
Σχεδίαση κβαντικών κυκλωμάτων με χρήση του κβαντικού
μετασχηματισμού Fourier

Πτυχιακή Εργασία



*Τμήμα Πληροφορικής
Ιόνιο Πανεπιστήμιο*

Στοιχεία Φοιτητή:
Σοφία Τσιβούλα Π2018130

Κέρκυρα, 2022

Επιβλέπων καθηγητής:
Θεόδωρος Ανδρόνικος

Επιτροπή:
Θεμιστοκλής Έξαρχος
Μιχαήλ Στεφανιδάκης

Περιεχόμενα

1	Περίληψη	4
2	Εισαγωγή	5
3	Εισαγωγή στα κβαντικά κυκλώματα	8
3.1	Βασικοί όροι	8
3.1.1	Συστήματα δύο καταστάσεων	8
3.1.2	Διανύσματα bra και ket	10
3.1.3	Κβαντικό δυαδικό ψηφίο (qubit)	11
3.1.4	Κβαντικές καταστάσεις	12
3.2	Κβαντικές πύλες	13
3.3	Πύλες που δρουν σε ένα qubit	13
3.4	Πύλες που δρουν σε δύο qubits	16
4	Κβαντικός μετασχηματισμός Fourier	18
4.1	Εισαγωγή	18
4.2	Περιγραφή	18
4.3	Μετασχηματισμός φάσης	20
4.4	Μετατροπή βάσης	20
4.4.1	Κωδικοποίηση βάσης	20
4.5	Λογική	21
4.6	Κβαντικό κύκλωμα	21
4.6.1	Αντίστροφος κβαντικός μετασχηματισμός Fourier	24
4.7	Μαθηματική περιγραφή	25
4.8	Κώδικας	27
5	Αλγόριθμος Deutsch Jozsa	31
5.1	Περιγραφή	31
5.2	Υλοποίηση σε κλασικό υπολογιστή	31
5.2.1	Η $f(x)$ είναι σταθερή	32
5.2.2	Η $f(x)$ είναι ισορροπημένη	32
5.3	Υλοποίηση σε κβαντικό υπολογιστή	33
5.4	Περιγραφή αλγορίθμου	33
5.5	Περιγραφή των Oracles	33
5.5.1	Constant Oracle	33
5.5.2	Balanced Oracle	34
5.6	Μαθηματική περιγραφή	35
5.7	Κώδικας	37

6	Αλγόριθμος Bernstein Vazirani	41
6.1	Εισαγωγή	41
6.2	Περιγραφή προβλήματος	41
6.3	Υλοποίηση σε κλασικό υπολογιστή	41
6.4	Παράδειγμα υλοποίησης σε κλασικό υπολογιστή	42
6.5	Υλοποίηση σε κβαντικό υπολογιστή	44
6.6	Μαθηματική περιγραφή	45
6.7	Κώδικας	46
7	Χρήση κβαντικού μετασχηματισμού Fourier στον αλγόριθμο των Deutsch Jozsa	51
7.1	Ολοκληρωμένος κώδικας	52
7.2	Περίπτωση 1	56
7.2.1	Constant Oracle	56
7.2.2	Balanced Oracle	58
7.3	Περίπτωση 2	60
7.3.1	Constant Oracle	60
7.3.2	Balanced Oracle	63
7.4	Περίπτωση 3	65
7.4.1	Constant Oracle	65
7.4.2	Balanced Oracle	69
8	Χρήση κβαντικού μετασχηματισμού Fourier στον αλγόριθμο των Bernstein Vazirani	71
8.1	Πλήρες κώδικας	71
8.2	Περιγραφή κώδικα για τις τρεις διαφορετικές περιπτώσεις	74
8.3	Περίπτωση 1: Το μήκος της συμβολοσειράς s έχει μήκος 2 qubits	82
8.4	Περίπτωση 2: Το μήκος της συμβολοσειράς s έχει μήκος 3 qubits	84
8.5	Περίπτωση 3: Το μήκος της συμβολοσειράς s έχει μήκος 4+ qubits	87

1. Περίληψη

Η παρούσα εργασία επικεντρώνεται στην χρήση του κβαντικού μετασχηματισμού Fourier σε κλασικούς κβαντικούς αλγόριθμους. Οι αλγόριθμοι στους οποίους έχει πραγματοποιηθεί η μετατροπή είναι οι αλγόριθμοι των Deutsch Jozsa και των Bernstein Vazirani. Στόχος της εργασίας είναι η δημιουργία κβαντικών κυκλωμάτων με τους εν λόγω αλγόριθμους, ο πειραματισμός και ο σχολιασμός των αποτελεσμάτων που προέκυψαν. Η εργασία χωρίζεται σε δύο βασικά μέρη, το πρώτο μέρος που αφορά το ερευνητικό υπόβαθρο και το δεύτερο μέρος που αφορά τον πειραματισμό και την δημιουργία κυκλωμάτων. Στο πρώτο μέρος, περιγράφονται αναλυτικά τα εργαλεία και οι αλγόριθμοι πάνω στους οποίους πραγματοποιήθηκε η δημιουργία κβαντικών κυκλωμάτων, με παραδείγματα, με μαθηματική περιγραφή και με χρήση κώδικα σε γλώσσα προγραμματισμού Python. Το δεύτερο μέρος της εργασίας, επικεντρώνεται στην κατασκευή των κυκλωμάτων, στην κατασκευή νέων αλγορίθμων και στις δυσκολίες που βρέθηκαν κατά την διαδικασία παρασκευής των κυκλωμάτων. Τα κυκλώματα κατασκευάστηκαν σε γλώσσα προγραμματισμού Python, με χρήση της βιβλιοθήκης Qiskit για την προσομοίωση του κβαντικού υπολογιστή.

Abstract

The present work uses the Quantum Fourier transform in classical quantum algorithms. The algorithms in which the conversion has taken place are the algorithms of Deutsch Jozsa and Bernstein Vazirani. This work aims to create quantum circuits with these algorithms, experiment, and comment on the obtained results. The work is divided into two main parts, the first part focuses on the research background, and the second part focuses on the experimentation and the creation of circuits. The first part shows the tools and the algorithms on which the quantum circuits were created and are described in great detail. This section also shows examples and mathematical descriptions and shows code in a Python programming language to make the reader understand the concepts more accessible. The second part of the work focuses on constructing the circuits, the construction of new algorithms and the difficulties found during the process. The circuits were built in Python programming language, using the Qiskit library to simulate the quantum computer.

2. Εισαγωγή

Η παρούσα πτυχιακή εργασία έχει ως στόχο την χρήση του κβαντικού μετασχηματισμού Fourier [20] σε κλασικούς κβαντικούς αλγορίθμους. Στόχος της εργασίας αποτελεί η εξοικείωση με τα κβαντικά κυκλώματα και με την δημιουργία κβαντικών αλγορίθμων. Μέσα από την παρούσα εργασία, δίνεται η δυνατότητα εκμάθησης δύο πολύ βασικών κβαντικών αλγορίθμων, των Deutsch Jozsa [8] και των Bernstein Vazirani [7] και η μετάφρασή τους σε εκτελέσιμο κώδικα. Η περιγραφή του κώδικα και του μαθηματικού υποβάθρου, σε συνδυασμό με τα διαγράμματα αποτελεί έναν οδηγό για την εισαγωγή στον κβαντικό υπολογισμό και ειδικότερα, στην κατασκευή κβαντικών κυκλωμάτων. Για την ανάγνωση της εργασίας απαιτείται βασική γνώση εννοιών του κβαντικού υπολογισμού και των κλασικών αλλά και κβαντικών κυκλωμάτων, διότι θεωρούνται ως προαπαιτούμενη γνώση για την πλήρης κατανόηση της εργασίας.

Η ανάγκη για υπολογιστικά συστήματα υψηλών επιδόσεων, ικανά να λύνουν πολύ δύσκολα προβλήματα, οδήγησε την ερευνητική κοινότητα να στραφεί στη χρήση μη συμβατικών υπολογιστών. Αναμφίβολα, οι περισσότεροι υποσχόμενοι μη συμβατικοί υπολογιστές είναι οι κβαντικοί υπολογιστές που υπόσχονται να δώσουν δυνατότητες που είναι αδιανόητες για τους συμβατικούς υπολογιστές.

Οι κβαντικοί υπολογιστές που είναι διαθέσιμοι για χρήση σήμερα διαφοροποιούνται ανάλογα με το μοντέλο κβαντικού υπολογισμού που ακολουθούν. Τα κυριότερα μοντέλα κβαντικού υπολογισμού είναι αυτό της κβαντικής ανόπτησης και αυτό του κυκλωματικού μοντέλου. Οι κβαντικοί υπολογιστές που χρησιμοποιούν την κβαντική ανόπτηση έχουν τη δυνατότητα να επιλύουν ιδιαίτερα δύσκολα προβλήματα υπολογιστικής βελτιστοποίησης [16], [19]. Οι κβαντικοί υπολογιστές που βασίζονται στο κυκλωματικό μοντέλο είναι πιο ευέλικτοι, αλλά στην παρούσα φάση περιορίζονται από τον σχετικά μικρό αριθμό qubits που έχουν. Οι δυνατότητές τους τους επιτρέπουν: (i) να χρησιμοποιηθούν για να εκτελέσουν πρωτοποριακούς κβαντικούς αλγόριθμους, ανώτερους από τους αντίστοιχους κλασικούς, (ii) να υλοποιήσουν πρωτόκολλα που εγγυώνται την ασφάλεια και την κβαντική κρυπτογράφηση [1], και (iii) να προσομοιώσουν κβαντικά παίγνια που επεκτείνουν τα κλασικά παίγνια [2], [3], [13], [4], [12].

Θα ήταν παράλειψη αν σε αυτή τη σύντομη εισαγωγή δεν γινόταν αναφορά σε μία άλλη κυρίαρχη τάση στο μη συμβατικό υπολογισμό, και συγκεκριμένα, στην τάση που βασίζεται και εμπνέεται από τον μοριακό και γενικότερο τον

βιολογικό υπολογισμό. Ο μοριακός και ο βιο-εμπνευσμένος υπολογισμός έχουν επίσης δώσει καινοτόμα και ελπιδοφόρα αποτελέσματα, ανοίγοντας νέες προοπτικές στην περιοχή του μη συμβατικού υπολογισμού [10], [11].

Η δομή της εργασίας χωρίζεται σε κεφάλαια ανάλογα τις θεματικές που αναλύει το κάθε κεφάλαιο. Τα κεφάλαια της εργασίας αναγράφονται ονομαστικά παρακάτω:

1. Εισαγωγή στα κβαντικά συστήματα
2. Κβαντικός μετασχηματισμός Fourier
3. Αλγόριθμος Deutsch Jozsa
4. Αλγόριθμος Bernstein Vazirani
5. Χρήση κβαντικού μετασχηματισμού Fourier στον αλγόριθμο των Deutsch Jozsa
6. Χρήση κβαντικού μετασχηματισμού Fourier στον αλγόριθμο Bernstein Vazirani

Σε κάθε ένα από τα αναφερόμενα κεφάλαια αναφέρεται ο εν λόγω αλγόριθμος ή η εν λόγω μεθοδολογία.

Στο κεφάλαιο “Εισαγωγή στα κβαντικά συστήματα” περιγράφονται συνοπτικά βασικοί όροι της κβαντικής υπολογιστικής και αναφέρεται το επιστημονικό υπόβαθρο. Εξηγούνται βασικοί όροι, περιγράφονται οι κβαντικές πύλες που χρησιμοποιούνται στην εν λόγω εργασία και αναφέρονται παραδείγματα χρήσης των πυλών μέσω κώδικα και διαγραμμάτων. Αποτελεί εισαγωγικό κεφάλαιο στον κβαντικό υπολογισμό και περιγράφει όρους που θα χρειαστούν για την κατανόηση της υπόλοιπης εργασίας.

Στο επόμενο κεφάλαιο με τίτλο “Κβαντικός μετασχηματισμός Fourier” γίνεται περιγραφή του κβαντικού μετασχηματισμού Fourier για την αλλαγή κβαντικής βάσης όπου περιγράφεται η μεθοδολογία που ακολουθήθηκε στην εργασία. Παρουσιάζονται επίσης τρόποι υλοποίησης σε κώδικα, με χρήση της γλώσσας προγραμματισμού Python καθώς και αναλύεται μαθηματικά η μεθοδολογία. Στο κεφάλαιο αυτό επίσης, επεξηγείται ο τρόπος μετασχηματισμού ενός κβαντικού κυκλώματος από την βάση z σε βάση Fourier καθώς και ο τρόπος επαναφοράς του κυκλώματος στην αρχική βάση υπολογισμού, την z .

Έπειτα, περιγράφονται οι αλγόριθμοι των Deutsch Jozsa και των Bernstein Vazirani που θα χρησιμοποιηθούν στο δεύτερο μέρος της εργασίας ως οι βασικοί αλγόριθμοι που θα εφαρμοστεί ο κβαντικός μετασχηματισμός Fourier. Για κάθε ένα από αυτούς τους αλγορίθμους, θα γίνει περιγραφή του προβλήματος που επιλύουν, της πολυπλοκότητας τους και του ίδιου του αλγορίθμου. Η περιγραφή του αλγορίθμου θα πραγματοποιηθεί σε τρεις φάσεις:

- την φάση της επεξήγησης, όπου θα παρουσιαστούν αναλυτικά τα βήματα υλοποίησης
- την φάση του κώδικα, όπου θα αναλυθεί ο κώδικας υλοποίησης σε γλώσσα προγραμματισμού Python με χρήση της βιβλιοθήκης Qiskit [5] που χρησιμοποιείται για την κατασκευή κβαντικών κυκλωμάτων
- την φάση της μαθηματικής περιγραφής, όπου θα αναλυθεί μαθηματικά κάθε βήμα του εκάστοτε αλγορίθμου

Σε κάθε φάση θα αναλύονται βασικοί όροι και εργαλεία που έχουν χρησιμοποιηθεί για την περιγραφή των αλγορίθμων.

Έπειτα, θα παρουσιαστεί με χρήση κώδικα σε γλώσσα προγραμματισμού Python, η κατασκευή των κβαντικών κυκλωμάτων με χρήση κβαντικού μετασχηματισμού Fourier στους παραπάνω αλγορίθμους, αντί της κβαντικής πύλης Hadamard. Στο κεφάλαιο αυτό, θα παρουσιαστούν τα κυκλώματα που δημιουργήθηκαν, οι δυσκολίες που αντιμετωπίστηκαν και τα αποτελέσματα που προέκυψαν από τα πειράματα. Για κάθε έναν από τους δύο αλγορίθμους θα ακολουθήσει ερμηνεία των αποτελεσμάτων με διαγράμματα και θα παρουσιαστούν αναλυτικές περιγραφές.

3. Εισαγωγή στα κβαντικά κυκλώματα

Στο παρόν κεφάλαιο θα παρουσιαστεί το θεωρητικό υπόβαθρο που θεωρείται απαραίτητο για την κατανόηση της παρούσας εργασίας. Αρχικά θα παρουσιαστούν οι βασικοί όροι του κβαντικού υπολογισμού [14], θα σχολιαστούν γνωστές κβαντικές πύλες και θα γίνει περιγραφή της δράσης τους και τέλος, θα γίνει μια μικρή ανάλυση βασικών αλγοριθμικών εννοιών [18].

3.1. Βασικοί όροι

Για την κατανόηση του κβαντικού υπολογισμού και των κβαντικών αλγορίθμων, πρέπει να γίνει περιγραφή ορισμένων βασικών όρων. Οι όροι αυτοί αναλύονται κατά μήκος της εργασίας και θεωρούνται προαπαιτούμενοι για την κατανόηση της. Οι βασικοί όροι που θα αναλυθούν στο παρόν κεφάλαιο είναι:

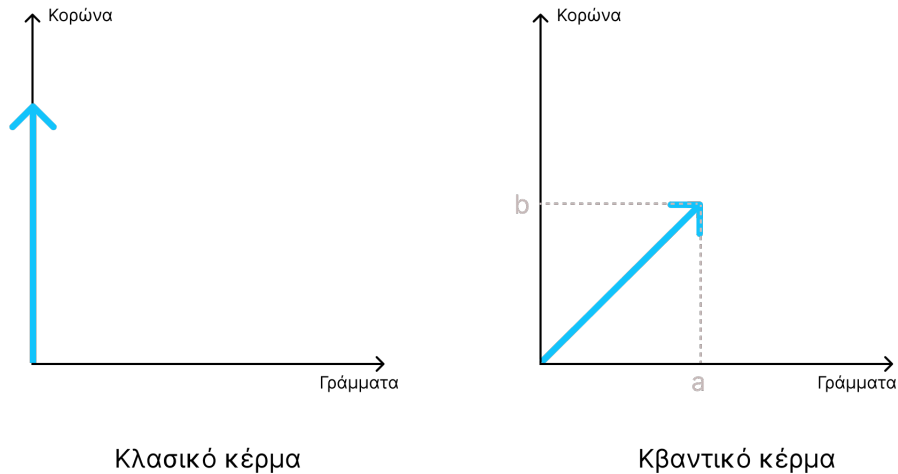
- Διανύσματα bra και ket
- Συστήματα δύο καταστάσεων
- Κβαντικό δυαδικό ψηφίο (qubit)
- Κβαντικές καταστάσεις
 - Κατάσταση ηρεμίας
 - Υπέρθθεση κατάστασης (quantum superposition)
- Κβαντική βάση

Στόχος του κεφαλαίου είναι οι κατανόηση των παραπάνω όρων μέσα από παραδείγματα και μαθηματικές περιγραφές.

3.1.1. Συστήματα δύο καταστάσεων

Ένα σύστημα δύο καταστάσεων, αποτελεί ένα πραγματικό ή θεωρητικό σύστημα που βάση των δεδομένων εισόδου του, μπορεί να έχει δύο διαφορετικά αποτελέσματα. Γνωστά συστήματα δύο καταστάσεων στην πραγματική ζωή είναι η ρίψη κέρματος ή η εξίσωση $f(x) = \text{mod}2$ ή το θεωρητικό παράδειγμα της γάτας του Schrödinger. Για την περιγραφή των συστημάτων δύο καταστάσεων, θα δοθεί ένα παράδειγμα ρίψης κέρματος.

Ένα κέρμα μπορεί να έχει δύο πιθανές καταστάσεις, είτε να είναι κορώνα, είτε γράμματα. Η πιθανότητα για το κάθε ένα αποτέλεσμα είναι ισοπίθανη, δηλαδή η περίπτωση το κέρμα να προσγειωθεί και να είναι κορώνα είναι 50% και να είναι γράμματα επίσης, είναι 50%. Σε ένα κβαντικό σύστημα όμως, οι καταστάσεις στις οποίες μπορεί να βρεθεί το κέρμα διαφέρουν. Για την ακρίβεια, το σύστημα είναι διαφορετικό. Ένα συμβατικό κέρμα, μπορεί να έχει μόνο δύο καταστάσεις, ενώ ένα κβαντικό μπορεί να περιέχει και όλες τις ενδιάμεσες καταστάσεις. Στην πραγματικότητα, ένα κβαντικό κέρμα, μπορεί να βρίσκεται την ίδια στιγμή, σε όλες τις ενδιάμεσες καταστάσεις μέχρι να πάρει την τελική τιμή του. Όταν το κβαντικό κέρμα πάρει την τελική του τιμή, μπορεί να είναι είτε στην κατάσταση κορώνα, είτε στην κατάσταση γράμματα. Η στιγμή που το κέρμα παίρνει μια από τις δύο τιμές ονομάζεται "κατάρρευση κβαντικού συστήματος" και η κατάσταση του κβαντικού κέρματος δεν μπορεί να αλλάξει. Παρακάτω, στο σχήμα 1 υπάρχει παράδειγμα με την ρίψη ενός συμβατικού και ενός κβαντικού νομίσματος πριν την κατάρρευση του συστήματος.



Σχήμα 1: Ρίψη ενός κλασικού και ενός κβαντικού νομίσματος πριν την κατάρρευση του συστήματος

Όπως φαίνεται στο σχήμα 1 στη περίπτωση του κβαντικού κέρματος, το κέρμα δεν βρίσκεται ούτε στην κατάσταση "κορώνα" ούτε στην κατάσταση "γράμματα", αλλά σε μία ενδιάμεση, των δύο κατάσταση. Όσο το κβαντικό κέρμα βρίσκεται ανάμεσα στις δύο καταστάσεις δεν είναι δυνατή η πρόβλεψη του αποτελέσματος που θα έχει μετά την κατάρρευση της κατάστασης. Είναι όμως δυνατό να βρεθεί μια σχέση πρόβλεψης της πιθανότητας. Ειδικότερα, η πιθανότητα το κέρμα να πάρει την τιμή κορώνα ή γράμματα φαίνεται παρακάτω:

$$|A\rangle = a|K\rangle + b|\Gamma\rangle$$

Στην παραπάνω σχέση, το "A" συμβολίζει το αποτέλεσμα της ρίψης του κβαντικού κέρματος, το "a" τη πιθανότητα το αποτέλεσμα να είναι κορώνα και το "b" την πιθανότητα το αποτέλεσμα της ρίψης να είναι γράμματα. Τα "a" και "b" δεν είναι απαραίτητα ισοπίθανα στην περίπτωση του κβαντικού κέρματος, αλλά ισχύει η παρακάτω σχέση:

$$|a|^2 + |b|^2 = 1$$

3.1.2. Διανύσματα bra και ket

Τα διανύσματα bra και ket χρησιμοποιούνται για την ευκολότερη επεξήγηση πινάκων. Ειδικότερα, αποτελούν εργαλεία για την ευκολότερη απεικόνιση και περιγραφή πινάκων. Τα διανύσματα ket αναγράφονται με τον συμβολισμό " $|a\rangle$ " ενώ τα διανύσματα bra απεικονίζονται με τον συμβολισμό " $\langle a|$ ". Το κάθε ένα από τα διανύσματα έχει διαφορετική χρήση και αναπαριστά διαφορετικό πίνακα. Τα διανύσματα bra και ket μπορούν ανήκουν στην κατηγορία των διανυσμάτων Dirac [15].

