps to the circuit changes its layout to the one represented in figure 4.2.

4.3 Circuit's Implementation

Having the gates necessary to build the circuit been determined, the circuit was implemented using Qiskit. After implementation, Qiskit's integration with IBM provide a way to simulate the circuit, run on a quantum computer, and check whether the produced results correspond to the expected ones. Given the number of users daily running programs on IBM's quantum computers and the small size of the circuit,

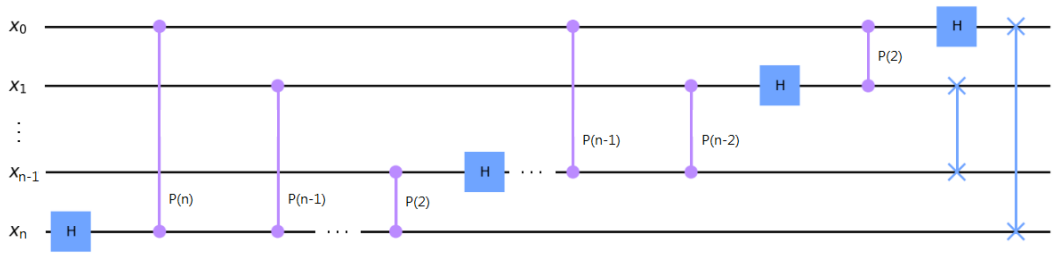


Figure 4.2: Circuit to implement QFT with swaps.

the circuit was first be tested on Qiskit's Aer simulator. Measuring a qubit on a quantum computer causes its state to collapse. However, by using a quantum simulator it is possible to access and represent the final state of the qubits using a state vector representation, in the Bloch sphere. Both on Qiskit simulators as well as on IBM quantum computers and quantum simulators, all qubits are initialized in the state $|0\rangle$ so, to simulate the QFT of a chosen state it is first necessary to force the qubits to that state, which will require adding more quantum gates to the previous design.

Applying the Pauli-X gate to a qubit forces a rotation of π around the x-axis, flipping the qubits from $|0\rangle$ to $|1\rangle$, making it ideal to manipulate the initial state into the desired value. For instance, forcing the initial state $|x\rangle = |11_{10}\rangle = |1011_2\rangle$ can be achieved by applying a Pauli-X gate to qubits q_0, q_1 and q_3. Applying these gates before the circuit for the QFT results in figure 4.3 and, simulating the circuit using Qiskit Aer produces the state vector represented in figure 4.4.

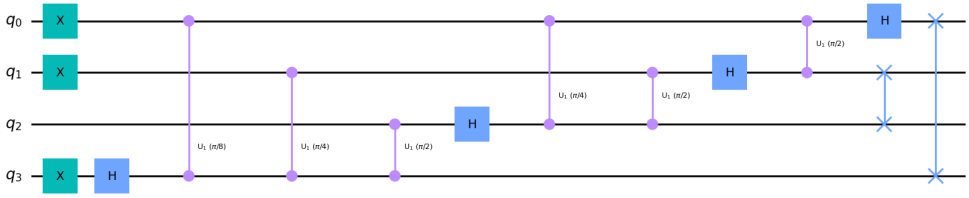


Figure 4.3: Circuit implemented to simulate QFT of $|1011\rangle$.

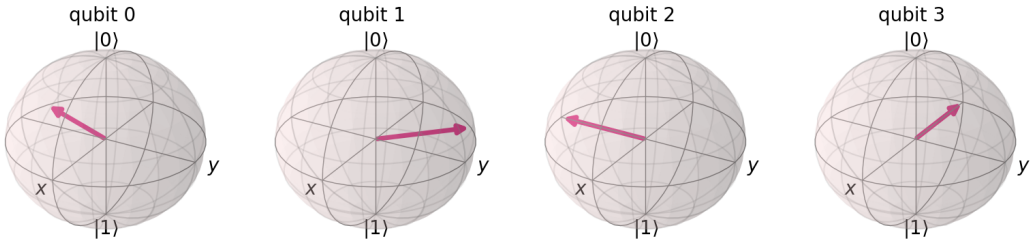


Figure 4.4: State vector correspondent to QFT($|1011\rangle$).

The correctness of state vector in figure 4.4 can be confirmed by calculating the expected states pro-

duced by applying the QFT to $|1011\rangle$. This result can be obtained by solving equation 4.7 for $x = 1011$.

$$
\begin{aligned}
q_0 &= \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.1011}\,|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{\frac{11}{8}\pi}\,|1\rangle) \\
q_1 &= \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.011}\,|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{\frac{3}{4}\pi}\,|1\rangle) \\
q_2 &= \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.11}\,|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{\frac{3}{2}\pi}\,|1\rangle) \\
q_3 &= \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.1}\,|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{\pi}\,|1\rangle)
\end{aligned}
\tag{4.14}
$$

Comparing the state for each qubit in equation 4.14 with the respective Bloch sphere representation of each qubit in figure 4.4 it is shown that the circuit from figure 4.3 applies the QFT, as intended.

After confirming the correctness of the circuit using a quantum simulator, the circuit was tested in a real quantum device. However, when measuring a state, quantum computers can only read either $|0\rangle$ or $|1\rangle$ collapsing states in superposition. This poses a problem for the implemented circuit, because all the qubits in the expected output state are in a superposition state. In order to properly test it, the circuit had to be build in such way that the output state would be composed only of qubits in the basis states. This was achieved by, instead of testing the QFT which takes a basis state to a superposition state, using the inverse QFT with a prepared superposition state as input producing as output a basis state. Considering the properties of the quantum gates, the inverse QFT can be applied simply by inverting the circuit that applies the QFT, resulting in the one represented in figure 4.5. Since this circuit applies the inverse QFT, using the state vector from figure 4.4 as input should produce the state $|1011\rangle$. To prepare the desired input, to each qubit will be applied an Hadamard gate to force it to the state $|+\rangle$, and a rotation corresponding to the angles calculated in equation4.14. Meaning that, after the Hadamard gate, qubit 0 will be applied a rotation of $\frac{11}{8}\pi$, qubit 1 a rotation of $\frac{3}{4}\pi$, qubit 2 a rotation of $\frac{3}{2}\pi$ and qubit 3 a rotation of π. To get the final result, each qubit will also need to be measured and the result stored in a classical buffer. This can done by adding a measurement gate to each qubit at the end of the circuit description. The circuit that prepares the initial state and applies the inverse QFT and measures the output is represented in figure 4.6.

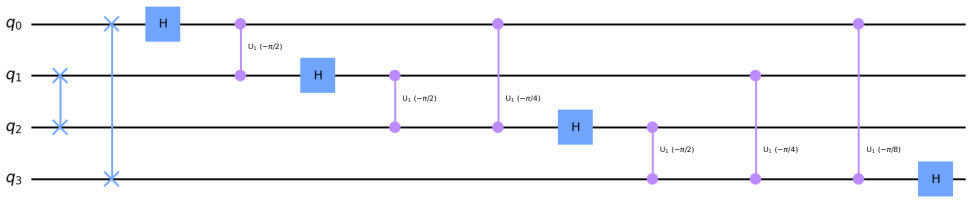


Figure 4.5: Circuit to implement the Inverse Quantum Fourier Transform.

Given the probabilistic nature of quantum computers, the circuit needs to be ran multiple times. The

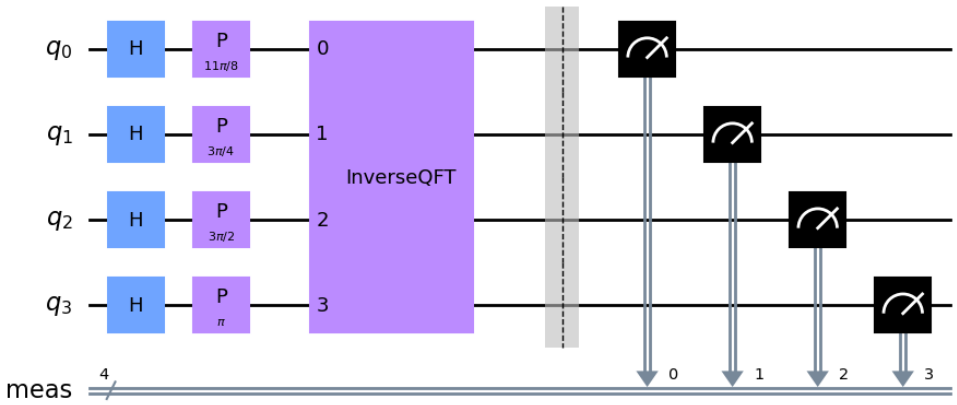
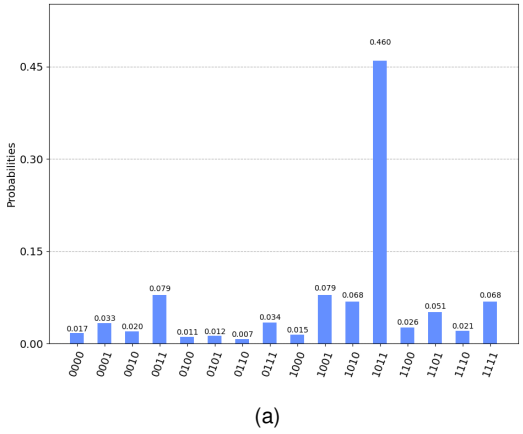
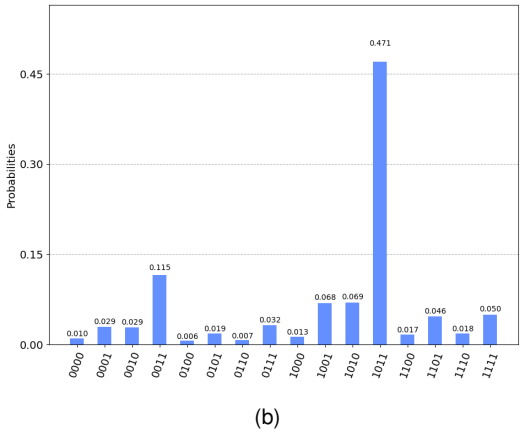


Figure 4.6: Circuit to test the QFT on a real device.

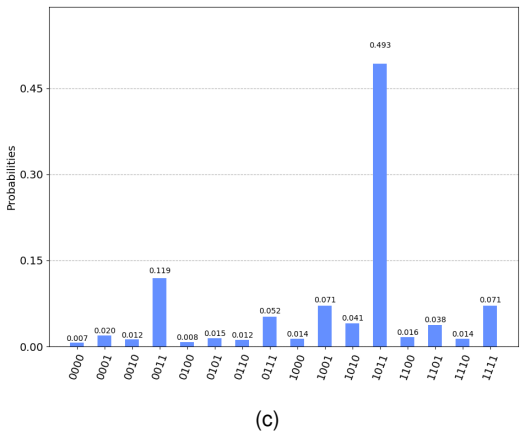
histograms represented in figure 4.7 contain the results obtained from running the circuit 2048 times in three different quantum computers from IBM. These histograms show the probability of measuring each possible value of x. The fact that the probability of measuring 1011 is not 1, evidentiates the importance of running the circuit multiple times.



