Performance evaluation of quantum Fourier transform on a modern quantum device

MARCUS GRANSTRÖM, SVEN ANDERZÉN RODENKIRCHEN

Performance evaluation of quantum Fourier transform on a modern quantum device

MARCUS GRANSTRÖM, SVEN ANDERZÉN RODENKIRCHEN

Bachelor in Computer Science
Date: June 5, 2018
Supervisor: Stefano Markidis
Examiner: Örjan Ekeberg
Swedish title: Prestandautvärdering av kvant-Fouriertransform på en modern kvantdator
School of Electrical Engineering and Computer Science

Abstract

Quantum computing has in the last decades gone from a purely theoretical concept to being implemented on actual devices in the real world. However, one of the major challenges of quantum computing still remains; its sensitivity to noise. Great progress has been made in the last couple of years in reducing the effects of noise on a quantum computation which raises the question of how good a non error corrected quantum computation actually performs on a modern quantum device.

In this work we perform quantum Fourier transform, a common operation found in many quantum algorithms on a real quantum device, the IBMQX4. We compare the results obtained from the real quantum device against the same error-free results obtained through simulation and measure the impact the error has had on the period.

We show that although it is possible to distinguish a correct result for trivial problem instances, error caused by noise is still to large to obtain a correct result for any non-trivial problems. We conclude that quantum error correcting techniques are a necessity for quantum computations of any non-trivial problem size.

Sammanfattning

Kvantberäkningar har under de senaste decennierna gått från att vara ett teoretiskt koncept till att implementeras på riktiga enheter. En stor utmaning finns dock kvar; kvantdatorernas känslighet för störningar. Under de senaste åren har stora framsteg gjorts för att minska påverkan av störningar i kvantberäkningar vilket öppnar upp för frågan; hur bra fungerar en icke-felrättande kvantberäkning på en modern kvantdator?

I detta arbete utför vi kvant-Fouriertransform, en vanlig operation i många kvantalgoritmer, på den riktig kvantdatorn IBMQX4. Vi jämför resultaten från den riktiga kvantdatorn mot felfria simuleringar och mäter den inverkan felen haft på perioden.

Vi visar att även om det är möjligt att urskilja ett korrekt resultat för triviala probleminstanser så är fel orsakat av störningar fortfarande för stort för att erhålla ett korrekt resultat för icke-triviala problem. Vi fastställer att tekniker för felkorrigering inom kvantberäkningar är en nödvändighet för icke-triviala probleminstanser.

Contents

Chapter 1

Introduction

During the past few decades great effort has gone into realizing a new form of computation known as *quantum computing*. Unlike classical computers where a single bit can be in one of two distinct states at any given point in time a quantum bit, or *qubit*, has the ability to be in a *superposition* of both zero and one at the same time. Together with the quantum mechanical phenomenon known as *entanglement*, quantum computers provide a new way of solving problems not otherwise realistically solvable by classical computers. One of these problems is integer factorization, which is heavily used as a one-way function in public-key cryptography and can be efficiently solved by *Shor's algorithm*.

Shor's algorithm is a quantum algorithm that consists of a classical part and a quantum mechanical part, where the quantum mechanical part uses *Quantum Fourier Transform* (QFT) to find the period of a certain function. However, by utilizing superposition, the algorithm is prohibited from partially solving the problem according to the no-cloning theorem. This means that in order for Shor's algorithm to provide the quantum speedup theory predicts, errors must be kept at a minimum during the entire run of the computation since any errors that do occur propagates throughout the circuit and could lead to a wrong result.

One of the sources of error in a quantum computation is *decoherence*, or the loss of information due to environmental disturbances. This disturbance can be divided into two types of errors; *energy relaxation*, measured by the time constant T_1 and *dephasing*, measured by a the time constant T_2. During the last decade great progress has been made in improving these measurements which raises the question of how well a typical quantum operation such as the quantum Fourier transform performs today, on a modern quantum device.

1.1 Problem definition

Being able to factorize large integers would completely break public-key cryptography protocols such as RSA and have severe implications on online security. The purpose of this work is to investigate how well the quantum Fourier transform, which is an integral part of Shor's algorithm, works on a modern quantum device in order to evaluate the current state of the field.

The following question will be explored in this report:

> How well does the quantum Fourier transform perform on a modern quantum device?

In this report we define performance as how close the output of a real quantum device is to an error-free simulated output.

1.2 Scope and constraints

This thesis will focus on quantum circuits consisting of three and five quantum bit registers and will run on the same quantum device provided by IBM. Due to hardware limitations no logical qubits or error correcting codes have been implemented or used since it would require more quantum bits than is currently available to the community and would increase the complexity of this work. The authors recognize this as a limitation and reasons further about this in section 5.3.

1.3 Thesis overview

This thesis is divided into four main sections, each covering a different topic of the work. These sections are:

- **Background:** providing vital information for interpreting later sections.
- **Method:** describing the experiments that was conducted and the reasoning behind the experimental setup.

- **Result:** presenting a comparison between the error-free simulated results and the results acquired from a real quantum device.
- **Discussion:** the authors thoughts about the result, further improvements and possible future work.

1.4 Acknowledgements

We acknowledge use of the IBM Q experience for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Q experience team.

Chapter 2

Background

2.1 Quantum bits

A quantum bit, or *qubit*, is a two-dimensional quantum mechanical system and a basic unit of quantum information (Yanofsky and Mannucci, 2008, p. 139). Unlike a classical system, which can only be in one of two distinct states (represented as 0 or 1), quantum mechanics allow a quantum bit to be in *superposition* of both states at the same time. Although this can seem counter-intuitive from a classical viewpoint this property is fundamental to the theory of quantum computations.

The state of a given two-state quantum mechanical system (and in extension the state of a qubit) can be described by a state vector in a two-dimensional Hilbert space [1] (Yanofsky and Mannucci, 2008, p. 311). Given the space basis vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.1}$$

every state vector $|\Psi\rangle$ can be described as a linear combination of the two basis vectors (Yanofsky and Mannucci, 2008, p. 140):

[1] A Hilbert space is a complex inner product space that is complete, i.e. any sequence of vectors accumulating somewhere converges to a point (Yanofsky and Mannucci, 2008, p. 60).

$$|\Psi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \alpha\begin{pmatrix}1\\ 0\end{pmatrix} + \beta\begin{pmatrix}0\\ 1\end{pmatrix} = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.2}$$

Multiple qubits can also be combined to form the basis of higher dimensions (Yanofsky and Mannucci, 2008, pp. 142-144). For a two-qubit system the space basis vectors are:

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \quad |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix} \quad |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \quad |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \tag{2.3}$$

Upon measuring a qubit in superposition, the fragile quantum state present in the system will collapse to one of the space basis vectors (Yanofsky and Mannucci, 2008, p. 139). The probability of measuring an outcome of $|0\rangle$ is equal to $|\alpha|^2$ and the probability of measuring an outcome of $|1\rangle$ is equal to $|\beta|^2$. Since the coefficients represents the states probabilities the value of the coefficients are constrained under the condition:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.4}$$

2.2 Bloch sphere

One way to visualize the state of a single quantum bit is by using the *Bloch sphere*. A generic qubit $|\Psi\rangle$ on the form

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.5}$$

has two complex numbers and in extension four real number dimensions and degrees of freedom (Yanofsky and Mannucci, 2008, p. 160). Rewriting the qubit state on polar form gives us:

$$|\Psi\rangle = r_0 e^{i\phi_0}\,|0\rangle + r_1 e^{i\phi_1}\,|1\rangle \quad \text{where} \quad \alpha = r_0 e^{i\phi_0} \quad \beta = r_1 e^{i\phi_1} \tag{2.6}$$

Since a quantum physical state does not change if we multiply its corresponding vector by an arbitrary complex number we can obtain the following equivalent state for the qubit [2](Yanofsky and Mannucci, 2008, p. 160)

$$e^{-i\phi_0} |\Psi\rangle = e^{-i\phi_0}(r_0 e^{i\phi_0} |0\rangle + r_1 e^{i\phi_1} |1\rangle) = r_0 |0\rangle + r_1 e^{i(\phi_1 - \phi_0)} |1\rangle \tag{2.7}$$

By using the fact that the sum of all outcome probabilities of a quantum state equals one and that the length of $e^{i\phi}$ equals one we get:

$$\begin{aligned} 1 &= |\alpha|^2 + |\beta|^2 &(2.8)\\ &= |r_0 e^{i\phi_0}|^2 + |r_1 e^{i\phi_1}|^2 &(2.9)\\ &= |r_0|^2 |e^{i\phi_0}|^2 + |r_1|^2 |e^{i\phi_1}|^2 &(2.10)\\ &= r_0^2 + r_1^2 &(2.11) \end{aligned}$$

Noting that the above equality resembles the equation for the unit circle we can rewrite the variables to eliminate one of the dimensions with the help of the Pythagorean trigonometric identity:

$$r_0 = \cos(\theta) \qquad r_1 = \sin(\theta) \tag{2.12}$$

Together with Equation 2.7 we arrive at the final two parameter canonical form (Yanofsky and Mannucci, 2008, p. 160):

$$|\Psi\rangle = \cos(\theta) |0\rangle + e^{i\phi} \sin(\theta) |1\rangle \tag{2.13}$$

Given that there is only two parameters in this form, any single qubit state can be visualized as a point on a three dimensional sphere, where θ and ϕ represents the amplitude (latitude) and phase (longitude) respectively (Yanofsky and Mannucci, 2008, p. 161). Figure 2.1 shows a visualization of this sphere known as the *Bloch sphere*.

[2] A vector $|\Psi\rangle$ and all its complex scalar multiples $c|\Psi\rangle$ describe the same physical state (Yanofsky and Mannucci, 2008, pp. 108-109). When multiplying a vector $|\Psi\rangle$ with a scalar the relative probability of an outcome does not change and thereby neither its quantum mechanical state.

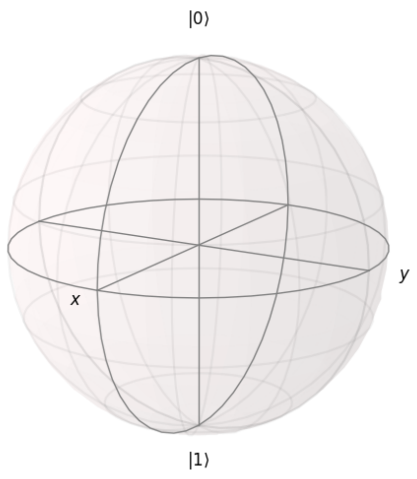


Figure 2.1: The Bloch sphere

2.3 Quantum gates

Every operation that is not a measurement in a quantum computer is reversible and can be represented by a unitary matrix [3](Yanofsky and Mannucci, 2008, p. 151). The reason why a measurement is not reversible is because the measurement itself forces the quantum bit in superposition to collapse to a classical state which can not be undone without violating the *no-cloning theorem* 2.6.3 (Yanofsky and Mannucci, 2008, p. 96).

