# Quantum Computing: From Algorithms to Implementation

Gonçalo Barbosa Valentim
Instituto Superior Técnico
Universidade de Lisboa
Lisbon, Portugal
Email: goncalovalentim@tecnico.ulisboa.pt

***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, 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.**

***Keywords*—**
**Quantum Computing; Quantum Fourier Transform; Shor's Algorithm; Qiskit.**

## I. Introduction

Classical computation has evolved from computers with the size of a room to day to day common objects persons can carry on themselves. 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 quantum computation [1].

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 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 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 exists a compiler smart enough to swap qubits around to help simulate a system where all the qubits are entangled.

The research object of this master thesis will be on developing and running an algorithm on a quantum computer. Furthermore, this work aims to operate with quantum computation: analyse ways to manipulate single qubits and use them to build quantum circuits; develop a quantum algorithm and simulate it on a real device.

## II. Theoretical Background

### A. 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,

$$
\begin{aligned}
|a\rangle &= \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \\
\langle b| &= \begin{pmatrix} b_1^* & b_2^* \end{pmatrix}, \\
\langle b|a\rangle &= a_1 b_1^* + a_2 b_2^*, \\
|a\rangle\langle b| &= \begin{pmatrix} a_1 b_1^* & a_1 b_2^* \\ a_2 b_1^* & a_2 b_2^* \end{pmatrix} \\
&.
\end{aligned} \tag{1}
$$

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 and represented as:

$$
|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \alpha, \beta \in \mathbb{C} \tag{2}
$$

Quantum mechanics define that when measuring a qubit the only possible outcomes are the computational basis states, and as such, measuring a superposition state gives $|0\rangle$, with probability $|\alpha|^2$, or $|1\rangle$, with probability $|\beta|^2$. Since the global phase of a qubit cannot be measured, the state representation in equation 2 can be simplified by using real numbers to represent $\alpha$ and $\beta$ and representing the difference in phase in the expression. Using the trigonometric identity $\sqrt{\sin^2 x + \cos^2 x}$ on the relation between them ($|\alpha|^2 + |\beta|^2 = 1$), allows the qubit to be represented by

$$
|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle\,. \tag{3}
$$

This equation defines a point on a three-dimensional unit sphere called Bloch sphere, represented in figure 1.

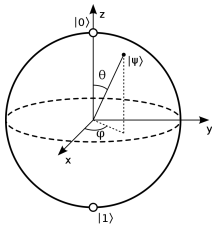



Fig. 1. Bloch sphere [3]

Looking at figure 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,

$$
\begin{aligned}
z\text{-axis} &\rightarrow \begin{cases} |0\rangle & \theta = 0, \varphi \text{ arbitrary} \\ |1\rangle & \theta = \pi, \varphi \text{ arbitrary} \end{cases}, \\
x\text{-axis} &\rightarrow \begin{cases} |+\rangle & \theta = \frac{\pi}{2}, \varphi = 0 \\ |-\rangle & \theta = \frac{\pi}{2}, \varphi = \pi \end{cases}, \\
y\text{-axis} &\rightarrow \begin{cases} |+i\rangle & \theta = \frac{\pi}{2}, \varphi = \frac{\pi}{2} \\ |-i\rangle & \theta = \frac{\pi}{2}, \varphi = \frac{3\pi}{2} \end{cases}.
\end{aligned} \tag{4}
$$

### *B. 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. For the qubit, flipping its value is swapping the probability of measuring each of its possible states so, the behaviour of the quantum not gate is given by

$$
\alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \beta\,|0\rangle + \alpha\,|1\rangle\,. \tag{5}
$$

This quantum version of the NOT gate is one of the Pauli gates, a set of single qubit gates that apply a rotation of $\pi$ around the axis, and whose name come from each of those axis. Each of these gates can be described by a $2 \times 2$ matrix.

$$
\begin{aligned}
\text{Pauli-X} = \text{X} &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\
\text{Pauli-Y} = \text{Y} &= \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \\
\text{Pauli-Z} = \text{Z} &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}
\end{aligned} \tag{6}
$$

In quantum computing, the initial state of a qubit is defined to be $|0\rangle$ and so, producing a superposition state requires manipulation. The gate that does this is the Hadamard gate:

$$
\text{Hadamard} = \text{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{7}
$$

Besides the already referred gates another single qubit gate worth mentioning is the Phase gate, defined in equation 8

$$
\text{Phase} = \text{P}(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}. \tag{8}
$$

The Phase gate, has a parameter $\varphi$ corresponding to the rotation around the $z$-axis the gate applies.