Τα διανύσματα bra και ket, όπως τα περισσότερα διανύσματα μπορούν να περιγραφούν με την χρήση πινάκων. Για παράδειγμα, το διάνυσμα ket του κεφαλαίου 3.1.1 μπορεί να μεταφραστεί στον παρακάτω πίνακα:

$$|A\rangle = a|K\rangle + b|\Gamma\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$$

Υπενθυμίζεται πως τα "a" και "b" αποτελούν αριθμούς από το 0 έως το 1 που ανήκουν στο σύνολο των μιγαδικών αριθμών.

Για την κατανόηση των διανυσμάτων bra, αρχικά ορίζονται οι μιγαδικοί συζυγείς των "a" και "b" ως " a^* " και " b^* " αντίστοιχα. Ομοίως, για τα διανύσματα bra, η αντίστοιχη αναπαράσταση πινάκων δίνεται από την σχέση:

$$\langle A| = a^* \langle K| + b^* \langle \Gamma| = [a^* \quad b^*]$$

Τέλος, θα παρουσιαστεί η εύρεση του εσωτερικού γινομένου δύο διανυσμάτων bra και ket. Έστω πως υπάρχουν τα διανύσματα $|R\rangle = s|B\rangle + t|C\rangle$ και $\langle L| = m^* \langle K| + n^* \langle \Gamma|$. Το εσωτερικό τους γινόμενο βρίσκεται από τον τύπο:

$$\langle R|A\rangle = [m^* \quad n^*] \begin{bmatrix} s \\ t \end{bmatrix} = (m^*s + n^*t)$$

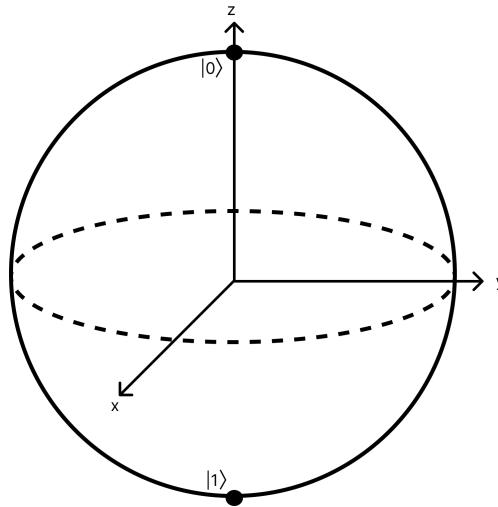
3.1.3. Κβαντικό δυαδικό ψηφίο (qubit)

Ένας κλασικός ηλεκτρονικός υπολογιστής λειτουργεί με την χρήση των bits. Τα bits αποτελούν δυαδικά ψηφία που μπορούν να πάρουν δύο καταστάσεις, είτε 0 είτε 1. Ένα κβαντικό σύστημα όμως, στην πραγματικότητα είναι πολύ πιο περίπλοκο από ότι ένα κλασικό. Ένα κβαντικό σύστημα, ένας κβαντικός υπολογιστής, λειτουργεί με την χρήση των κβαντικών διαδικασιών ψηφίων, δηλαδή με την χρήση των qubits.

Οι καταστάσεις των qubits μπορούν βρεθούν φυσιολογικά είναι 2, η κατάσταση $|0\rangle$ και η κατάσταση $|1\rangle$. Όπως ειπώθηκε και στο κεφάλαιο 3.1.2 τα qubits είναι σε μορφή διανυσμάτων ket, επομένως, σε ανεπτυγμένη μορφή, οι δύο καταστάσεις είναι:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ και } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Κάθε ένα qubit, όπως αναφέρθηκε και στην ενότητα 3.1.1 μπορεί να παίρνει τιμές από το 0 έως το 1 με όλες τις ενδιάμεσες. Κάθε διάνυσμα κατάστασης ενός qubit μπορεί να αναπαρασταθεί μέσω της σφαίρας του Bloch. Η σφαίρα Bloch, μπορεί να χρησιμοποιηθεί για την αναπαράσταση ενός qubit σε κατάσταση ηρεμίας. Η σφαίρα αυτή, αποτελείται από 2 πόλους, όπου ο κάθε ένας αντιστοιχεί στις δύο πιθανές καταστάσεις του qubit $|0\rangle$ και $|1\rangle$. Τέλος η ακτίνα της σφαίρας είναι πάντα ίση με την μονάδα, όσο δηλαδή το μέγιστο μήκος του διανύσματος ket. Παρακάτω, στο σχήμα 2, γίνεται απεικόνιση της σφαίρας Bloch.



Σχήμα 2: Σφαίρα Bloch

3.1.4. Κβαντικές καταστάσεις

Στο κεφάλαιο [3.1.1](#) έγινε αναφορά στον όρο "κατάρρευση κβαντικού συστήματος". Για την επεξήγηση του όρου πρέπει πρώτα να γίνει αναφορά στους όρους κατάσταση ηρεμίας και κατάσταση υπέρθεσης.

Κατάσταση ηρεμίας

Όταν ένα κβαντικό κύκλωμα είναι στην αρχική του κατάσταση βρίσκεται σε κατάσταση ηρεμίας. Για την ακρίβεια, όταν ένα qubit εισέρχεται σε ένα κύκλωμα, η κατάστασή του είναι $|0\rangle$. Στην κατάσταση αυτή το qubit βρίσκεται σε μια σταθερή και ήμερη κατάσταση, η οποία μπορεί να μεταβληθεί μόνο με την χρήση κβαντικών πυλών.

Κατάσταση υπέρθεσης

Η κατάσταση υπέρθεσης ή αλλιώς quantum superposition, είναι μια ιδιαίτερη κατάσταση στην οποία μπορεί να βρεθεί ένα qubit. Για την ακρίβεια, όσο το qubit βρίσκεται σε κατάσταση υπέρθεσης, μπορεί να πάρει όλες τις ενδιάμεσες τιμές των καταστάσεων $|0\rangle$ και $|1\rangle$. Όσο χρονικό διάστημα τα qubits βρίσκονται σε κατάσταση υπέρθεσης, συμπεριφέρονται ως κύματα και έχουν τις παρόμοιες ιδιότητες με τα φωτόνια. Στην πραγματικότητα, στην ενότητα [3.1.2](#) όταν δόθηκαν παραδείγματα διανυσμάτων ket, τα διανύσματα που περιγράφηκαν βρίσκονταν σε υπέρθεση εφόσον δεν είχαν μια σταθερή κατάσταση, αλλά βρίσκονταν σε μια ενδιάμεση.

Όταν το qubit για οποιοδήποτε λόγο βγει από την κατάσταση υπέρθεσης, τότε το κβαντικό σύστημα καταρρέει και το qubit παίρνει μια τελική τιμή.

3.2. Κβαντικές πύλες

Σε έναν κλασικό υπολογιστή, η κατάσταση των bits μπορεί να αλλάξει με την χρήση πυλών. Ομοίως, σε ένα κβαντικό υπολογιστή, η κατάσταση των qubits μπορεί να αλλάξει με την χρήση κβαντικών πυλών. Οι κβαντικές πύλες [6], όπως και οι συμβατικές, μπορούν να δρουν σε ένα, σε δύο ή σε περισσότερα qubits. Στην ενότητα αυτή θα παρουσιαστούν ορισμένες κβαντικές πύλες που χρησιμοποιούνται κατά μήκος της εργασίας χωρισμένες σε κατηγορίες βάση του πλήθους των qubits πάνω στα οποία επιδρούν.

3.3. Πύλες που δρουν σε ένα qubit

Στην ενότητα αυτή θα γίνει αναφορά και μαθηματική περιγραφή των πυλών που δρουν σε ένα μόνο qubit. Αναφορικά, θα γίνει περιγραφή των πυλών:

- Identity gate
- Hadamard gate
- Πύλες Pauli
 - Pauli-X gate
 - Pauli-Y gate
 - Pauli-Z gate
- S gate
- T gate

Identity gate

Η κβαντική πύλη Identity ή η κβαντική πύλη ταυτότητας, επιδρά σε ένα μόνο qubit και δεν τροποποιεί την κατάσταση των πυλών. Για παράδειγμα, εάν σε ένα qubit που βρίσκεται στην κατάσταση $|0\rangle$ εφαρμοστεί μια πύλη ταυτότητας, η κατάσταση του qubit θα παραμείνει $|0\rangle$. Παρακάτω φαίνεται η μαθηματική περιγραφή της πύλης και ο συμβολισμός της:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

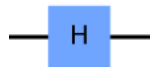


Σχήμα 3: Identity gate

Hadamard gate

Η κβαντική πύλη Hadamard επιδρά σε ένα qubit με σκοπό να το φέρει σε υπέρθεση. Συγκεκριμένα, αλλάζει την κατάσταση $|0\rangle$ σε $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ και την κατάσταση $|1\rangle$ σε $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ φέρνοντας έτσι τις καταστάσεις σε υπέρθεση.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$



Σχήμα 4: Hadamard gate

Pauli gates

Στον κόσμο των κβαντικών κυκλωμάτων υπάρχουν 3 πύλες Pauli, η πύλη X, η πύλη Y και η πύλη Z. Στην παρούσα εργασία, θα σχολιαστούν και οι τρεις πύλες αλλά χρησιμοποιούνται μόνο οι πύλες X και Z. Τέλος, πύλες Pauli, εκτελούν επίσης περιστροφές αντίστοιχα κατά π γύρω από τον άξονα y, z και z της σφαίρας Bloch.

Pauli-X gate

Η κβαντική πύλη Pauli-X χρησιμοποιείται για την εκτέλεση μιας περιστροφής κατά π στον άξονα x της σφαίρας Bloch.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

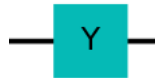


Σχήμα 5: Pauli-X gate

Pauli-Y gate

Η κβαντική πύλη Pauli-Y χρησιμοποιείται για την εκτέλεση μιας περιστροφής κατά π στον άξονα y της σφαίρας Bloch.

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

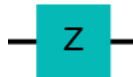


Σχήμα 6: Pauli-Y gate

Pauli-Z gate

Η κβαντική πύλη Pauli-Z χρησιμοποιείται για την εκτέλεση μιας περιστροφής κατά π στον άξονα z της σφαίρας Bloch.

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

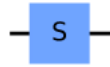


Σχήμα 7: Pauli-Z gate

S gate

Η πύλη S επιδρά σε ένα qubit και χρησιμοποιείται για την περιστροφή του qubit κατά 90° γύρω από τον άξονα z στην σφαίρα του Bloch. Η πύλη S συνδέεται με την πύλη T από τον τύπο $S = T^2$.

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

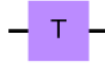


Σχήμα 8: S gate

T gate

Η πύλη T επιδρά σε ένα qubit και ανήκει στην οικογένεια των πυλών αλλαγής φάσης. Συγκεκριμένα, η πύλη T αλλάζει την φάση ενός qubit κατά $\phi = \frac{\pi}{4}$.

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$$



Σχήμα 9: T gate

3.4. Πύλες που δρουν σε δύο qubits

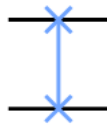
Στην ενότητα αυτή θα γίνει αναφορά και μαθηματική περιγραφή των πυλών που δρουν σε ένα μόνο qubit. Αναφορικά, θα γίνει περιγραφή των πυλών:

- Swap gate
- Controlled NOT gate

Swap gate

Η κβαντική πύλη Swap επιδρά σε δύο qubits ανταλλάσσοντας την κατάσταση των δύο qubits που εμπλέκονται στη λειτουργία.

$$Swap = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

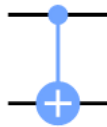


Σχήμα 10: Swap gate

Controlled-Not gate

Η κβαντική πύλη Controlled-Not [17] επιδρά σε δύο qubits έχοντας το πρώτο qubit ως Control qubit ενώ το δεύτερο ως Target. Συγκεκριμένα, αφήνει το Control qubit αμετάβλητο και προσθέτει μια πύλη Pauli-X στο Target qubit όταν το Control qubit είναι σε κατάσταση $|1\rangle$. Όταν το Control qubit είναι σε κατάσταση $|0\rangle$, αφήνει το Target qubit αμετάβλητο.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$



Σχήμα 11: Controlled-Not gate

4. Κβαντικός μετασχηματισμός Fourier

4.1. Εισαγωγή

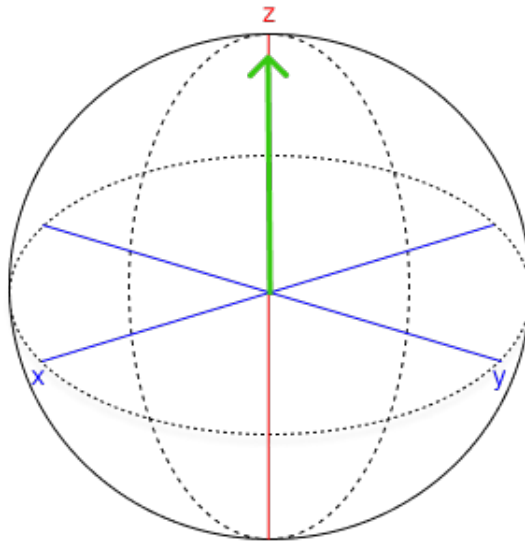
Ο κβαντικός μετασχηματισμός Fourier έχει ως κύριο στόχο την αλλαγή της κβαντικής φάσης ενός κβαντικού κυκλώματος. Στόχος του παρόντος κεφαλαίου είναι η κατανόηση του κβαντικού μετασχηματισμού Fourier, η κατανόηση του τρόπου χρήσης του και θα δοθούν και παραδείγματα για την καλύτερη παρουσίαση του.

4.2. Περιγραφή

Η εργασία θα επικεντρωθεί κυρίως στον υπολογισμό της κβαντικής βάσης και στην αλλαγή της, με χρήση του κβαντικού μετασχηματισμού Fourier. Ο κβαντικός μετασχηματισμός Fourier (QFT) επιδρά σε δύο υπολογιστικές βάσεις, την βάση του Fourier και την υπολογιστική βάση z . Ο κβαντικός μετασχηματισμός Fourier είναι συνάρτηση που μπορεί να εφαρμόσει τον μετασχηματισμό μεταξύ των δύο βάσεων, δηλαδή μπορεί να καταστήσει εφικτή την αλλαγή της βάσης από z σε βάση Fourier.

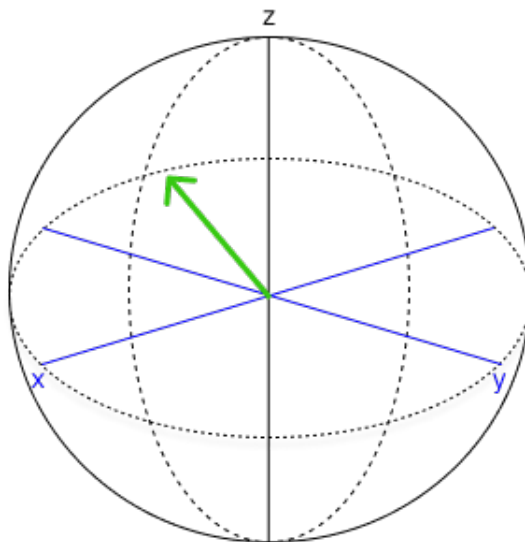
Σε περιπτώσεις όπου το qubit πάνω στο οποίο επιδρά ο Fourier είναι μόνο ένα, τότε τα αποτελέσματα που προκύπτουν είναι τα ίδια με την χρήση της κβαντικής πύλης Hadamard. Έχει γίνει λοιπόν η διαπίστωση πως σε περιπτώσεις όπου υπάρχει μόνο ένα qubit, εάν στο κβαντικό κύκλωμα υλοποιηθεί ο QFT, τότε η δράση του, επιφέρει τα ίδια αποτελέσματα με την χρήση της κβαντικής πύλης Hadamard.

Πριν την χρήση του QFT, τα qubits κινούνται στην υπολογιστική βάση z όπως φαίνεται στην εικόνα [12](#).



Σχήμα 12: Κίνηση qubits στην υπολογιστική βάση z

Όπως παρατηρείται, οι κινήσεις των qubits πραγματοποιούνται κατά μήκος του άξονα z . Όταν επιδράσει στο κύκλωμα ο QFT οι κινήσεις των qubits μεταβάλλονται σε μία νέα βάση, την βάση Fourier. Παρακάτω, στην εικόνα 13 φαίνεται αναλυτικά η κίνηση των qubits σε διαφορετική βάση.



Σχήμα 13: Κίνηση qubits στην υπολογιστική βάση Fourier

4.3. Μετασχηματισμός φάσης

Για την καλύτερη εξήγηση του τρόπου και την χρησιμότητα της αλλαγής βάσης, θα γίνει μια ανάλυση των παραπάνω όρων. Για να υλοποιηθεί η αλλαγή βάσης, είναι αναγκαία η ύπαρξη ενός μοναδιαίου τελεστή U και ενός ιδιοδιανύσματος $|\psi\rangle$ για τα οποία θα ισχύει η παρακάτω σχέση:

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$$

Στην παραπάνω σχέση, το $|\psi\rangle$ είναι το ιδιοδιάνυσμα και το $e^{2\pi i\theta}$ αποτελεί την ιδιοτιμή του ιδιοδιανύσματος με το θ να αποτελεί την γωνία αλλαγής φάσης. Εφόσον, τα ιδιοδιανύσματα μπορούν να συμβάλλουν στην κατανόηση του γραμμικού μετασχηματισμού, με την χρήση τους μπορεί να γίνει η εξήγηση του μετασχηματισμού της φάσης, από z σε Fourier.

4.4. Μετατροπή βάσης

Για την αλλαγή της βάσης σε ένα κύκλωμα, βάση του τύπου $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, χρειάζεται να κατασκευαστεί ένα μοναδιαίος τελεστής U και ένα ιδιοδιάνυσμα $|\psi\rangle$. Σκοπός είναι η εύρεση της γωνίας θ , δηλαδή η γωνία αλλαγής της φάσης. Για την κατασκευή του κβαντικού κυκλώματος, είναι απαραίτητη η παρουσία t qubits. Ο αριθμός t συμβολίζει το πλήθος των qubits για την κατασκευή του κβαντικού κυκλώματος στο συγκεκριμένο παράδειγμα.

4.4.1. Κωδικοποίηση βάσης

Η αλλαγή κβαντικής βάσης, όπως περιγράφεται και στο κεφάλαιο Μετασχηματισμός φάσης πραγματοποιείται από τον τύπο:

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$$

Στην πραγματικότητα όμως, πως μπορεί να γίνει η αλλαγή βάσης από Fourier σε z και το αντίστροφο. Αρχικά, χρειάζεται να οριστεί το πλήθος των qubits που υπάρχουν στον κβαντικό καταχωρητή. Εφόσον τα οριστεί, θα πρέπει να επιλεγεί η κατάσταση (το state) που θα γίνει η μετατροπή.

Έστω για παράδειγμα πως υπάρχει ένας καταχωρητής με 3 qubits και πρέπει να γίνει η μετατροπή του state $|5\rangle$ της βάσης Fourier σε βάση z . Ο τύπος με τον οποίο πραγματοποιείται την αλλαγή είναι:

$$\frac{x}{2^t} \cdot 2\pi$$

Όπου το x συμβολίζει την κατάσταση στην βάση Fourier, το t συμβολίζει το πλήθος των qubits και το 2π είναι η πλήρης περιστροφή. Στο παράδειγμα λοιπόν, η κωδικοποίηση της βάσης Fourier σε z είναι για το πρώτο qubit $\frac{5}{8} \cdot 2\pi$. Για τα επόμενα qubits διπλασιάζεται ο αριθμητής του προηγούμενου qubit. Παρακάτω φαίνεται αναλυτικά, η μετατροπή για όλο το κύκλωμα.

- Πρώτο qubit: $\frac{5}{8} \cdot 2\pi$
- Πρώτο qubit: $\frac{10}{8} \cdot 2\pi = \frac{5}{4} \cdot 2\pi$
- Τρίτο qubit: $\frac{20}{8} \cdot 2\pi = \frac{5}{2} \cdot 2\pi$

Με τον τρόπο αυτό είναι δυνατή η κωδικοποίηση μιας φάσης από Fourier σε z .