2.3.1 Single-qubit gates

Pauli gates

The Pauli gates are a set of operators that rotates a quantum bit π radians around the x, y or z axis. More general forms of these operators also exists that makes it possible to rotate θ radians around one of the axis.

[3]A unitary matrix is a complex square matrix whose conjugate transpose is equal to its inverse.

Pauli-X gate

The Pauli-X gate rotates the qubit π radians around the x axis. In the case where the qubit is positioned in one of the classical states, $|0\rangle$ or $|1\rangle$, the X gate will simply act as a "bit-flip" resembling the NOT gate used in classical computing (Nielsen and Chuang, 2010, p. 18). The operation is represented by the Pauli X matrix 2.14 (Yanofsky and Mannucci, 2008, p. 158). A visual representation of the operation can be seen in figure 2.2.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.14}$$

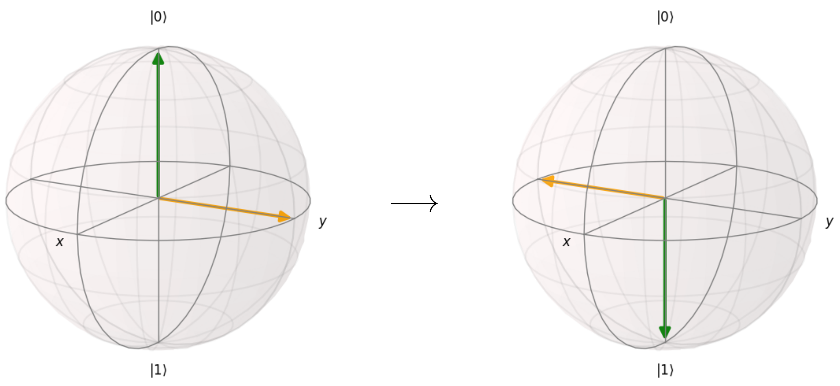


Figure 2.2: The qubits state before and after a Pauli-X gate.

Pauli-Y gate

The Pauli-Y gate rotates the qubit π radians around the y axis. This operation performs the transformation $|0\rangle \rightarrow i\,|1\rangle\,, |1\rangle \rightarrow -i\,|0\rangle$ and is represented by the Pauli Y matrix 2.15 (Yanofsky and Mannucci, 2008, p. 158). A visual representation of the operation can be seen in figure 2.3.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.15}$$

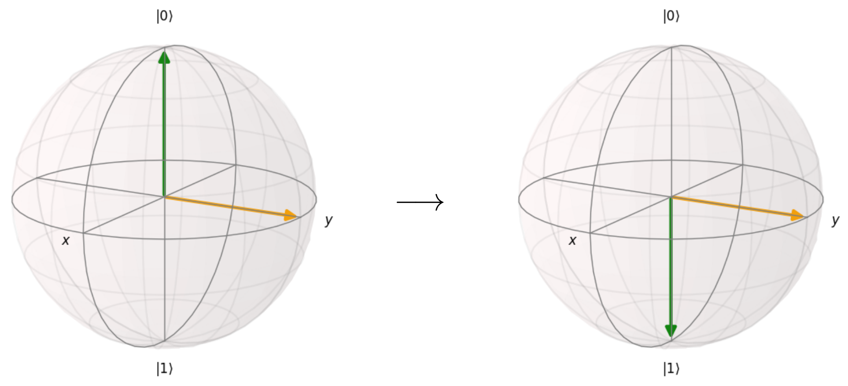


Figure 2.3: The qubits state before and after a Pauli-Y gate.

Pauli-Z gate

The Pauli-Z gate rotates the qubit π radians around the z axis. This leaves the amplitude of the qubit unchanged, i.e. $|0\rangle$ will still be $|0\rangle$ and $|1\rangle$ will still be $|1\rangle$, but the phase of the qubit will become shifted by π radians. Phase shifting does not affect the probability of which classical state the qubit will collapse to but it does change the quantum state of the qubit (Yanofsky and Mannucci, 2008, p. 163). When the qubit is in superposition with an equal probability of collapsing to $|0\rangle$ and $|1\rangle$, the result of a phase shift of π radians will be the transformation $|+\rangle \to |-\rangle, |-\rangle \to |+\rangle$. The operation is represented by the Pauli Z matrix 2.16 (Yanofsky and Mannucci, 2008, p. 158). A visual representation of the operation can be seen in figure 2.4.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.16}$$

Hadamard gate

The Hadamard gate is used to force a qubit into superposition. The operation rotates the qubit π radians around the $x + z$ axis and can be visualized in the Bloch sphere as a $\pi/2$ radians rotation around the y axis followed by a π radians rotation around the x axis (Nielsen and Chuang, 2010, p. 19). If the Hadamard gate is applied to a qubit in a

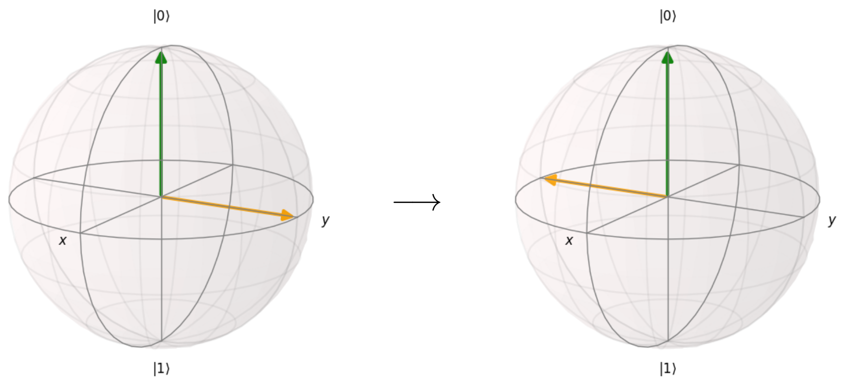


Figure 2.4: The qubits state before and after a Pauli-Z gate.

classical state of either $|0\rangle$ or $|1\rangle$ the operation will force the qubit into a superposition with equal probability amplitudes of $|0\rangle$ and $|1\rangle$.

The Hadamard gate is represented by the Hadamard matrix 2.17 (Yanofsky and Mannucci, 2008, p. 158). A visual representation of the operation can be seen in figure 2.5.

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \tag{2.17}$$

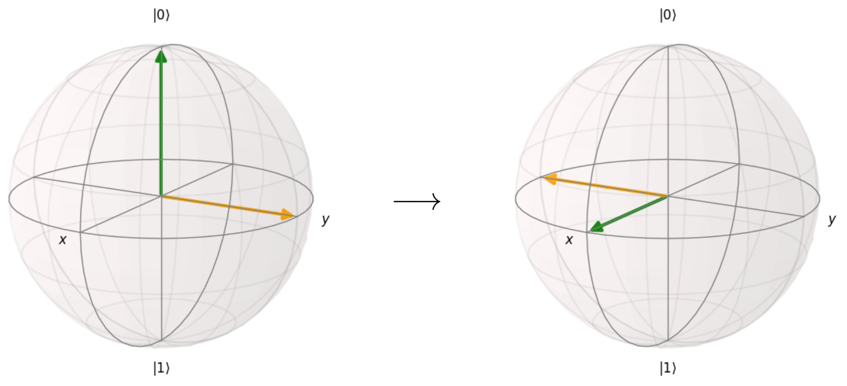


Figure 2.5: The qubits state before and after a Hadamard gate.

2.3.2 Multi-qubit gates

Controlled-NOT gate

The controlled-NOT (CNOT) gate is a multi-qubit gate that can be used to *entangle* and *disentangle* two different qubits. The operation is represented by the CNOT matrix 2.18 (Nielsen and Chuang, 2010, p. 178).

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.18}$$

The controlled-NOT gate can be used for the quantum phenomenon known as *entanglement* and is comprised of a *control* qubit, denoted $\bullet$ and a *target* qubit, denoted $\oplus$ (Nielsen and Chuang, 2010, p. 178). The gate performs the Pauli-X operation on the target qubit if and only if the control qubit is in state $|1\rangle$. Circuit 2.6 provides an example of a circuit using the controlled-NOT gate.

In the example circuit (2.6) the quantum bits are initialized to state $|00\rangle$ giving the classical outcome state 00 with a probability of 100 percent. After applying the Hadamard gate to qubit q_0, the qubit will be set to state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, giving it an equal probability of being in state $|0\rangle$ and $|1\rangle$. By applying the controlled-NOT gate, where qubit q_0 is the controller and qubit q_1 is the target, we entangle the qubits and can determine the value of q_1 just by observing q_0 [4](Nielsen and Chuang, 2010, p. 97).

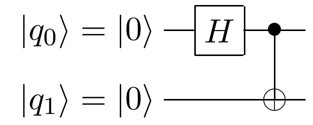


Figure 2.6: An example of a circuit using the controlled-NOT gate.

[4]Qubit q_1 will be altered if and only if qubit q_0 collapses to state $|1\rangle$. This means that if q_0 is in state $|0\rangle$, q_1 will be in state $|0\rangle$ and if q_0 is in state $|1\rangle$, q_1 will be in state $|1\rangle$. This will give an even probability distribution between the outcomes $|00\rangle$ and $|11\rangle$.

Controlled phase gate

The controlled phase gate (CPHASE) is similar to the controlled-NOT gate but performs a general Pauli-Z gate rotation by ϕ radians on the target qubit instead of a Pauli-X gate. The operation is represented by the CPHASE matrix 2.19 (Yanofsky and Mannucci, 2008, pp. 163-165).

$$CPHASE = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix} \tag{2.19}$$

Circuit 2.7 provides an example of a circuit using the controlled phase gate. The subscript ϕ indicates the rotation of the Pauli-Z operation.

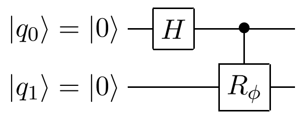


Figure 2.7: An example of a circuit using the controlled phase gate.

2.4 Shor's algorithm

Shor's algorithm is a quantum algorithm able to solve the integer factorization problem in polynomial time. One of the main application areas for the algorithm is for factoring the public key number N used in RSA cryptography, where two large prime numbers p and q together form the prime factors of N (Yanofsky and Mannucci, 2008, pp. 204-217). The algorithm can be described by the following steps:

Step 1 Choose a random co-prime integer $a < N$. If a and N are not co-prime then a non-trivial factor has already been found and the algorithm can be terminated.

Step 2 Find the smallest period r of the function $f(x) = a^x \mod N$, such that $f(x) = f(x + r)$. If r is an odd integer pick a new integer a from **Step 1**.

Step 3 If $a^{r/2} \equiv -1 \mod N$ pick a new integer a from **Step 1**.

Step 4 Calculate and return the prime factors $p = gcd((a^{r/2} - 1), N)$ and $q = gcd((a^{r/2} + 1), N)$.

The above steps, with the exception of **Step 2**, can all be calculated relatively quickly on a classical computer by using the Euclidean algorithm. However, at the time of writing, there does not exist an efficient algorithm to find the period of the function $f(x)$ in **Step 2**, even with modest key sizes. It is in this step the quantum Fourier transform is used in Shor's algorithm to give the exponential speed up.