### *C. Multiple Qubit Gates*

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). This can be achieved by utilising two qubit gates like the C-NOT, a controlled version of the Pauli-X gate whose operation is given by:

$$
\text{C-NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{9}
$$

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.
The existence of the C-NOT, facilitated the creation of other multi-qubit gates. Using three sequential C-NOT gates, alternating the target and control qubits creates a gate that swaps the states of the qubits, the Swap gate. Controlled version of

other single qubit gates can also be done using the C-NOT, such as the Controlled Pauli-Z gate or the controlled Phase gate. Circuits acting on more than two qubits are also possible to build using C-NOT gates. The Toffoli is a complex circuit that uses the C-NOT gate to apply a doubly-controlled version of the Pauli-X gate. Similarly to how the C-NOT gate is used to create controlled versions of the single qubit gates, the Toffoli gate can also be used to create a controlled version of the two qubit gates, such as the Fredkin gate. A controlled version of the swap gate obtained by replacing the middle C-NOT gate from the swap gate by a Toffoli gate.

## III. Quantum computating

### A. Algorithms

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. The chosen algorithm was Shor's algorithm for factoring large numbers [4], one of the most important quantum algorithms ever invented.

Factoring algorithms scale 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.

### B. Quantum Computers

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 [5], a cloud application that allows users to program real quantum computers. There are twenty-one quantum systems made available by IBM [6] ranging from one to sixty-five available qubits. However, only eight of these are available for ordinary users with a maximum size of 5 qubits. IBM also provides a framework that allows users to run quantum circuits on their quantum computers. This framework comes in the form of Qiskit [7], an open-source software development kit that provides tools for the design and simulation of quantum programs that runs on Python.

## IV. Shor's Algorithm

Shor's Algorithm, proposed by Peter Shor [4], allows the factorization of large prime numbers in polynomial time using a quantum computer. 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; and, solving this order-finding problem using a quantum algorithm. The algorithm is given in [8] as:

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

The order-finding problem can be reduced to that of finding the period of $f(x) = g^x \mod N$. In classical computation, this means calculating every value of $f(x)$ for every possible value of $x$, however, quantum superposition makes it possible to write the qubits state as a linear combination of states, allowing the computation of all those images at the same time. Given that $f(x)$ is periodic, the produced state is composed of equal values separated by a period. When applied the QFT it creates 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, $|1/p\rangle$. The development of a quantum circuit to solve Shor's algorithm is, as aforementioned, the implementation of a quantum algorithm to solve the order-finding subroutine.

### A. Circuit Design

As described by Peter Shor [4], 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 $x$ ($|x\rangle$), has to be composed by $log_2(q)$ qubits which, taking into account the possible values of $q$, will be initialized with size $2n$ qubits, where $n = \lfloor log_2(N) \rfloor + 1$. Next, 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 $|x\rangle$, leaving the register in the state:

$$\frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle |0\rangle . \tag{10}$$

Having defined all possible values of x, the next step is to compute $f(x) = g^x \mod N$. In order to apply this function, a complex circuit was designed. This circuit is referred to as

$U_{x,N}$ gate and will be analysed in the next section. Applying this gate to equation 10 produces the state in equation 11

$$\frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle |g^x \mod N\rangle . \tag{11}$$

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

$$QFT^\dagger |x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{-\frac{2\pi i}{N} xy} |y\rangle , \tag{12}$$

which when applied to the state produced in equation 11 creates the state:

$$QFT^\dagger |x\rangle |g^x \mod N\rangle = \frac{1}{q} \sum_{x=0}^{q-1} \sum_{y=0}^{q-1} e^{-\frac{2\pi i}{q} xy} |y\rangle |g^x \mod N\rangle . \tag{13}$$

After building a quantum gate to implement the inverse QFT, the circuit to implement the order-finding subroutine can be assembled. Applying the gates, as described, to quantum register $|x\rangle$ for $n = 3$, creates the circuit in figure 2.

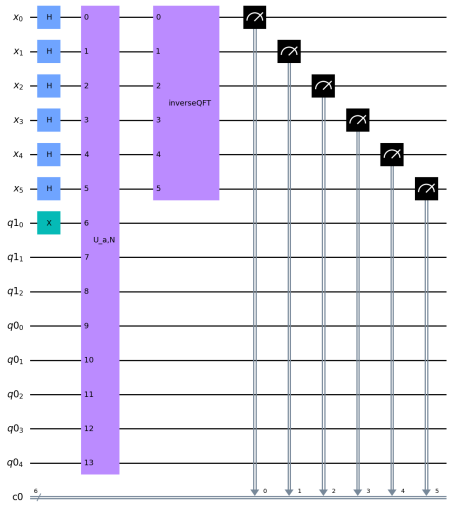



Fig. 2. Circuit built to implement the order-finding subroutine of Shor's Algorithm for $n = 3$.