(a)



(b)



(c)

Figure 4.7: QFT simulated on a real quantum computer.(a) Histogram produced by running the circuit 2048 times on ibm_q_athens; (b) Histogram produced by running the circuit 2048 times on ibm_q_vigo; (c) Histogram produced by running the circuit 2048 times on ibm_q_valencia;

5

Shor's Algorithm

Contents

Shor's Algorithm is an algorithm proposed by Peter Shor which allows the factorization of large prime numbers in polynomial time using a quantum computer [13]. It enables to break RSA based cryptosystems which is widely used to secure data sent via insecure means of communication such as the internet. Shor's algorithm for factoring can be split into two problems: reducing the problem to an order-finding problem, which can be done on a classical computer using the Euclidean Algorithm; and the second by is using a quantum algorithm to solve the order-finding problem. The algorithm is given in [12] as:

1. If N is even, return the factor 2.
2. Determine whether N = p^q for integers p $\geq$ 1 and q $\geq$ 2, and if so return the factor p.
3. Randomly choose x in the range 1 to N-1. If gcd(x, N) $>$ 1 then return the factor gcd(x, N).
4. Use the order-finding subroutine to find the order r of x modulo N.
5. If r is even and $x^{r/2}$ = - 1(mod N) then compute gcd($x^{r/2} - 1$, N) and gcd($x^{r/2}$ + 1, N), and test to see if one of these is a non-trivial factor, returning that factor if so. Otherwise, the algorithm fails.

In this chapter, Shor's Algorithm will be explained and implemented using Qiskit [18]. Various optimizations will be discussed, implemented and lastly compared.

5.1 Shor's Algorithm in classical computation

Most internet protocols rely on asymmetric encryption to establish secure connections which are based on the difficulty of factoring large numbers. An algorithm that solves this factoring problem is Shor's Algorithm, which factors a number by guessing one of it's factors and improving that guess in case it is not. Shor's Algorithm starts by taking a random guess g and checking if it shares a factor with the large number N using the Euclid's Algorithm [26]. If that is the case, the other factor can be obtained by simply dividing N by g. However, since N is the product of two very large "random" prime numbers [27] finding a factor is extremely unlikely [28] and so, the algorithm will instead transform g into a better guess. This new guess comes from the application of Euler's theorem [29] to the relation between g and N: g is not a factor of N they are coprime and through Euler's theorem can be written as,

$$\underset{(coprime)}{g, N} \longrightarrow g^p = m \cdot N + 1 \equiv (g^p/2 + 1)(g^p/2 - 1) = m \cdot N. \tag{5.1}$$

According to equation 5.1, the improved guess is of the form $g^p/2 \pm 1$, and the only thing left to do is determine the value of p. By taking the relation $g^p = m \cdot N + 1$ from 5.1 it's possible to write a function $f(x) = g^x \mod N$. This function is periodic, with period p, and the period is given by the smallest integer larger than zero such that $g^p \mod N = 1$. Finding p is not an easy task as computing all images

of f(x) for a large value of N is extremely costly. At time of writing the recommended key size for RSA encryption is 2048 bits [30] and the biggest size RSA publicly broken is the RSA-250 (829 bits) [28].

5.2 Shor's Algorithm in quantum computing

Shor's Algorithm classical implementation does not pose a threat to modern cryptography, given the immense amount of computational time required to factor a large number. This problem comes from the fact that finding the periodicity of $f(x) = g^x \mod N$ requires computing separately each value of $f(x)$. However, in a quantum system this issue can be overcome thanks to quantum superposition and interference. Quantum superposition makes it possible to write the qubits state as a linear combination of states, allowing the computation of all the images of $f(x) = g^x \mod N$ at the same time. Given that $f(x)$ is periodic, the produced state is composed of equal values separated by the period. When applied the QFT it results in a state composed by the QFT of every image of $f(x)$, that thanks to quantum interference add together resulting in a state that contains the frequency at which each value repeats, $|1/p\rangle$.

5.2.1 Circuit Design

Shor's algorithm isn't purely quantum. As it was aforementioned, it is composed by Euclid's algorithm which classical implementation is already efficient at determining whether or not two numbers share a factor. As such, the quantum circuit will only be required to implement step 4 of the algorithm: "use the order-finding subroutine to find the order r of x modulo N.". Looking to step 4, it uses a random guess $g < N$ and computes $f(x) = g^x \mod N$ to find its period. As described by Peter Shor in [13], the first step of the order-finding algorithm is to find q, a power of 2 with $N^2 \leqslant q < 2N^2$, and put the register x in a uniform superposition state representing all numbers modulo q, which will constitute the domain of $f(x) : x \in [0, q-1]$. To store this information the register containing x, denoted by $|x\rangle$, has to be composed by $log_2(q)$ qubits which, taking into account the possible values of q, will be initialized with a size of $2n$ qubits, where $n = \lfloor log_2(N) \rfloor + 1$.

Subsequently,, it is necessary to put $|x\rangle$ in a uniform superposition of states to represent all numbers x ($\mod q$), this can was done by applying an Hadamard gate to every qubit of $|x\rangle$, as it can be seen in the circuit diagram represented in figure 5.1. Doing this, leaves the machine in the state

$$\frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle |0\rangle . \tag{5.2}$$

Having defined all possible values of x, the next step is to compute $f(x) = g^x \mod N$. By looking at figure 5.1 one can easily deduce that $f(x)$ is computed by the gate $U_{x,N}$. This gate takes the state from

equation 5.2 to

$$\frac{1}{\sqrt{q}}\sum_{x=0}^{q-1}|x\rangle\,|(w\cdot g)^x \mod N\rangle\,, \tag{5.3}$$

By using a Pauli-X gate on the least significant qubit of $|w\rangle$, it states changes from $|0\rangle$ to $|1\rangle$ and equation 5.3 becomes

$$\frac{1}{\sqrt{q}}\sum_{x=0}^{q-1}|x\rangle\,|g^x \mod N\rangle\,. \tag{5.4}$$

Now what is left is to extract the periodicity of $f(x)$ from the current state by applying an inverse QFT to $|x\rangle$. The inverse QFT is given by

$$QFT^\dagger\,|x\rangle = \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}e^{-\frac{2\pi i}{N}xy}\,|y\rangle\,, \tag{5.5}$$

which when applied to the state produced in equation 5.4 creates the state:

$$QFT^\dagger\,|x\rangle\,|g^x \mod N\rangle = \frac{1}{\sqrt{q}}\sum_{x=0}^{q-1}\frac{1}{\sqrt{q}}\sum_{y=0}^{q-1}e^{-\frac{2\pi i}{q}xy}\,|y\rangle\,|g^x \mod N\rangle = \frac{1}{q}\sum_{x=0}^{q-1}\sum_{y=0}^{q-1}e^{-\frac{2\pi i}{q}xy}\,|y\rangle\,|g^x \mod N\rangle\,. \tag{5.6}$$

5.2.2 The $U_{x,N}$ gate

As explained in the previous section, the circuit designed to implement the order-finding subroutine of Shor's Algorithm is composed by an array of Hadamard gates acting on the $|x\rangle$ register, a $U_{x,N}$ gate that applies $f(x) = g^x \mod N$ and the QFT circuit described and implemented in chapter 4 to determine the period of the function. Since both the Hadamard gate and the QFT have already been studied in previous chapters, this section will focus on implementing and described the $U_{x,N}$ gate. This gate's implementation is based on the circuit described by Stéphane Beauregard [31].

The first element of the circuit to built the $U_{x,N}$ gate is an adder gate that takes the QFT of a quantum register $|b\rangle$ and produces the QFT of the sum of that register to a given constant a. This transformation can be described as

$$QFT(|b\rangle) \rightarrow QFT(|a+b\rangle). \tag{5.7}$$

The principle behind equation 5.10 is that adding two QFTs corresponds just to add the phases of each individual qubit. This can be confirmed by taking, for instance, the QFT of $|4\rangle$ and $|7\rangle$ represented

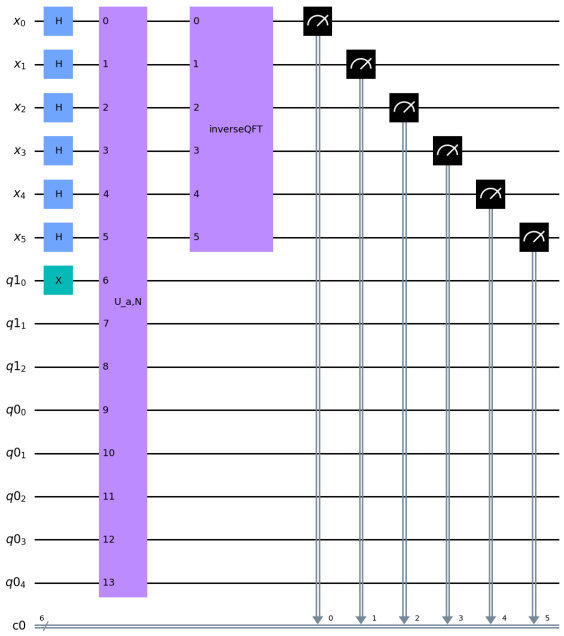


Figure 5.1: Circuit built to implement the order-finding subroutine of Shor's Algorithm for $n = 3$.

as a state vector in figure 5.2 and figure 5.3, respectively, and adding the phases of the qubits. The phases for each qubit can be calculated as demonstrated in equation 4.14, or simply inferred from figures 5.2 and 5.3 and represented as $\alpha_{|S_n\rangle}$, where $|S_n\rangle$ is the n^{th} qubit of state S, resulting in equation 5.8.

$$\begin{aligned} \alpha_{|4_0\rangle} &= \frac{\pi}{2}, \quad \alpha_{|4_1\rangle} = \pi, \quad \alpha_{|4_2\rangle} = 0, \quad \alpha_{|4_3\rangle} = 0, \\ \alpha_{|7_0\rangle} &= \frac{7\pi}{8}, \quad \alpha_{|7_1\rangle} \frac{7\pi}{4}, \quad \alpha_{|7_2\rangle} = \frac{3\pi}{2}, \quad \alpha_{|7_3\rangle} = \pi. \end{aligned} \tag{5.8}$$

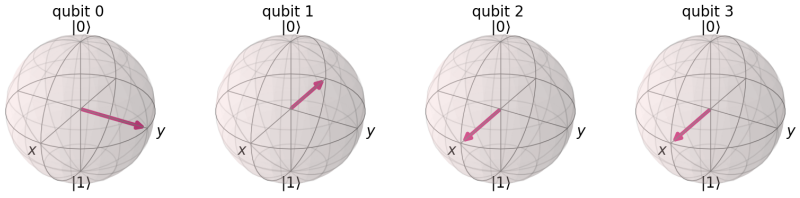


Figure 5.2: Quantum Fourier Transform of $|4\rangle$.

Adding these angles pairwise results in the angles correspondent to QFT($|11\rangle$) as can be concluded by comparing the phases from equation 4.14 with the angles in equation 5.8.