2.5 Fourier Transform

Quantum Fourier transform (QFT) is the quantum equivalent of classical discrete Fourier transform (DFT).

2.5.1 Discrete Fourier Transform

Classical Fourier transform is a method of transforming a function from the time-plane (a signal) to the frequency-plane, thereby getting the sinusoidal functions that make up the signal. The discrete Fourier transform takes a fixed N dimensional vector of complex numbers $\bar{x} = x_0, \ldots, x_{N-1}$ as input and outputs the transformed data $\bar{y} = y_0, \ldots, y_{N-1}$ (Nielsen and Chuang, 2010, p. 37). This function is defined as:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi ijk/N} \tag{2.20}$$

By letting $\omega_N = e^{2\pi i/N}$ be an N-th root of unity we can define the $j \times k$ matrix F_N, where $j \in \{0, 1, \ldots, N-1\}, k \in \{0, 1, \ldots, N-1\}$ (de Wolf, 2011, p. 1). This matrix is defined as:

$$F_N = \frac{1}{\sqrt{N}} \begin{pmatrix} & \vdots & \\ \cdots & \omega_N^{jk} & \cdots \\ & \vdots & \end{pmatrix} \tag{2.21}$$

This gives us the discrete Fourier transformation for the input $\bar{x} = x_0, \ldots, x_{N-1}$ on matrix form as:

$$\begin{pmatrix} F(x_0) \\ F(x_1) \\ F(x_2) \\ \vdots \\ F(x_{N-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{N-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{N-2} & \cdots & \omega \end{pmatrix} \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \end{pmatrix} \tag{2.22}$$

2.5.2 Quantum Fourier Transform

It can be shown that F_N is a unitary matrix allowing us to view it as a quantum mechanical mapping between two vectors of amplitudes (de Wolf, 2011, p. 3). This mapping is called the *Quantum Fourier Transform*. Unlike classical Fourier transform which works on a given vector of inputs, quantum Fourier transform works on a quantum mechanical state, a vector of amplitudes corresponding to the probability of each outcome of a superposition (de Wolf, 2011, p. 3).

The quantum Fourier transform is a linear transformation on each basis $|k\rangle$ of a quantum mechanical state (de Wolf, 2011, p. 3). It is defined by the function:

$$|k\rangle \longrightarrow F_N |k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{jk} |j\rangle \tag{2.23}$$

In the case of $N = 2^n$, where n is the number of qubits in the input, this will be an n-qubit unitary and can be implemented with a quantum circuit requiring $\mathcal{O}(n^2)$ quantum gates (de Wolf, 2011, p. 3). An example of the 3-qubit quantum Fourier transform circuit is depicted in figure 2.8 (de Wolf, 2011, p. 5).

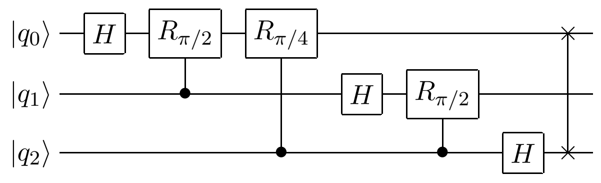


Figure 2.8: Quantum Fourier transform circuit for $n = 3$.

2.6 Errors in quantum computations

2.6.1 Pure and mixed state

We call a quantum mechanical state $|\psi\rangle$ a *pure state* if it is in a known and well-defined state. In this case the density operator is $\rho = |\psi\rangle \langle\psi|$ (Nielsen and Chuang, 2010, p. 100). If a pure state is exposed to noise from the surrounding environment this single, well-defined pure state will become a probabilistic mixture of multiple different pure states, called a *mixed state*. The phenomenon causing loss of purity is called *decoherence* (Yanofsky and Mannucci, 2008, pp. 306-308).

2.6.2 Decoherence

One of the sources of error in a quantum computation is *decoherence*, or the loss of information due to environmental disturbances (Yanofsky and Mannucci, 2008, pp. 306-308). This disturbance can be divided into two sources of errors; *energy relaxation*, measured by the time constant T_1 and *dephasing*, measured by a the time constant T_2 (Nielsen and Chuang, 2010, p. 280).

Energy relaxation

Energy relaxation is the decoherence process where a quantum bit in the higher energy state $|1\rangle$ decays towards the lower energy state $|0\rangle$. This relaxation time represents the lifetime of classical states and is denoted with the constant T_1 (Nielsen and Chuang, 2010, p. 280).

Dephasing

Dephasing is the process of a single quantum bit in an arbitrary superposition state losing its quantum mechanical state. This relaxation time is usually shorter than the energy relaxation time T_1 and is denoted with the constant T_2 (Nielsen and Chuang, 2010, p. 280).

2.6.3 No-cloning theorem

The *no-cloning theorem* is a quantum mechanical theorem stating that it is impossible to clone an exact quantum mechanical state without first destroying the original state (Yanofsky and Mannucci, 2008, p. 166). This is due to the measurement itself forcing the system to collapse into a classical state without the possibility of restoring the original quantum mechanical state. Noson S. Yanofsky and Mirco A. Mannucci, authors of *Quantum Computing for Computer Scientists*, phrases the phenomenon as following:

> In "computerese," this says that we can "cut" and "paste" a quantum state, but we cannot "copy" and "paste" it. "Move is possible. Copy is impossible." (Yanofsky and Mannucci, 2008, p. 166)

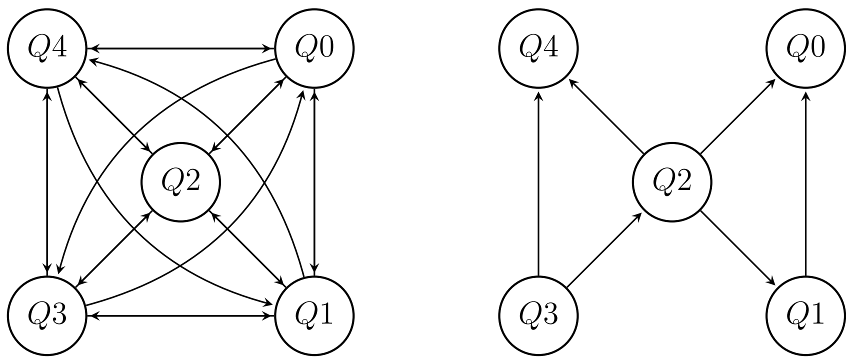
This property prevents quantum computers from using classical error correcting techniques since no backup of the quantum mechanical state can be made within a computation. However, certain quantum error correcting techniques exist which does not rely on copying the quantum mechanical state while still allowing for a certain amount of error correction (Yanofsky and Mannucci, 2008, p. 303).

2.7 Coupling map

The quantum registers in a real quantum device (QPU) are arranged in a planar manner and connected to their neighbors (*Compiling and running a quantum program* n.d.). However, each quantum register is only connected to *some* of the other quantum register making it impossible to directly entangle those registers who are not connected. In order to run circuits which use controlled-NOT gates on quantum registers that are not connected *coupling maps* can be used.

A coupling map consists of a directed graph specifying which registers can be connected as well as specifying which register is the controller and which is the target, since the relation is not always bidirectional. By using the information in the coupling map the circuit can be rewritten to be able to run on a QPU by inserting *swap* gates to switch the states between two connected qubits (*Compiling and running a quantum program* n.d.).

Given that inserting gates increases the number of operations that have to be run, one wants to limit the number of swap gate that is required. This becomes an optimization problem and is an active research topic at the time of writing (Zulehner et al., 2018).



(a) Register coupling for a simulator. (b) Register coupling for IBMQX4.

Figure 2.9: Difference in quantum register coupling between a simulator and a real device.

2.8 Software

There exist several different software interfaces for running quantum computations, both on simulators as well as on real quantum devices. Two of these interfaces are *PyQuil* [5] from Rigetti Computing and *QISKit* [6] from IBM.

[5]https://github.com/rigetticomputing/pyquil
[6]https://github.com/QISKit/qiskit-sdk-py

2.8.1 PyQuil

PyQuil is a Python based library used for generating instructions in the Quantum Instruction Language (Quil). Quil is a low level quantum instruction set language able to run on both a simulator (QVM) as well as on real quantum processing units (QPU) on Rigetti Forest (Smith et al., 2016, p. 2). The simulator from Rigetti provide an error free environment to test quantum circuits, with the ability to simulate custom noise models such as decoherence and gate error if the user chooses (*Noise and Quantum Computation* n.d.).

At the time of writing the latest stable release is version 1.8.0 which is also the version that has been used in this project. The documentation can be found at http://pyquil.readthedocs.io/en/latest/.

2.8.2 QISKit

QISKit is a Python based library used for generating instructions in the Quantum Assembly Language (QASM). QASM is a low level quantum instruction set language able to run on a both a simulator as well as on a real quantum device on IBM Quantum Experience (Cross et al., 2017, p. 1). The simulator from IBM provide random noise model resembling a real quantum device by default.

At the time of writing the latest stable release is version 0.4.14 which is also the version that has been used in this project. The documentation can be found at https://qiskit.org/documentation/

2.9 Previous work

Previous research conducted by Lockheed Martin Advanced Technology Laboratories indicate that the polynomial scaling of Shor's algorithm vanishes in the presence of operator imprecision errors (Hill and Viamontes, 2008, p. 1). In their research they state that as input errors are introduced to the period finding subroutine, the ability of the overall algorithm to find short periods gets degraded (Hill and Viamontes, 2008, p. 6). They further conclude that in the presence of error the period finding subroutine break the polynomial scaling of Shor's algorithm making it no more efficient than classical algorithms for factoring integers (Hill and Viamontes, 2008, p. 13).

Chapter 3

Method

The following sections aims to explain the methods that has been used as well as describe the decisions that has been made throughout the project.

3.1 Setup

3.1.1 Software

Both software stacks described in the background has been used in the project. PyQuil was used for the simulation part in order to get an error-free baseline result and QISKit was used for real life results. The source code for the experiment can be found in appendix A.

3.1.2 Quantum device

All experiments performed on a real quantum device was conducted on the five qubit quantum device IBMQX4, provided through the IBM Quantum Experience [7]. The quantum device was calibrated on *2018-05-14 11:55:30* with the error measurements shown in figure 3.1 and 3.2 when the experiments was conducted.

[7]https://quantumexperience.ng.bluemix.net/

	Q_0	Q_1	Q_2	Q_3	Q_4
Frequency (GHz)	5.242	5.307	5.352	5.411	5.189
T1 (μs)	50.9	49.1	34.3	29.9	60.4
T2 (μs)	14.3	73.6	46.4	16.9	28.1
Gate error (10^{-3})	0.6868	6.273	1.288	4.209	1.03
Readout error (10^{-2})	5.7	5.6	9.2	3.9	5.9

Figure 3.1: Device readouts for IBMQX4

Gate error (10^{-2})	
cx $1 \to 0$	2.30
cx $2 \to 0$	2.73
cx $2 \to 1$	2.86
cx $3 \to 2$	13.55
cx $3 \to 4$	8.34
cx $4 \to 2$	4.91

Figure 3.2: Multi-qubit gate error for each qubit connection

3.1.3 Quantum Fourier Transform

In order to be able to perform Fourier transform on the input states we have implemented the quantum Fourier transform circuit according to figure 3.3 and 3.4. The two circuits have been implemented in both PyQuil and QISKit and the full implementation details can be found in the source code in appendix A.