### B. The $U_{x,N}$ gate

The $U_{x,N}$ gate applied in the circuit represented in figure 2 computes $f(x) = g^x \mod N$. This gate was implemented based on the circuit described by Stéphane Beauregard [9].

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

$$QFT(|b\rangle) \rightarrow QFT(|a + b\rangle). \tag{14}$$

Adding two QFTs corresponds just to add the phases of each individual qubit. 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 3. This phases can be classically calculated since $a$ is a given constant.

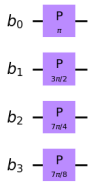



Fig. 3. Quantum circuit for addition in the Fourier space, with a=7.

The adder gate is composed only by unitary gates. As such, the inverse of this gate can be build by inverting the circuit. However, unlike when adding $a$ and $b$, subtracting them might result in a negative number and as such, it maps in equation 15.

$$QFT(|b\rangle) \rightarrow \begin{cases} QFT(|b - a\rangle), & b \geqslant a. \\ QFT(|2^{n+1} - a + b\rangle), & b < a. \end{cases} \tag{15}$$

A controlled or doubly controlled version of the adder can also be made using controlled versions of the Phase gates.

Using the implemented adder (figure 3) 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 4.

By assuming $|q0_0\rangle |q1_0\rangle = |11\rangle$, the circuit in figure 4 starts by applying an adder gate to the register containing $QFT(|b\rangle)$ switching the state as defined in equation 15. 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 16.

$$|b\rangle_5 |b_5\rangle = QFT(|a + b - N\rangle) |0\rangle . \tag{16}$$

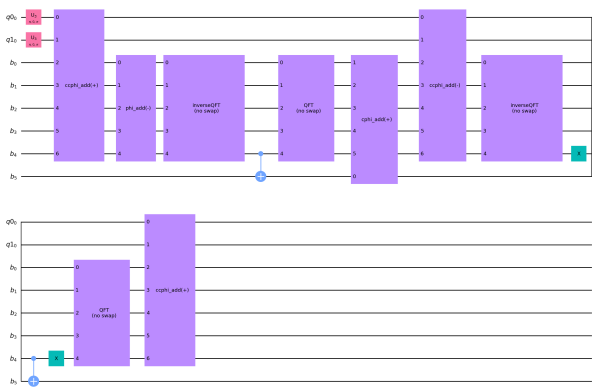


Fig. 4. Quantum circuit for the modular addition.

If $(a+b) \geqslant N$ the state from equation 16 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 17 summarizes the behaviour of the circuit for the two possible outcomes.

$$\begin{cases} |b_5\rangle = 1 & \text{if } (a+b) < N \xrightarrow{\text{N is added}} |b\rangle_5 |b_5\rangle = \\ & = QFT(|a+b\rangle) |1\rangle = \\ & = QFT(|(a+b) \mod N\rangle) |1\rangle, \\ |b_5\rangle = 0 & \text{if } (a+b) \geqslant N \xrightarrow{\text{N is not added}} |b\rangle_5 |b_5\rangle = \\ & = QFT(|a+b-N\rangle) |0\rangle = \\ & = QFT(|(a+b) \mod N\rangle) |0\rangle. \end{cases} \tag{17}$$

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 17 equation 18 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.

$$\begin{cases} |b\rangle_5 = a+b-a = b & \text{if } |b_5\rangle = 1, \\ |b\rangle_5 = a+b-N-a = b-N & \text{if } |b_5\rangle = 0. \end{cases} \tag{18}$$

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 19.

$$|b\rangle_5 |b_5\rangle = QFT(|(a+b) \mod N\rangle) |0\rangle. \tag{19}$$

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. As such, the multiplication of an arbitrary number $x$ by an arbitrary constant $a$ can be defined by,

$$x \cdot a = 2^{n-1}ax_{n-1} + 2^{n-2}ax_{n-2} + \cdots + 2^0 ax_0. \tag{20}$$

Given the relation described in equation 20, 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 $QFT(|(b+2^j a x_j) \mod N\rangle)$, 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.