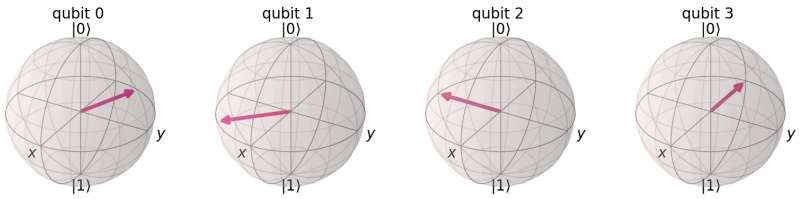


Figure 5.3: Quantum Fourier Transform of $|7\rangle$.

$$\begin{aligned}
\alpha_{|11_0\rangle} &= \frac{\pi}{2} + \frac{7\pi}{8} = \frac{11\pi}{8}, \\
\alpha_{|11_1\rangle} &= \pi + \frac{7\pi}{4} = \frac{3\pi}{4}, \\
\alpha_{|11_2\rangle} &= 0 + \frac{3\pi}{2} = \frac{3\pi}{2}, \\
\alpha_{|11_3\rangle} &= 0 + \pi = \pi.
\end{aligned} \tag{5.9}$$

Given that property, the adder was made by taking QFT($|b\rangle$) and applying to each qubit a phase rotation corresponding to the phases of QFT($|a\rangle$), as represented in figure 5.4. This phases can be classically calculated since a is a given constant and not a quantum register.

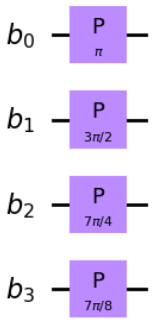


Figure 5.4: Quantum circuit for addition in the Fourier space, with a=7.

To test this adder gate, a circuit was built using: a circuit composed only by Pauli-x gates that forces a desired state to $|b\rangle$; the circuit to perform the QFT of $|b\rangle$; the adder gate that adds a to b; and the circuit for the inverse QFT that puts the resulting state into the basis form allowing for it to be measured. This circuit, represented in figure 5.5 was tested using IBM qasm simulator for the values of a and b used to calculate equation 5.9, respectively, 4 and 7. Figure 5.9 contains the state vector corresponding to the output of the simulation. As expected, the final state corresponds to $|b_3...b_0\rangle = |1011\rangle = |11\rangle_4$. It is also important to note that, since the adder gate is composed only by unitary gates, the inverse of this gate

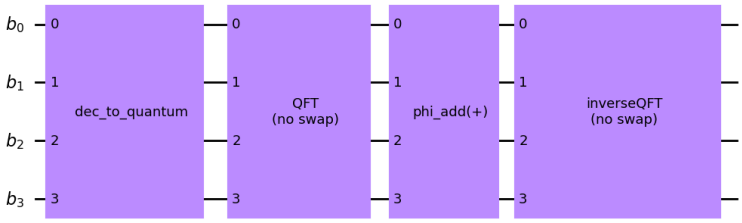


Figure 5.5: Quantum circuit to test the adder gate.

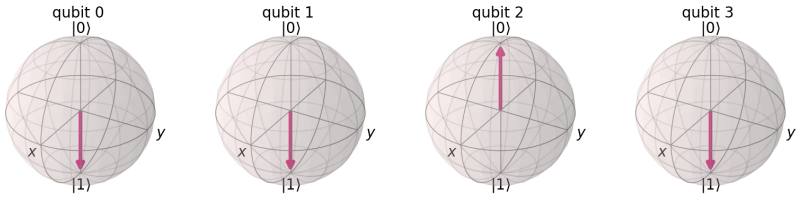


Figure 5.6: State vector produced by adder gate where qubit 3 represent the most significant bit of $|b\rangle$.

can be build. However, unlike when adding a and b, subtracting them might result in a negative number and as such, it maps in equation 5.10.

$$QFT(|b\rangle) \rightarrow \begin{cases} QFT(|b-a\rangle), & b \geqslant a. \\ QFT(|2^{n+1}-a+b\rangle), & b < a. \end{cases} \tag{5.10}$$

A controlled or doubly controlled version of the adder can also be made using, instead of normal rotation, single our doubly controlled phase shift gates, respectively.

Using the implemented adder (figure 5.4) it is possible to build a modular adder. Adding two numbers modulo N when both of them are smaller than N can be easily done by summing them together and, if the result surpasses N, subtracting N from it. The circuit for the modular adder needs an extra bit, set to $|0\rangle$, in order to perform the comparisons. This gate will also be designed with two additional control qubits that will be needed at a later stage of building the $U_{x,N}$ gate. It is also important to notice that, to allow chaining multiple modular adders together, the comparison bit will need to be restored to its initial state. To design this circuit, both the adder gate and its inverse, used for subtraction, are needed. This circuit was designed using $n+4$ qubits; two used as control qubits, $n+1$ to store $a+b$ accounting for overflowing and a clean qubit used to perform comparisons. The full architecture of the circuit implemented to perform modular addition is represented in figure 5.7.

By assuming $|q0_0\rangle\,|q1_0\rangle = |11\rangle$, the circuit in figure 5.7 starts by applying an adder gate to the register containing $QFT(|b\rangle)$ switching the state as defined in equation 5.10. Note that $QFT(|b\rangle)$ is only stored in the first five qubits of register $|b\rangle_6$ as $|b_5\rangle$ is used to do the comparison. After the addition, N is subtracted from the resulting state, originating the state in equation 5.11.

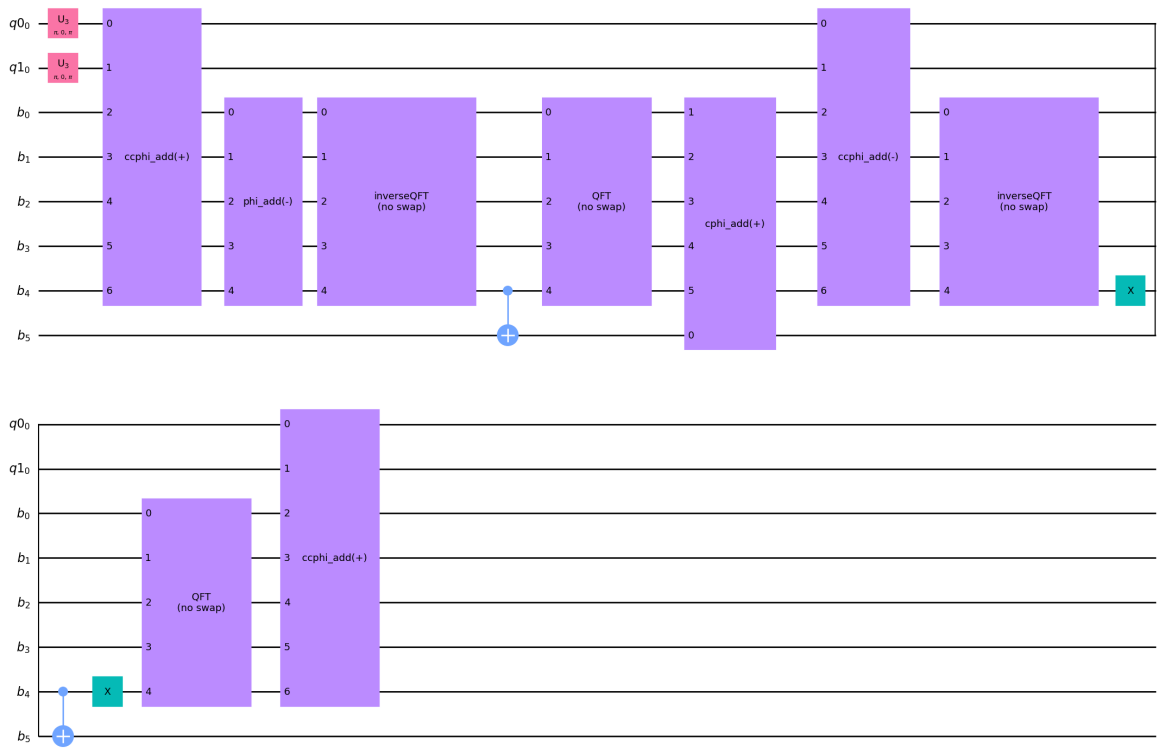


Figure 5.7: Quantum circuit for the modular addition.

$$|b\rangle_5 |b_5\rangle = QFT(|a+b-N\rangle) |0\rangle . \tag{5.11}$$

If $(a+b) \geqslant N$ the state from equation 5.18 corresponds to $|(a+b) \mod N\rangle$. However, if $(a+b) < N$ the state corresponding to $|(a+b) \mod N\rangle$ is simply $|a+b\rangle$ and as such N needs to be added back this state. The way used to compare N with (a+b) was by checking the most significant bit of $|a+b-N\rangle$ to see if it is $|0\rangle$ and so $(a+b) \geqslant N$, or if it is $|1\rangle$ and $(a+b) < N$. Since measuring the qubit would collapse its state, the approach taken was to used it to control a C-NOT targeting $|b_5\rangle$. This way, if $(a+b) < N$, the comparison qubit $|b_5\rangle$ will be flipped to $|1\rangle$ and used to control the adder that sums N back to $a+b-N$, otherwise, the bit is not flipped and N doesn't need to be added back. Since the register contains $QFT(|a+b-N\rangle)$ the inverse QFT needs to be applied first in order to use it as a control qubit. Equation 5.12 summarizes the behaviour of the circuit for the two possible outcomes.

$$\begin{cases} |b_5\rangle = 1 & \text{if } (a+b) < N \xrightarrow{\text{N is added back}} |b\rangle_5 |b_5\rangle = QFT(|a+b\rangle) |1\rangle = QFT(|(a+b) \mod N\rangle) |1\rangle , \\ |b_5\rangle = 0 & \text{if } (a+b) \geqslant N \xrightarrow{\text{N is not added back}} |b\rangle_5 |b_5\rangle = QFT(|a+b-N\rangle) |0\rangle = QFT(|(a+b) \mod N\rangle) |0\rangle . \end{cases} \tag{5.12}$$

After computing $|(a+b) \mod N\rangle$ all that's left is to restore $|b_5\rangle$ to $|0\rangle$, which is only needed when

the state is $QFT(|a+b\rangle)\,|1\rangle$. By subtracting a from the states described in equation 5.12 equation 5.13 is created. This equation shows that, by doing so, one is left with either b or $b-N$, a positive and a negative number, respectively. As such, the signal qubit can again be used to control the flipping of the comparison qubit.

$$\begin{cases} |b\rangle_5 = a+b-a = b & \text{if } |b_5\rangle = 1, \\ |b\rangle_5 = a+b-N-a = b-N & \text{if } |b_5\rangle = 0. \end{cases} \tag{5.13}$$

This time, however, $|b_5\rangle$ needs to be flipped when $|b\rangle_5 = b$ - which is a positive number -, so before and after applying the C-NOT gate, a Pauli-X gate is applied to $|b_4\rangle$. This ensures that, when the signal qubit is $|1\rangle$, and therefore the number is negative, $|b_4\rangle$ is not flipped, and when the signal is $|0\rangle$ it is flipped. After that. a is added back, producing the final state described in equation 5.14.

$$|b\rangle_5\,|b_5\rangle = QFT(|(a+b) \mod N\rangle)\,|0\rangle\,. \tag{5.14}$$

For the test of the modular adder, a circuit similar to the one used for testing the adder gate was used, as expressed in figure 5.8. This gate was simulated, in the same quantum simulator as the adder gate, for $N=15$, $b=4$ and $a=16$ which is supposed to produce $(4+16) \mod 15 = 5$. This means that for the input $|q_0\rangle\,|q_1\rangle\,|b\rangle = |1\rangle\,|1\rangle\,|000100\rangle$ the circuit is expected to produce $|q_0\rangle\,|q_1\rangle\,|b\rangle = |1\rangle\,|1\rangle\,|000101\rangle$. Figure 5.9 shows that the resulting state corresponds to the expected result.