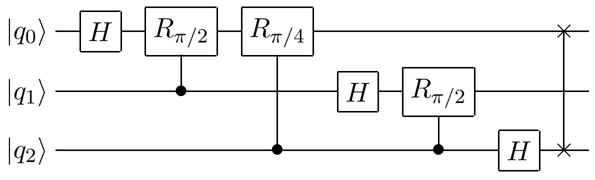


Figure 3.3: 3-qubit quantum Fourier transform circuit.

3.2 Experiments

Three separate experiments has been conducted, each with a different input state to the quantum Fourier transform. The experiments was

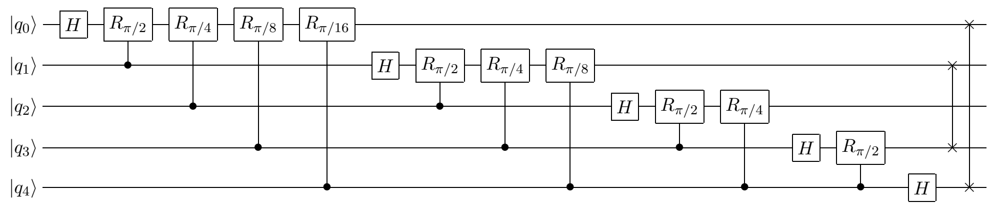


Figure 3.4: 5-qubit quantum Fourier transform circuit.

conducted on the same device with the same calibrations for all runs (see section 3.1.2). Each quantum computation was conducted 8192 times in order to get the distribution of outcome amplitudes.

3.2.1 Input states

Three different input states has been used in the experiments, each highlighting a certain property of the quantum Fourier transform. Each input state and its properties are described more in detail in the sections below. The circuits used to implement the states can be found in appendix D. The detailed state amplitudes for each state can be found in appendix B.

All experiments was conducted on a five qubit device. In the three qubit experiments the two unused quantum registers $|q_3\rangle$ and $|q_4\rangle$ was initialized and set to $|0\rangle$ during the entire computation. Since these quantum registers always will be in state $|0\rangle$ they have been removed from the diagrams and tables to provide a better overview.

Input state 1

The first input state is a quantum state with evenly distributed amplitudes on even binary outcomes. Given that the largest possible outcome of the Fourier transform is a full period of 2π, or 2^n measured in the number of states where n is the number of qubits, the smallest meaningful period measured in the number of states is 2. An example of this state with three qubits is shown in figure 3.5.

Since this input state represents a function with only two sine basis functions, $\sin(0x)$ and $\sin(4x)$, intuition suggests that this state should be highly sensitive to noise during the computation. Given an error

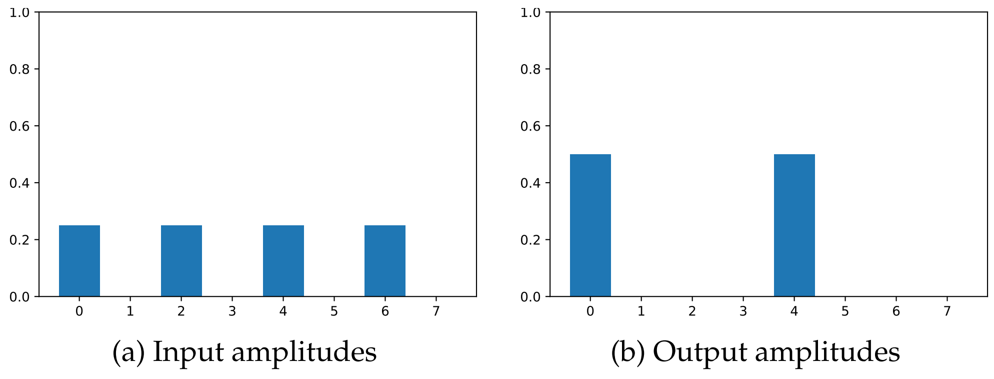


Figure 3.5: Amplitudes of input state 1 with 3 qubits.

that produces a slightly different outcome there is a high chance that another sinusoidal is introduced, thereby destroying the period.

Input state 2

The second input state is a classical state resulting in an quantum state of evenly distributed amplitudes as seen in figure 3.6. Since the sum of all amplitudes always is equal to one, each outcome has an amplitude of $1/2^n$, where n is the number of qubits used in the quantum Fourier transform. Although this input state is impossible to find in any non-trivial instance of Shor's algorithm it highlights the effect error has on the amplitudes without affecting the period. This input state can therefore be seen as the state that is the most tolerant to errors in the computation, measured in the period.

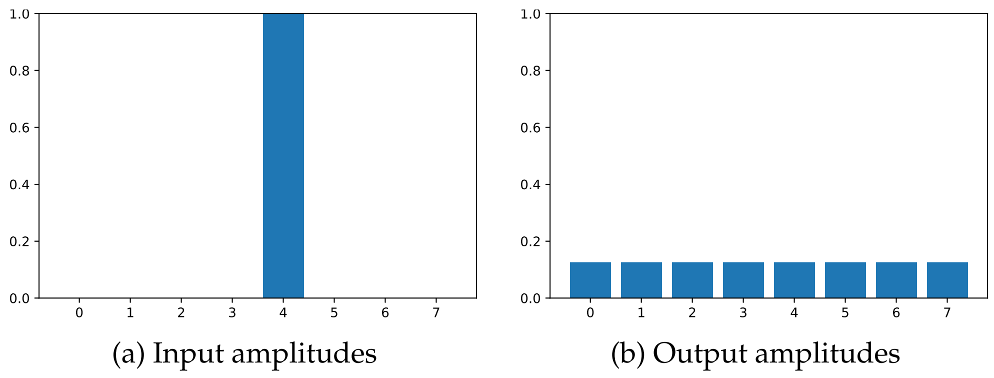
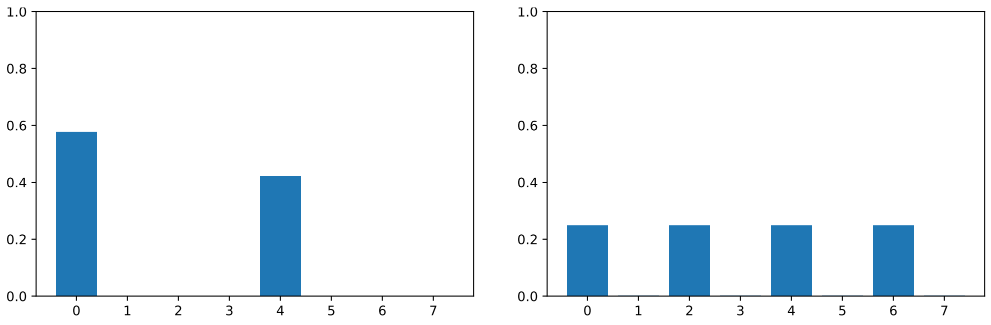


Figure 3.6: Amplitudes of input state 2 with 3 qubits.

Input state 3

The third input state is a quantum state that has been slightly rotated around the Y axis. This state highlights the quantum Fourier transforms tolerance to input errors in the Y axis were the input will be unevenly distributed between two states as seen in figure 3.7. This state was chosen because when the quantum Fourier transform is used to find periods, small amplitudes are more prone to disappear since there is a smaller probability that it will be part of the output from a real quantum device.



(a) Input amplitudes (b) Output amplitudes

Figure 3.7: Amplitudes of input state 3 with 3 qubits.

3.3 Evaluation

Two different methods have been used to evaluate the error of each quantum computation, each providing its own perspective of the final error. These two methods are *residual sum of squares* (RSS) and *period finding*.

3.3.1 Residual sum of squares

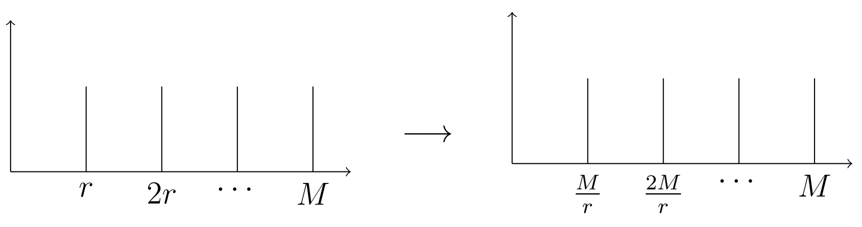
Residual sum of squares (RSS) or sum of squared error (SSE) is a measurement of the differences between a set of predicted values and a set of observed values. For n observations each with a predicted value y_i and an observed value $\hat{y}_i$ the residual sum of squares is defined as

$$RSS = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

The above evaluation metric was chosen to give a sense of the magnitude of the error in the calculation. This metric gives a squared growth for errors on observed amplitudes further away from their predicted amplitude, but does not consider the case of different sinusoidal functions all together.

3.3.2 Period finding

The output of the Fourier transform is a set of quantum amplitudes corresponding to the amplitudes of the sinusoidal functions that the original function consists of. When a periodic function of input length M and with period r is used as input, and no errors are present in the Fourier transform, the output will consist of nonzero amplitudes on multiples of $\frac{M}{r}$. See figure 3.8 for a visualization of this transformation.



(a) Input before Fourier transform. (b) Output after Fourier transform.

Figure 3.8: Periodic function before and after Fourier transform.

In order to extract the period r we calculate the greatest common divisor between the resulting values after the Fourier transform. When the greatest common divisor has been calculated the period r of the original function can be found by equation 3.1.

$$r = \frac{M}{gcd(\frac{M}{r}, \frac{2M}{r}, \ldots, M)} \tag{3.1}$$

The above evaluation metric was chosen because of its use in different quantum algorithms. Shor's algorithm is one example, where the aim is to find the period of the original function that was input to the quantum Fourier transform. This evaluation metric provides us with the information whether or not we can find the correct period of the original function, but is not able to give us any information about how far off the correct solution we was or the error of the amplitudes.

Chapter 4

Result

The following results are all from experiments conducted on the five qubit device IBMQX4, provided through IBM Quantum Experience. The three qubit experiments was performed on the same quantum device and the two unused quantum registers $|q_3\rangle$ and $|q_4\rangle$ was initialized and set to $|0\rangle$ during the entire computation. Since these quantum registers always will be in state $|0\rangle$ they have been removed from the results to provide a better overview.

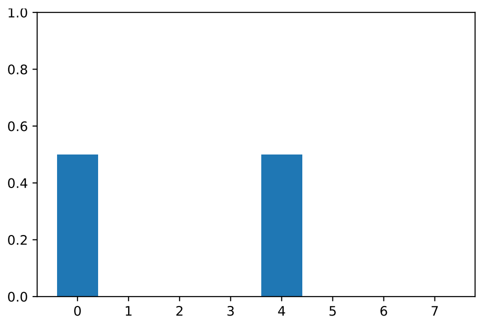
The output amplitudes from the real quantum device are compared side by side with the error-free simulated output amplitudes from section 3.2.1. The exact output amplitude values for each experiment can be found in appendix C.