4.5. Λογική

Στην ενότητα αυτή, θα πραγματοποιηθεί η εξήγηση του κυκλώματος που θα πραγματοποιηθεί σε λογικό επίπεδο, ενώ στις υπό-ενότητες θα γίνει περιγραφή της μαθηματικής υλοποίησης και της υλοποίησης σε κώδικα, μέσω της γλώσσας προγραμματισμού Python.

Σκοπός του QFT είναι η αλλαγή της κβαντικής βάσης από Fourier σε z . Για την δημιουργία του κυκλώματος, αρχικά πρέπει να οριστεί το πλήθος των qubits που θα μπουν ως είσοδος στον κβαντικό καταχωρητή. Εφόσον οριστεί το πλήθος, θα πρέπει να δημιουργηθεί ένα κύκλωμα με όσα qubits έχει ορίσει ο χρήστης, συν ένα. Το ένα παραπάνω qubit θα χρησιμοποιηθεί για τον ορισμό της ιδιοκατάστασης του μοναδιαίου τελεστή. Ως μοναδιαίος τελεστής, θα δηλωθεί η κβαντική πύλη T . Η κβαντική πύλη T αποτελεί μια κβαντική πύλη η οποία χρησιμοποιείται για την αλλαγή φάσης σε μία κατάσταση. Συγκεκριμένα, η πύλη T αλλάζει την φάση κατά $\phi = \frac{\pi}{4}$ σε κάθε περιστροφή, δηλαδή μπορεί να προσθέτει $e^{\frac{i\pi}{4}}$ σε κάθε νέα φάση. Ο πίνακας που περιγράφει την πύλη είναι:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & \exp(\frac{i\pi}{4}) \end{pmatrix}$$

4.6. Κβαντικό κύκλωμα

Για την δημιουργία κβαντικού κυκλώματος θα πρέπει να οριστεί το πλήθος των qubits που μπαίνουν ως είσοδος στο κύκλωμα. Συνολικά, το κύκλωμα θα αποτελείται από $n + 1$ qubits, όπου n είναι το πλήθος των qubits και το

ένα παραπάνω θα χρησιμοποιηθεί για την προσομοίωση του ιδιοδιανύσματος. Αρχικά, σε κάθε ένα από τα n qubits θα τοποθετηθεί μια πύλη Hadamard ενώ στο τελευταίο qubit, που έχει τον ρόλο του ιδιοδιανύσματος, θα τοποθετηθεί μια X-gate. Στην συνέχεια, για κάθε ένα qubit, θα πραγματοποιηθούν rotations με χρήση της πύλης CPhaseGate.

Η CPhaseGate αποτελεί μια διαγώνια και συμμετρική πύλη που προκαλεί αλλαγή στην φάση του target qubit, ανάλογα με την κατάσταση ελέγχου. Ως qubit στόχος ορίζεται το ιδιοδιάνυσμα, δηλαδή το τελευταίο qubit του κβαντικού κυκλώματος. Κατά την εφαρμογή της πύλης, οι παράμετροι που παίρνει είναι:

- Γωνία περιστροφής
- Control qubit
- Target qubit

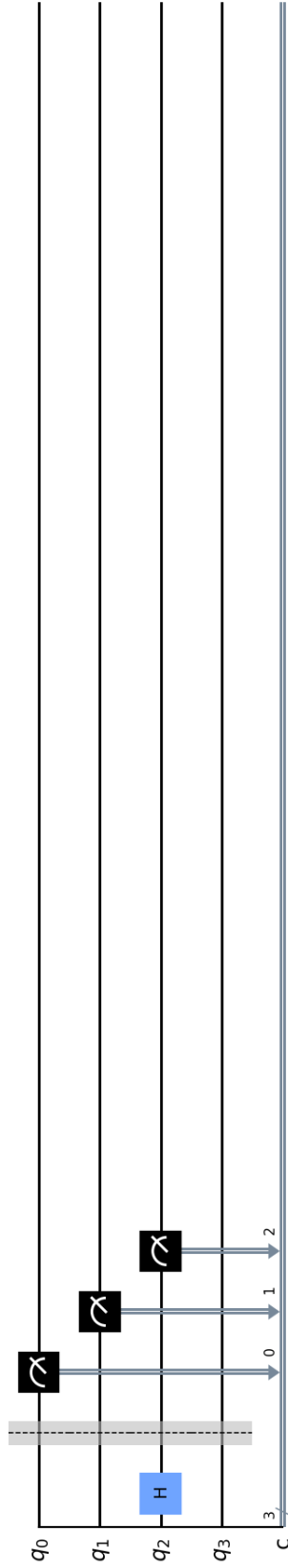
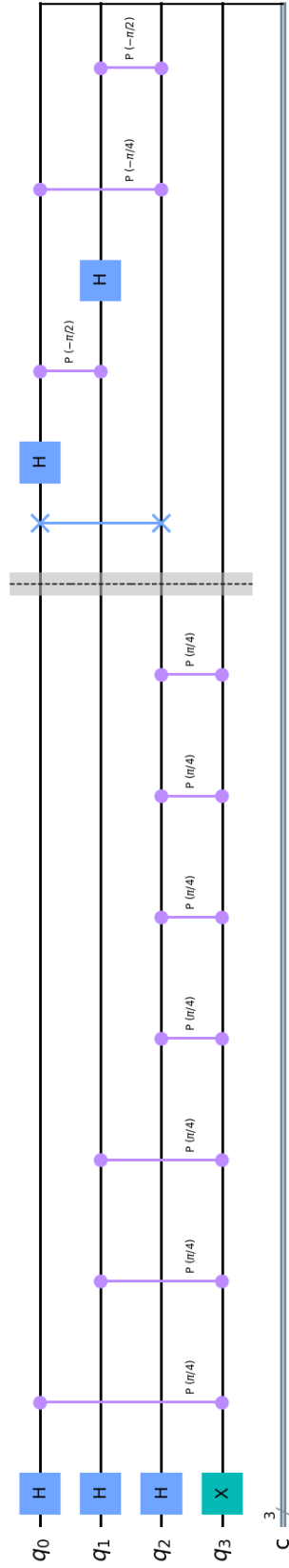
Η πύλη έχει την παρακάτω μορφή:

$$CPhase = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\lambda} \end{pmatrix}$$

Η πύλη θα εφαρμοστεί με τον εξής τρόπο:

- Επιλέγεται πόσες φορές θα πραγματοποιηθούν οι περιστροφές (δηλαδή, πόσες φορές θα εφαρμοστεί η πύλη CPhaseGate σε κάθε qubit), έστω πως θα εφαρμοστεί k φορές
- Για κάθε ένα qubit που υπάρχει στο κύκλωμα (με εξαίρεση το ιδιοδιάνυσμα), εφαρμόζεται η CPhaseGate όπου, τίθεται ως Control qubit το εκάστοτε qubit που εξετάζεται και ως Target το ιδιοδιάνυσμα. Η γωνία περιστροφής ορίζεται στα $\frac{\pi}{4}$.
- Η διαδικασία εφαρμογής της πύλης επαναλαμβάνεται k φορές και το k διπλασιάζεται για το επόμενο qubit.
- Εφαρμόζεται ο αντίστροφος κβαντικός μετασχηματισμός Fourier για την μετατροπή της κατάστασης του καταχωρητή μέτρησης.
- Γίνονται οι μετρήσεις.

Παρακάτω, στην εικόνα 14, φαίνεται το κύκλωμα του QFT για ένα κύκλωμα με τρία qubits.



Σχήμα 14: QFT σε κύκλωμα με 3 qubits

Για καλύτερη κατανόηση περιγράφεται ο κώδικας σε γλώσσα προγραμματισμού Python:

```
1 for qubit in range(3):
2     circuit.h(qubit)
3
4 repetitions = 1
5 for qubits in range(3):
6     for i in range(repetitions):
7         circuit.cp(math.pi/4, qubits, 3)
8     repetitions *= 2
```

4.6.1. Αντίστροφος κβαντικός μετασχηματισμός Fourier

Στην ενότητα 4.6 έγινε αναφορά στον όρο “αντίστροφος κβαντικός μετασχηματισμός Fourier”. Ο αντίστροφος κβαντικός μετασχηματισμός Fourier, χρησιμοποιείται για την μετατροπή της κατάστασης του καταχωρητή μέτρησης, από βάση Fourier σε βάση z . Η αλλαγή της βάσης, για δεύτερη φορά γίνεται με σκοπό την εξαγωγή ερμηνεύσιμων αποτελεσμάτων από το κύκλωμα.

Για την δημιουργία του αντίστροφου μετασχηματισμού Fourier, χρειάζεται ένα κύκλωμα, στο οποίο έχει δράσει ο κβαντικός μετασχηματισμός Fourier και το πλήθος των qubits που υπάρχουν στο κύκλωμα. Για κάθε ένα qubit που υπάρχει στο κύκλωμα, εκτός από το τελευταίο που αποτελεί το ιδιοδιάνυσμα, εφαρμόζεται ο αντίστροφος μετασχηματισμός Fourier. Για τον αντίστροφο μετασχηματισμό Fourier γίνεται χρήση της πύλης “Swap”.

Η κβαντική πύλη “Swap” εφαρμόζεται σε δύο qubits. Το είναι qubit αποτελεί το Control qubit, ενώ το δεύτερο qubit αποτελεί το Target qubit. Με την εφαρμογή της πύλης Swap στα δύο qubits, ανταλλάσσεται η κατάστασή τους. Με τον τρόπο αυτό, η κατάσταση του Control qubit μεταφέρεται στο Target qubit. Το αποτέλεσμα που προκύπτει είναι πως, το Target qubit, στην περίπτωση που εξετάζεται, το ιδιοδιάνυσμα, έχει την κατάσταση του Control qubit. Η μαθηματική περιγραφή της πύλης Swap φαίνεται στο παρακάτω σχήμα:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ο τρόπος λειτουργίας του αντίστροφου μετασχηματισμού Fourier περιγράφεται παρακάτω:

1. Υπολογίζεται το συνολικό πλήθος των qubits που συμμετέχουν στο κύκλωμα
2. Το πλήθος των qubits το διαιρούμε δια δύο και κρατάμε το ακέραιο μέρος
3. Η διαδικασία του βήματος 2 επαναλαμβάνεται για κάθε qubit στο σύστημα
4. Για κάθε qubit που πραγματοποιείται το βήμα 2, εφαρμόζεται μια Swap πύλη, με Control qubit το εκάστοτε qubit που εξετάζεται και με Target το qubit που προκύπτει από την παρακάτω πράξη: $n - \text{qubit} - 1$
5. Για κάθε qubit (έστω j) που υπάρχει στο κύκλωμα δίνεται ένας αριθμός και για το εύρος του αριθμού αυτού (έστω k) τοποθετείται μια CPhaseGate πύλη. Οι παράμετροι είναι αρχικά n γωνία περιστροφής και υπολογίζεται στα $-(\frac{\pi}{2^{j-k}})$. Ως Control qubit ορίζεται το qubit με τον αριθμό j και ως Target qubit ορίζεται το qubit με τον αριθμό k .
6. Τέλος, προστίθεται μια πύλη Hadamard σε κάθε qubit με τον αριθμό j .

Παρακάτω εμφανίζεται ο κώδικας σε γλώσσα προγραμματισμού Python:

```

1 def QFT_dagger(circuit, n):
2     # Don't forget the Swaps!
3     for qubit in range(n//2):
4         circuit.swap(qubit, n-qubit-1)
5     for j in range(n):
6         for k in range(j):
7             circuit.cp(-math.pi/float(2**(j-k)), k, j)
8         circuit.h(j)

```

4.7. Μαθηματική περιγραφή

Στο τρέχων κεφάλαιο θα γίνει μαθηματική περιγραφή του αλγορίθμου αλλαγής φάσης. Για την μαθηματική περιγραφή πρέπει να τεθούν οι παρακάτω όροι:

- Ιδιοδιάνυσμα: $|\psi\rangle$
- Μοναδιαίος τελεστής U : $e^{2\pi i\theta}|\psi\rangle$

Για την δημιουργία του κυκλώματος χρειάζεται το πλήθος των qubit καταχωρητών και ένα qubit για να οριστεί ως ιδιοδιάνυσμα:

$$|\psi\rangle = 0^{\otimes n}|\psi\rangle$$

Το επόμενο βήμα είναι, το υπάρχον κύκλωμα, να βρεθεί σε κατάσταση υπέρ-θεσης. Για τον λόγο αυτό, εφαρμόζεται μια πύλη Hadamard σε κάθε qubit του κυκλώματος:

$$|\psi\rangle = \frac{1}{2^{\frac{n}{2}}}(|0\rangle + |1\rangle)^{\otimes n}|\psi\rangle$$

Έπειτα, εφαρμόζονται οι αλλαγές της γωνίας περιστροφής στο κύκλωμα. Για την υλοποίηση του βήματος χρειάζεται η εφαρμογή του μοναδιαίου τελεστή U στο Target qubit μόνο στις περιπτώσεις όπου το Control qubit έχει την τιμή $|1\rangle$. Εφόσον το U αποτελεί μοναδιαίο τελεστή με ιδιοτιμή $|\psi\rangle$, τέτοια ώστε να ισχύει $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, σχηματίζεται η παρακάτω σχέση:

$$U^{2^j}|\psi\rangle = U^{2^j-1}U|\psi\rangle = U^{2^j-1}e^{2\pi i\theta}|\psi\rangle = \dots = e^{2\pi i2^j\theta}|\psi\rangle$$

Σημειώνεται πως στην παραπάνω σχέση, το j αποτελεί έναν αριθμό που σε κάθε επανάληψη η τιμή του αλλάζει, Μπορεί να πάρει τιμή από το 0 έως το $n-1$. Χρησιμοποιώντας την σχέση $|0\rangle \otimes |\psi\rangle + |1\rangle \otimes e^{2\pi i\theta}|\psi\rangle = (|0\rangle + e^{2\pi i\theta}|1\rangle) \otimes |\psi\rangle$ στην παραπάνω σχέση προκύπτει:

$$|\psi\rangle = \frac{1}{2^{\frac{n}{2}}}(|0\rangle + e^{2\pi i\theta 2^{(n-1)}}|1\rangle) \otimes \dots \otimes (|0\rangle + e^{2\pi i\theta 2^1}|1\rangle) \otimes (|0\rangle + e^{2\pi i\theta 2^0}|1\rangle) \otimes |\psi\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{x=0}^{2^n-1} e^{2\pi i\theta x} |x\rangle \otimes |\psi\rangle$$

Όπου το x δηλώνει την αναπαράσταση των δυαδικών αριθμών μήκους n -bit.

Έπειτα, εφαρμόζεται ο αντίστροφος μετασχηματισμός Fourier στην παραπάνω σχέση:

$$|\psi\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{x=0}^{2^n-1} e^{2\pi i\theta x} |x\rangle \otimes |\psi\rangle \xrightarrow{QTF_n^{-1}} \frac{1}{2^n} \sum_{k=0}^{2^n-1} \sum_{x=0}^{2^n-1} e^{-\frac{2\pi i x}{2^n}(k-2^n\theta)} |k\rangle \otimes |\psi\rangle$$

Τελευταίο βήμα είναι η μέτρηση του κυκλώματος. Η παραπάνω σχέση φτάνει έως το $2^n\theta$. Προκύπτουν δύο περιπτώσεις βάσει της τιμής του $2^n\theta$.

- $2^n\theta =$ ακέραιος αριθμός
- $2^n\theta =$ μη ακέραιος αριθμός

Στην περίπτωση όπου το αποτέλεσμα είναι ακέραιος, με την μέτρηση δίνεται το αποτέλεσμα αλλαγής της φάσης στο βοηθητικό qubit.

Για την περίπτωση που το $2^n\theta$ δεν είναι ακέραιος, προκύπτει ότι η παραπάνω έκφραση δίνει το αποτέλεσμα αλλαγής φάσης στο βοηθητικό qubit με μικρότερο ποσοστό ακρίβειας.

4.8. Κώδικας

Η παρούσα ενότητα περιέχει τον κβαντικό μετασχηματισμό Fourier σε μορφή εκτελέσιμου κώδικα με χρήση της γλώσσας προγραμματισμού Python. Για την δημιουργία των κβαντικών κυκλωμάτων, γίνεται χρήση της βιβλιοθήκης Qiskit. Το πρώτο βήμα είναι η εισαγωγή των απαραίτητων βιβλιοθηκών.

```
1 #initialization
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import math
5
6 # importing Qiskit
7 from qiskit import IBMQ, Aer, transpile, assemble
8 from qiskit import QuantumCircuit, ClassicalRegister,
   QuantumRegister
9
10 # import basic plot tools
11 from qiskit.visualization import plot_histogram
```

Μετά την εισαγωγή των βιβλιοθηκών γίνεται προσθήκη των βασικών συναρτήσεων. Οι συναρτήσεις που θα δημιουργηθούν είναι:

- `reverse_QFT`
- `apply_gft`

Η συνάρτηση `reverse_QFT` έχει ως βασικό στόχο την δημιουργία του αντίστροφου κβαντικού μετασχηματισμού Fourier για την επιστροφή του κυκλώματος στην κβαντική βάση z . Ομοίως, η συνάρτηση `apply_gft` χρησιμοποιείται για την δημιουργία του κβαντικού κυκλώματος και για την εφαρμογή του κβαντικού μετασχηματισμού Fourier, δηλαδή την αλλαγή της κβαντικής βάσης του κυκλώματος από z σε κβαντική βάση Fourier.

Παρακάτω φαίνεται ο κώδικας της συνάρτησης `reverse_QFT`, η οποία έχει σχολιαστεί στην ενότητα [4.6.1](#).

```
1 def reverse_QFT(circuit, number_of_qubits):
2     # add the swaps!
3     for qubit in range(number_of_qubits//2):
4         circuit.swap(qubit, number_of_qubits-qubit-1)
5     for j in range(number_of_qubits):
6         for k in range(j):
7             circuit.cp(-math.pi/float(2*(j-k)), k, j)
8         circuit.h(j)
```

Η επόμενη συνάρτηση είναι η συνάρτηση *apply_gft*. Στην ενότητα 4.6 ως ένα βαθμό έγινε περιγραφή της συνάρτησης, κυρίως για να δοθεί έμφαση στην ιδιότητα των swaps. Η συνάρτηση αυτή, αρχικά δημιουργεί ένα κβαντικό κύκλωμα, με το πλήθος των qubits να ορίζεται ως παράμετρος. Έπειτα, προσθέτει μια πύλη Pauli-X στο qubit που έχει τον ρόλο του ιδιοδιανύσματος και προσθέτει πύλες Hadamard στα υπόλοιπα qubits του κβαντικού κυκλώματος.