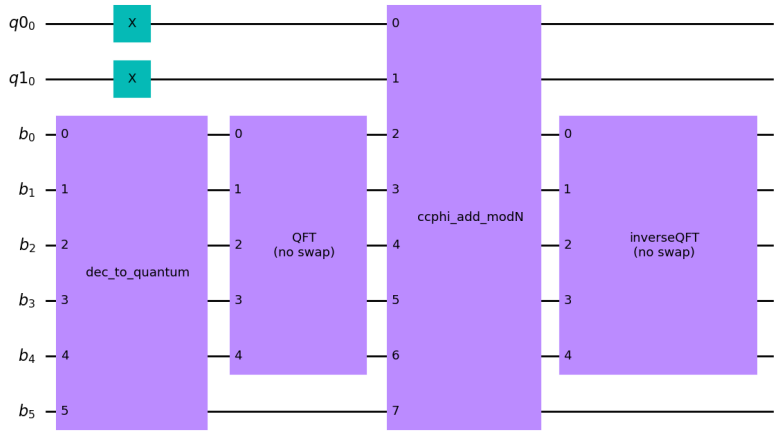


Figure 5.8: Quantum circuit to test the modular adder.

Every number can be written in the form of a sum of coefficients multiplied by the corresponding base power, for instance, $236_{10} = 2\times10^2+3\times10^1+6\times10^0$. A multiplication of two numbers can be easily done by multiplying each of those sums by the other number used in the multiplication (e.g.$236_{10} \times 14_{10} = 14\times2\times10^2 + 14\times3\times10^1 + 14\times6\times10^0$). As such, the multiplication of an arbitrary number x by an arbitrary constant a can be defined by,

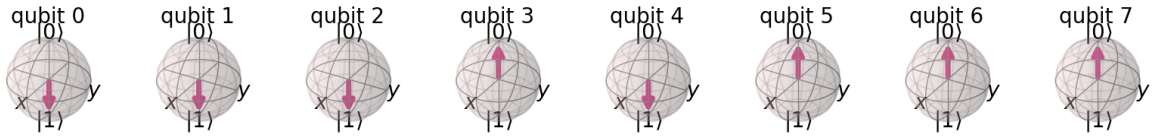


Figure 5.9: State vector produced by modular adder gate, where qubit 0 $\rightarrow |q_0\rangle$, qubit 1 $\rightarrow |q_1\rangle$ and qubit 2:7 $\rightarrow |b\rangle$ with qubit 7 representing the most significant bit of $|b\rangle$.

$$x \cdot a = 2^{n-1}ax_{n-1} + 2^{n-2}ax_{n-2} + \cdots + 2^0 ax_0, \tag{5.15}$$

where $x = x_{n-1}x_{n-2}\cdots x_0 = 2^{n-1}x_{n-1} + 2^{n-2}x_{n-2} + \cdots + 2^0 x_0$.

Given the relation described in equation 5.15, it is possible to construct a modular multiplication gate using the previously described modular adder gate. Taking a closer look at that referred equation it is noticeable that, for $j \in [0, n-1]$, every time $x_j = 0 \rightarrow 2^j a x_j = 0$ and so that term of the addition is null meaning that, one can instead use x_j to control whether or not the term $2^j a$ is added into the equation. Taking this into account, the modular multiplier was built using n modular adder gates in series and each controlled by a different qubit of $|x\rangle$. This way, by classically calculating $2^j a$, each gate produces $QFT(\left|(b + 2^j a x_j) \mod N\right\rangle)$, where j is the order of the qubit of $|x\rangle$ used as control. Drawing this circuit using Qiskit produces the diagram in figure 5.10.

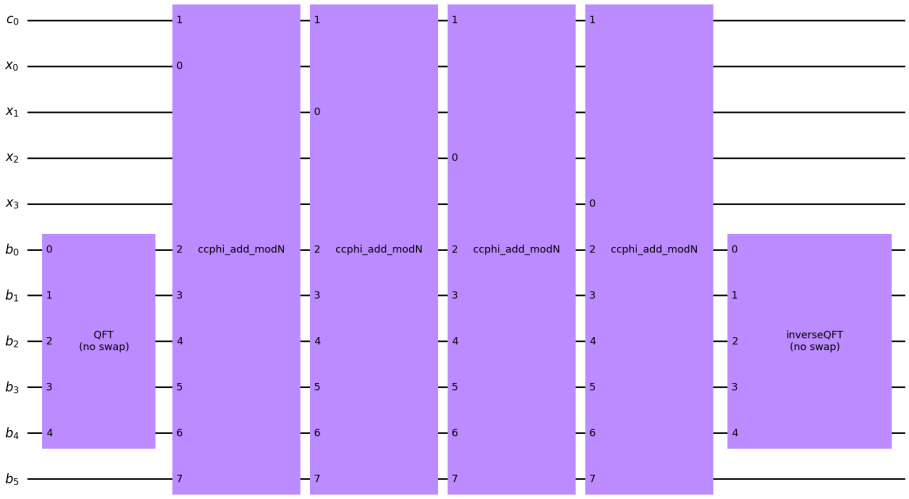


Figure 5.10: Quantum circuit for the modular multiplier.

To analyze the behaviour of the modular multiplier circuit it is important to note that, one adder gate maps

$$|x_j\rangle\,|b\rangle \rightarrow QFT(\left|(b + 2^j a x_j) \mod N\right\rangle), \tag{5.16}$$

and, by connecting them in series, the output of one is the input of the next, making the final state for the circuit of figure 5.10

$$|x\rangle |b\rangle = QFT(\big|(((((b + 2^0 a x_0) \mod N) + 2^1 a x_1 \mod N) + 2^2 a x_2 \mod N) + 2^3 a x_3 \mod N)\big\rangle). \tag{5.17}$$

Furthermore, since applying the modulus after every addition or simply at the end of all produces the exact same result, the expression from 5.18 can be written as

$$|x\rangle |b\rangle = QFT(\big|((b + 2^0 a x_0 + 2^1 a x_1 + 2^2 a x_2 + 2^3 a x_3) \mod N)\big\rangle) = QFT(|(b + a \cdot x) mod N\rangle). \tag{5.18}$$

To test the operation of the modular multiplier, the circuit in figure 5.11 was used. The simulation was ran in a quantum simulator for $N = 15$, $b = 6$, $a = 2$ and $x = 5$ and the ensure the gate operation the control qubit $|c\rangle_1$ is set to $|1\rangle_1$. The result from this simulation is the state vector represented in figure 5.12.

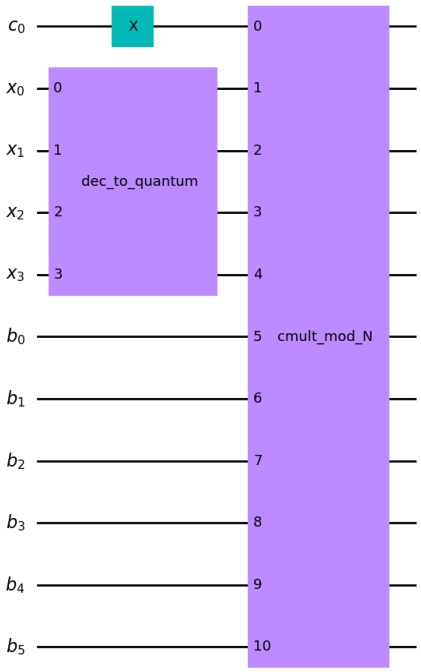


Figure 5.11: Quantum circuit to test the modular multiplier.

The correct behaviour of the gate can be confirmed by comparing the state vector from figure 5.12 with the expect result $|c\rangle_1 |x\rangle_4 |b\rangle_6 \rightarrow |1\rangle_1 |5\rangle_4 |(6 + 2 \cdot 5) \mod 15\rangle_6 = |1\rangle_1 |5\rangle_4 |1\rangle_6$. Unfortunately, this

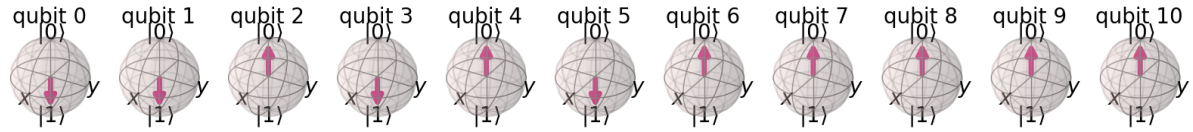


Figure 5.12: State vector produced by modular multiplier where, qubit 0 $\rightarrow |c_0\rangle$, qubit 1:4 $\rightarrow |x\rangle$ and qubit 5:10 $\rightarrow |b\rangle$ with qubits 4 and 10 representing the most significant bit of $|x\rangle$ and $|b\rangle$ respectively.

gate is a modular multiplier so, instead of mapping $|x\rangle|b\rangle \rightarrow |x\rangle|b + a \cdot x \mod N\rangle$ it needs to map $|x\rangle \rightarrow |a \cdot x \mod N\rangle$. This can easily be achieved by having $b = 0$ however, the result of the computation is stored on the register $|b\rangle$. Removing $|b\rangle$ from the equation can be done by treating it as an clean ancilla register -a register with a known state used only to provide qubits to calculation whose initial state must be preserved-. The computed value of $|a \cdot x \mod N\rangle$ can easily be stored in $|x\rangle$ by swapping the values from $|x\rangle$ and $|b\rangle$ using swap gates, but this leaves $|b\rangle$ storing the value of x. This can be solved by applying the inverse of the modular multiplier gate to the circuit as is shown in equation 5.19. Doing this creates a circuit that performs the modular multiplication of its input x by a constant a and stores the result in the input register, the U_a gate. This steps were implemented using Qiskit and the resulting circuit is the diagram in figure 5.13.

$$\begin{aligned}|x\rangle|0\rangle \xrightarrow{multiplier} |x\rangle|(ax) \mod N\rangle \xrightarrow{swap} |(ax) \mod N\rangle|x\rangle \\ |(ax) \mod N\rangle|x\rangle \xrightarrow{inv\_multiplier} |(ax) \mod N\rangle\left|(x - a^{-1}ax) \mod N\right\rangle \\ |(ax) \mod N\rangle\left|(x - a^{-1}ax) \mod N\right\rangle = |(ax) \mod N\rangle|0\rangle \end{aligned} \quad (5.19)$$

Similarly to the other gates, the U_a gate was tested in a quantum simulator using the circuit in figure 5.14. The state vector produced for $N = 15$, $b = 0$, $a = 4$ and $x = 8$ is represented in figure 5.15.