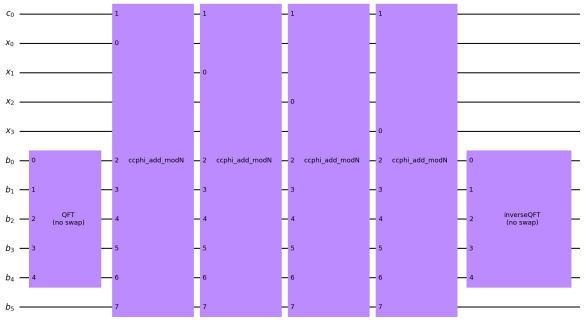


Fig. 5. Quantum circuit for the modular multiplier.

This 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 $|b\rangle$. Given this, the circuit needs to be changed so it instead stores the result in the input register. 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 21. 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. The circuit implementing the $U_a$ gate using Qiskit is shown in figure 6.

$$
\begin{aligned}
&|x\rangle|0\rangle \xrightarrow{multiplier} |x\rangle|(ax) \mod N\rangle \\
&|x\rangle|(ax) \mod N\rangle \xrightarrow{swap} |(ax) \mod N\rangle|x\rangle \\
&|(ax) \mod N\rangle|x\rangle \xrightarrow{inv\_mult} |(ax) \mod N\rangle \otimes \\
&\otimes \left|(x - a^{-1}ax) \mod N\right\rangle = |(ax) \mod N\rangle|0\rangle
\end{aligned}
\tag{21}
$$

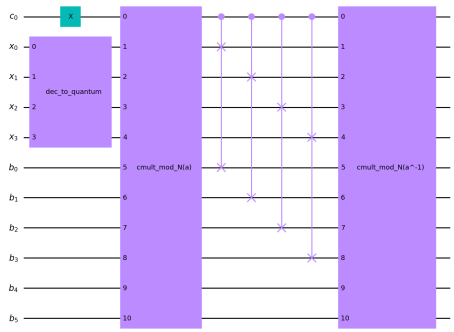



Fig. 6. Quantum circuit for the modular multiplier gate acting on the input register.

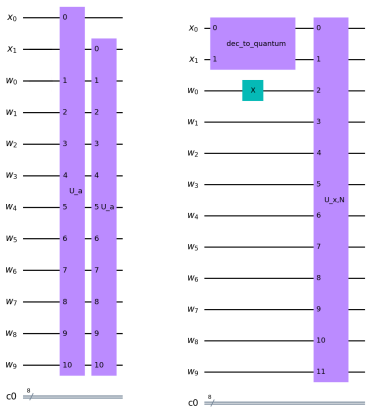
The function applied by the $U_{x,N}$ gate is not modular multiplication but modular exponentiation, in the form of $f(x) = g^x \mod N$. This can be accomplished using the same controlling approach as for the modular multiplier, since:

$$
\begin{aligned}
&|g^x \mod N\rangle = \\
&= \left|g^{2^0 x_0 + x^1 x_1 + \cdots + 2^{n-1} x_{n-1}} \mod N\right\rangle = \\
&= \left|(((g^{2^0 x_0} \mod N) g^{2^1 x_1} \mod N) \cdots g^{2^{n-1} x_{n-1}} \mod N)\right\rangle .
\end{aligned}
\tag{22}
$$

Being $g$ a classical constant, $g^{2^j}$ can be computed classically and as such, the $U_{x,N}$ gate can be 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 7(a).
The $U_{x,N}$ was be tested using the circuit in figure 7(b). In this circuit, a Pauli-X gate was used to force $|w\rangle$ to $|1\rangle$ since the $U_{x,N}$ computes $f(x) = (w \cdot g)^x \mod N$ instead of $f(x) = g^x \mod N$.Running the circuit in a quantum simulator for $N = 15$, $x = 4$ and $g = 7$, produced the state vector in in figure 8 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.

### C. Quantum Fourier Transform

The last gate needed to implement the circuit for the order-finding subroutine of Shor's algorithm is the Quantum Fourier Transform (QFT). 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.




(a) (b)

Fig. 7. Quantum circuits for the $U_{x,N}$ gate. (a) Quantum circuit to implement the $U_{x,N}$ gate; (b) Quantum circuit to test the $U_{x,N}$ gate;

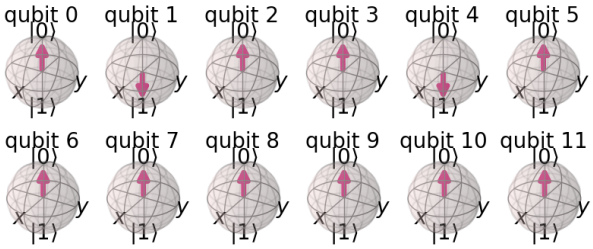



Fig. 8. State vector produced by the $U_{x,N}$ gate where, qubit $0 \rightarrow |c_0\rangle$, qubit $1{:}4 \rightarrow |x\rangle$ and qubit $5{:}10 \rightarrow |b\rangle$ with qubits 4 and 10 representing the most significant bit of $|x\rangle$ and $|b\rangle$ respectively.