Στο επόμενο βήμα ορίζεται το πλήθος των επαναλήψεων και πραγματοποιείται η αλλαγή της γωνίας περιστροφής. Τέλος, προστίθεται μια μπάρα για την καλύτερη απεικόνιση των διαδικασιών του κυκλώματος, καλείται η συνάρτηση *reverse_gft*, γίνεται η μέτρηση του συστήματος και επιστρέφεται το κύκλωμα στην τελική του κατάσταση.

```
1 def apply_gft(number_of_qubits):
2     # create quantum circuit
3     circuit = QuantumCircuit(number_of_qubits, number_of_qubits-1)
4
5     # apply an X gate to the eigenstate qubit
6     circuit.x(number_of_qubits-1)
7
8     # apply Hadamard gates to the counting qubits
9     for qubit in range(number_of_qubits-1):
10         circuit.h(qubit)
11
12     # select number of repetitions
13     repetitions = 1
14     for counting_qubit in range(number_of_qubits-1):
15         for i in range(repetitions):
16             circuit.cp(math.pi/4, counting_qubit,
17                        number_of_qubits-1);
18             repetitions *= 2
19
20     # add barrier for visuals
21     circuit.barrier()
22     # apply inverse QFT
23     reverse_qtf(circuit, number_of_qubits-1)
24     # measure
25     circuit.barrier()
26     for n in range(number_of_qubits-1):
27         circuit.measure(n,n)
28     return circuit
```

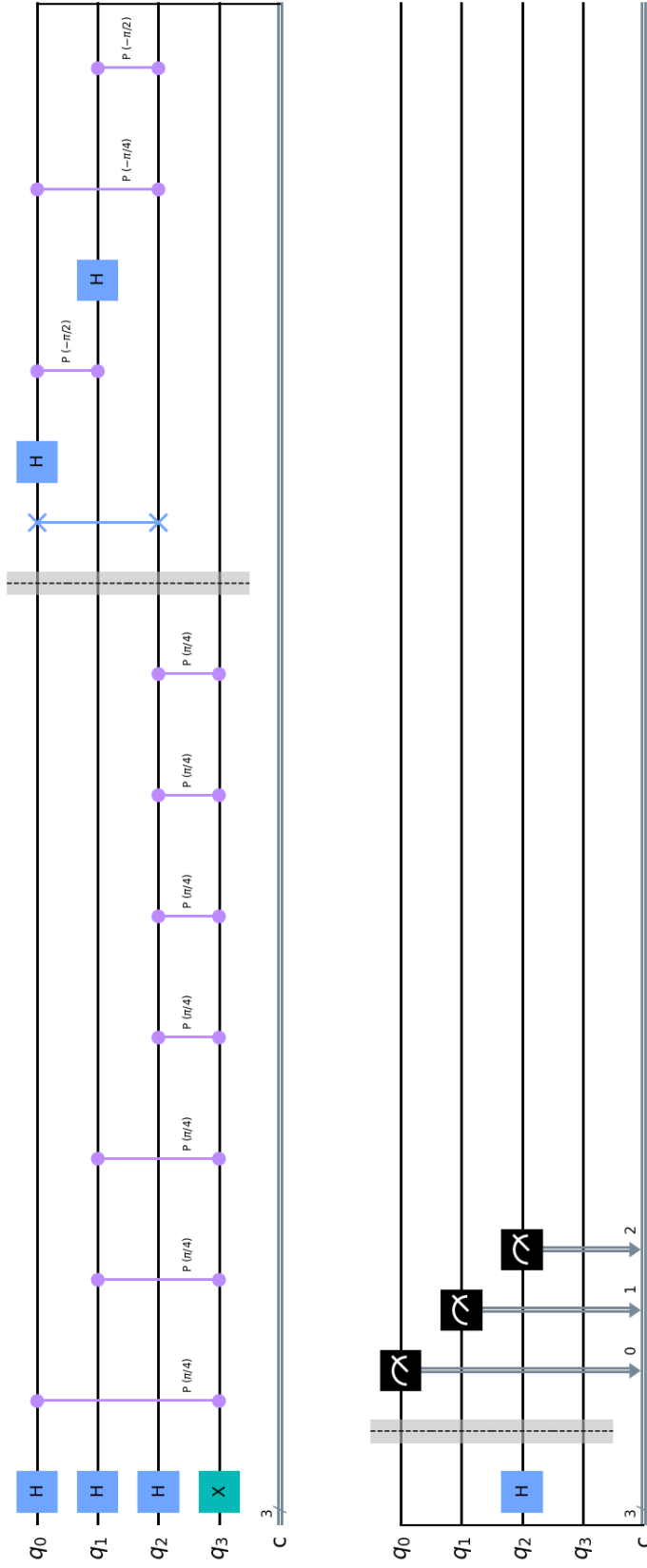
Έπειτα ορίζονται οι παράμετροι του συστήματος, όπως το πλήθος των qubits που θα συμμετέχουν στην μετατροπή της κβαντικής βάσης και καλείται η συνάρτηση *apply_gft*.

```
1 # select number of qubits, in the example it is set to 5
2 number_of_qubits = 4
3
4 # call the apply_gft function to create the quantum circuit and
  apply the QFT
5 circuit = apply_gft(number_of_qubits)
```

Τελευταίο βήμα είναι η επεξεργασία των δεδομένων που προέκυψαν από το κύκλωμα, ο ορισμός του αριθμού των επαναλήψεων για την διεξαγωγή ασφαλών αποτελεσμάτων και η προσομοίωση της εκτέλεσής του κώδικα σε κβαντικό σύστημα.

```
1 # select quantum simulator
2 aer_sim = Aer.get_backend('aer_simulator')
3 # select number of repetitions
4 shots = 2048
5 # run on quantum device
6 t_circuit = transpile(circuit, aer_sim)
7 qobj = assemble(t_circuit, shots=shots)
8 results = aer_sim.run(qobj).result()
9 answer = results.get_counts()
10
11 # plot histogram with answers
12 plot_histogram(answer)
```

Τέλος, παρακάτω φαίνεται το τελικό κύκλωμα, ολοκληρωμένο, και με τις δύο εναλλαγές μεταξύ των βάσεων.



Σχήμα 15: Το κύκλωμα που προκύπτει από την εκτέλεση του κβαντικού μετασχηματισμού Fourier σε κύκλωμα με 3 qubits

5. Αλγόριθμος Deutsch Jozsa

Ο αλγόριθμος των Deutsch-Jozsa υλοποιήθηκε το 1992 από τους David Deutsch και Richard Jozsa. Ο αλγόριθμος αυτός ήταν ο πρώτος όπου απέδειξε πως σε συγκεκριμένες κατηγορίες αλγορίθμων, οι κβαντικοί υπολογιστές μπορούν να επιλύσουν τα ίδια προβλήματα με πολύ μεγαλύτερη ταχύτητα. Ειδικότερα, ο αλγόριθμος των Deutsch-Jozsa έχει καλύτερη εξαιρετικά καλύτερη πολυπλοκότητα σε σχέση με την βέλτιστη λύση του σε κλασσικό υπολογιστή.

5.1. Περιγραφή

Παρακάτω, αναγράφεται η περιγραφή του προβλήματος που θέλει να επιλύσει ο αλγόριθμος και παρουσιάζεται ένα παράδειγμα χρήσης.

Το πρόβλημα που καλείται να επιλύσει ο αλγόριθμος αυτός είναι η κατηγοριοποίηση της συνάρτησης $f(x)$ ως σταθερή (constant) ή ως ισορροπημένη (balanced). Παρακάτω αναφέρονται ενδεικτικά παραδείγματα balanced και constant συναρτήσεων.

- Παράδειγμα constant συνάρτησης: $f(x) = 0$ ή $f(x) = 1$. Στην περίπτωση αυτή, για κάθε τιμή του x η συνάρτηση θα έχει το ίδιο τελικό αποτέλεσμα. Δηλαδή, τα δεδομένα εισόδου (οι διάφορες τιμές του x) δεν επηρεάζουν την τιμή της συνάρτησης.
- Παράδειγμα balanced συνάρτησης: $f(x) = \text{mod}2$. Στην περίπτωση αυτή, το αποτέλεσμα της $f(x)$ σε κάποιες περιπτώσεις (στις μισές για παράδειγμα), η τιμή της $f(x)$ έχει την τιμή 0 και στις υπόλοιπες έχει την τιμή ένα. Χαρακτηριστικό παράδειγμα είναι η συνάρτηση $f(x) = \text{mod}2$, όπως αναφέρθηκε και παραπάνω, όπου στις περιπτώσεις που το x είναι περιττός το αποτέλεσμα είναι ίσο με 0 ενώ όταν το x είναι άρτιος έχει την τιμή 1.

5.2. Υλοποίηση σε κλασσικό υπολογιστή

Το πρόβλημα που θέλει να επιλύσει ο αλγόριθμος των Deutsch – Jozsa μπορεί να επιλυθεί και σε κλασσικό υπολογιστή. Ο τρόπος επίλυσης του αλγορίθμου προϋποθέτει πως η συνάρτηση $f(x)$ πρέπει να τρέξει τόσες φορές ώστε να έχουν τρέξει τα 50%+1 παραδείγματα για τις διάφορες τιμές του x . Για την καλύτερης κατανόηση του αλγορίθμου θα υλοποιηθεί ένα παράδειγμα.

Έστω πως υπάρχει η συνάρτηση $f(x)$ με το μήκος του x να είναι ίσο με 3. Οι πιθανές τιμές που μπορεί να πάρει το x είναι: 000, 001, 010, 011, 100, 101, 110, 111. Επειδή όμως, η σειρά των διαφορετικών τιμών είναι τυχαία θα πρέπει τουλάχιστον μια τιμή της $f(x)$ να δίνει διαφορετικό αποτέλεσμα εξόδου. Τα παραδείγματα θα υλοποιηθούν και για τις δύο περιπτώσεις, δηλαδή και για την περίπτωση που η $f(x)$ είναι σταθερή και για την περίπτωση που η $f(x)$ είναι ισορροπημένη.

5.2.1. Η $f(x)$ είναι σταθερή

Σε κάθε περίπτωση σε έναν κλασικό υπολογιστή, πρέπει να υπολογιστεί η τιμή της συνάρτησης για τις μισές συν μια περιπτώσεις του x . Στην περίπτωση που η $f(x)$ είναι σταθερή η έξοδος θα έχει πάντα το ίδιο αποτέλεσμα. Θα επιλεγθούν τυχαία κάποιες από τις τιμές του x :

- $f(000) = 0$
- $f(010) = 0$
- $f(100) = 0$
- $f(110) = 0$
- $f(001) = 0$

Στην περίπτωση αυτή είναι σίγουρο ότι η $f(x)$ είναι σταθερή

5.2.2. Η $f(x)$ είναι ισορροπημένη

Η διαδικασία που θα πραγματοποιηθεί θα είναι η ίδια. Υπολογίζουμε την τιμή της $f(x)$ για τα μισά συν ένα δεδομένα εισόδου.

- $f(000) = 0$
- $f(010) = 0$
- $f(100) = 0$
- $f(110) = 0$
- $f(001) = 1$

Παρατηρείται πως στην περίπτωση που η $f(x)$ έτρεχε για τα μισά μόνο παραδείγματα δεν θα ήταν ασφαλές να διεξαχθεί κάποιο αποτέλεσμα.

Και στις δύο περιπτώσεις, παρατηρείται πως για την διεξαγωγή ασφαλούς συμπεράσματος χρειάζονται $n+1$ βήματα. Για την υλοποίηση όμως σε κβαντικό υπολογιστή, χρειάζεται αποκλειστικά ένα μόνο βήμα. Παρακάτω, εξηγείται αναλυτικά ο τρόπος υλοποίησης του.

5.3. Υλοποίηση σε κβαντικό υπολογιστή

Η συνέχεια της εργασίας θα επικεντρωθεί στην δημιουργία και την περιγραφή του αλγορίθμου Deutsch - Jozsa για την επίλυση του παραπάνω προβλήματος. Θα γίνει περιγραφή των δεδομένων εισόδου, περιγραφή του αλγορίθμου, απεικόνιση του αλγορίθμου και περιγραφή του αλγορίθμου σε γλώσσα προγραμματισμού Python με χρήση της βιβλιοθήκης Qiskit.

5.4. Περιγραφή αλγορίθμου

Ο αλγόριθμος των Deutsch Jozsa υλοποιείται σε κβαντικούς υπολογιστές για να μπορέσει να προβλέψει αν μια συνάρτηση είναι σταθερή ή όχι, γνωρίζοντας μόνο τα δεδομένα εισόδου. Για την υλοποίηση του αλγορίθμου, χρειάζεται αρχικά, να γίνει η περιγραφή του Oracle.

Το Oracle στην περίπτωση που περιγράφεται, αποτελεί ένα μαύρο κουτί, στο οποίο πραγματοποιούνται διαδικασίες άγνωστες στο κοινό. Ο ρόλος τους είναι να τρέξουν κάποιες “κρυφές” πράξεις ώστε να μπορέσουν να επεξεργαστούν τα δεδομένα εισόδου του αλγορίθμου. Υπάρχουν δύο είδη Oracles: Το Balanced Oracle και το Constant Oracle. Στις αντίστοιχες υπο-ενότητες θα σχολιαστούν οι διεργασίες που πραγματοποιούνται στο κάθε ένα.

5.5. Περιγραφή των Oracles

Τα Oracles μπορούν να είναι είτε Balanced είτε Constant, όπως αναφέρθηκε στην ενότητα 5.4. Παρακάτω θα εξηγηθεί ο τρόπος υλοποίησης τους σε θεωρητικό επίπεδο και σε ένα κβαντικό κύκλωμα.

5.5.1. Constant Oracle

Όταν ο τύπος του Oracle χαρακτηρίζεται ως Constant, η συνάρτηση $f(x)$, για κάθε τιμή του x έχει σταθερό αποτέλεσμα. Αυτό σημαίνει πως για κάθε πιθα-

νή τιμή του x , το αποτέλεσμα θα είναι είτε αποκλειστικά 0 είτε αποκλειστικά 1. Επομένως η συνάρτηση $f(x)$ χαρακτηρίζεται ως σταθερή.

Έστω για παράδειγμα πως έχουμε την συνάρτηση $f(x)$ όπου το μήκος της συμβολοσειράς είναι 4. Ενδεικτικά, παίρνουμε τις τιμές : 0110, 1100, 0101, 1010 . Στην περίπτωση που το Oracle είναι Constant, σημαίνει πως:

- $f(0110) = 0$
- $f(1100) = 0$
- $f(0101) = 0$
- $f(1010) = 0$

Στο σημείο αυτό αξίζει να σημειωθεί πως το 0 ως αποτέλεσμα επιλέχθηκε τυχαία. Στην θέση του, θα μπορούσε να είναι και ο αριθμός 1.

Υλοποίηση σε κβαντικό κύκλωμα

Για την υλοποίηση του Constant Oracle σε κβαντικό κύκλωμα χρειάζεται ένας αριθμός σε δυαδική μορφή. Έστω, για παράδειγμα, ο αριθμός 0110. Η διαδικασία που ακολουθείτε για την επεξεργασία του αριθμού από το Constant Oracle είναι η εξής:

- Για κάθε ένα στοιχείο του αριθμού γίνεται ένας έλεγχος
- Αν το στοιχείο αυτό είναι ίσο με 1 τότε, προστίθεται στο κύκλωμα μια X πύλη.
- Αν το στοιχείο αυτό είναι ίσο με το 0 τότε δεν πραγματοποιείται κάποια ενέργεια και προχωράει ο αλγόριθμος στο επόμενο qubit μέχρι και να ελεγχθεί το τελευταίο qubit.

5.5.2. *Balanced Oracle*

Όταν το Oracle χαρακτηρίζεται ως Balanced σημαίνει πως σε κάποιες περιπτώσεις το αποτέλεσμα της $f(x)$ θα είναι ίσο με 0 και κάποιες άλλες ίσο με 1. Στην περίπτωση αυτή, τα δεδομένα εισόδου επηρεάζουν την τιμή της συνάρτησης εφόσον η τιμή της δεν είναι σταθερή. Στην περίπτωση αυτή η συνάρτηση $f(x)$ ονομάζεται Balanced.

Έστω πως στη συνάρτηση $f(x)$ το x έχει μήκος συμβολοσειράς ίσο με 4. Ενδεικτικά, επιλέγονται οι αριθμοί: 0110, 1100, 0101, 1010. Στην περίπτωση που το Oracle είναι Balanced τότε, η $f(x)$ έχει διαφορετική τιμή για κάθε εκτέλεση.

- $f(0110) = 0$
- $f(1100) = 1$
- $f(0101) = 1$
- $f(1010) = 0$

Επομένως, παρατηρείται πως στις μισές περιπτώσεις, η $f(x)$ έχει την τιμή 0 και στις άλλες μισές έχει την τιμή 1.

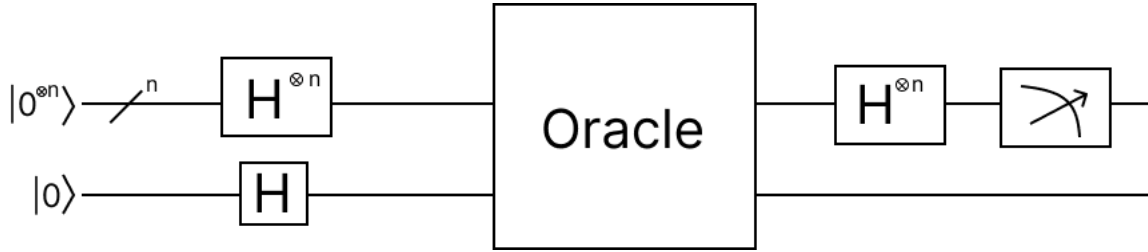
Υλοποίηση σε κβαντικό κύκλωμα

Για την υλοποίηση του Balanced Oracle σε κβαντικό κύκλωμα χρειάζεται ένας αριθμός σε δυαδική μορφή. Έστω, για παράδειγμα, ο αριθμός 0110. Η διαδικασία που ακολουθείτε για την επεξεργασία του αριθμού από το Balanced Oracle είναι η εξής:

- Για κάθε ένα στοιχείο του αριθμού γίνεται ένας έλεγχος
- Αν το στοιχείο αυτό είναι ίσο με 1 τότε, προστίθεται στο κύκλωμα μια X πύλη.
- Έπειτα, για κάθε ένα qubit που υπάρχει στο κύκλωμα, μπαίνει μια Controlled Not πύλη με Control qubit το qubit που εξετάζεται και Target κάποιο qubit εξόδου που έχουμε θέσει.
- Τέλος, γίνεται ένας ακόμα έλεγχος για κάθε ένα στοιχείο
- Αν το στοιχείο αυτό είναι ίσο με 1 τότε, προστίθεται στο κύκλωμα μια X πύλη.

5.6. Μαθηματική περιγραφή

Το παρακάτω κβαντικό κύκλωμα περιγράφει τον αλγόριθμο των Deutsch - Jozsa σχηματικά. Το σχήμα αυτό περιγράφει πως στην αρχή τα δυαδικά δεδομένα εισόδου εισάγονται στο σύστημα. Σε κάθε ένα από τα qubits, εφαρμόζεται η κβαντική πύλη Hadamard και έπειτα εφαρμόζεται το Oracle (όποιο και αν είναι αυτό). Τέλος, εφαρμόζεται ξανά η κβαντική πύλη Hadamard σε κάθε qubit.



Σχήμα 16: Κύκλωμα του αλγορίθμου των Deutsch Jozsa

Για την μαθηματική επεξήγηση του αλγορίθμου αρχικά περιγράφονται τα δεδομένα εισόδου στο κβαντικό κύκλωμα:

$$|y\rangle = |0\rangle^{\otimes n}|1\rangle$$

Έπειτα, εφαρμόζεται η πύλη Hadamard σε κάθε qubit:

$$|y\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^{n-1}} |x\rangle (|0\rangle - |1\rangle)$$

Έπειτα εφαρμόζεται το Oracle:

$$|y\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^{n-1}} |x\rangle (|f(x)\rangle - |1 \oplus f(x)\rangle) = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^{n-1}} (-1)^{f(x)} |x\rangle (|0\rangle - |1\rangle)$$

Στη συνέχεια, εφαρμόζεται για δεύτερη φορά η κβαντική πύλη Hadamard για κάθε qubit:

$$\begin{aligned} |y\rangle &= \frac{1}{\sqrt{2}} \sum_{x=0}^{2^{n-1}} (-1)^{f(x)} \left[\frac{1}{\sqrt{2}} \sum_{x=0}^{2^{n-1}} (-1)^{xy} |y\rangle \right] = \\ &= \frac{1}{\sqrt{2}} \sum_{y=0}^{2^{n-1}} \left[\sum_{x=0}^{2^{n-1}} (-1)^{f(x)} (-1)^{xy} \right] |y\rangle \end{aligned}$$

Και εν τέλει γίνεται η μέτρηση.