The correct operation of the gate can be confirmed by comparing the expected result $|(ax) \mod N\rangle = (4 \times 8) \mod 15 = 2$ with the state vector from figure 5.15.

However, the function applied by the $U_{x,N}$ gate is not modular multiplication but modular exponentiation, in the form of $f(x) = g^x \mod N$. This can be accomplished using the same controlling approach as for the modular multiplier, since:

$$\begin{aligned}|g^x \mod N\rangle = \left|g^{2^0x_0+2^1x_1+\cdots+2^{n-1}x_{n-1}} \mod N\right\rangle = \\ = \left|((g^{2^0x_0} \mod N)g^{2^1x_1} \mod N)\cdots g^{2^{n-1}x_{n-1}} \mod N)\right\rangle. \end{aligned} \quad (5.20)$$

Being g a classical constant, g^{2^j} can be computed classically and as such, the $U_{x,N}$ gate can be

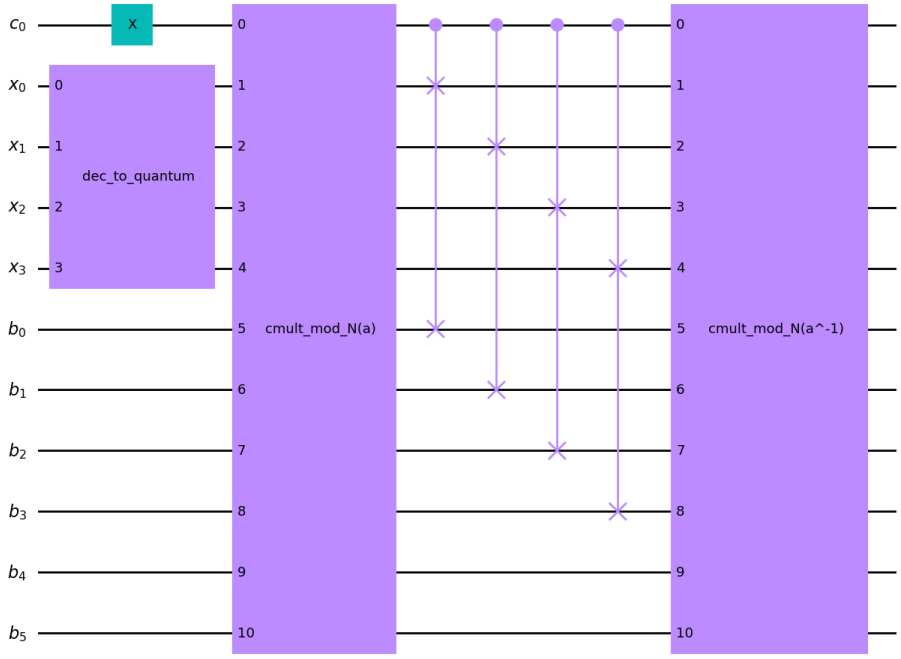


Figure 5.13: Quantum circuit for the modular multiplier gate acting on the input register.

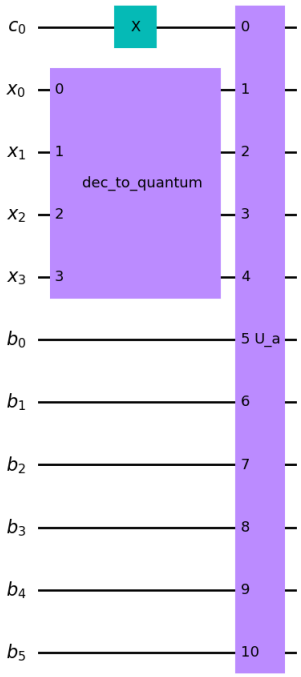


Figure 5.14: Quantum circuit to test the modular multiplier gate acting on the input register.

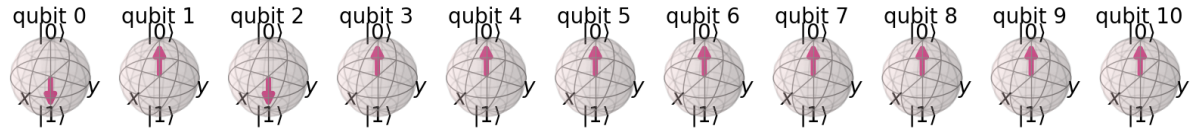


Figure 5.15: State vector produced by the modular multiplier gate acting on the input register where, qubit 0 $\rightarrow |c_0\rangle$, qubit 1:4 $\rightarrow |x\rangle$ and qubit 5:10 $\rightarrow |b\rangle$ with qubits 4 and 10 representing the most significant bit of $|x\rangle$ and $|b\rangle$ respectively.

implemented by applying the U_a gates acting on the input register $|w\rangle$, and controlled by $|x\rangle$ making the circuit represented in figure 5.16.

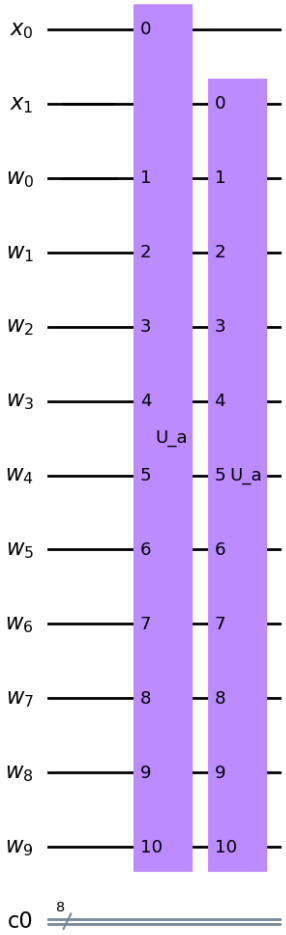


Figure 5.16: Quantum circuit for the $U_{x,N}$ gate

The behaviour of the $U_{x,N}$ can be tested using the circuit in figure 5.17. The $U_{x,N}$ computes $f(x) = (w \cdot g)^x \mod N$ instead of $f(x) = g^x \mod N$. This is overcome by setting $|w\rangle$ to $|1\rangle$ before applying the

gate, hence the Pauli-X gate in the circuit used for simulation. Running the circuit in a quantum simulator for $N = 15$, $x = 4$ and $g = 7$, produced the state vector in in figure 5.18 which, when compared with the expected value $\left|7^4 \mod 15\right\rangle = 1$ leads to the conclusion that the gate is working properly.

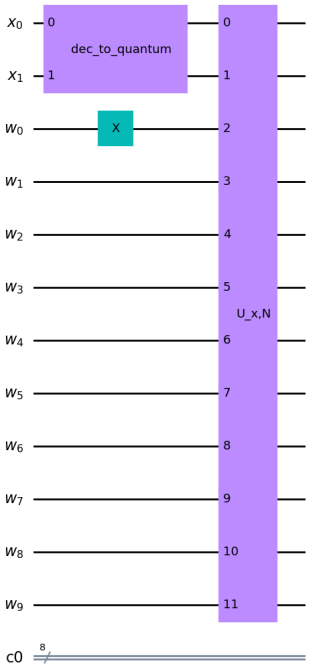


Figure 5.17: Quantum circuit to test the $U_{x,N}$ gate

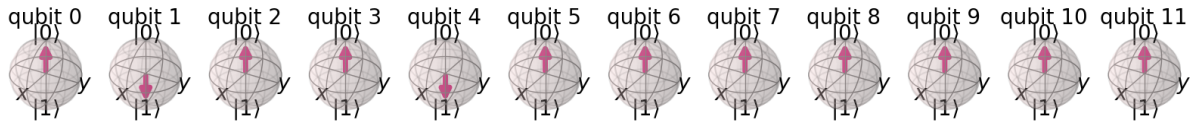


Figure 5.18: State vector produced by the $U_{x,N}$ gate where, qubit 0 $\rightarrow |c_0\rangle$, qubit 1:4 $\rightarrow |x\rangle$ and qubit 5:10 $\rightarrow |b\rangle$ with qubits 4 and 10 representing the most significant bit of $|x\rangle$ and $|b\rangle$ respectively.

Concluded the construction of the $U_{x,N}$ all the requirements to build the order-finding subroutine circuit are met and the circuit from figure 5.1 can be built and tested.

5.2.3 Circuit Implementation

To implement the circuit one needs to first choose the values of N and g. Since Shor's algorithm main purpose is to factor numbers which are a product of two prime numbers, N equal to fifteen is the smallest odd number that is also the product of two different prime numbers - three and five. Any other number meeting the same criteria could have been chosen, but fifteen being the smallest uses less qubits and

as such requires smaller and faster circuits. Additionally, the equations describing the circuit will also be smaller, providing a better reading. For g, any number smaller than fifteen and bigger than one is good, and as such, two was chosen. Having N been chosen, q can be determined by finding the power of two such that $225 \leqslant q < 450$, meaning $q = 256 = 2^8$ and so $n = 4$. Since the circuit uses $4n + 2$ qubits, the total amount of qubits required to build the circuit sums up to eighteen. Unfortunately, this conclusion means that the circuit cannot be tested on a real quantum computer, as as the maximum number of available qubits from IBM Quantum Experience is five. However, IBM also provides a set of advanced cloud-based quantum simulators with up to five thousand qubits which will be used instead to make the essential test.

Figure 5.19 illustrates the architecture of the circuit implemented to compute the order-finding subroutine of Shor's algorithm for $N = 15$. In this figure there are also four vertical lines representing different moments in time (t_0, t_1, t_2 and t_3), which are used to represent the state of the circuit in those given moments. For instance, the state $|t_0\rangle$ corresponds to the state $|x\rangle\,|w\rangle$ in the moment t_0, the initial state of the circuit.

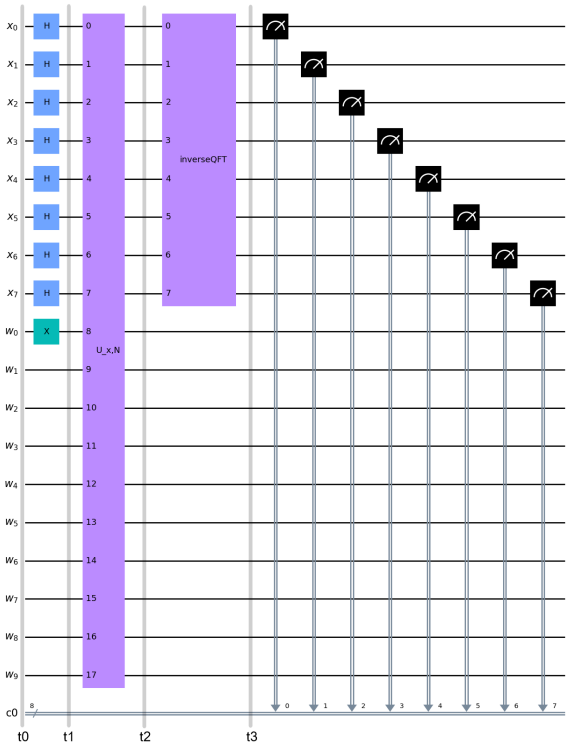


Figure 5.19: Circuit to implement order-finding subroutine of Shor's Algorithm for N=15.