4.1 Output amplitudes

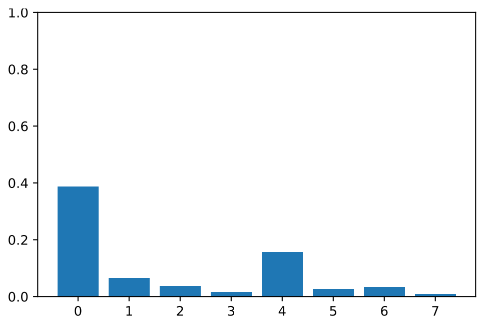
4.1.1 3-qubit quantum Fourier transform

Below figures provide a side by side comparison between the simulated and real output amplitudes of the three qubit quantum Fourier transform. Since the experiments was conducted on a five qubit quantum device, were quantum registers $|q_3\rangle$ and $|q_4\rangle$ was initialized and set to $|0\rangle$, output amplitudes on states greater than seven have been removed from the results to provide a better overview.

Input state 1 - Output amplitudes



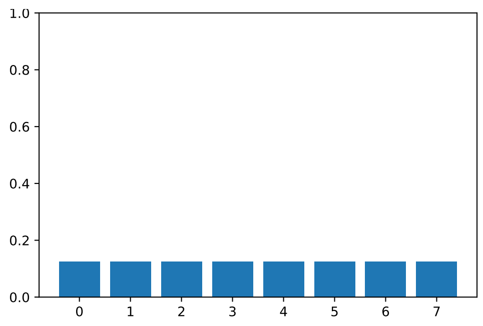
(a) Simulated output amplitudes



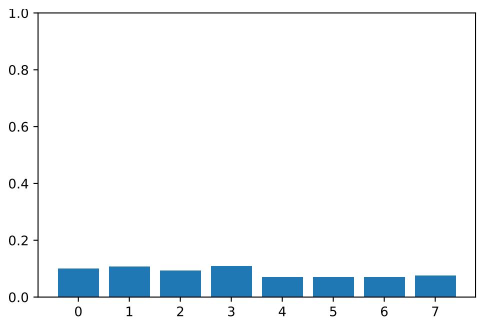
(b) Real output amplitudes

Figure 4.1: Output amplitudes of input state 1

Input state 2 - Output amplitudes



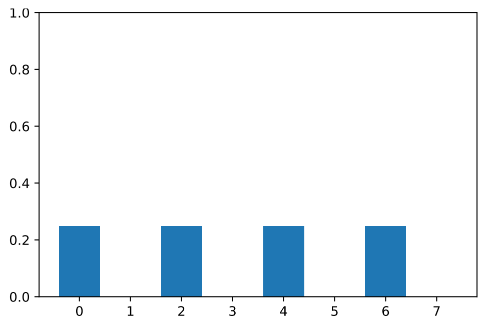
(a) Simulated output amplitudes



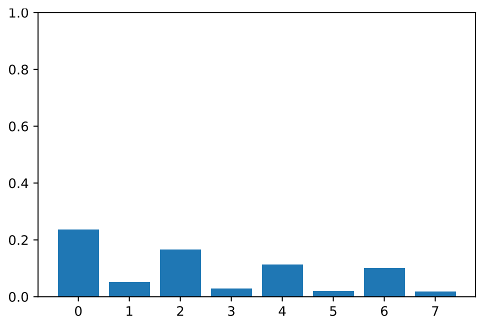
(b) Real output amplitudes

Figure 4.2: Output amplitudes of input state 2

Input state 3 - Output amplitudes



(a) Simulated output amplitudes



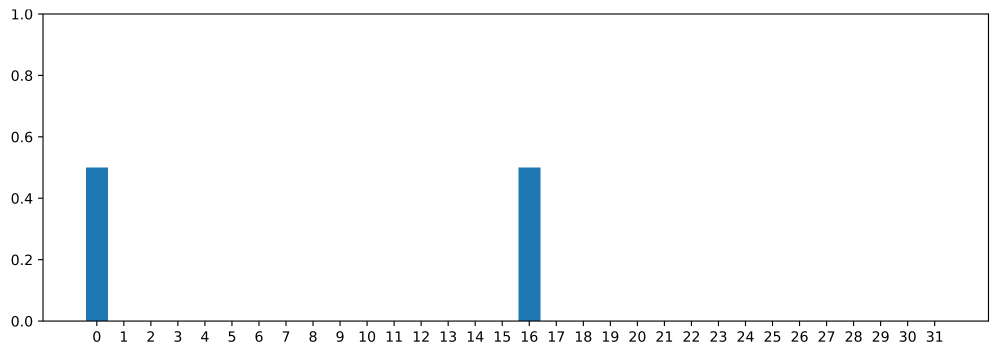
(b) Real output amplitudes

Figure 4.3: Output amplitudes of input state 3

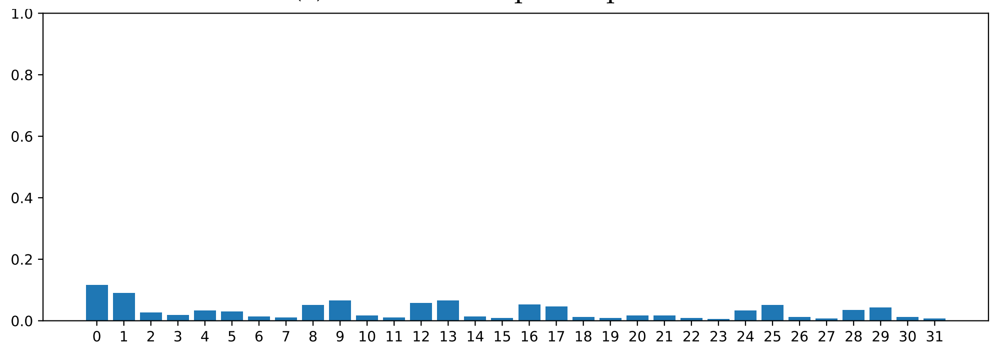
4.1.2 5-qubit quantum Fourier transform

Below figures provide a side by side comparison between the simulated and real output amplitudes of the five qubit quantum Fourier transform.

Input state 1 - Output amplitudes



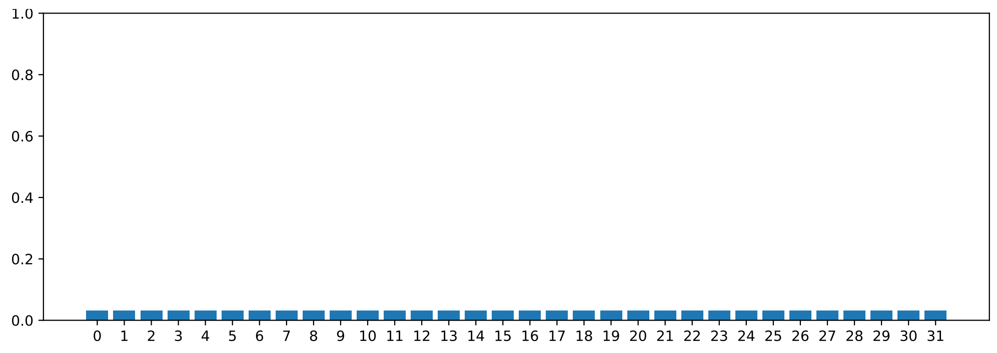
(a) Simulated output amplitudes



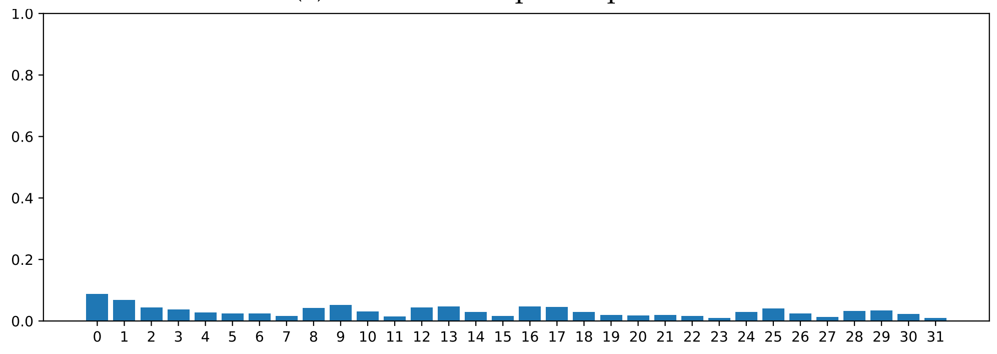
(b) Real output amplitudes

Figure 4.4: Output amplitudes of input state 1

Input state 2 - Output amplitudes



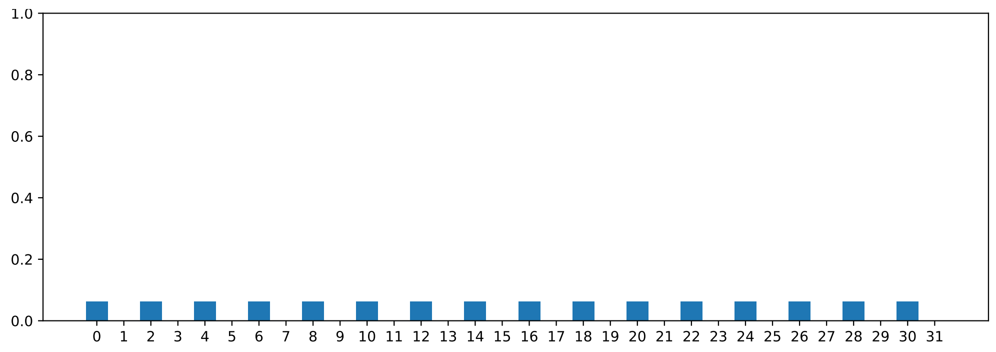
(a) Simulated output amplitudes



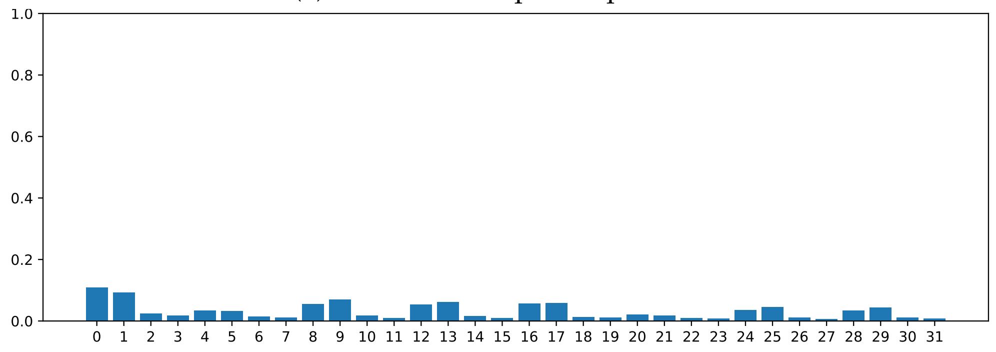
(b) Real output amplitudes

Figure 4.5: Output amplitudes of input state 2

Input state 3 - Output amplitudes



(a) Simulated output amplitudes



(b) Real output amplitudes

Figure 4.6: Output amplitudes of input state 3

4.2 Error analysis

Table 4.7 and table 4.8 show the sum of squared errors (SSE) and period of each conducted experiment. The sum of squared error was calculated for each individual experiment between the error-free simulated output amplitudes and the real output amplitudes. Period finding was performed both on the error-free simulated output amplitudes in order to get the correct period (real period) and on the real output amplitudes to get the period found in the experiment on a real quantum device (found period).