5.7. Κώδικας

Η παρούσα ενότητα περιέχει τον αλγόριθμο των Deutsch Jozsa σε μορφή εκτελέσιμου κώδικα με χρήση της γλώσσας προγραμματισμού Python. Για την δημιουργία των κβαντικών κυκλωμάτων, γίνεται χρήση της βιβλιοθήκης Qiskit. Το πρώτο βήμα είναι η εισαγωγή των απαραίτητων βιβλιοθηκών.

```
1 # Import basic libraries
2
3 # initialization
4 import numpy as np
5
6 # importing Qiskit
7 from qiskit import IBMQ, Aer
8 from qiskit.providers.ibmq import least_busy
9 from qiskit import QuantumCircuit, assemble, transpile
10
11 # import basic plot tools
12 from qiskit.visualization import plot_histogram
```

Έπειτα, δημιουργείται μια συνάρτηση για την κατασκευή των Oracles:

```
1 # function to select quantum oracles
2 def dj_oracle(case, n):
3     oracle_qc = QuantumCircuit(n+1)
4
5     # if the selected oracle is balanced
6     if case == "balanced":
7         # generate a random integer and convert it to binary
8         b = np.random.randint(1,2**n)
9         b_str = format(b, '0'+str(n)+'b')
10
11         # for each qubit whose values is one apply Not gate
12         for qubit in range(len(b_str)):
13             if b_str[qubit] == '1':
14                 oracle_qc.x(qubit)
15
16         # for each qubit apply controlled not gate and use as
17         # controlled
18         # qubit the current qubit and as target the output qubit
19         for qubit in range(n):
20             oracle_qc.cx(qubit, n)
21
22         # for each qubit whose values is one apply Not gate
```

```

22     for qubit in range(len(b_str)):
23         if b_str[qubit] == '1':
24             oracle_qc.x(qubit)
25
26     # if the selected oracle is balanced
27     if case == "constant":
28         # generate a random binary number (0 or 1)
29         output = np.random.randint(2)
30         # if the generated qubit is one apply Not gate
31         if output == 1:
32             oracle_qc.x(n)
33
34     # return the selected Oracle
35     oracle_gate = oracle_qc.to_gate()
36     oracle_gate.name = "Oracle"
37     return oracle_gate

```

Παρακάτω, ακολουθεί η συνάρτηση για την δημιουργία του κυκλώματος των Deutsch - Jozsa:

```

1 # the Deutsch - Jozsa function
2 def dj_algorithm(oracle, n):
3
4     # generate quantum circuit with n+1 qubits
5     # the amount of qubits is selected by the user
6     # the plus one qubit is used as output qubit
7     dj_circuit = QuantumCircuit(n+1, n)
8
9     # gates to simulate the output qubit
10    dj_circuit.x(n)
11    dj_circuit.h(n)
12
13    # apply Hadamard gate to each qubit
14    for qubit in range(n):
15        dj_circuit.h(qubit)
16
17    # apply the selected oracle
18    dj_circuit.append(oracle, range(n+1))
19
20    # apply Hadamard gate the second time
21    for qubit in range(n):
22        dj_circuit.h(qubit)
23
24    # measure the circuit

```

```

25     for i in range(n):
26         dj_circuit.measure(i, i)
27
28     # return the circuit
29     return dj_circuit

```

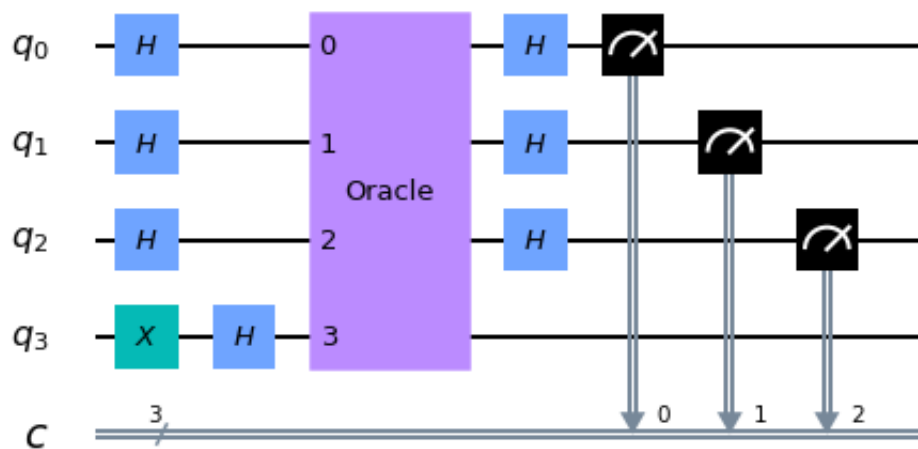
Έπειτα καλείται η συνάρτηση Deutsch - Jozsa και δημιουργούνται οι απεικονίσεις:

```

1 n = length of binary qubits
2
3 # choose the Oracle type
4 # in this code the chosen oracle is balanced
5 oracle_gate = dj_oracle('balanced', n)
6
7 # call the Deutsch - Jozsa function
8 dj_circuit = dj_algorithm(oracle_gate, n)
9
10 # draw the circuit
11 dj_circuit.draw(output='mpl')
12
13 # select quantum simulator
14 aer_sim = Aer.get_backend('qasm_simulator')
15 transpiled_dj_circuit = transpile(dj_circuit, aer_sim)
16 qobj = assemble(transpiled_dj_circuit)
17 results = aer_sim.run(qobj).result()
18 answer = results.get_counts()
19
20 # plot the results
21 plot_histogram(answer)

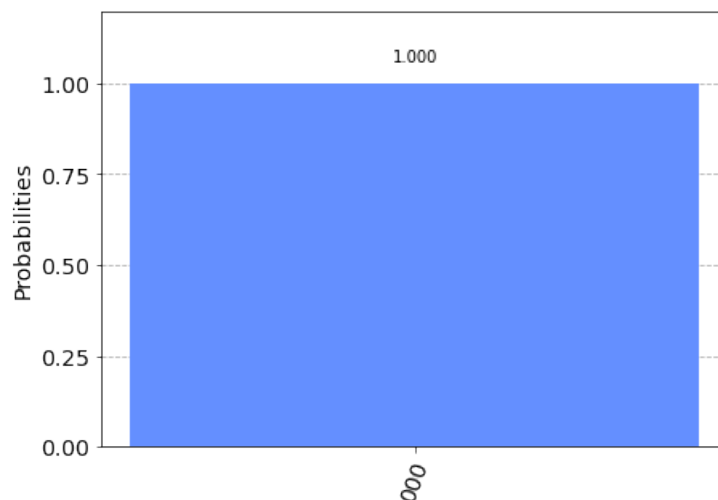
```

Για την υλοποίηση σε $n = 3$, δηλαδή στην περίπτωση όπου το μήκος του δυαδικού αριθμού που θα μελετηθεί είναι ίσο με 3, τα διαγράμματα και τα σχήματα που παράγονται φαίνονται παρακάτω:



Σχήμα 17: Κύκλωμα του αλγορίθμου των Deutsch - Jozsa για 3 qubits

Τέλος, για το κύκλωμα που δημιουργήθηκε από τον παραπάνω κώδικα, το τελικό αποτέλεσμα φαίνεται στο Σχήμα 18.



Σχήμα 18: Διάγραμμα του αλγορίθμου των Deutsch Jozsa για 3 qubits με την χρήση Constant Oracle

6. Αλγόριθμος Bernstein Vazirani

6.1. Εισαγωγή

Ο αλγόριθμος των Bernstein Vazirani αποτελεί μια επέκταση του αλγορίθμου των Deutsch Jozsa που περιγράφηκε στο προηγούμενο κεφάλαιο. Στόχος του αλγορίθμου είναι η απόδειξη πως οι κβαντικοί υπολογιστές μπορούν να χρησιμοποιηθούν για το υπολογισμό πολύπλοκων πράξεων και την επίλυση δύσκολων προβλημάτων. Ο αλγόριθμος αναπτύχθηκε το 1992 από τους Ethan Bernstein και Umesh Vazirani [9]. Η βασική διαφορά του, από τον αλγόριθμο των Deutsch Jozsa είναι πως αντί να κάνει διάκριση μεταξύ δύο διαφορετικών κατηγοριών συναρτήσεων, της Constant και της Balanced συνάρτησης, χρησιμοποιεί μία παραγόμενη συμβολοσειρά για να καταλάβει το είδος της συνάρτησης. Η σημασία της παραπάνω πρότασης, περιγράφεται αναλυτικά στην ενότητα 6.2.

6.2. Περιγραφή προβλήματος

Στο κύκλωμα που παράχθηκε για τον αλγόριθμο των Deutsch Jozsa γινόταν επιλογή του Oracle κατά την υλοποίηση του κυκλώματος και τα αποτελέσματα που παράχθηκαν ανάλογα την έξοδο χαρακτήριζαν την συνάρτηση εισόδου ως Constant ή ως Balanced. Ο αλγόριθμος των Bernstein Vazirani εμπνευσμένος από τον αλγόριθμο των Deutsch Jozsa, προσπάθησε να επεκτείνει τις προοπτικές του αλγορίθμου και να αναδείξει την πολυπλοκότητα των κβαντικών υπολογιστών. Η ιδέα καταγράφεται ως εξής: Αντί να χαρακτηρίζεται μια συνάρτηση ως Constant ή Balanced, θα παράγεται μια συμβολοσειρά s από την συνάρτηση $f(x)$ για την οποία θα ισχύει: $f(x) = s \times x(mod2)$

Στόχος του αλγορίθμου είναι να βρεθεί η συμβολοσειρά s , η οποία θα είναι και η έξοδος του αλγορίθμου. Στην παραπάνω σχέση, το x συμβολίζει τα δεδομένα εισόδου, η $f(x)$ συμβολίζει την συνάρτηση που δέχεται τα δεδομένα εισόδου και το $mod2$ αποτελεί την πράξη modulo 2. Η πράξη modulo παράγει το υπόλοιπο μιας διαίρεσης ακέραιου αριθμού, στην περίπτωσή που εξετάζεται, υπολογίζει το υπόλοιπο της διαίρεσης $(s \times x)/2$.

6.3. Υλοποίηση σε κλασικό υπολογιστή

Η υλοποίηση του αλγορίθμου σε κλασικό υπολογιστή πραγματοποιείται με τον εξής τρόπο. Αρχικά, γίνεται γνωστό στον χρήστη το μήκος της δυαδικής συμβολοσειράς s που πρόκειται να βρεθεί, έστω πως η συμβολοσειρά έχει μήκος 5.

Ο χρήστης δεν δύναται να γνωρίζει την συμβολοσειρά αλλά σκοπός του είναι να την μάθει. Το ποιόν της συμβολοσειράς, το γνωρίζει το Oracle. Το Oracle, όπως είχε ειπωθεί και σε προηγούμενα κεφάλαια, αποτελεί ένα μαύρο κουτί, στο οποίο δεν γνωρίζουμε τις διαδικασίες που πραγματοποιούνται. Στο εν λόγω Oracle πραγματοποιούνται πράξεις, για την ακρίβεια πολλαπλασιασμοί, μεταξύ των ψηφίων της κρυμμένης συμβολοσειράς s και αριθμών που θέτει το πρόγραμμα.

Συνολικά το πρόγραμμα θα πραγματοποιήσει τόσους πολλαπλασιασμούς, όσους το μήκος της συμβολοσειράς s , στο παράδειγμά που παρουσιάζεται θα χρειαστούν συνολικά πέντε πολλαπλασιασμοί. Ο πρώτος πολλαπλασιασμός που θα πραγματοποιηθεί στο Oracle θα είναι ο πολλαπλασιασμός της συμβολοσειράς s με τον αριθμό 00001. Το αποτέλεσμα που θα επιστρέψει ο πολλαπλασιασμός θα είναι ή το 0, στην περίπτωση που η πρώτη θέση της συμβολοσειράς s είναι 0 και 1 στην περίπτωση που η πρώτη θέση της συμβολοσειράς s είναι 1.

Στο δεύτερο πολλαπλασιασμό, θα είναι ο πολλαπλασιασμός της συμβολοσειράς s με τον αριθμό 00010. Στο τρίτο πολλαπλασιασμό, θα είναι ο πολλαπλασιασμός της συμβολοσειράς s με τον αριθμό 00100 στον τέταρτο πολλαπλασιασμό με τον αριθμό 01000 και στον τελευταίο πολλαπλασιασμό με τον αριθμό 10000. Σε κάθε ένα βήμα, το αποτέλεσμα του πολλαπλασιασμού θα αποκαλύπτει εάν το ψηφίο που εξετάζεται έχει την τιμή 0 ή την τιμή 1. Το τελικό αποτέλεσμα είναι η συμβολοσειρά s .

6.4. Παράδειγμα υλοποίησης σε κλασικό υπολογιστή

Έστω πως η συμβολοσειρά s είναι η συμβολοσειρά $s = "01101"$, η οποία έχει μήκος 5. Ο πρώτος πολλαπλασιασμός θα είναι ο πολλαπλασιασμός των ψηφίων της s με τα ψηφία του αριθμού 00001, όπως δηλαδή, η χρήση της πύλης AND στον κλασικό υπολογισμό. Παρακάτω φαίνεται η διαδικασία:

$$\begin{array}{r} 01101 \\ \times 00001 \\ \hline 00001 \end{array}$$

Ο παραπάνω πολλαπλασιασμός, πραγματοποιείται εντός του Oracle. Η μόνη πληροφορία που εξάγεται από το Oracle είναι ο αριθμός 00001, ο οποίος φανερώνει πως το ψηφίο στη πρώτη θέση είναι 1. Ομοίως πραγματοποιείται και ο δεύτερος πολλαπλασιασμός με τον αριθμό 00010:

$$\begin{array}{r}
 01101 \\
 \times 00010 \\
 \hline
 00000
 \end{array}$$

Ο δεύτερος πολλαπλασιασμός φανερώνει πως το δεύτερο ψηφίο της συμβολοσειράς είναι το ψηφίο 0. Έπειτα πραγματοποιείται ο τρίτος πολλαπλασιασμός με τον δυαδικό αριθμό 00100:

$$\begin{array}{r}
 01101 \\
 \times 00100 \\
 \hline
 00100
 \end{array}$$

Ο τρίτος πολλαπλασιασμός φανερώνει πως το τρίτο ψηφίο της συμβολοσειράς είναι το ψηφίο 1. Σειρά έχει ο τέταρτος πολλαπλασιασμός με τον αριθμό 01000:

$$\begin{array}{r}
 01101 \\
 \times 01000 \\
 \hline
 01000
 \end{array}$$

Ο τέταρτος πολλαπλασιασμός φανερώνει πως το τέταρτο ψηφίο της συμβολοσειράς είναι το ψηφίο 1. Τέλος, ο πέμπτος πολλαπλασιασμός με τον αριθμό 10000:

$$\begin{array}{r}
 01101 \\
 \times 10000 \\
 \hline
 00000
 \end{array}$$

Όπου φανερώνει πως το τελευταίο ψηφίο είναι 0. Διατάσσοντας τα αποτελέσματα των πολλαπλασιασμών προκύπτει πως η συμβολοσειρά s είναι 01101, η οποία είναι η ίδια συμβολοσειρά με αυτή που είχε εισαχθεί στο Oracle.

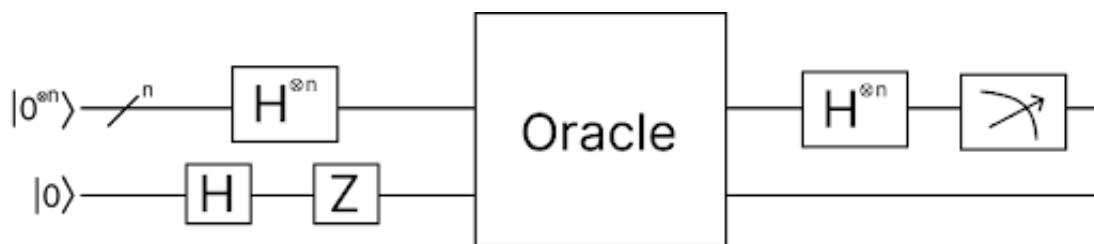
Όπως εύκολα παρατηρήθηκε ο αλγόριθμος που χτίστηκε έλυσε αυτό το πρόβλημα σε 5 βήματα, όσα δηλαδή και το μήκος της συμβολοσειράς. Σε περιπτώσεις δηλαδή που η συμβολοσειρά είχε μήκος n, ο αλγόριθμος θα επιλέγονται σε n βήματα. Συνεπώς η πολυπλοκότητα σε συμβατικό υπολογιστή είναι O(n).

6.5. Υλοποίηση σε κβαντικό υπολογιστή

Στο παραπάνω κεφάλαιο με τίτλο “Παράδειγμα υλοποίησης σε κλασικό υπολογιστή” βρέθηκε πως ο αλγόριθμος σε κλασικό υπολογιστή έχει πολυπλοκότητα $O(n)$. Η επίλυση του ίδιου προβλήματος μπορεί να επιλυθεί σε ένα μόνο βήμα με την χρήση κβαντικού υπολογιστή. Παρακάτω στο κεφάλαιο, θα γίνει περιγραφή του αλγορίθμου Bernstein Vazirani για την επίλυση του προβλήματος σε κβαντικό υπολογιστή.

Αρχικά, όπως και στο κλασικό σύστημα, πρέπει να δοθεί η πληροφορία του μήκους της συμβολοσειράς s . Κάθε ένα ψηφίο αναπαριστάται με ένα qubit, επομένως, αρχικά το κύκλωμα έχει ως δεδομένα εισόδου το μήκος της συμβολοσειράς s αναπαραστατημένο με qubits. Το επόμενο βήμα είναι η προσθήκη πυλών Hadamard σε κάθε ένα qubit εισόδου.

Μετά την προσθήκη των πυλών Hadamard, γίνεται προσθήκη του Oracle. Όπως έχει αναφερθεί και σε προηγούμενα κεφάλαια, δεν υπάρχει γνώση για τις πράξεις που πραγματοποιούνται εντός του Oracle. Η μόνη γνώση που μπορεί να δοθεί είναι η έξοδος του Oracle, στην οποία τα qubits βρίσκονται ακόμα σε κατάσταση υπέρθεσης. Το τελευταίο βήμα πριν την μέτρηση του συστήματος είναι η προσθήκη πυλών Hadamard για την κατάργηση της υπέρθεσης στο κβαντικό σύστημα. Μετά την δεύτερη και τελευταία προσθήκη της Hadamard στο κάθε qubit πραγματοποιείται η μέτρηση του κυκλώματος από όπου και προκύπτει η συμβολοσειρά s . Παρακάτω φαίνεται η απεικόνιση του κβαντικού κυκλώματος:



Σχήμα 19: Κύκλωμα του αλγορίθμου των Bernstein Vazirani

Όπως πολύ εύκολα παρατηρείται, το κύκλωμα που έχει σχεδιαστεί έχει πολλές ομοιότητες με το κύκλωμα του αλγορίθμου των Deutsch Jozsa. Οι βασικές διαφορές παρατηρούνται στον τρόπο που αρχικοποιείται το βοηθητικό qubit όπου εκτός της Hadamard προστίθεται και μια Pauli z πύλη. Επίσης μια ακόμα βασική διαφορά είναι στον τρόπο κατασκευής του Oracle.

Για την κατασκευή του Oracle αρχικά δημιουργείται ένας επαναληπτικός βρόχος για κάθε qubit που αντιπροσωπεύει ένα ψηφίο της συμβολοσειράς s . Έπειτα πραγματοποιείται έλεγχος για την κατάσταση του εκάστοτε qubit που εξετάζεται. Εάν το στοιχείο στην κρυμμένη συμβολοσειρά s είναι 0, τότε τοποθετείται μια Identity πύλη στο αντίστοιχο qubit του κυκλώματος. Εάν το στοιχείο στην κρυμμένη συμβολοσειρά είναι 1, τότε τοποθετείται μια Controlled Not πύλη με Control qubit το αντίστοιχο qubit του κυκλώματος και με Target qubit το βοηθητικό qubit.

6.6. Μαθηματική περιγραφή

Στην παρούσα ενότητα θα πραγματοποιηθεί μαθηματική περιγραφή του αλγορίθμου των Bernstein Vazirani και θα αποδειχθεί ο τρόπος με τον οποίο ο αλγόριθμος αυτός λειτουργεί με μαθηματική απόδειξη.

Αρχικά, προστίθεται η πύλη Hadamard σε κάθε qubit που συμμετέχει στον αλγόριθμο. Για την καλύτερη επεξήγηση θα παρουσιαστεί η επίδραση της Hadamard σε ένα τυχαίο qubit :

$$|a\rangle \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{a \cdot x} |x\rangle$$

Για τον αλγόριθμο των Bernstein Vazirani στο πρώτο βήμα, ο κβαντικός καταχωρητής είναι σε κατάσταση $|000\dots 0\rangle$. Με την προσθήκη της πύλης Hadamard, όπως έδειξε το πάνω βήμα, ο όρος $(-1)^{a \cdot x}$ χάνεται εφόσον η τιμή του είναι ίση με 0, επομένως, $(-1)^{0 \cdot x} = 1$. Επομένως η Oracle επιστρέφει 1 για κάθε δεδομένο εισόδου για το οποίο ισχύει $s \cdot x \bmod 2 = 1$ και επιστρέφει 0 σε οποιαδήποτε άλλη περίπτωση. Παρακάτω φαίνεται η κατάσταση για το πρώτο βήμα του αλγορίθμου:

$$|000\dots 0\rangle \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

Στο επόμενο βήμα, θα εφαρμοστεί αλλαγή φάσης μέσω του Oracle. Υπενθυμίζεται πως η μέθοδος αλλαγής φάσης σε ένα qubit μέσω του Oracle είναι η ακόλουθη:

$$|x\rangle \xrightarrow{f_s} (-1)^{s \cdot x} |x\rangle$$

Παρακάτω φαίνεται η εφαρμογή της αλλαγής φάσης με χρήση του Oracle, στο βήμα 2:

$$|000\dots 0\rangle \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{a \cdot x} |x\rangle$$

Στο τελευταίο βήμα, προστίθεται ξανά η πύλη Hadamard σε κάθε qubit που συμμετέχει στο κύκλωμα για να βρεθεί το qubit $|a\rangle$. Παρακάτω, φαίνεται η διαδικασία:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{a \cdot x} |x\rangle \xrightarrow{H^{\otimes n}} |a\rangle$$

6.7. Κώδικας

Για την δημιουργία του κώδικα για τον αλγόριθμο των Bernstein Vazirani θα γίνει χρήση της βιβλιοθήκης Qiskit με χρήση της γλώσσας προγραμματισμού Python. Αρχικά, θα γίνει προσθήκη των απαραίτητων βιβλιοθηκών:

```

1 # initialization
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # importing Qiskit
6 from qiskit import IBMQ, Aer
7 from qiskit.providers.ibmq import least_busy
8 from qiskit import QuantumCircuit, ClassicalRegister,
   QuantumRegister, transpile, assemble
9
10 # import basic plot tools
11 from qiskit.visualization import plot_histogram

```

Έπειτα, ακολουθεί η κυρίως συνάρτηση του αλγορίθμου. Η συνάρτηση αυτή παίρνει ως παράμετρο την συμβολοσειρά s , την οποία την θέτει ο χρήστης σε επόμενο βήμα. Αρχικά, βρίσκεται το μήκος της συμβολοσειράς και ορίζεται σε μια μεταβλητή με όνομα "number_of_qubits". Έπειτα, στο βοηθητικό qubit προστίθενται οι πύλες Hadamard και Pauli-Z.