It is also important to notice that, although the circuit requires $4n + 2$ qubits, the equations describing the behaviour of the circuit will only use $2n + 2$ qubits. As explained by Gidney [32], the maximum period of a number N with n bits is given by 2^n and only one sample is required to determine the periodicity

using continued fractions. Yet, for huge periods, the fractions start to get close together urging the need of having a frequency space large enough to distinguish those samples. This space can be achieved using an input register of $2 * log_2(P)$ qubits, where P is the maximum period of N, making it necessary to represent $|x\rangle$ using $2 * log_2(2^n) = 2n$ qubits, as referred in the algorithm description. However, in this case, since the number to factor is small, no overlapping of frequencies exist and as such, to provide a better readability, the register will be of n qubits and the states will be represented in decimal with the size of the register in subscript, (e.g. state $|0101\rangle$ is represented by $|7\rangle_4$).

As all qubits are initialized in the state $|0\rangle$, the initial state is given by

$$|t_0\rangle = |x\rangle_4 |w\rangle_6 = |0\rangle_4 |0\rangle_6 . \tag{5.21}$$

The first step of the order-finding is to apply an Hadamard gate to every single qubit of the first register, which causes the state to shift to an uniform superposition. At the same time, a Pauli-X gate is applied to the least significant qubit of the second register causing it to go from $|0\rangle$ to $|1\rangle$. Given the action of these gates, the state goes from equation 5.21 to

$$|t_1\rangle = \frac{1}{\sqrt{16}} \sum_{x=0}^{15} |x\rangle_4 |1\rangle_6 \equiv \frac{1}{4}\Big[|0\rangle_4 + |1\rangle_4 + |2\rangle_4 + \cdots + |15\rangle_4 \Big] |1\rangle_6 . \tag{5.22}$$

From here the gate $U_{x,N}$ is applied to equation 5.22 taking the first register as an input and producing in the second register the value of the function $f(x) = 2^x \mod 15$ leading to the state:

$$|t_2\rangle = \frac{1}{4}\Big[|0\rangle_4 \big|2^0 \mod 15\big\rangle_6 + |1\rangle_4 \big|2^1 \mod 15\big\rangle_6 + \cdots + |15\rangle_4 \big|2^{15} \mod 15\big\rangle_6 \Big], \tag{5.23}$$

which can be developed to make

$$\begin{aligned} |t_2\rangle = \frac{1}{4}\Big[& |00\rangle_4 |1\rangle_6 + |01\rangle_4 |2\rangle_6 + |02\rangle_4 |4\rangle_6 + |03\rangle_4 |8\rangle_6 + \\ & |04\rangle_4 |1\rangle_6 + |05\rangle_4 |2\rangle_6 + |06\rangle_4 |4\rangle_6 + |07\rangle_4 |8\rangle_6 + \\ & |08\rangle_4 |1\rangle_6 + |09\rangle_4 |2\rangle_6 + |10\rangle_4 |4\rangle_6 + |11\rangle_4 |8\rangle_6 + \\ & |12\rangle_4 |1\rangle_6 + |13\rangle_4 |2\rangle_6 + |14\rangle_4 |4\rangle_6 + |15\rangle_4 |8\rangle_6 \Big]. \end{aligned} \tag{5.24}$$

Just by looking at the state from equation 5.24, one can immediately see the periodicity of $f(x)$. Nevertheless, applying the inverse QFT is still needed to extract this information from the state. However, considering what it means for a state to be in an uniform superposition the state $|t_2\rangle$ can be simplified by assuming the second register was measured. Taking an arbitrary basis state $|0\rangle$ and putting it into uniform superposition creates the state:

$$|x\rangle_1 = \frac{1}{\sqrt{2}}\Big[\,|0\rangle + |1\rangle\,\Big], \tag{5.25}$$

that when measured, gives either $|0\rangle$ or $|1\rangle$ with equal probability: measuring collapses the superposition state and as such, by measuring the the second register of t_2 the state collapses, resulting into a simpler state that will facilitate the calculation of the QFT without impacting the outcome. By analyzing equation 5.24, it is concluded that the second register of t_2, $|w\rangle_6$, can take the values $|1\rangle_6$, $|2\rangle_6$, $|4\rangle_6$ and $|8\rangle_6$ with equal probability. By assuming the measurement result was $|4\rangle$ the state becomes

$$|t_2\rangle = \frac{1}{2}\Big[\,|2\rangle_4|4\rangle_6 + |6\rangle_4|4\rangle_6 + |10\rangle_4|4\rangle_6 + |14\rangle_4|4\rangle_6\,\Big] \equiv \frac{1}{2}\Big[\,|2\rangle_4 + |6\rangle_4 + |10\rangle_4 + |14\rangle_4\,\Big]|4\rangle_6\,. \tag{5.26}$$

To extract the periodicity of the function, the inverse QFT,

$$QFT^\dagger|\tilde{x}\rangle = |x\rangle = \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1} e^{-\frac{2\pi i}{N}xy}|y\rangle\,, \tag{5.27}$$

is applied to the first register transforming the state $|t_2\rangle$ in:

$$|t_3\rangle = QFT^\dagger(|x\rangle_4)\,|4\rangle_6 \equiv \frac{1}{2}\Big[QFT^\dagger|2\rangle_4 + QFT^\dagger|6\rangle_4 + QFT^\dagger|10\rangle_4 + QFT^\dagger|14\rangle_4\,\Big]|4\rangle_6\,. \tag{5.28}$$

By substituting the values of $|x\rangle$ in equation 5.27, every inverse QFT from equation 5.28 can be written as:

$$\begin{aligned}
QFT^\dagger|2\rangle &= \frac{1}{4}\sum_{y=0}^{15} e^{-\frac{\pi i}{8}2y}|y\rangle \\
QFT^\dagger|6\rangle &= \frac{1}{4}\sum_{y=0}^{15} e^{-\frac{\pi i}{8}6y}|y\rangle \\
QFT^\dagger|10\rangle &= \frac{1}{4}\sum_{y=0}^{15} e^{-\frac{\pi i}{8}10y}|y\rangle \\
QFT^\dagger|14\rangle &= \frac{1}{4}\sum_{y=0}^{15} e^{-\frac{\pi i}{8}14y}|y\rangle\,.
\end{aligned} \tag{5.29}$$

These equations applied to equation 5.28, provide the inverse Fourier transform of the first register as:

$$QFT^{\dagger}\left|x\right\rangle = \frac{1}{8}\sum_{y=0}^{15}\left[e^{-\frac{\pi i}{8}2y} + e^{-\frac{\pi i}{8}6y} + e^{-\frac{\pi i}{8}10y} + e^{-\frac{\pi i}{8}14y}\right]\left|y\right\rangle_4 . \tag{5.30}$$

Solving these equations is not trivial, it was done in this Thesis by software programmed in python, producing the results presented on figure 5.20, that when applied to equation 5.30 leads to the final state:

$$\left|t_3\right\rangle = \frac{1}{8}\Big[4\left|0\right\rangle_4 + (-4)\left|4\right\rangle_4 + 4\left|8\right\rangle_4 + (-4)\left|12\right\rangle_4\Big]. \tag{5.31}$$

```
inverse QFT |X>= 1/8[
(4+0j)|0>
0|1>
0|2>
0|3>
(-4-1.959434878635765e-15j)|4>
0|5>
0|6>
0|7>
(4+3.918869757271153e-15j)|8>
0|9>
0|10>
0|11>
(-4-5.878304635907295e-15j)|12>
0|13>
0|14>
0|15>

Process finished with exit code 0
```

Figure 5.20: Solution of equation 5.30.

When measuring the first register of $\left|t_3\right\rangle$, the value has an equal probability of being $\left|0\right\rangle_4$, $\left|4\right\rangle_4$, $\left|8\right\rangle_4$ or $\left|12\right\rangle_4$ and so the histogram from figure 5.21: as explained in [32], through continued fractions, will result in a period equal to the number of peaks (in this case four).

This implementation of the order-finding subroutine was validated by comparing the result produced by the circuit with the calculated period. Figure 5.22 shows the histogram built from 2048 measurements of the first register from figure 5.19 and, as expected, the number of peaks is four. Additionally, the distance from the peaks of the histogram show that the simplification of using an $\left|x\right\rangle$ register of size four for the calculations had no impact on the final result.

5.2.4 Semi-classical implementation

Shor's algorithm is indeed relevant to the world of quantum computation, being directly tied with cyber security. However, to factorize big numbers (RSA-2048), there are still a great deal of advances to be made in the quantum computation area, which can be achieved by either building bigger systems with many more qubits or optimizing the algorithms to require less qubits. Fortunately, both are being

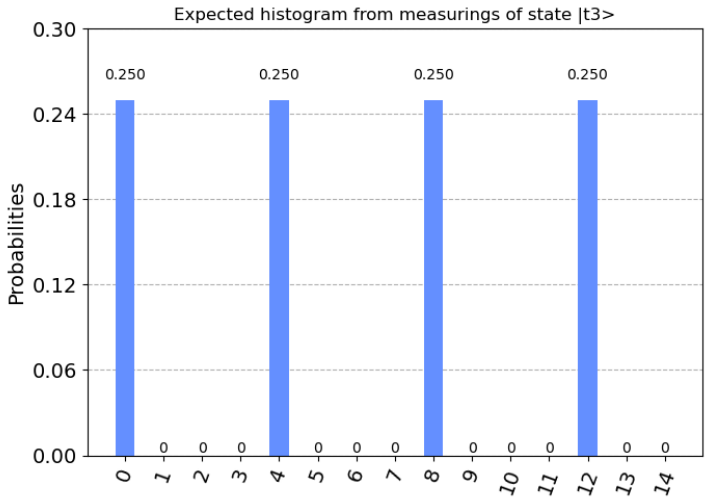


Figure 5.21: Expected histogram from measuring state $|t_3\rangle$.