Both periods in table 4.7 and 4.8 are presented relative to their own state space size. The state space size is the number of possible classical outcomes and is equal to 2^n, where n is the number of qubits used in the computation. This means that the maximum period 2π will be interpreted as period 8 for three qubits and 32 for five qubits.

	State 1	State 2	State 3
SSE (10^{-3})	140	14	51
Real period	2	8	8
Found period	8	8	8

Figure 4.7: SSE and period from the three qubit circuit

	State 1	State 2	State 3
SSE (10^{-3})	390	9.2	51
Real period	2	32	32
Found period	32	32	32

Figure 4.8: SSE and period from the five qubit circuit

Chapter 5

Discussion

From the conducted experiments we can conclude that the sensitivity to noise is dependent on both the structure of the input state as well as on the number of quantum bits used in the computation. In some cases it is possible to distinguish and recover the correct result even though they have been distorted. In other cases the result is affected by the noise to such an extent that it is impossible to distinguish the correct solution. In these cases we could try to run the computation more times in the hope of receiving a result with greater distinction between correct output states and states caused by random error. However, given a high enough number of computations this would defeat the main purpose with many quantum algorithms, namely the quantum speedup.

5.1 Quantum and discrete Fourier transform

Based on the results in section 4 we suspect that error tolerance of the quantum Fourier transforms decreases as the problem size increases. One possible explanation for this could stem from the difference in output amplitudes between the quantum Fourier transform and the discrete Fourier transform. Since the output of the quantum Fourier transform is another quantum mechanical state as shown by equation 2.23, the sum of the output amplitudes have to equal one. This means that every wrong answer caused by errors will affect the whole sample space. This makes correct answers appear to be less likely, which is not the case with the discrete Fourier transform.

This constraint leads to smaller output amplitudes as the number of sinusoidal functions increases, thereby making it harder to distinguish the correct output amplitudes from ones created by random noise. An example of this difference can be seen in figure 4.1 and figure 4.4. In the three qubit version the two sinusoidal functions $\sin(0x)$ and $\sin(4x)$ are easily distinguishable, which is not the case for the five qubit version where the sinusoidal functions $\sin(0x)$ and $\sin(16x)$ are indistinguishable from the surrounding noise.

Since we are only able to view a collapsed quantum state we have to run the same computation multiple times in order to get the distribution of output amplitudes. Because of this, for some states, the output amplitudes will spread more evenly and thereby provide lower amplitudes as the problem size increases. This could lead to longer running times in order to receive distinguishable results.

5.2 Amplitude threshold

The results from the conducted experiments indicates that different states differ in how susceptible they are to noise. For the given input states the trend is that the output of the smaller circuits better resembles the output of a perfect device than the larger circuits. For some experiments with the smaller circuits it is even possible to distinguish the correct result in the disturbed output state.

Input state one for the three qubit quantum Fourier transform is one example of this. Based on the output amplitudes it is possible to distinguish the correct result by eliminating the lower amplitudes, which in this case are caused by errors. This is however not possible for the five qubit quantum Fourier transform. In this case the errors have had a greater effect on the result and the amplitudes caused by error are in some cases larger than the correct ones.

One naive approach for getting correct readouts from distorted states could be to introduce an amplitude threshold. This would work as a noise filter, where all amplitudes under a given threshold is filtered out from the output amplitudes. A method like this would be successful for cases where the found solution is easy to validate, which is the case for Shor's algorithm. This approach would work well for the three qubit quantum Fourier transform on input state one, where all amplitudes caused by noise are lower than the correct ones. For the larger five qubit quantum Fourier transform it is however not possible to recover from the distorted output amplitudes since the correct amplitudes are indistinguishable from the incorrect ones by simply analyzing the amplitude levels.

Although great improvements have been made in the last decades in reducing decoherence times in quantum computers, the results clearly show that the current level of noise is to substantial to get a correct result for any non-trivial problem instance. This clearly highlights the need for other means of reducing the effects of noise in quantum computations. One solution to this problem could be that instead of working to reduce the noise levels, try to correct errors as they occur. Error correcting techniques provides a theoretical framework to achieve this.

5.3 Logical qubits and error correcting codes

Based on the results from the conducted experiments we see that error correcting techniques are needed in order to be able to get a correct result. Logical qubits and quantum error correcting codes (QECC) could be a way to achieve better performance in regards to error. However, current error correcting techniques require the addition of more quantum bits in a circuit to achieve error correction for any non-trivial problem instances, something that is currently a scarce resource.

5.4 Improvements and possible future work

This work has barely scratched the surface of the current state of the field and has not accounted for any error correcting techniques when conducting the experiments. Given that the number of quantum bits available in a quantum device is likely to rise in the coming years a natural evolution of this work would be incorporating error correcting techniques when conducting the experiments.

Chapter 6

Conclusion

We have performed the quantum Fourier transform on a real quantum device (IBMQX4) and compared it to an error free simulated outcome. We were able to distinguish the correct period of the quantum Fourier transform only for the most trivial of input states. The result further shows that the decoherence error for any bigger quantum states than the most trivial one is to high to distinguish a correct result. We conclude that it is currently a necessity to use logical qubits and quantum error correcting codes in order to utilize the potential of quantum computing to solve any non-trivial problem.

Bibliography

Compiling and running a quantum program. (n.d.). QISKit GitHub repository, commit: c96d95d8c9defb7e1630b19a48163f4bd22ba44c. Retrieved April 20, 2018, from https://github.com/QISKit/qiskit-tutorial/blob/42ce5793dcc9e862e56a763dbee9a9222e10e6c1/appendix/advanced_qiskit/compiling_and_running.ipynb

Cross, A. W., Bishop, L. S., Smolin, J. A., & Gambetta, J. M. (2017). Open quantum assembly language. *ArXiv e-prints*. arXiv: 1707.03429 `[quant-ph]`. Retrieved May 10, 2018, from https://arxiv.org/pdf/1707.03429.pdf

de Wolf, R. (2011). The classical and quantum fourier transform. Centrum Wiskunde & Informatica. Retrieved March 24, 2018, from https://homepages.cwi.nl/~rdewolf/qfourierintro.pdf

Hill, C., & Viamontes, G. F. (2008). Operator imprecision and scaling of shor's algorithm. *ArXiv e-prints*. arXiv: 0804.3076 `[quant-ph]`. Retrieved May 10, 2018, from https://arxiv.org/pdf/0804.3076.pdf

Nielsen, M. A., & Chuang, I. (2010). *Quantum computation and quantum information*. Cambridge University Press Cambridge.

Noise and Quantum Computation. (n.d.). Retrieved April 21, 2018, from http://pyquil.readthedocs.io/en/latest/noise.html

Smith, R. S., Curtis, M. J., & Zeng, W. J. (2016). A practical quantum instruction set architecture. arXiv: 1608.03355 `[quant-ph]`. Retrieved May 10, 2018, from https://arxiv.org/pdf/1608.03355.pdf

Yanofsky, N. S., & Mannucci, M. A. (2008). *Quantum computing for computer scientists*. Cambridge University Press Cambridge.

Zulehner, A., Paler, A., & Wille, R. (2018). An efficient methodology for mapping quantum circuits to the ibm qx architectures. *ArXiv e-prints*. arXiv: 1712.04722 `[quant-ph]`. Retrieved April 20, 2018, from https://arxiv.org/pdf/1712.04722.pdf