Το επόμενο βήμα είναι η προσθήκη των πυλών Hadamard σε κάθε qubit που συμμετέχει στον αλγόριθμο. Για αισθητικούς λόγους, προστίθεται ένα barrier για την ευκολότερη απεικόνισή των διαγραμμάτων.

Συνέχεια έχει η δημιουργία του Oracle. Αρχικά, γίνεται αντιστροφή της σειράς των ψηφίων της συμβολοσειράς s για να ταιριάζει με την σειρά επεξεργασίας

των qubits από την Qiskit.

Στο Oracle συμβαίνουν οι εξής διαδικασίες:

1. Για κάθε qubit που υπάρχει στο κύκλωμα και συμμετέχει στην δημιουργία της συμβολοσειράς, γίνεται έλεγχος του qubit.
2. Εάν, η τιμή του qubit είναι ίση με το 0, προστίθεται μια Identity πύλη στο qubit, διαφορετικά, προστίθεται μια Control Not πύλη, με Control qubit το qubit που ελέγχεται και ως Target το βοηθητικό qubit.

Παρακάτω φαίνεται ο κώδικας της συνάρτησης Bernstein Vazirani:

```
1 def bernstein_vazirani(s):
2     number_of_qubits = len(s)
3     # We need a circuit with n qubits, plus one auxiliary qubit
4     # Also need n classical bits to write the output to
5     circuit = QuantumCircuit(number_of_qubits+1, number_of_qubits)
6
7     # put auxiliary in state |->
8     circuit.h(number_of_qubits)
9     circuit.z(number_of_qubits)
10
11    # Apply Hadamard gates before querying the oracle
12    for i in range(number_of_qubits):
13        circuit.h(i)
14
15    # Apply barrier
16    circuit.barrier()
17
18    # Apply the inner-product oracle
19    # reverse s to fit Qiskit's qubit ordering
20    s = s[::-1]
21    for q in range(number_of_qubits):
22        if s[q] == '0':
23            circuit.i(q)
24        else:
25            circuit.cx(q, number_of_qubits)
26
27    # Apply barrier
28    circuit.barrier()
29
30    #Apply Hadamard gates after querying the oracle
31    for i in range(number_of_qubits):
```

```

32     circuit.h(i)
33
34     # Measurement
35     for i in range(number_of_qubits):
36         circuit.measure(i, i)
37
38     return circuit

```

Στο επόμενο βήμα, ορίζεται η συμβολοσειρά s και καλείται η παραπάνω συνάρτηση με παράμετρο την συμβολοσειρά. Στο πλαίσιο της εργασίας, η συμβολοσειρά που επιλέχθηκε είναι με 6 qubits. Παρακάτω φαίνεται ο κώδικας υλοποίησης:

```

1 s = '011001'    # the hidden binary string
2 circuit = bernstein_vazirani(s)

```

Τέλος, γίνονται οι μετρήσεις στο κύκλωμα, και παράγονται τα διαγράμματα με τις αιτιολογήσεις.

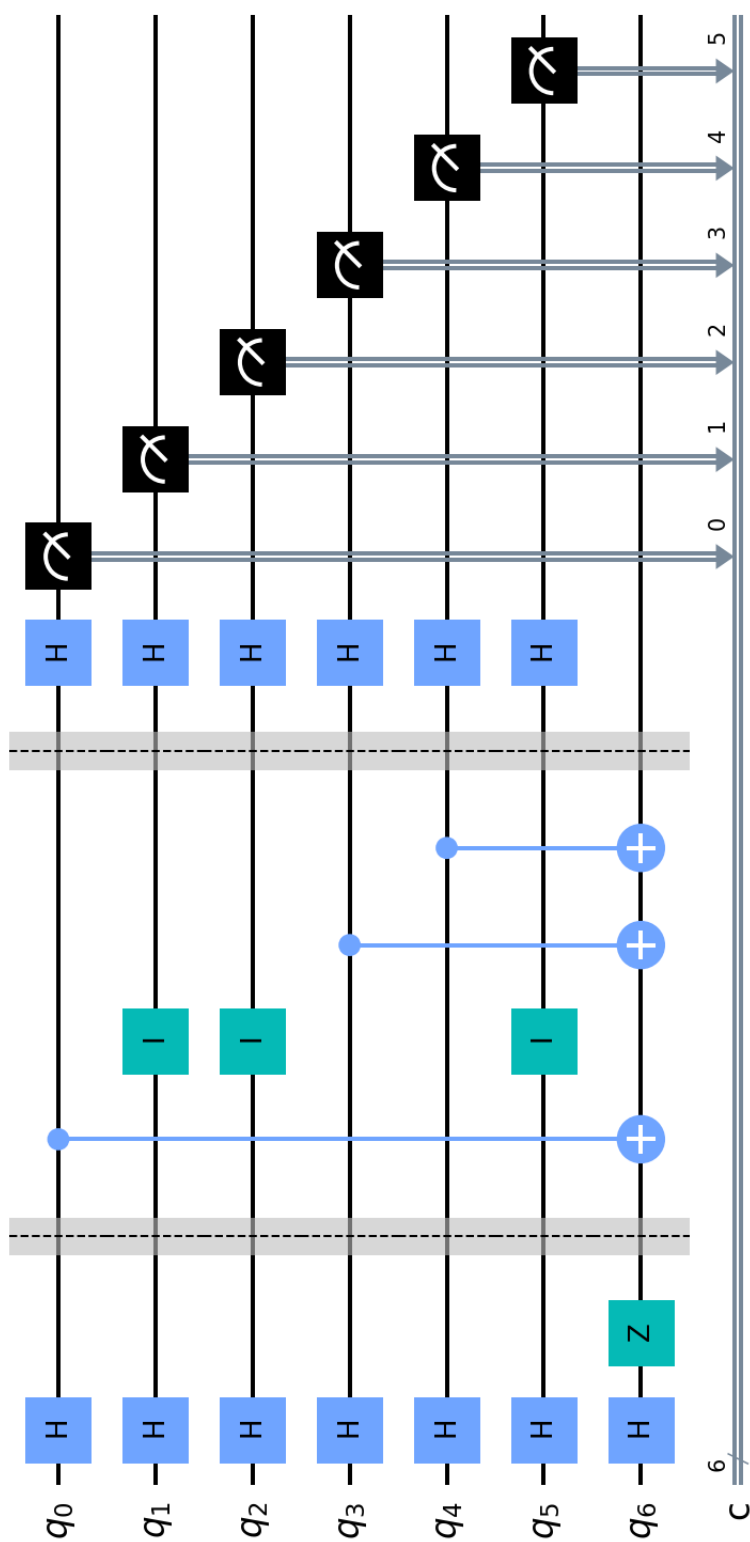
```

1 # use local simulator
2 aer_sim = Aer.get_backend('aer_simulator')
3 shots = 1024
4 qobj = assemble(circuit)
5 results = aer_sim.run(qobj).result()
6 answer = results.get_counts()
7
8 plot_histogram(answer)

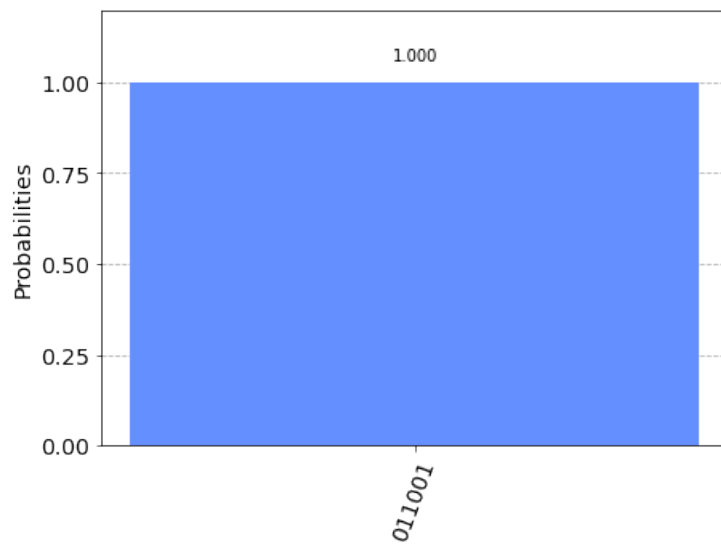
```

Παρακάτω φαίνεται το κύκλωμα που παράχθηκε από το παράδειγμα που δόθηκε για την συμβολοσειρά $s = 011001$.

Και τέλος φαίνεται το διάγραμμα με την παραγόμενη συμβολοσειρά s ως έξοδο:



Σχήμα 20: Κύκλωμα του αλγορίθμου των Bernstein Vazirani για την συμβολοσειρά s



Σχήμα 21: Το διάγραμμα με την συμβολοσειρά s που προκύπτει από τον αλγόριθμο των Bernstein Vazirani

7. Χρήση κβαντικού μετασχηματισμού Fourier στον αλγόριθμο των Deutsch Jozsa

Στόχος της εργασίας είναι η μελέτη των αποτελεσμάτων κατά την χρήση του κβαντικού μετασχηματισμού Fourier σε κλασσικούς κβαντικούς αλγορίθμους. Οι αλγόριθμοι που θα γίνει χρήση του κβαντικού μετασχηματισμού Fourier είναι οι αλγόριθμοι των Deutsch Jozsa και των Bernstein Vazirani. Η ιδέα βασίστηκε στην παρατήρηση της συμπεριφοράς του κβαντικού μετασχηματισμού Fourier και της κβαντικής πύλης Hadamard. Μάλιστα, ο ισχυρισμός αυτός ενισχύθηκε από το γεγονός πως σε ένα qubit, η χρήση της πύλης Hadamard έχει τα ίδια αποτελέσματα με την χρήση του κβαντικού μετασχηματισμού Fourier σε ένα qubit. Για τον λόγο αυτό, στους παραπάνω αλγορίθμους, έγινε αλλαγή των κυκλωμάτων και στην θέση της πύλης Hadamard τοποθετήθηκε ο κβαντικός μετασχηματισμός Fourier. Στο πλαίσιο των κεφαλαίων θα πραγματοποιηθεί η περιγραφή του κώδικα που δημιουργήθηκε, διάφορες παραδοχές που έγιναν στο πλαίσιο της εργασίας και θα παρουσιαστούν διάφορες παρατηρήσεις κατά την εκτέλεση των κυκλωμάτων.

Αρχικά, για την χρήση του κβαντικού μετασχηματισμού Fourier στον αλγόριθμο των Deutsch Jozsa, θα γίνει προβολή του κώδικα για την δημιουργία των κυκλωμάτων. Ο αλγόριθμος δημιουργήθηκε με χρήση της γλώσσας προγραμματισμού Python και τα κβαντικά κυκλώματα κατασκευάστηκαν με χρήση της βιβλιοθήκης Qiskit.

Για την καλύτερη περιγραφή των αποτελεσμάτων θα παρουσιαστεί ο κώδικας που δημιουργήθηκε στα προηγούμενα κεφάλαια με μερικές τροποποιήσεις ανάλογα την περίπτωση. Σε κάθε παράδειγμα θα παρουσιάζεται μόνο το μέρος του κώδικα που τροποποιείται. Στο κεφάλαιο 7.1 θα παρουσιαστεί συνολικά ο κώδικας έχοντας σε σχόλια τμήματα του που θα τροποποιηθούν στις ανάλογες περιπτώσεις. Για την περίπτωση του Constant Oracle θα παρουσιαστούν οι παρακάτω εκδοχές του κώδικα.

- Στο κύκλωμα θα έχει την μορφή: Hadamard, Oracle, Κβαντικός μετασχηματισμός Fourier
- Στο κύκλωμα θα έχει την μορφή: Κβαντικός μετασχηματισμός Fourier, Oracle, Hadamard
- Το κύκλωμα θα έχει την μορφή: Κβαντικός μετασχηματισμός Fourier, Oracle, Κβαντικός μετασχηματισμός Fourier για 4 qubits

Ομοίως, για το Balanced Oracle θα τρέξουν οι ίδιες υλοποιήσεις, όπως και στο Constant Oracle.

7.1. Ολοκληρωμένος κώδικας

Αρχικά, θα γίνει η προσθήκη των απαιτούμενων βιβλιοθηκών. Η προσθήκη γίνεται με τον εξής τρόπο:

```
1 # import libraries
2
3 #initialization
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import math
7
8 # importing Qiskit
9 from qiskit import IBMQ, Aer, transpile, assemble
10 from qiskit import QuantumCircuit, ClassicalRegister,
    QuantumRegister
11 from qiskit.providers.ibmq import least_busy
12
13 # import basic plot tools
14 from qiskit.visualization import plot_histogram
```

Έπειτα, θα γίνει προσθήκη των συναρτήσεων για τον Κβαντικό μετασχηματισμό Fourier και τον Αντίστροφο μετασχηματισμό Fourier, όπως περιγράφηκε στην ενότητα 4.8.

```
1 # Reverse QFT
2 def reverse_QFT(circuit, number_of_qubits):
3     # add the swaps!
4     for qubit in range(number_of_qubits//2):
5         circuit.swap(qubit, number_of_qubits-qubit-1)
6     for j in range(number_of_qubits):
7         for k in range(j):
8             circuit.cp(-math.pi/float(2**(j-k)), k, j)
9         circuit.h(j)
10
11 # Basic quntum fourier transform function
12 def apply_gft(number_of_qubits,circuit):
13
14     # number_of_qubits = number_of_qubits + 1
15     # apply an X gate to the eigenstate quibit
16     qtf_circuit = circuit
```

```

7   qtf_circuit.x(number_of_qubits-1)
8
9   # apply Hadamard gates to the counting qubits
10  for qubit in range(number_of_qubits-1):
11      qtf_circuit.h(qubit)
12
13  # repetitions is set to 1
14  repetitions = 1
15
16  # apply all the nesecessary gates as many times as the repetitions
17  for counting_qubit in range(number_of_qubits-1):
18      for i in range(repetitions):
19          qtf_circuit.cp(math.pi/4, counting_qubit,
20              number_of_qubits-1); # This is CU
21          repetitions *= 2
22      # qtf_circuit.draw()
23
24  qtf_circuit.barrier()
25  # Apply inverse QFT
26  qft_dagger(qtf_circuit, number_of_qubits-1)
27
28  # return the circuit
29  return qtf_circuit

```

Έπειτα γίνεται η δημιουργία των Oracles:

```

1  # function to select quantum oracles
2  def dj_oracle(case, n):
3      oracle_qc = QuantumCircuit(n+1)
4
5      # if the selected oracle is balanced
6      if case == "balanced":
7          # generate a random integer and convert it to binary
8          b = np.random.randint(1,2**n)
9          b_str = format(b, '0'+str(n)+'b')
10
11         # for each qubit whose values is one apply Not gate
12         for qubit in range(len(b_str)):
13             if b_str[qubit] == '1':
14                 oracle_qc.x(qubit)
15
16         # for each qubit apply controlled not gate and use as
17         controlled
18         # qubit the current qubit and as target the output qubit

```

```

18     for qubit in range(n):
19         oracle_qc.cx(qubit, n)
20
21     # for each qubit whose values is one apply Not gate
22     for qubit in range(len(b_str)):
23         if b_str[qubit] == '1':
24             oracle_qc.x(qubit)
25
26     # if the selected oracle is balanced
27     if case == "constant":
28         # generate a random binary number (0 or 1)
29         output = np.random.randint(2)
30         # if the generated qubit is one apply Not gate
31         if output == 1:
32             oracle_qc.x(n)
33
34     # return the selected Oracle
35     oracle_gate = oracle_qc.to_gate()
36     oracle_gate.name = "Oracle"
37     return oracle_gate

```

Στην συνέχεια γίνεται προσθήκη της συνάρτησης "circuit_creation". Η συνάρτηση αυτή θα αλλάξει στην συνέχεια των κεφαλαίων, ανάλογα με την περίπτωση που εξετάζεται. Οι γραμμές που είναι σε σχόλια θα αλλάξουν στα επόμενα κεφάλαια ανάλογα με την περίπτωση.

```

1 # Function to create the quantum circuits
2 def circuit_creation(oracle, n):
3     dj_circuit = QuantumCircuit(n+1, n)
4     dj_circuit.x(n)
5     dj_circuit.h(n)
6
7     number_of_qubits = n+1
8     for qubit in range(n):
9         dj_circuit.h(qubit)
10
11     # apply_gft(number_of_qubits ,dj_circuit)
12
13     dj_circuit.append(oracle, range(n+1))
14
15     for qubit in range(n):
16         dj_circuit.h(qubit)
17

```

```

18 # apply_gft(number_of_qubits,dj_circuit)
19
20 dj_circuit.barrier()
21
22 for i in range(n):
23     dj_circuit.measure(i, i)
24
25 return dj_circuit

```

Έπειτα, καλούνται οι συναρτήσεις, εντυπώνονται τα κυκλώματα και εκτελούνται σε προσομοιωτές κβαντικών υπολογιστών. Για την περιγραφή του κυκλώματος βάζουμε να εκτελεστούν τα κυκλώματα αρχικά με 4 qubits.

```

1 # set number of qubits
2 n = 4
3
4 oracle_gate = dj_oracle('balanced', n)
5 dj_circuit = circuit_creation(oracle_gate, n)
6 dj_circuit.draw()
7
8 # run simulations on quantum system
9 aer_sim = Aer.get_backend('qasm_simulator')
10
11 transpiled_dj_circuit = transpile(dj_circuit, aer_sim)
12 qobj = assemble(transpiled_dj_circuit)
13
14 # get results and create histogram
15 results = aer_sim.run(qobj).result()
16 answer = results.get_counts()
17 plot_histogram(answer)

```

Για την καλύτερη ερμηνεία των αποτελεσμάτων, γίνονται ορισμένες πράξεις, κυρίως η εναλλαγή των δυαδικών αποτελεσμάτων σε δεκαδικά.

```

1 # create empty dictionary
2 keys = []
3 # for each key in the dictionary
4 for key in answer.keys():
5     # convert str to integer and add the integers
6     # to the empty dictionary
7     key = int(key,2)
8     keys.append(key)
9
10 print(keys)

```

Στα επόμενα κεφάλαια, για κάθε μια από τις περιπτώσεις που θα σχολιαστούν θα γίνει η περιγραφή των αποτελεσμάτων για το Constant και το Balanced Oracle.