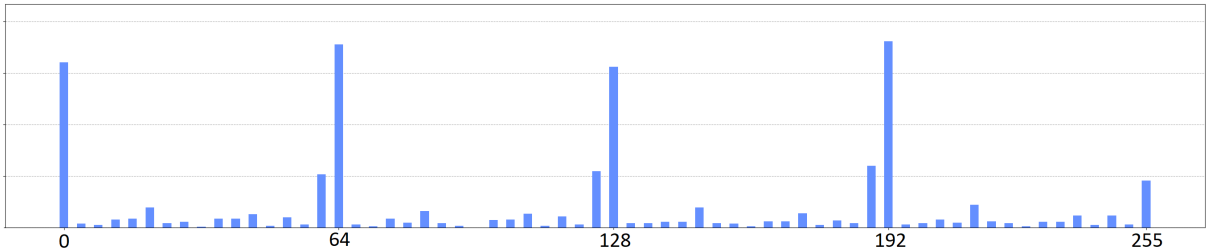


Figure 5.22: Histogram build from 2048 measurements of the circuit from figure 5.19 for N=15 and g=2 (measurements are summed in groups of 4 to fit the page.)

pursued, such as the semi-classical implementation of the order-finding subroutine of Shor's algorithm by S.Parker and M.B.Plenio [33]. This implementation proves that periodicity of $f(x) = g^x \mod N$ can be extracted using a single qubit inverse QFT.

Taking a look at the circuit in figure 4.5, it's noticeable that the inverse-QFT is applied to the target register by acting on each qubit sequentially using conditional rotations based on values from previous qubits. It can be inferred that if the values of the previous qubits were known before starting to apply the gates to a qubit, those controlled rotations could be replaced by single qubit rotations. This is exactly the principle behind the implementation of a semi-classical sequential QFT.

The circuit in figure 5.1 applies the order finding subroutine or Shor's Algorithm. The complex gates of this circuit, namely the $U_{x,N}$ and the QFT, can be represented by their components, making the circuit in figure 5.23. Considering an inverse QFT using "uncontrolled" phase gates, the qubits of register $|x\rangle$ in figure 5.23 would be independent of each other, meaning that all the gates acting on one qubit could be applied without waiting for gates to be applied to the others, allowing the order-finding sub routine to be built using only one bit to store the value of x. This circuit was implemented by analysing which gates are applied to each qubit of $|x\rangle$ and applying them sequentially to it, measuring the result before

passing on to the next qubit. This raises a problem, the swap gate applied at the beginning of the inverse QFT forces an interaction between qubits of $|x\rangle$. Since what that gate does is switching the value of two states before applying them the inverse QFT gates, the problem can be solved by simply applying the gates that would target the qubit after swapping, to it directly (e.g. instead of swapping x_2 and x_0 and applying and Hadamard gate to x_0, that can would just be applied to x_2). This way, the gates should be applied from MSB to LSB. After establishing the order at which to apply the gates, it is possible to build the circuit.

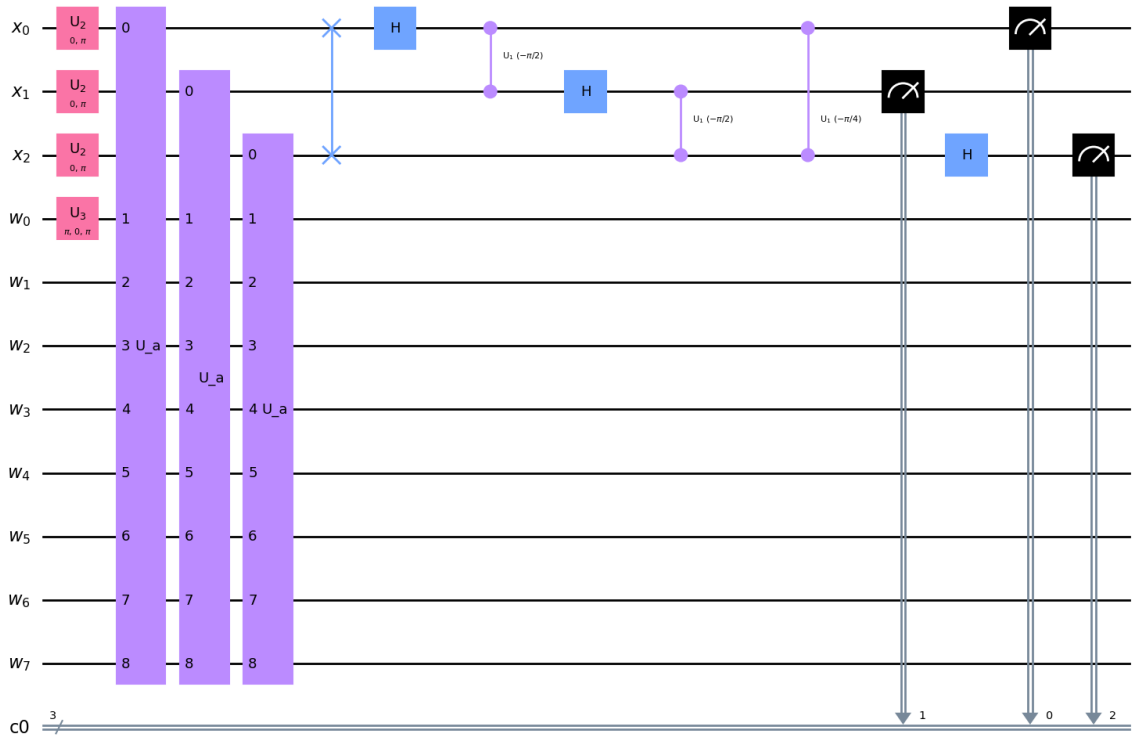


Figure 5.23: Quantum circuit for order-finding subroutine with $n = 3$.

Using only one qubit for $|x\rangle$ makes it impossible to parallelize the circuit, this causes the depth of the circuit to be equal to the number of used gates, increased by the additional Pauli-X and two measurement gates required per removed qubit of $|x\rangle$. The circuit diagram in figure 5.24 corresponds to the implementation of the semi-classical order-finding subroutine of Shor's Algorithm for $n = 2$. Although it makes no sense factoring a 2 qubit number, building the circuit for a larger number would make it impossible to represent. This circuit was build using Qiskit. As usual it starts by defining the size of the registers, which, for the semi-classical implementation is 1 for $|x\rangle$ and $2n + 2$ for $|w\rangle$. Since this circuit requires the use of measurements in the middle of the circuit to control other gates, two classical register were created to store these values, one of size 1, to store the value of the qubit used to control

the conditional Pauli-X gate ,and another of size 2^n, to store the measurements corresponding to all possible values of x. After defining the size of the register, the gates were applied sequentially starting with the MSB: an Hadamard gate, the U_a gate, the classically controlled Phase gates and Hadamard gate corresponding to the inverse QFT, the two measurements and a Pauli-X gate that resets $|x\rangle$ to $|0\rangle$ in case $|1\rangle$ was measured.

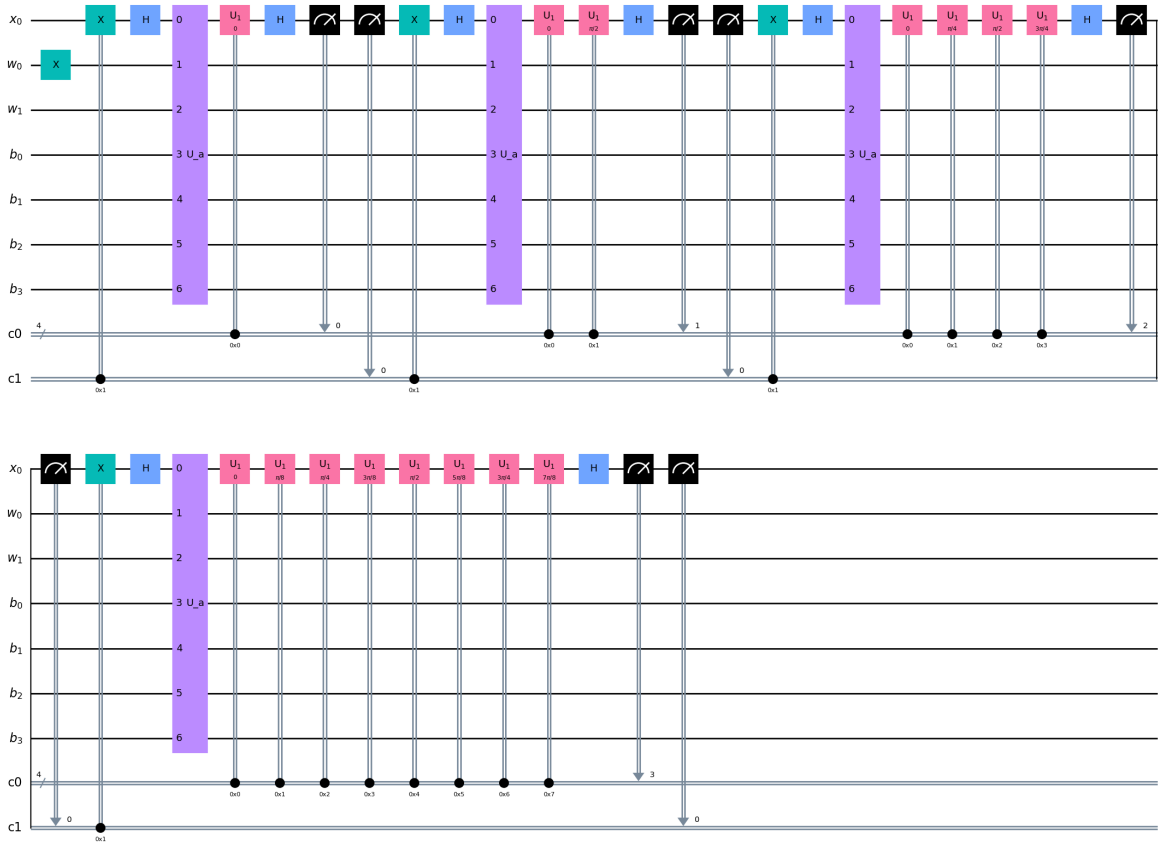


Figure 5.24: Quantum circuit for the semi-classical order finding subroutine for $n = 2$.

To test this implementation, the same code was used to build a circuit to solve the order-finding subroutine for $N = 15$ and $g = 2$ and ran in the same quantum simulator. This was done to ensure the same conditions were used to test both implementations (quantum and semi-classical) and that, as such, both would produce the same results. In figure 5.25 is the histogram corresponding to the measurements resulting from running the semi-classical implementation 2048 times. As expected, the number of peaks from the histogram is four, just like on figure 5.22, meaning that both implementations produce the same results.

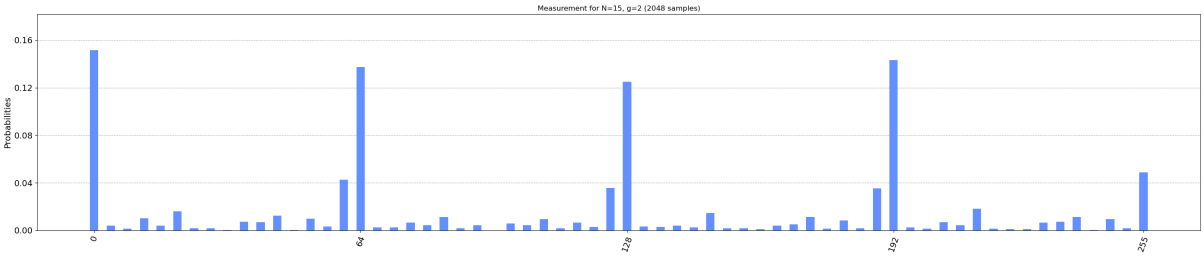


Figure 5.25: Histogram build from 2048 measurements of the circuit for the semi-classical implementation with N=15 and g=2 (measurements are summed in groups of 4 to fit the page).