Appendices

Appendix A

Source code

```
# coding: utf-8

from functools import reduce
from itertools import product
from math import gcd, pi, sqrt
from matplotlib import pyplot
from pyquil.api import QVMConnection
from pyquil.gates import *
from pyquil.quil import Program
from qiskit import QuantumProgram, backends
from qiskit.tools.visualization import plot_histogram

import argparse
import numpy as np
import sys


def pyquil_qft(number_of_qubits):
    """Return a PyQuil program implementing the Quantum
     ↪  Fourier Transform."""
    program = Program()

    for i in reversed(range(number_of_qubits)):
        for j in range((number_of_qubits - 1) - i):
            program.inst(CPHASE(
                pi/2.0**((number_of_qubits - 1) - i - j),
                i,
                (number_of_qubits - 1) - j
            ))

        program.inst(H(i))

    for i in range(int(number_of_qubits / 2)):
        program.inst(SWAP(i, (number_of_qubits - 1) - i))

    return program

def pyquil_run(number_of_qubits, state):
    qvm = QVMConnection()

    if number_of_qubits == 3:
        if state == 1:
            program = Program(I(0), H(1), H(2), I(3), I(4))
        elif state == 2:
            program = Program(I(0), I(1), X(2), I(3), I(4))
        else:
            program = Program(I(0), I(1), RY(pi / 2 * 0.9,
             ↪  2), I(3), I(4))
    else:
        if state == 1:
            program = Program(I(0), H(1), H(2), H(3), H(4))
        elif state == 2:
            program = Program(I(0), I(1), I(2), I(3), X(4))
        else:
            program = Program(I(0), I(1), I(2), I(3), RY(pi /
             ↪  2 * 0.9, 4))

    # Perform the QFT without error to get the correct output
     ↪  state
    outcome_state = qvm.wavefunction(program +
     ↪  pyquil_qft(number_of_qubits))
    return outcome_state.get_outcome_probs()


def qiskit_qft(q_circ, q_reg, number_of_qubits):
    """Define the Quantum Fourier Transform """
    for i in reversed(range(number_of_qubits)):
        for j in range((number_of_qubits - 1) - i):
            q_circ.cu1(
                pi/2.0**((number_of_qubits - 1) - i - j),
                q_reg[i],
                q_reg[(number_of_qubits - 1) - j]
            )

        q_circ.h(q_reg[i])

    for i in range(int(number_of_qubits / 2)):
        q_circ.swap(q_reg[i], q_reg[(number_of_qubits - 1) -
        ↪  i])

def qiskit_run(api_key, number_of_qubits, state, shots):
    prog = QuantumProgram()
    prog.set_api(api_key,
    ↪   "https://quantumexperience.ng.bluemix.net/api")

    q_reg = prog.create_quantum_register("q", 5)
    c_reg = prog.create_classical_register("c", 5)
    q_circ = q_circ = prog.create_circuit("qft", [q_reg],
    ↪   [c_reg])

    if number_of_qubits == 3:
        if state == 1:
            q_circ.h(q_reg[1]); q_circ.h(q_reg[2])
        elif state == 2:
            q_circ.x(q_reg[2])
        else:
            q_circ.ry(pi / 2 * 0.9, q_reg[2])
    else:
        if state == 1:
            q_circ.h(q_reg[1]); q_circ.h(q_reg[2]);
            q_circ.h(q_reg[2]); q_circ.h(q_reg[4]);
        elif state == 2:
            q_circ.x(q_reg[4])
        else:
            q_circ.ry(pi / 2 * 0.9, q_reg[4])

    qiskit_qft(q_circ, q_reg, number_of_qubits)
    [q_circ.measure(q_reg[i], c_reg[i]) for i in range(5)]
```

```python

    backend = backends.get_backend_instance("ibmqx4")
    coupling_map = backend.configuration["coupling_map"]
    result = prog.execute(["qft"], "ibmqx4", max_credits=5,
    ↪  timeout=6000, shots=shots, coupling_map=coupling_map)
    outcome_result = result.get_counts("qft")

    # Fill in the missing binary value outcomes
    binary_values = ["{0:b}".format(x).zfill(5) for x in
    ↪  range(2**5)]
    tmp_res = dict(zip(binary_values, [0] * 2**5))
    tmp_res.update(outcome_result)
    outcome_result = {key: value / shots for (key, value) in
    ↪  tmp_res.items()}

    return outcome_result


def sse_error(correct_states, disturbed_states):
    """Returns the SSE error between the two provided
    ↪  states."""
    correct_values = [correct_states[key] for key in
    ↪  sorted(correct_states.keys())]
    disturbed_values = [disturbed_states[key] for key in
    ↪  sorted(disturbed_states.keys())]

    correct_values = np.array(correct_values)
    disturbed_values = np.array(disturbed_values)

    return np.sum(np.square(np.subtract(correct_values,
    ↪  disturbed_values)))


def find_period(number_of_qubits, outcome_probs,
↪   amplitude_threshold=0):
    """Return the period of the outcome probabilities."""
    # Unzip the values
    _, y = zip(*sorted(outcome_probs.items()))
    periods = []

    # Select periods with higher amplitude than the amplitude
    ↪   threshold.
    # Amplitudes on zero is not interesting as it does not
    ↪   affect the period.
    for i, value in enumerate(y):
        if value > amplitude_threshold and i > 0:
            periods.append(i)

    # Find greatest common denominator
    denominator = reduce(lambda a, b: gcd(a, b), periods)

    period = 2**number_of_qubits / denominator
    return period


def plot_probs(title, number_of_qubits, state,
↪   outcome_probs):
    with open("{n}_qubit_state_{s}_{t}_output.csv".format(
        n=number_of_qubits, s=state, t=title), "w") as csv:
        csv.write("State, Outcome amplitude\n")
        [csv.write("{} ({}), {:.3}\n".format(int(x, 2), x,
        ↪   y)) for (x, y) in outcome_probs.items()]

    # Unzip the values
    x, y = zip(*sorted(outcome_probs.items()))
    axes = pyplot.gca()
    axes.set_ylim([0, 1])
    pyplot.xticks(range(len(x)), [int(t, 2) for t in x])
    pyplot.bar(range(len(x)), y)
    pyplot.savefig("{n}_bits_state_{s}_{t}_out.png".format(
        n=number_of_qubits,
        s=state,
        t=title)
    )
    pyplot.show()


def main(api_key, number_of_qubits=3, state=1, shots=8192):
    # Get the simulated and real results
    pyquil_res = pyquil_run(number_of_qubits, state)
```

```python
    qiskit_res = qiskit_run(api_key, number_of_qubits, state,
     ↪  shots)

    # Remove q3 and q4 since they're zero
    if number_of_qubits == 3:
        pyquil_res = {k:v for k, v in pyquil_res.items() if
         ↪  k.startswith("00")}
        qiskit_res = {k:v for k, v in qiskit_res.items() if
         ↪  k.startswith("00")}

    sse = sse_error(pyquil_res, qiskit_res)
    pyquil_period = find_period(number_of_qubits, pyquil_res)
    qiskit_period = find_period(number_of_qubits, qiskit_res)

    # Plot the simulated and real results
    plot_probs("simulated", number_of_qubits, state,
     ↪  pyquil_res)
    plot_probs("real", number_of_qubits, state, qiskit_res)

    # Print the sse and period values to csv
    print_sse_and_period(number_of_qubits, state, sse,
     ↪  pyquil_period, qiskit_period)
```

Appendix B

Simulated state amplitudes

State	Amplitude
0 (000)	0.25
1 (001)	0.0
2 (010)	0.25
3 (011)	0.0
4 (100)	0.25
5 (101)	0.0
6 (110)	0.25
7 (111)	0.0

(a) Input amplitudes

State	Amplitude
0 (000)	0.5
1 (001)	0.0
2 (010)	0.0
3 (011)	0.0
4 (100)	0.5
5 (101)	0.0
6 (110)	0.0
7 (111)	0.0

(b) Output amplitudes

Figure B.1: Simulated state amplitudes for three qubit quantum Fourier transform on input state 1.

State	Amplitude
0 (000)	0.0
1 (001)	0.0
2 (010)	0.0
3 (011)	0.0
4 (100)	1.0
5 (101)	0.0
6 (110)	0.0
7 (111)	0.0

(a) Input amplitudes

State	Amplitude
0 (000)	0.125
1 (001)	0.125
2 (010)	0.125
3 (011)	0.125
4 (100)	0.125
5 (101)	0.125
6 (110)	0.125
7 (111)	0.125

(b) Output amplitudes

Figure B.2: Simulated state amplitudes for three qubit quantum Fourier transform on input state 2.

State	Amplitude
0 (000)	0.578
1 (001)	0.0
2 (010)	0.0
3 (011)	0.0
4 (100)	0.422
5 (101)	0.0
6 (110)	0.0
7 (111)	0.0

(a) Input amplitudes

State	Amplitude
0 (000)	0.248
1 (001)	0.00154
2 (010)	0.248
3 (011)	0.00154
4 (100)	0.248
5 (101)	0.00154
6 (110)	0.248
7 (111)	0.00154

(b) Output amplitudes

Figure B.3: Simulated state amplitudes for three qubit quantum Fourier transform on input state 3.

State	Amplitude
0 (00000)	0.0625
1 (00001)	0.0
2 (00010)	0.0625
3 (00011)	0.0
4 (00100)	0.0625
5 (00101)	0.0
6 (00110)	0.0625
7 (00111)	0.0
8 (01000)	0.0625
9 (01001)	0.0
10 (01010)	0.0625
11 (01011)	0.0
12 (01100)	0.0625
13 (01101)	0.0
14 (01110)	0.0625
15 (01111)	0.0
16 (10000)	0.0625
17 (10001)	0.0
18 (10010)	0.0625
19 (10011)	0.0
20 (10100)	0.0625
21 (10101)	0.0
22 (10110)	0.0625
23 (10111)	0.0
24 (11000)	0.0625
25 (11001)	0.0
26 (11010)	0.0625
27 (11011)	0.0
28 (11100)	0.0625
29 (11101)	0.0
30 (11110)	0.0625
31 (11111)	0.0

(a) Input amplitudes

State	Amplitude
0 (00000)	0.5
1 (00001)	0.0
2 (00010)	0.0
3 (00011)	0.0
4 (00100)	0.0
5 (00101)	0.0
6 (00110)	0.0
7 (00111)	0.0
8 (01000)	0.0
9 (01001)	0.0
10 (01010)	0.0
11 (01011)	0.0
12 (01100)	0.0
13 (01101)	0.0
14 (01110)	0.0
15 (01111)	0.0
16 (10000)	0.5
17 (10001)	0.0
18 (10010)	0.0
19 (10011)	0.0
20 (10100)	0.0
21 (10101)	0.0
22 (10110)	0.0
23 (10111)	0.0
24 (11000)	0.0
25 (11001)	0.0
26 (11010)	0.0
27 (11011)	0.0
28 (11100)	0.0
29 (11101)	0.0
30 (11110)	0.0
31 (11111)	0.0

(b) Output amplitudes

Figure B.4: Simulated state amplitudes for five qubit quantum Fourier transform on input state 1.

State	Amplitude
0 (00000)	0.0
1 (00001)	0.0
2 (00010)	0.0
3 (00011)	0.0
4 (00100)	0.0
5 (00101)	0.0
6 (00110)	0.0
7 (00111)	0.0
8 (01000)	0.0
9 (01001)	0.0
10 (01010)	0.0
11 (01011)	0.0
12 (01100)	0.0
13 (01101)	0.0
14 (01110)	0.0
15 (01111)	0.0
16 (10000)	1.0
17 (10001)	0.0
18 (10010)	0.0
19 (10011)	0.0
20 (10100)	0.0
21 (10101)	0.0
22 (10110)	0.0
23 (10111)	0.0
24 (11000)	0.0
25 (11001)	0.0
26 (11010)	0.0
27 (11011)	0.0
28 (11100)	0.0
29 (11101)	0.0
30 (11110)	0.0
31 (11111)	0.0

(a) Input amplitudes

State	Amplitude
0 (00000)	0.0312
1 (00001)	0.0312
2 (00010)	0.0312
3 (00011)	0.0312
4 (00100)	0.0312
5 (00101)	0.0312
6 (00110)	0.0312
7 (00111)	0.0312
8 (01000)	0.0312
9 (01001)	0.0312
10 (01010)	0.0312
11 (01011)	0.0312
12 (01100)	0.0312
13 (01101)	0.0312
14 (01110)	0.0312
15 (01111)	0.0312
16 (10000)	0.0312
17 (10001)	0.0312
18 (10010)	0.0312
19 (10011)	0.0312
20 (10100)	0.0312
21 (10101)	0.0312
22 (10110)	0.0312
23 (10111)	0.0312
24 (11000)	0.0312
25 (11001)	0.0312
26 (11010)	0.0312
27 (11011)	0.0312
28 (11100)	0.0312
29 (11101)	0.0312
30 (11110)	0.0312
31 (11111)	0.0312

(b) Output amplitudes

Figure B.5: Simulated state amplitudes for five qubit quantum Fourier transform on input state 2.