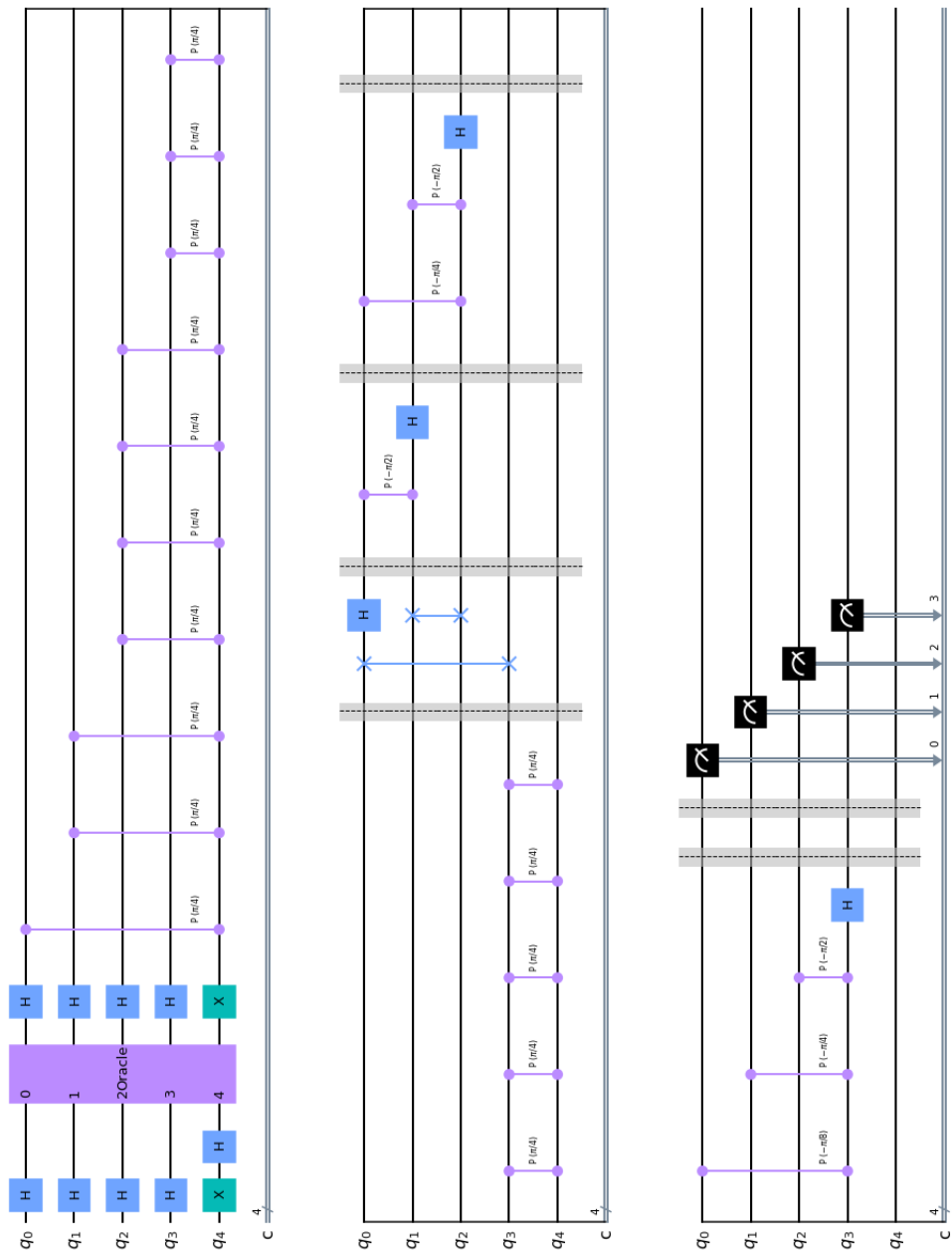
7.2. Περίπτωση 1

7.2.1. Constant Oracle

Στην πρώτη περίπτωση του κυκλώματος που σχεδιάστηκε, το κύκλωμα θα έχει την μορφή: Hadamard, Oracle, Κβαντικός μετασχηματισμός Fourier και θα εκτελεστεί για 4 qubits. Ο κώδικας της συνάρτησης "circuit_creation" έχει την παρακάτω μορφή:

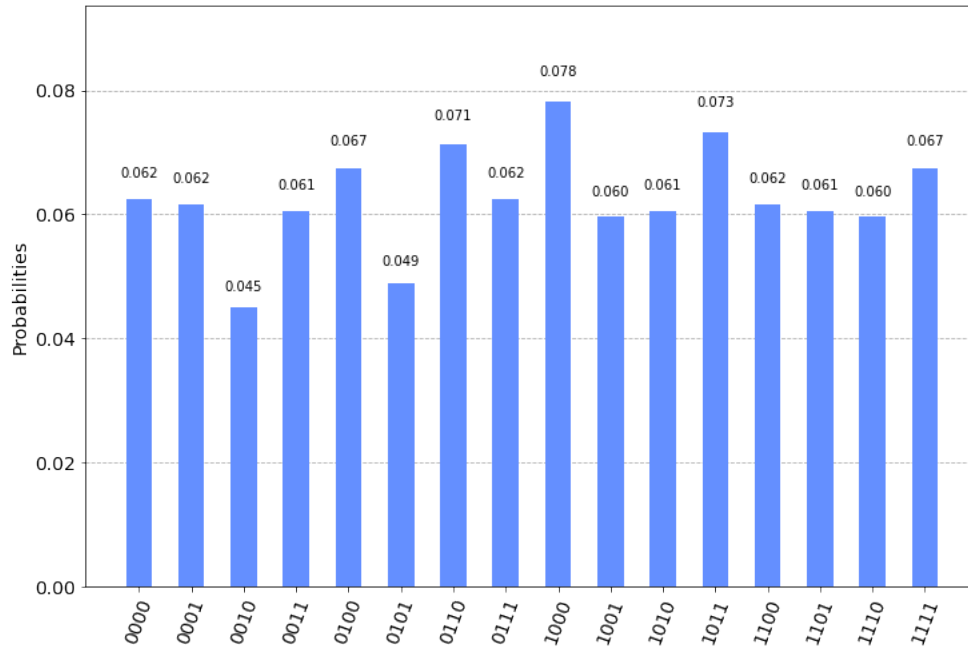
```
1 # basic deutsch jozsa function
2 def circuit_creation(oracle, n):
3     dj_circuit = QuantumCircuit(n+1, n)
4     dj_circuit.x(n)
5     dj_circuit.h(n)
6
7     number_of_qubits = n+1
8     for qubit in range(n):
9         dj_circuit.h(qubit)
10
11     # apply_gft(number_of_qubits ,dj_circuit)
12
13     dj_circuit.append(oracle, range(n+1))
14
15     # for qubit in range(n):
16     #     dj_circuit.h(qubit)
17
18     apply_gft(number_of_qubits,dj_circuit)
19
20     dj_circuit.barrier()
21
22     for i in range(n):
23         dj_circuit.measure(i, i)
24
25     return dj_circuit
```

Αρχικά, το Oracle θα είναι Constant. Στην περίπτωση αυτή, για 4 qubits το κύκλωμα έχει την παρακάτω μορφή.



Σχήμα 22: Κύκλωμα που προκύπτει για την πρώτη περίπτωση με χρήση του Constant Oracle, για 4 qubits

Στο παρακάτω κύκλωμα, πριν την Oracle, γίνεται προσθήκη των πυλών Hadamard, όπως είναι φυσιολογικά η υλοποίηση του αλγορίθμου, αλλά μετά την Oracle, γίνεται προσθήκη του κβαντικού μετασχηματισμού Fourier. Η εικόνα 23, παρακάτω δείχνει τα δεδομένα εξόδου από την υλοποίηση του παραπάνω κώδικα με την χρήση του Constant Oracle.

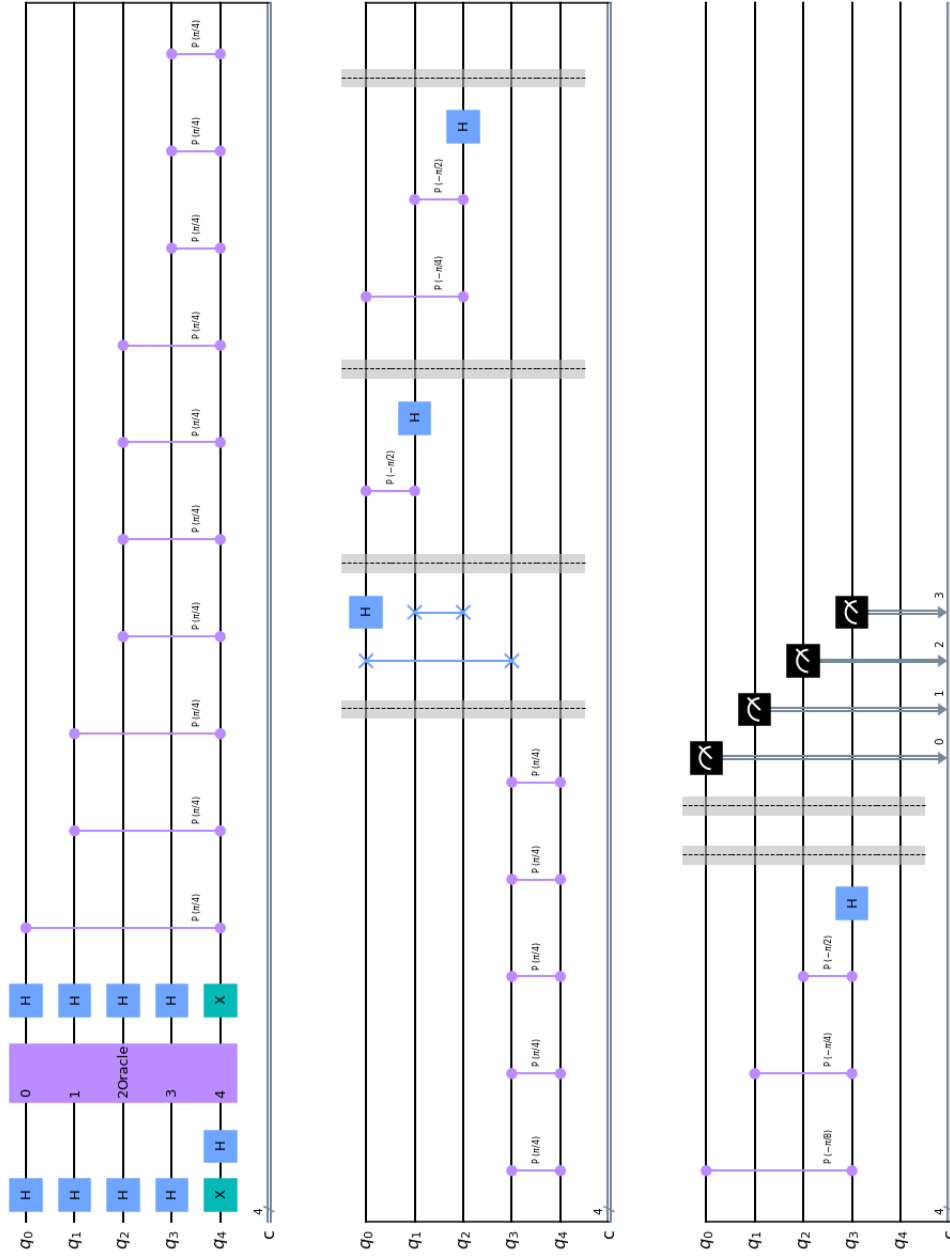


Σχήμα 23: Αποτέλεσμα που προκύπτει για την πρώτη περίπτωση με χρήση του Constant Oracle, για 4 qubits

Από τα αποτελέσματα εξόδου του κώδικα, προκύπτουν σε δυαδική μορφή οι αριθμοί από το "0000" έως το "1111" οι οποίοι σε δεκαδική μορφή είναι οι αριθμοί από το 0 έως το 15. Παρόμοιο αποτέλεσμα προκύπτει για μεγαλύτερο πλήθος qubits. Το αποτέλεσμα αυτό δηλώνει πως για το Constant Oracle, ο κβαντικός μετασχηματισμός Fourier εξουδετερώνει τις πύλες Hadamard και βγαίνουν ως έξοδος, με ίδια περίπου πιθανότητα, οι αριθμοί από το 0 έως το 15.

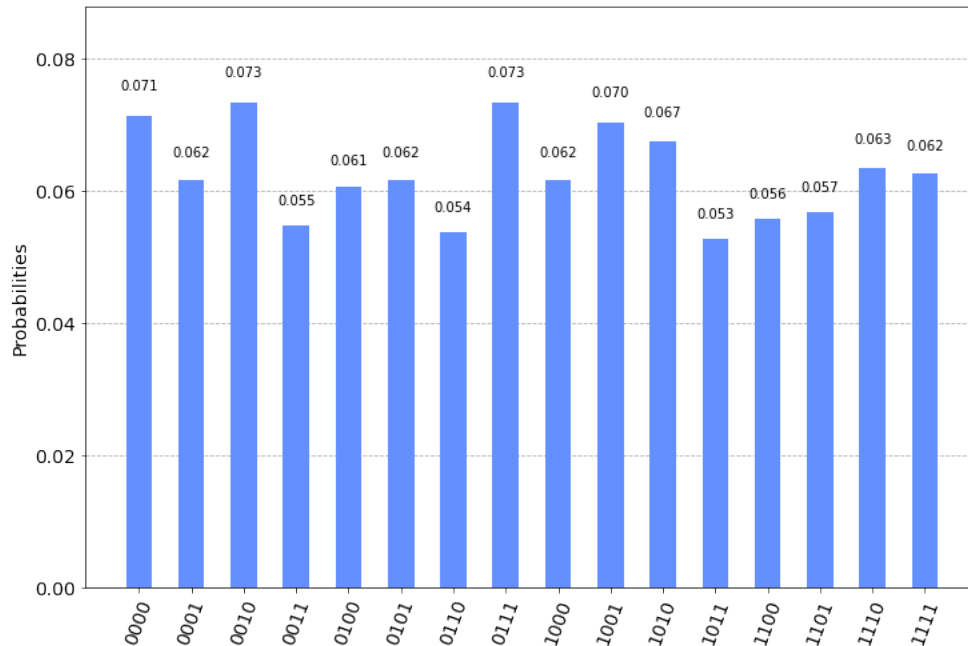
7.2.2. *Balanced Oracle*

Ομοίως, η περίπτωση για το Balanced Oracle. Ο κώδικας της συνάρτησης "circuit_creation" παραμένει ο ίδιος με την ενότητα 7.2.1. Το κύκλωμα που θα σχεδιαστεί είναι της παρακάτω μορφής 24:



Σχήμα 24: Κύκλωμα που προκύπτει για την πρώτη περίπτωση με χρήση του Balanced Oracle, για 4 qubits

Παρακάτω φαίνεται το διάγραμμα με τα δεδομένα εξόδου:



Σχήμα 25: Αποτέλεσμα που προκύπτει για την πρώτη περίπτωση με χρήση του Balanced Oracle, για 4 qubits

Τα αποτελέσματα που προκύπτουν από την εκτέλεση αυτή είναι ίδια με αυτά του κεφαλαίου 7.2.1. Για την ακρίβεια, το μοτίβο που έχει παρατηρηθεί, είναι πως προκύπτουν οι αριθμοί από το 0 έως το 15 σε δεκαδικό σύστημα, με ίση πιθανότητα εμφάνισης του κάθε στοιχείου.

7.3. Περίπτωση 2

7.3.1. Constant Oracle

Στην δεύτερη περίπτωση του κυκλώματος που σχεδιάστηκε, το κύκλωμα θα έχει την μορφή: Κβαντικός μετασχηματισμός Fourier, Oracle, Hadamard και θα εκτελεστεί για 4 qubits. Ο κώδικας της συνάρτησης "circuit_creation" έχει την παρακάτω μορφή:

```

1 # basic deutsch jozsa function
2 def circuit_creation(oracle, n):
3     dj_circuit = QuantumCircuit(n+1, n)
4     dj_circuit.x(n)

```

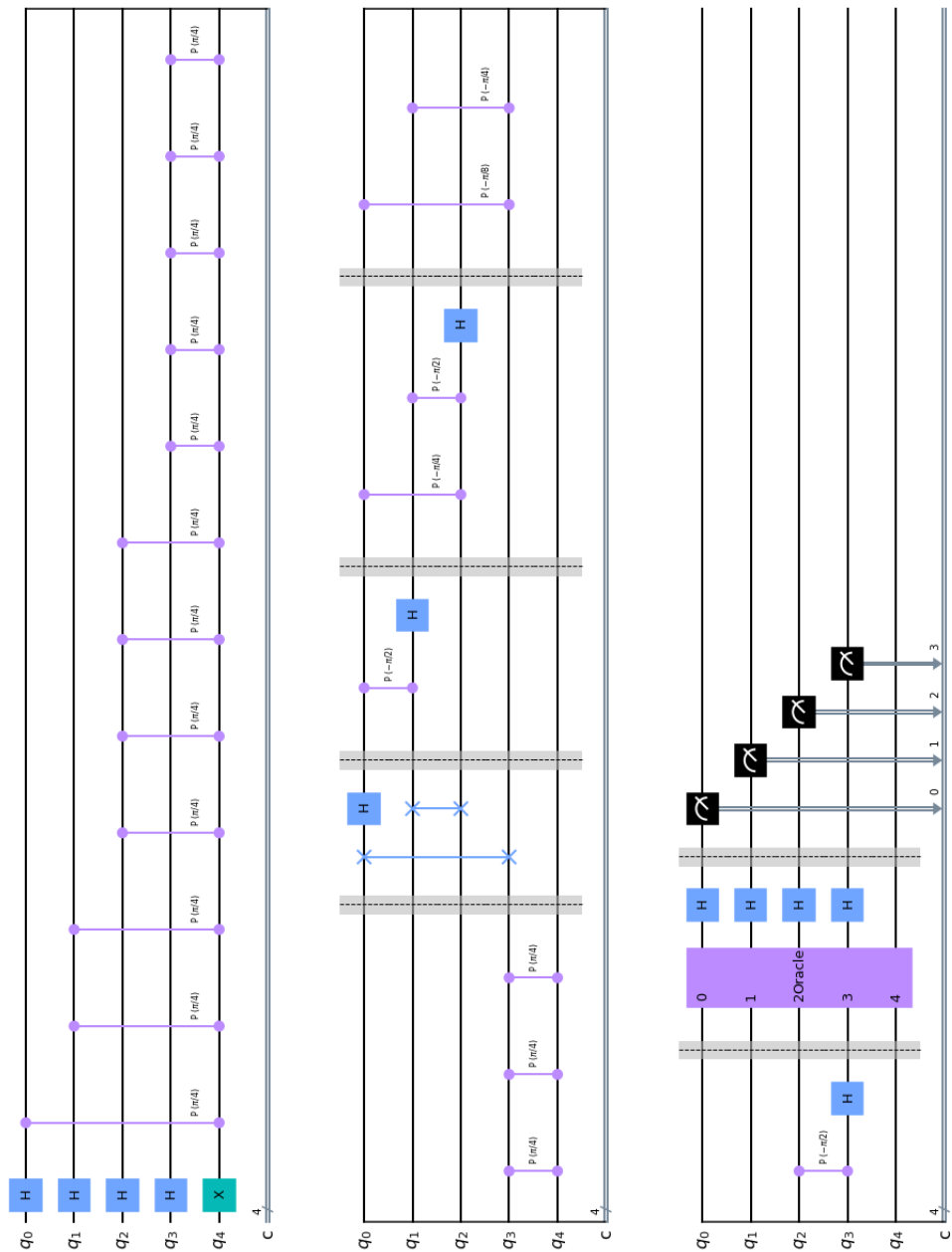
```

5  # dj_circuit.h(n)
6
7  number_of_qubits = n+1
8  # for qubit in range(n):
9  #    dj_circuit.h(qubit)
10
11  apply_gft(number_of_qubits ,dj_circuit)
12
13  dj_circuit.append(oracle, range(n+1))
14
15  for qubit in range(n):
16      dj_circuit.h(qubit)
17
18  # apply_gft(number_of_qubits,dj_circuit)
19
20  dj_circuit.barrier()
21
22  for i in range(n):
23      dj_circuit.measure(i, i)
24
25  return dj_circuit

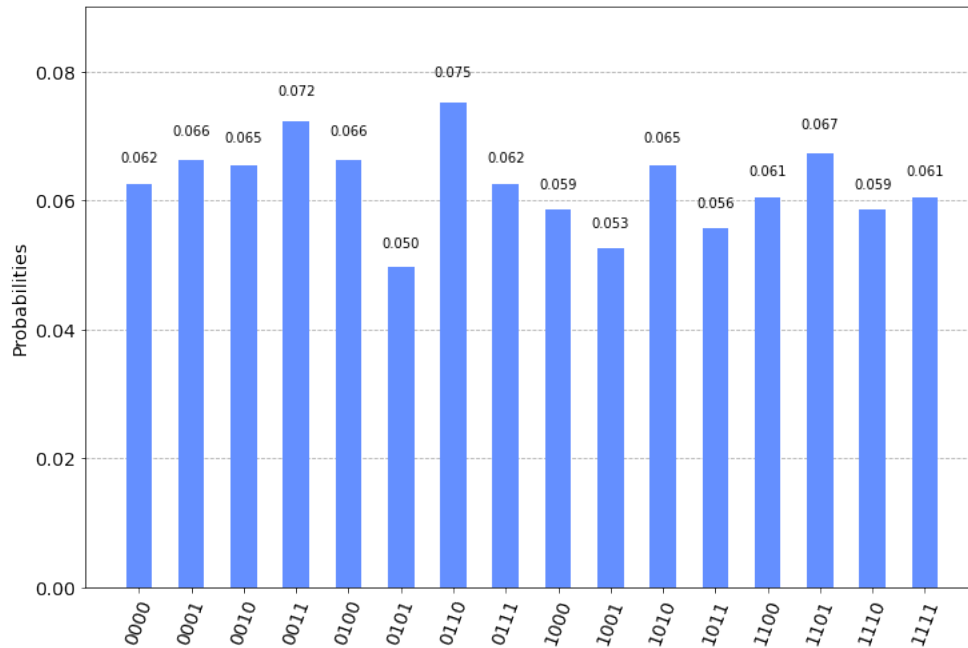
```

Αρχικά, το Oracle θα είναι Constant. Στην περίπτωση αυτή, για 4 qubits το κύκλωμα έχει την παρακάτω μορφή [26](#).

Στο παρακάτω κύκλωμα, πριν την Oracle, γίνεται προσθήκη προσθήκη του κβαντικού μετασχηματισμού Fourier, μετά γίνεται προσθήκης της Oracle και τέλος, γίνεται προσθήκη των πυλών Hadamard. Η εικόνα [27](#), παρακάτω, δείχνει τα δεδομένα εξόδου από την υλοποίηση του παραπάνω κώδικα με την χρήση του Constant Oracle.