This quantum algorithm is an efficient implementation of the discrete Fourier transform (DFT) in a quantum computer. The DFT takes an input vector of complex numbers ($x$) and outputs a transformed vector of complex numbers ($y$).

$$
y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k / N}. \tag{23}
$$

The QFT is the quantum implementation of the DFT and as such, instead of acting on complex numbers, it acts on qubits. 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 DFT can be written as

$$
|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x y / N} |y\rangle . \tag{24}
$$

Although qubits can exist in many different states, they can only be measured in two. This, leads to qubits states being represented using binary notation. Equation 24 can be re-written to account for this fact and, as such, the QFT of the state $|x\rangle$ where $x = x_n ... x_2 x_1$, is described by:

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x \sum_{k=1}^{n} y_k/2^k} |y\rangle . \tag{25}$$

Equation 25 can be further simplified through algebraic manipulation to the expression considered by Nielsen, Michael A., and Isaac Chuang as the *definition* of QFT in [8],

$$\begin{aligned} QFT(|x\rangle) = \frac{1}{\sqrt{N}} \Big[ (|0\rangle + e^{2\pi i 0.x_1} |1\rangle) \otimes \\ \otimes (|0\rangle + e^{2\pi i 0.x_2 x_1} |1\rangle) \otimes \cdots \\ \cdots \otimes (|0\rangle + e^{2\pi i 0.x_n \ldots x_2 x_1} |1\rangle) \Big]. \end{aligned} \tag{26}$$

The first term of equation 26 corresponds to the state of the most significant bit (MSB) of $|x\rangle$ and the last term to the least significant one.

After defining the function to be applied, the quantum circuit can be designed. Looking at 26 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 26 results in

$$\frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.x_n \ldots x_2 x_1} |1\rangle), \tag{27}$$

where the $\frac{1}{\sqrt{2}}$ in 27 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,

$$\frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i (\frac{x_1}{2^n} + \frac{x_2}{2^{n-1}} + \ldots + \frac{x_n}{2^1})} |1\rangle). \tag{28}$$

This simplified expression is very similar to the implemented by the Hadamard Gate, differing only on the phase which varies depending on the values of the other qubits. Controlled Phase gates can be used to apply this changes to the phase of the qubits. Applying an Hadamard to the last qubit produces:

$$\frac{1}{\sqrt{2}} (|0\rangle + e^{\frac{2\pi i x_n}{2}} |1\rangle). \tag{29}$$

That, after applying $n-1$ controlled rotations becomes:

$$\begin{aligned} \frac{1}{\sqrt{2}} (|0\rangle + e^{\frac{2\pi i}{2^n} x_1 + \frac{2\pi i}{2^{n-1}} x_2 + \ldots + \frac{2\pi i}{2} x_n} |1\rangle) = \\ = \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.x_n \ldots x_2 x_1} |1\rangle). \end{aligned} \tag{30}$$

By analysing the state in equation 30 and comparing it with the definition of QFT in equation 26, it is noticeable that, the state produced by applying the Hadamard and controlled Phase gates to the most significant bit of $|x\rangle$, corresponds not to the QFT of the first term but to the last term of the equation, which is the QFT of the least significant bit (LSB). To correct this, after all gates are applied to both qubits, their value needs to be switched. This is done using a swap gate.

Having the gates necessary to build the circuit been determined, the circuit was implemented using Qiskit and the circuit represented in figure 9 was obtained. Simulating the behaviour of the circuit on a quantum computer is no possible because the final state is in an uniform superposition state, making its measurement result in $|0\rangle$ or $|1\rangle$ with equal probability. However, if instead of applying the QFT, the circuit is inverted to apply the inverse QFT ($QFT^\dagger$), the input state would be an uniform superposition and, the output, a basis state.

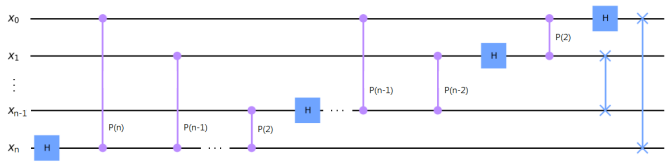



Fig. 9. Circuit to implement QFT with swaps.