State	Amplitude
0 (00000)	0.578
1 (00001)	0.0
2 (00010)	0.0
3 (00011)	0.0
4 (00100)	0.0
5 (00101)	0.0
6 (00110)	0.0
7 (00111)	0.0
8 (01000)	0.0
9 (01001)	0.0
10 (01010)	0.0
11 (01011)	0.0
12 (01100)	0.0
13 (01101)	0.0
14 (01110)	0.0
15 (01111)	0.0
16 (10000)	0.422
17 (10001)	0.0
18 (10010)	0.0
19 (10011)	0.0
20 (10100)	0.0
21 (10101)	0.0
22 (10110)	0.0
23 (10111)	0.0
24 (11000)	0.0
25 (11001)	0.0
26 (11010)	0.0
27 (11011)	0.0
28 (11100)	0.0
29 (11101)	0.0
30 (11110)	0.0
31 (11111)	0.0

(a) Input amplitudes

State	Amplitude
0 (00000)	0.0621
1 (00001)	0.000385
2 (00010)	0.0621
3 (00011)	0.000385
4 (00100)	0.0621
5 (00101)	0.000385
6 (00110)	0.0621
7 (00111)	0.000385
8 (01000)	0.0621
9 (01001)	0.000385
10 (01010)	0.0621
11 (01011)	0.000385
12 (01100)	0.0621
13 (01101)	0.000385
14 (01110)	0.0621
15 (01111)	0.000385
16 (10000)	0.0621
17 (10001)	0.000385
18 (10010)	0.0621
19 (10011)	0.000385
20 (10100)	0.0621
21 (10101)	0.000385
22 (10110)	0.0621
23 (10111)	0.000385
24 (11000)	0.0621
25 (11001)	0.000385
26 (11010)	0.0621
27 (11011)	0.000385
28 (11100)	0.0621
29 (11101)	0.000385
30 (11110)	0.0621
31 (11111)	0.000385

(b) Output amplitudes

Figure B.6: Simulated state amplitudes for five qubit quantum Fourier transform on input state 3.

Appendix C

State amplitude comparison

State	Amplitude
0 (00000)	0.5
1 (00001)	0.0
2 (00010)	0.0
3 (00011)	0.0
4 (00100)	0.5
5 (00101)	0.0
6 (00110)	0.0
7 (00111)	0.0

(a) Simulated output amplitudes

State	Amplitude
0 (00000)	0.386
1 (00001)	0.0656
2 (00010)	0.0359
3 (00011)	0.0151
4 (00100)	0.156
5 (00101)	0.0254
6 (00110)	0.0336
7 (00111)	0.00781

(b) Real output amplitudes

Figure C.1: State amplitudes for three qubit quantum Fourier transform on input state 1.

State	Amplitude
0 (00000)	0.125
1 (00001)	0.125
2 (00010)	0.125
3 (00011)	0.125
4 (00100)	0.125
5 (00101)	0.125
6 (00110)	0.125
7 (00111)	0.125

(a) Simulated output amplitudes

State	Amplitude
0 (00000)	0.0999
1 (00001)	0.107
2 (00010)	0.0938
3 (00011)	0.109
4 (00100)	0.0699
5 (00101)	0.0699
6 (00110)	0.0699
7 (00111)	0.0752

(b) Real output amplitudes

Figure C.2: State amplitudes for three qubit quantum Fourier transform on input state 2.

State	Amplitude
0 (00000)	0.248
1 (00001)	0.00154
2 (00010)	0.248
3 (00011)	0.00154
4 (00100)	0.248
5 (00101)	0.00154
6 (00110)	0.248
7 (00111)	0.00154

(a) Simulated output amplitudes

State	Amplitude
0 (00000)	0.236
1 (00001)	0.0525
2 (00010)	0.166
3 (00011)	0.0283
4 (00100)	0.113
5 (00101)	0.0201
6 (00110)	0.1
7 (00111)	0.0186

(b) Real output amplitudes

Figure C.3: State amplitudes for three qubit quantum Fourier transform on input state 3.

State	Amplitude
0 (00000)	0.5
1 (00001)	0.0
2 (00010)	0.0
3 (00011)	0.0
4 (00100)	0.0
5 (00101)	0.0
6 (00110)	0.0
7 (00111)	0.0
8 (01000)	0.0
9 (01001)	0.0
10 (01010)	0.0
11 (01011)	0.0
12 (01100)	0.0
13 (01101)	0.0
14 (01110)	0.0
15 (01111)	0.0
16 (10000)	0.5
17 (10001)	0.0
18 (10010)	0.0
19 (10011)	0.0
20 (10100)	0.0
21 (10101)	0.0
22 (10110)	0.0
23 (10111)	0.0
24 (11000)	0.0
25 (11001)	0.0
26 (11010)	0.0
27 (11011)	0.0
28 (11100)	0.0
29 (11101)	0.0
30 (11110)	0.0
31 (11111)	0.0

(a) Simulated output amplitudes

State	Amplitude
0 (00000)	0.116
1 (00001)	0.0912
2 (00010)	0.0264
3 (00011)	0.0183
4 (00100)	0.0336
5 (00101)	0.0294
6 (00110)	0.0137
7 (00111)	0.0103
8 (01000)	0.0514
9 (01001)	0.0653
10 (01010)	0.0175
11 (01011)	0.0106
12 (01100)	0.0581
13 (01101)	0.0665
14 (01110)	0.0137
15 (01111)	0.00842
16 (10000)	0.0524
17 (10001)	0.0468
18 (10010)	0.0122
19 (10011)	0.0094
20 (10100)	0.0175
21 (10101)	0.0166
22 (10110)	0.00842
23 (10111)	0.00598
24 (11000)	0.0331
25 (11001)	0.0511
26 (11010)	0.0123
27 (11011)	0.00659
28 (11100)	0.0344
29 (11101)	0.0438
30 (11110)	0.0125
31 (11111)	0.00659

(b) Real output amplitudes

Figure C.4: State amplitudes for five qubit quantum Fourier transform on input state 1.

State	Amplitude
0 (00000)	0.0312
1 (00001)	0.0312
2 (00010)	0.0312
3 (00011)	0.0312
4 (00100)	0.0312
5 (00101)	0.0312
6 (00110)	0.0312
7 (00111)	0.0312
8 (01000)	0.0312
9 (01001)	0.0312
10 (01010)	0.0312
11 (01011)	0.0312
12 (01100)	0.0312
13 (01101)	0.0312
14 (01110)	0.0312
15 (01111)	0.0312
16 (10000)	0.0312
17 (10001)	0.0312
18 (10010)	0.0312
19 (10011)	0.0312
20 (10100)	0.0312
21 (10101)	0.0312
22 (10110)	0.0312
23 (10111)	0.0312
24 (11000)	0.0312
25 (11001)	0.0312
26 (11010)	0.0312
27 (11011)	0.0312
28 (11100)	0.0312
29 (11101)	0.0312
30 (11110)	0.0312
31 (11111)	0.0312

(a) Simulated output amplitudes

State	Amplitude
0 (00000)	0.0875
1 (00001)	0.0671
2 (00010)	0.0426
3 (00011)	0.0375
4 (00100)	0.0275
5 (00101)	0.0243
6 (00110)	0.0234
7 (00111)	0.0153
8 (01000)	0.0414
9 (01001)	0.0514
10 (01010)	0.0304
11 (01011)	0.0144
12 (01100)	0.0439
13 (01101)	0.0468
14 (01110)	0.0293
15 (01111)	0.0157
16 (10000)	0.0465
17 (10001)	0.0458
18 (10010)	0.0294
19 (10011)	0.0183
20 (10100)	0.0171
21 (10101)	0.0186
22 (10110)	0.0157
23 (10111)	0.00928
24 (11000)	0.0286
25 (11001)	0.0397
26 (11010)	0.0236
27 (11011)	0.0129
28 (11100)	0.0312
29 (11101)	0.0339
30 (11110)	0.022
31 (11111)	0.00891

(b) Real output amplitudes

Figure C.5: State amplitudes for five qubit quantum Fourier transform on input state 2.

State	Amplitude
0 (00000)	0.0621
1 (00001)	0.000385
2 (00010)	0.0621
3 (00011)	0.000385
4 (00100)	0.0621
5 (00101)	0.000385
6 (00110)	0.0621
7 (00111)	0.000385
8 (01000)	0.0621
9 (01001)	0.000385
10 (01010)	0.0621
11 (01011)	0.000385
12 (01100)	0.0621
13 (01101)	0.000385
14 (01110)	0.0621
15 (01111)	0.000385
16 (10000)	0.0621
17 (10001)	0.000385
18 (10010)	0.0621
19 (10011)	0.000385
20 (10100)	0.0621
21 (10101)	0.000385
22 (10110)	0.0621
23 (10111)	0.000385
24 (11000)	0.0621
25 (11001)	0.000385
26 (11010)	0.0621
27 (11011)	0.000385
28 (11100)	0.0621
29 (11101)	0.000385
30 (11110)	0.0621
31 (11111)	0.000385

(a) Simulated output amplitudes

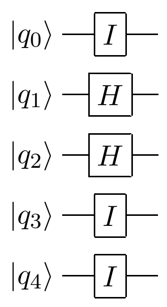
State	Amplitude
0 (00000)	0.108
1 (00001)	0.0923
2 (00010)	0.0234
3 (00011)	0.0171
4 (00100)	0.0336
5 (00101)	0.0316
6 (00110)	0.0133
7 (00111)	0.0104
8 (01000)	0.0548
9 (01001)	0.0695
10 (01010)	0.0173
11 (01011)	0.00854
12 (01100)	0.0533
13 (01101)	0.061
14 (01110)	0.0155
15 (01111)	0.00879
16 (10000)	0.0562
17 (10001)	0.0582
18 (10010)	0.0132
19 (10011)	0.0106
20 (10100)	0.0199
21 (10101)	0.0172
22 (10110)	0.00842
23 (10111)	0.00745
24 (11000)	0.0349
25 (11001)	0.0452
26 (11010)	0.0107
27 (11011)	0.00647
28 (11100)	0.0328
29 (11101)	0.0428
30 (11110)	0.01
31 (11111)	0.00732

(b) Real output amplitudes

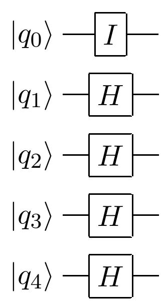
Figure C.6: State amplitudes for five qubit quantum Fourier transform on input state 3.

Appendix D

Input state circuits

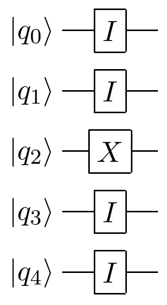


(a) Three qubit version

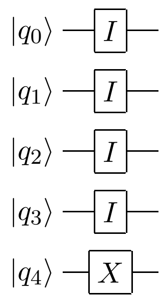


(b) Five qubit version

Figure D.1: Circuits used for setting up input state 1.

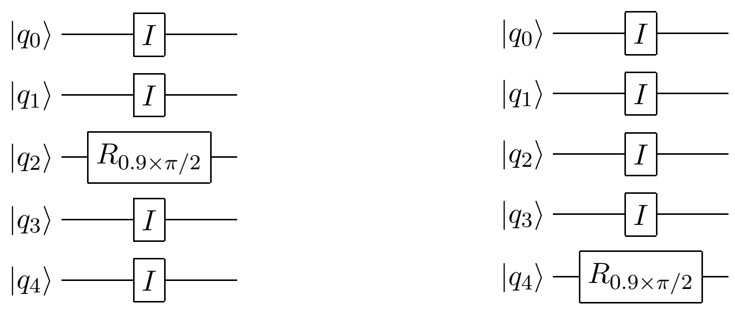


(a) Three qubit version



(b) Five qubit version

Figure D.2: Circuits used for setting up input state 2.



(a) Three qubit version (b) Five qubit version

Figure D.3: Circuits used for setting up input state 3.