6

Conclusions

Contents

6.1 Achievements

The main goal of this thesis was to delve into quantum computing. To achieve that, an extensive study of the basic principles of quantum computation was made along with a detailed analysis of both the quantum Fourier transform and the Shor's algorithm. To provide a better insight on the algorithms' implementation, a step by step analysis was conducted, always making parallelism between the mathematical equations and the corresponding quantum gates. This Thesis also featured a semi-classical implementation of Shor's algorithm that, by incorporating classical computing in the circuit design process, allowing the number of required qubits to be reduced from $4n + 2$ to $2n + 3$, roughly half the amount. Doing this comes with the cost of significantly increasing the quantum volume needed to run the quantum algorithm, which, just like the number of available qubits, is one of the bottlenecks for the implementation of quantum circuits.

Running the circuits on IBM's quantum computers raised some problems. The qubit range of the available computers was small which forced the simulations to be ran on quantum simulators instead of real devices. Given of how easy is to use IBM quantum computers and that they are built on a global network, the amount of users at the same time can be very high making running a simple circuit, which can be done in a split second, a task that can take hours. Even so, IBM's Qiskit a is very practical tool to use and provides graphical representation of circuits and simulation results facilitating their analysis and test. It works like a descriptive programming language and requires very basic Python knowledge to use, and, even with a limited number of available qubits for simulation, allows the development and implementation of quantum algorithms.

As aforementioned, the code for this project was developed in Python programming language. It was decided that it would be provided for free access and use for future investigation on quantum computing. The code for all implementations can be found at github.com/goncalobvalentim/fromalgtoimp.

6.2 Future Work

This Thesis serves as a introduction to quantum computing providing the necessary basic knowledge to pursue further studies in this area. While the QFT and Shor's algorithm were chosen to be implemented, there are others algorithms that can use this implementations in their circuit, just like Shor's uses QFT. This thesis also showed that optimizing the number of qubits necessary to implement a circuit comes with the cost of increasing its depth. Another good example of this, is the $2n + 1$ qubit implementation of Shor's algorithm by Craig Gidney [34], which detailed analysis and implementation using Qiskit could enrich the set of quantum algorithms implemented with Qiskit.

Bibliography

[1] L. Sousa, ”Nonconventional Computer Arithmetic Circuits, Systems and Applications,” in IEEE Circuits and Systems Magazine, vol. 21, no. 1, pp. 6-40, Firstquarter 2021

[2] Neill, Charles, et al. ”A blueprint for demonstrating quantum supremacy with superconducting qubits.” Science 360.6385 (2018): 195-199.

[3] Ebadi, S., Wang, T.T., Levine, H. et al. Quantum phases of matter on a 256-atom programmable quantum simulator. Nature 595, 227–232 (2021). doi.org

[4] Wootters, William K., and Wojciech H. Zurek. ”A single quantum cannot be cloned.” Nature 299.5886 (1982): 802-803.

[5] D. Deutsch. Quantum theory, the Church-Turing Principle and the universal quantum computer. Proc. R. Soc. Lond. A, 400:97, 1985.

[6] R. P. Feynman. Simulating physics with computers. Int. J. Theor. Phys., 21:467, 1982.

[7] D. Simon, On the power of quantum computation, in Proceedings of the 35th Annual Symposium on Foundations of Computer Science (IEEE Computer Society Press, Los Alamitos, CA, 1994) 116-123.

[8] S. Lloyd, Universal quantum simulators, Science 273 (1996) 1073.

[9] Schumacher, Benjamin (1995-04-01). ”Quantum coding”. Physical Review A. American Physical Society (APS). 51 (4): 2738–2747. doi:10.1103/physreva.51.2738

[10] ”File:Bloch sphere.svg”, Wikimedia, online via: commons.wikimedia.org/wiki/File:Bloch_sphere.svg (last accessed on 29 July 2021).

[11] Monroe, Chris, et al. ”Demonstration of a fundamental quantum logic gate.” Physical review letters 75.25 (1995): 4714.

[12] Nielsen, Michael A., and Isaac Chuang. ”Quantum computation and quantum information.” (2002): 558-559.

[13] Shor, Peter W. ”Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.” SIAM review 41.2 (1999): 303-332.

[14] Lenstra, Arjen K., et al. ”The number field sieve.” The development of the number field sieve. Springer, Berlin, Heidelberg, 1993. 11-42.

[15] IBM Quantum, online via: quantum-computing.ibm.com (last accessed 29 July 2021).

[16] IBM Quantum System One, online via: research.ibm.com/quantum-computing/system-one/ (last accessed on 29 July 2021).

[17] Gabemtta, J. (2020, September 15).*IBM's Roadmap For Scaling Quantum Technology* from www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap

[18] Qiskit - Open-Source Quantum Development, online via: qiskit.org (last accessed on 29 July 2021).

[19] Matplotlib: Visualization with Python, online via: matplotlib.org/ (last accessed on 29 July 2021).

[20] IBM Quantum systems, online via: quantum-computing.ibm.com/services/docs/services/manage/systems/ (last accessed on 29 July 2021).

[21] Cross, A. W., Bishop, L. S., Sheldon, S., Nation, P. D., & Gambetta, J. M. (2019). Validating quantum computers using randomized model circuits. Physical Review A, 100(3), 032328.

[22] IBM Quantum simulators, online via: quantum-computing.ibm.com/services/docs/services/manage/simulator/ (last accessed on 29 July 2021).

[23] Niemann, P., Wille, R., & Drechsler, R. (2014, January). Efficient synthesis of quantum circuits implementing Clifford group operations. In 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC) (pp. 483-488). IEEE.

[24] Perez-Garcia, D., Verstraete, F., Wolf, M. M., & Cirac, J. I. (2006). Matrix product state representations. arXiv preprint quant-ph/0608197.

[25] Jozsa, R., & Nest, M. V. D. (2013). Classical simulation complexity of extended Clifford circuits. arXiv preprint arXiv:1305.6190.

[26] Sunar B. (2005) Euclidean Algorithm. In: van Tilborg H.C.A. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA . doi.org

[27] A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, R. Rivest, A. Shamir, L. Adleman, Communications of the ACM, Vol. 21 (2), 1978, pages 120—126. web.archive.org

[28] Zimmermann, Paul (February 28, 2020). ”Factorization of RSA-250”. cado-nfs-discuss (Mailing list) lists.gforge.inria.fr.

[29] Translation of Euler’s ”Theorematum quorundam ad numeros primos spectantium demonstratio” by David Zhao, Department of Computer Science, University of Texas at Austin. scholarlycommons.pacific.edu

[30] Barker, E., & Dang, Q. (2016). Nist special publication 800-57 part 1, revision 4. NIST, Tech. Rep, 16. csrc.nist.rip

[31] Beauregard, Stephane. ”Circuit for Shor’s algorithm using 2n+3 qubits.” arXiv preprint quant-ph/0205095 (2002)

[32] Gidney, Craig. ”Shor’s Quantum Factoring Algorithm”. Algorithmic Assertions. Retrieved 05 July 2021. algassert.com

[33] Parker, S., and Martin B. Plenio. ”Efficient factorization with a single pure qubit and log N mixed qubits.” arXiv preprint quant-ph/0001066 (2005).

[34] Gidney, Craig. ”Factoring with n+2 clean qubits and n-1 dirty qubits.” arXiv preprint arXiv:1706.07884 (2017).

A

The *definition* of Quantum Fourier Transform

The Quantum Fourier Transform is described by an equation derived from the definition of the Discrete Fourier Transform and adapted to use quantum states instead of complex numbers. Since, at the time of writing, there is only a limited amount of unitary quantum gates available, it's important to, when designing a quantum circuit to implement some function, manipulate the function's expression to try and determine which quantum gates can be used to perform that duty. The QFT expression A.1 can be simplified in a way that eases the designing of the circuit using some basic algebraic notions.

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x \sum_{k=1}^{n} y_k/2^k} |y\rangle \tag{A.1}$$

Equation A.2 can be achieved by transforming the sum in the exponent of A.1 into a product. Then, since both quantum states are defined by binary qubits, the state $|y\rangle$ can be written as the binary state $|y_n ... y_2 y_1\rangle$ and the sum split to account for each different qubit, originating A.3 that by expanding the product results in A.4.

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \prod_{k=1}^{n} e^{2\pi i x y_k / 2^k} |y\rangle \tag{A.2}$$

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{y_1=0}^{1} \sum_{y_2=0}^{1} \cdots \sum_{y_n=0}^{1} \prod_{k=1}^{n} e^{2\pi i x y_k / 2^k} |y_n ... y_2 y_1\rangle \tag{A.3}$$

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{y_n=0}^{1} \cdots \sum_{y_2=0}^{1} \sum_{y_1=0}^{1} (e^{2\pi i x y_1 / 2^1} \cdot e^{2\pi i x y_2 / 2^2} \cdot \ldots \cdot e^{2\pi i x y_n / 2^n}) |y_n ... y_2 y_1\rangle \tag{A.4}$$

State $|y_n ... y_2 y_1\rangle$ is a tensor product of each individual qubit $|y_k\rangle$ and may be rewritten as $|y_n\rangle \otimes ... \otimes |y_2\rangle \otimes |y_1\rangle$. This notion can be applied to A.4 and rearranged into equation A.5 making trivial the expansion of the sums and thus resulting in A.6.

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{y_n=0}^{1} e^{2\pi i x y_n / 2^1} |y_n\rangle \otimes \cdots \otimes \sum_{y_2=0}^{1} e^{2\pi i x y_2 / 2^n - 1} |y_2\rangle \otimes \sum_{y_1=0}^{1} e^{2\pi i x y_1 / 2^n} |y_1\rangle \tag{A.5}$$

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} (|0\rangle + e^{\frac{2\pi i x}{2^1}} |1\rangle) \otimes (|0\rangle + e^{\frac{2\pi i x}{2^2}} |1\rangle) \otimes \cdots \otimes (|0\rangle + e^{\frac{2\pi i x}{2^n}} |1\rangle) \tag{A.6}$$

Similarly to $|y\rangle$, $|x\rangle$ is a quantum binary state and so, to improve readability, can be written in binary fractional notation A.7 resulting in the simplified expression A.8.

$$|x\rangle = |x_n ... x_2 x_1\rangle \therefore \frac{x}{2^k} = 0.x_n x_{n-1} ... x_1 \tag{A.7}$$

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} (|0\rangle + e^{2\pi i 0.x_1} |1\rangle) \otimes (|0\rangle + e^{2\pi i 0.x_2 x_1 |1\rangle}) \otimes \cdots \otimes (|0\rangle + e^{2\pi i 0.x_n ... x_2 x_1} |1\rangle) \tag{A.8}$$