Given that all used gates are unitary, inverting the circuit is just applying the gates in the reverse order (Figure 10 contains the diagram for a circuit implementing the inverse QFT on a 4 qubit number). And, the initial state can be prepared by putting all qubits in an uniform superposition state using Hadamard gates and applying the rotations using Phase gates. To determine the amplitude of the rotations to apply, the QFT can be calculated. Solving equation 26 for $x = 1011$ produces the states in equation 31.

$$\begin{aligned} q_0 &= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.1011} |1\rangle) = \frac{1}{\sqrt{2}} (|0\rangle + e^{\frac{11}{8}\pi} |1\rangle) \\ q_1 &= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.011} |1\rangle) = \frac{1}{\sqrt{2}} (|0\rangle + e^{\frac{3}{4}\pi} |1\rangle) \\ q_2 &= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.11} |1\rangle) = \frac{1}{\sqrt{2}} (|0\rangle + e^{\frac{3}{2}\pi} |1\rangle) \\ q_3 &= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.1} |1\rangle) = \frac{1}{\sqrt{2}} (|0\rangle + e^{\pi} |1\rangle) \end{aligned} \tag{31}$$

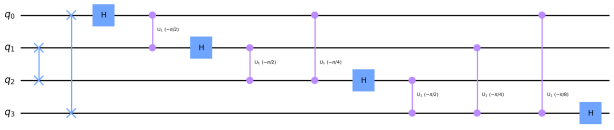



Fig. 10. Circuit to implement the Inverse Quantum Fourier Transform.

Applying the Hadamard gates followed by the Phase gates, with the phases calculated in equation 31, to the circuit shown in figure 10 and measuring the output produces the schematic in figure 11. This circuit is ready to be tested on a real quantum device however, given the probabilistic nature of quantum computation, it needs to be ran multiple times.

Figure 12 shows a histogram containing the results obtained by running the circuit from figure 11 2048 times on a quantum computer from IBM.

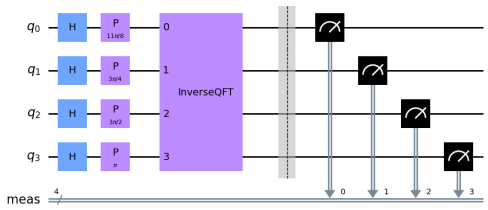



Fig. 11. Circuit to test the QFT on a real device.

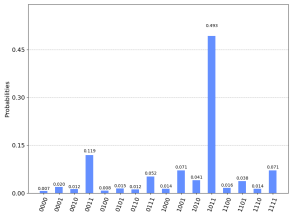


Fig. 12. Histogram produced by running the circuit on ibm_q_athens.

### D. Circuit Implementation

After implementing and testing all components from the circuit to implement the order-finding subroutine of Shor's algorithm, the next step is to assemble the circuit and test it. To implement the circuit one needs to first choose the values of $N$ and $g$. $N$ equal to fifteen is the smallest odd number that is also the product of two different prime numbers - three and five. 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 to implement the circuit from figure 13. This circuit computes the order-finding subroutine of Shor's algorithm for $N = 15$. The four vertical lines represent different moments in time and will be used to refer the circuit state on that given time.

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 [10], 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

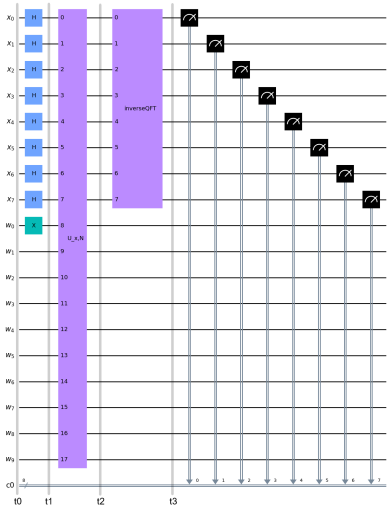



Fig. 13. Circuit to implement order-finding subroutine of Shor's Algorithm for N=15.

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. The initial state is given by

$$|t_0\rangle = |x\rangle_4 |w\rangle_6 = |0\rangle_4 |0\rangle_6 . \tag{32}$$

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 32 to

$$|t_1\rangle = \frac{1}{\sqrt{16}} \sum_{x=0}^{15} |x\rangle_4 |1\rangle_6 \equiv \frac{1}{4} \Big[ |0\rangle_4 + |1\rangle_4 + |2\rangle_4 + \cdots \\ \cdots + |14\rangle_4 + |15\rangle_4 \Big] |1\rangle_6 . \tag{33}$$

From here the gate $U_{x,N}$ is applied to equation 33 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:

$$|t_2\rangle = \frac{1}{4} \Big[ |0\rangle_4 \left|2^0 \mod 15\right\rangle_6 + |1\rangle_4 \left|2^1 \mod 15\right\rangle_6 + \cdots \\ \cdots + |15\rangle_4 \left|2^{15} \mod 15\right\rangle_6 \Big], \tag{34}$$

which can be developed to make

$$|t_2\rangle = \frac{1}{4} \Big[ |00\rangle_4 |1\rangle_6 + |01\rangle_4 |2\rangle_6 + |02\rangle_4 |4\rangle_6 + |03\rangle_4 |8\rangle_6 + \\ + \cdots + |12\rangle_4 |1\rangle_6 + |13\rangle_4 |2\rangle_6 + |14\rangle_4 |4\rangle_6 + |15\rangle_4 |8\rangle_6 \Big]. \tag{35}$$

Just by looking at the state from equation 35, 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:

$$|x\rangle_1 = \frac{1}{\sqrt{2}}\Big[\,|0\rangle + |1\rangle\,\Big], \tag{36}$$

When measured, it 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 35, 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. Assuming the measured result was $|4\rangle$:

$$\begin{aligned}|t_2\rangle = \frac{1}{2}\Big[\,|2\rangle_4|4\rangle_6 + |6\rangle_4|4\rangle_6 + |10\rangle_4|4\rangle_6 + |14\rangle_4|4\rangle_6\,\Big] \equiv \\ \equiv \frac{1}{2}\Big[\,|2\rangle_4 + |6\rangle_4 + |10\rangle_4 + |14\rangle_4\,\Big]|4\rangle_6\,.\end{aligned} \tag{37}$$

To extract the periodicity of the function, the inverse QFT, is applied to the first register transforming the state $|t_2\rangle$ in:

$$\begin{aligned}|t_3\rangle =& QFT^\dagger(|x\rangle_4)\,|4\rangle_6 \equiv \frac{1}{2}\Big[QFT^\dagger\,|2\rangle_4 + \\ &+ QFT^\dagger\,|6\rangle_4 + QFT^\dagger\,|10\rangle_4 + QFT^\dagger\,|14\rangle_4\Big]\,|4\rangle_6\,.\end{aligned} \tag{38}$$

$$QFT^\dagger\,|x\rangle = \frac{1}{4}\sum_{y=0}^{15} e^{-\frac{\pi i}{8}xy}\,|y\rangle \tag{39}$$

Solving equation 38 using the definition of QFT$^\dagger$ in equation 39, computes the inverse Fourier transform of the first register:

$$\frac{1}{8}\sum_{y=0}^{15}\Big[e^{-\frac{\pi i}{8}2y} + e^{-\frac{\pi i}{8}6y} + e^{-\frac{\pi i}{8}10y} + e^{-\frac{\pi i}{8}14y}\Big]\,|y\rangle_4\,. \tag{40}$$

Solving these equations is not trivial, it was done in this Thesis by software programmed in python and the results produced were applied to equation 40 leading to the final state:

$$|t_3\rangle = \frac{1}{8}\Big[4\,|0\rangle_4 + (-4)\,|4\rangle_4 + 4\,|8\rangle_4 + (-4)\,|12\rangle_4\,\Big]. \tag{41}$$

When measuring the first register of $|t_3\rangle$, the value has an equal probability of being $|0\rangle_4$, $|4\rangle_4$, $|8\rangle_4$ or $|12\rangle_4$ and so the histogram from figure 14: as explained in [10], results in a period equal to the number of peaks.
This implementation of the order-finding subroutine was validated by comparing the result produced by the circuit with

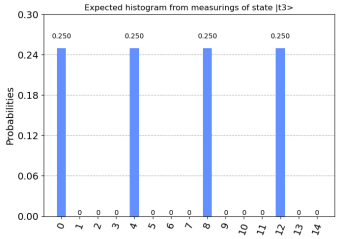



Fig. 14. Circuit to implement order-finding subroutine of Shor's Algorithm for N=15.

the calculated period. Figure 15 shows the histogram built from 2048 measurements of the first register from figure 13 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 $|x\rangle$ register of size four for the calculations had no impact on the final result.

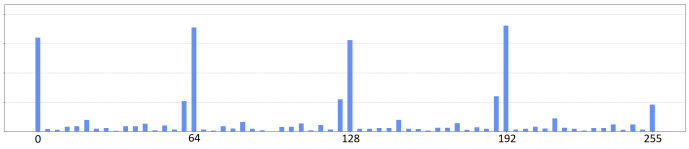



Fig. 15. Histogram build from 2048 measurements of the circuit from figure 13 for N=15 and g=2 (measurements are summed in groups of 4 to fit the page.)

### *E. Semi-classical implementation*

The semi classical implementation of Shor's algorithm uses a sequential implementation of the QFT as described by S.Parker and M.B.Plenio [11]. This implementation uses a single qubit to extract the periodicity of $f(x) = g^x \mod N$. This is done by using measurements in the middle of the circuit to vanquish the necessity of using conditional gates during the $QFT^\dagger$. Figure 16 shows that, not having the conditional Phase gates or the swap gate, allows all gates to be applied to one qubit before moving to another, which can be obtained by measuring the value of the qubits used as control and storing them classically. This qubits are then used to decide whether a not controlled Phase gate is applied in the $QFT^\dagger$. This process needs to be started with the qubit that is not applied any conditional Phase gates, but is used as control in all the gates; the most significant qubit. Having the order of the measurements been determined, the circuit can be implemented by applying all gates that act on a qubit and measuring it. It is also important to note that, since the qubit is reused, it needs to be restored to its initial state by using a Pauli-X gate conditioned on the measured value.
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. The circuit diagram in figure 17 corresponds to the implementation of the semi-classical order-finding subroutine of Shor's Algorithm for $n = 2$ implemented using Qiskit. As usual it starts by defining the size of the registers, which, for the semi-classical implementation is 1 for

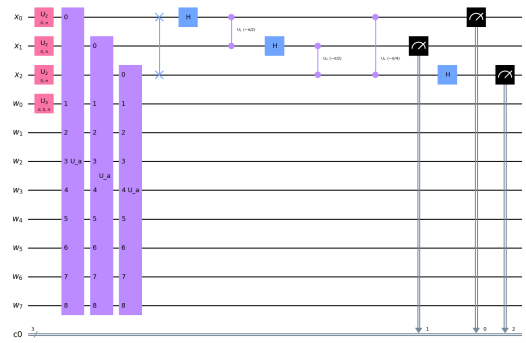


Fig. 16. Quantum circuit for order-finding subroutine with $n = 3$.

$|x\rangle$ and $2n + 2$ for $|w\rangle$. Since this circuit requires measuring qubits during computation to control other gates, two classical register were created to stores 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.

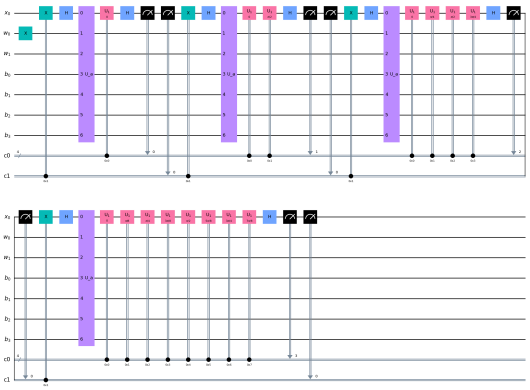


Fig. 17. Quantum circuit for the semi-classical order finding subroutine for $n = 2$.

The circuit represented in figure 17 is the implementation of the semi-classical order finding subroutine for a two qubit number. Since solving the algorithm for such numbers does not produce relevant results, the same code used to build this circuit was used to build another circuit to solve the order-finding subroutine for $N = 15$ and $g = 2$. This implementation was done in the same quantum simulator as the previous one using the same values for $N$ and $g$. Figure 18 contains 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 15.

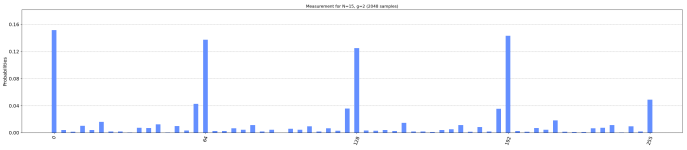


Fig. 18. Histogram build from 2048 measurements of the circuit for the semi-classical implementation with N=15 and g=2 (measurements are summed in groups of 4 to fit the page).

## V. Conclusion

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 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, allowed the number of necessary qubits to be reduced from $4n + 2$ to $2n + 3$, roughly half the amount. Doing this comes with the cost of greatly 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. All code produced in this project has been made available at github.com/goncalobvalentim/fromalgtoimp.

## References


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

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

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

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

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

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

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

[10] Gidney, Craig. "Shor's Quantum Factoring Algorithm". Algorithmic Assertions. Retrieved 05 July 2021. algassert.com

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