Σχήμα 26: Κώδικωμα που προκύπτει για την δεύτερη περίπτωση με χρήση του Constant Oracle, για 4 qubits



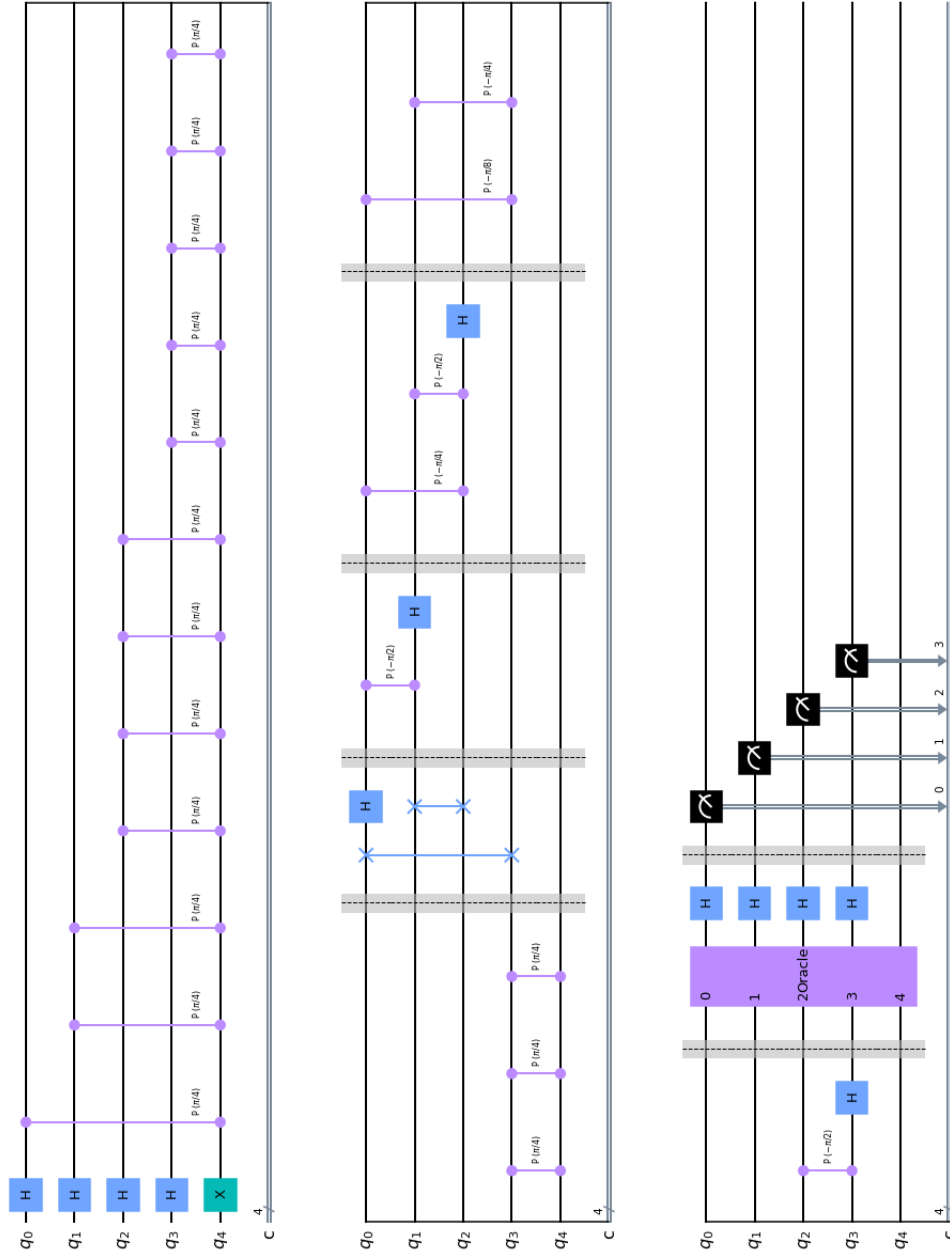
Σχήμα 27: Αποτέλεσμα που προκύπτει για την δεύτερη περίπτωση με χρήση του Constant Oracle, για 4 qubits

Στην περίπτωση αυτή, τα αποτελέσματα είναι ίδια με την ενότητα 7.2.1 εφόσον, όπως και πριν, ο κβαντικός μετασχηματισμός Fourier εξουδετερώνει την δράση των πυλών Hadamard.

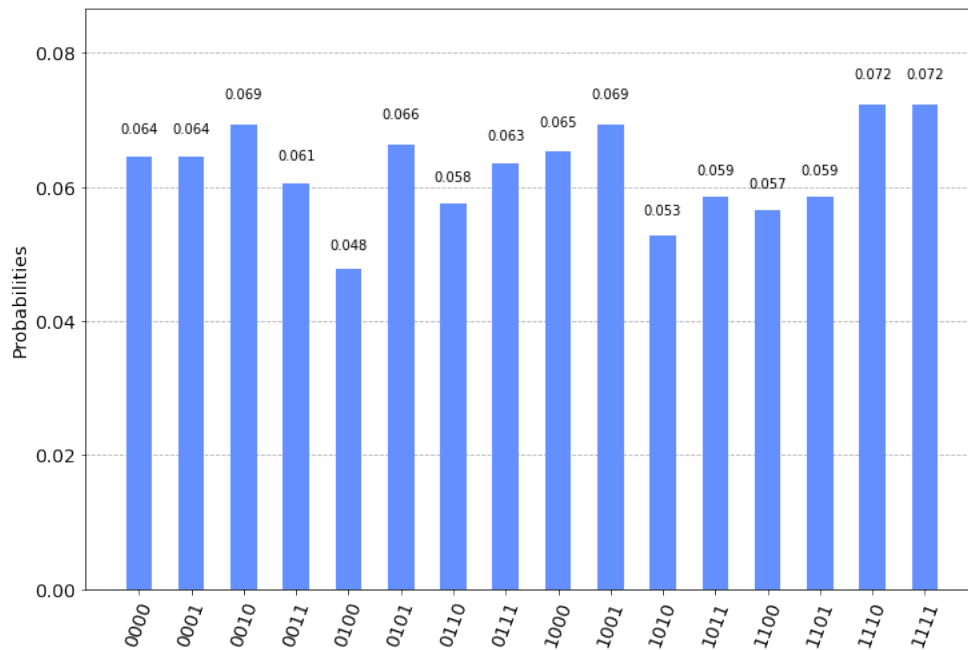
7.3.2. *Balanced Oracle*

Ομοίως, η περίπτωση για το Balanced Oracle. Ο κώδικας της συνάρτησης "circuit_creation" παραμένει ο ίδιος με την ενότητα 7.3.1. Το κύκλωμα που θα σχεδιαστεί είναι της παρακάτω μορφής: 28:

Τα παρακάτω αποτελέσματα, στο σχήμα 29, φανερώνουν πως σε δεκαδικό σύστημα, οι τιμές που μπορεί να πάρει το σύστημα είναι οι τιμές από το 0 έως το 15 και εμφανίζονται με ίση περίπου πιθανότητα, όπως και στην προηγούμενη περίπτωση.



Σχήμα 28: Κώδικωμα που προκύπτει για την δεύτερη περίπτωση με χρήση του Balanced Oracle, για 4 qubits



Σχήμα 29: Αποτέλεσμα που προκύπτει για την δεύτερη περίπτωση με χρήση του Balanced Oracle, για 4 qubits

7.4. Περίπτωση 3

7.4.1. Constant Oracle

Στην πρώτη περίπτωση του κυκλώματος που σχεδιάστηκε, το κύκλωμα θα έχει την μορφή: Κβαντικός μετασχηματισμός Fourier, Oracle, Κβαντικός μετασχηματισμός Fourier και θα εκτελεστεί για 4 qubits. Ο κώδικας της συνάρτησης "circuit_creation" έχει την παρακάτω μορφή:

```

1 # basic deutsch jozsa function
2 def circuit_creation(oracle, n):
3     dj_circuit = QuantumCircuit(n+1, n)
4     # dj_circuit.x(n)
5     # dj_circuit.h(n)
6
7     number_of_qubits = n+1
8     # for qubit in range(n):
9     #     dj_circuit.h(qubit)
10
11     apply_gft(number_of_qubits ,dj_circuit)

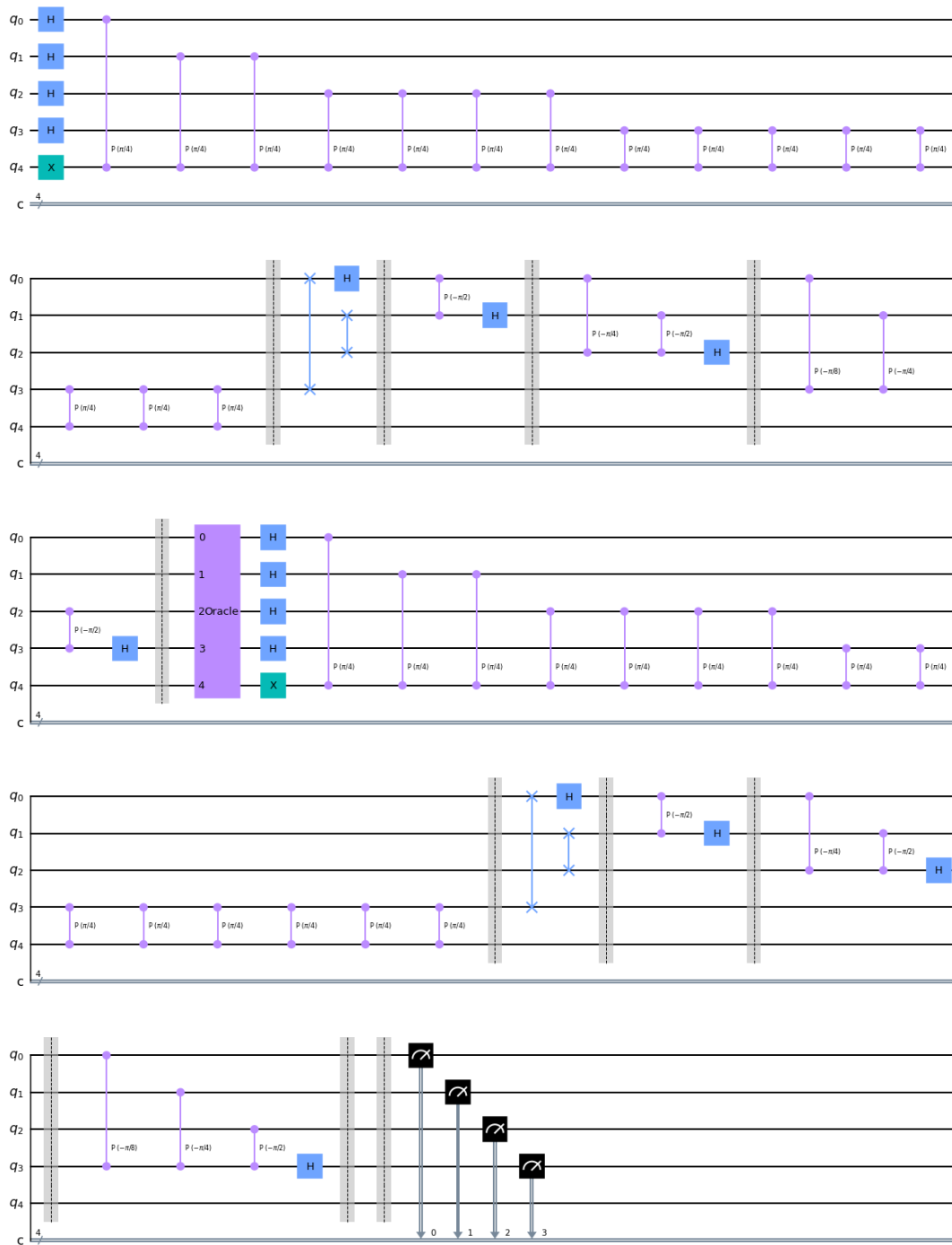
```

```

12
13  dj_circuit.append(oracle, range(n+1))
14
15  # for qubit in range(n):
16  #    dj_circuit.h(qubit)
17
18  apply_gft(number_of_qubits,dj_circuit)
19
20  dj_circuit.barrier()
21
22  for i in range(n):
23      dj_circuit.measure(i, i)
24
25  return dj_circuit

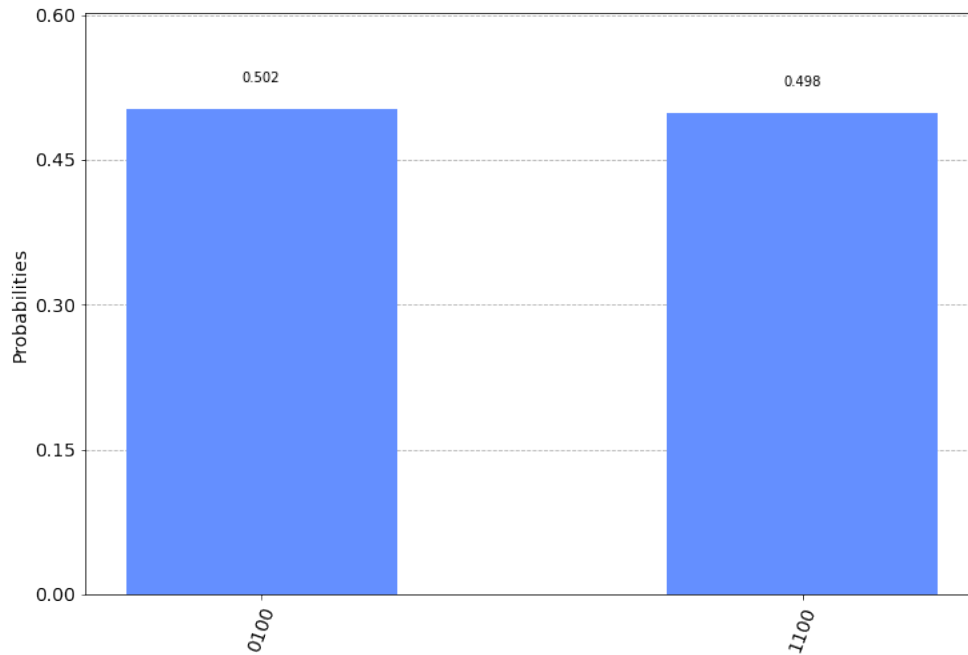
```

Αρχικά, το Oracle θα είναι Constant. Στην περίπτωση αυτή, για 4 qubits το κύκλωμα έχει την παρακάτω μορφή [30](#).



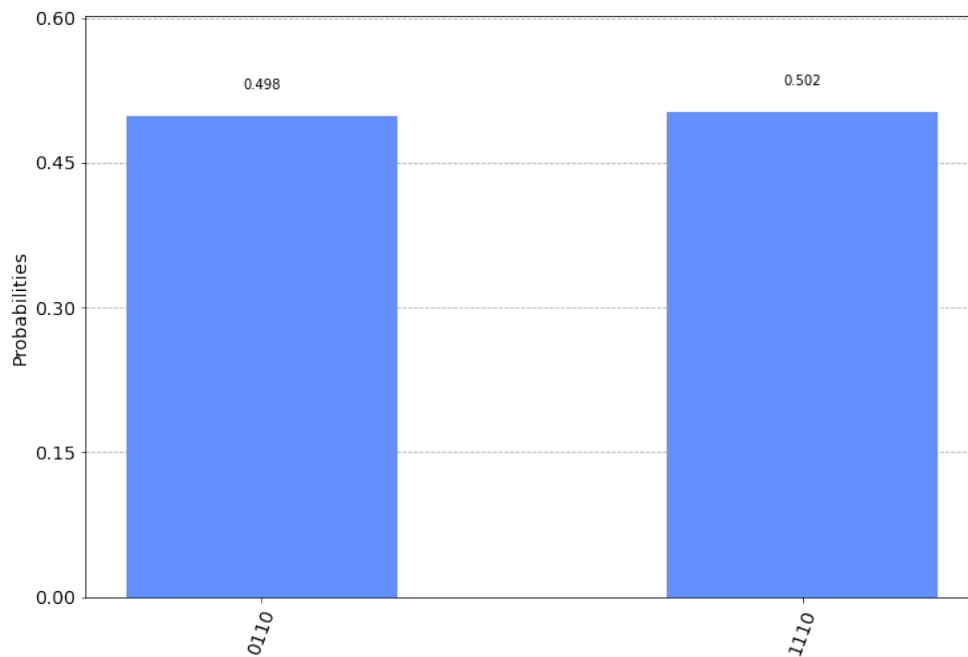
Σχήμα 30: Κύκλωμα που προκύπτει για την τρίτη περίπτωση, για 4 qubits

Στο παρακάτω κύκλωμα, γίνεται προσθήκη του Κβαντικού μετασχηματισμού Fourier, μετά γίνεται προσθήκη της Oracle και στο τέλος, για δεύτερη φορά, γίνεται προσθήκη του κβαντικού μετασχηματισμού Fourier. Η εικόνα 31, παρακάτω δείχνει τα δεδομένα εξόδου από την υλοποίηση του παραπάνω κώδικα με την χρήση του Constant Oracle.



Σχήμα 31: Αποτέλεσμα που προκύπτει για την τρίτη περίπτωση με χρήση του Constant Oracle, για 4 qubits

Στην περίπτωση αυτή, τα αποτελέσματα διαφέρουν αρκετά από τις προηγούμενες περιπτώσεις, όπως φαίνεται στο σχήμα 31. Για την ακρίβεια προκύπτει μόνο δύο πιθανές λύσεις, με ίση πιθανότητα εμφάνισης. Στην περίπτωση του συγκεκριμένου κυκλώματος του αριθμού προκύπτουν οι αριθμοί 4 και 12 σε δεκαδικό σύστημα. Στο σημείο αυτό αξίζει να σημειωθεί πως στο Constant Oracle, η τιμή του "output" στην συνάρτηση "dj_oracle" στην περίπτωση του Constant Oracle έχει την τιμή 0. Στην περίπτωση που η τιμή του "output" είναι 1 τότε το αποτέλεσμα που προκύπτει είναι ίσο με "0110" και "1110" δηλαδή σε δεκαδικό σύστημα 6 και 14 όπως φαίνεται στην εικόνα 32.

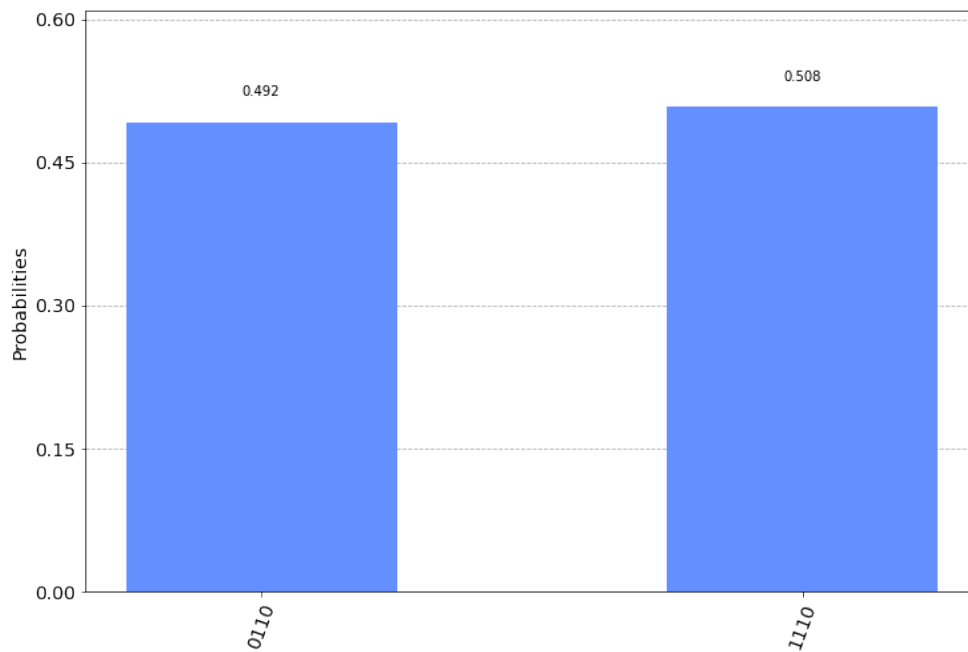


Σχήμα 32: Κύκλωμα που προκύπτει για την τρίτη περίπτωση, για 4 qubits, όταν το output έχει την τιμή 1

7.4.2. *Balanced Oracle*

Ομοίως, η περίπτωση για το Balanced Oracle. Ο κώδικας της συνάρτησης "circuit_creation" παραμένει ο ίδιος με την ενότητα 7.3.1 όπως και το κύκλωμα που προκύπτει είναι ίδιο με το σχήμα 7.2.2:

Παρακάτω φαίνεται το διάγραμμα με τα δεδομένα εξόδου:



Σχήμα 33: Αποτέλεσμα που προκύπτει για την τρίτη περίπτωση με χρήση του Balanced Oracle, για 4 qubits

Στην περίπτωση αυτή, τα αποτελέσματα εξόδου στο δεκαδικό σύστημα έχουν τις τιμές 4 και 12 σε ίσες περίπου πιθανότητες εμφάνισης.

8. Χρήση κβαντικού μετασχηματισμού Fourier στον αλγόριθμο των Bernstein Vazirani

Στο παρόν κεφάλαιο θα γίνει εκτέλεση και παρουσίαση των αποτελεσμάτων, από την προσθήκη του Κβαντικού μετασχηματισμού Fourier στον αλγόριθμο των Bernstein Vazirani. Στην πλειοψηφία των αποτελεσμάτων, δεν βρέθηκε κάποιο μοτίβο, όπως στην περίπτωση του αλγορίθμου των Deutsch Jozsa. Παρόλα αυτά, θα παρουσιαστούν 3 περιπτώσεις εκτέλεσης, για τυχαίες συμβολοσειρές s μήκους 2, 3 και 4^+ qubits.

Για την καλύτερη περιγραφή των αποτελεσμάτων θα παρουσιαστεί ο κώδικας που δημιουργήθηκε στα προηγούμενα κεφάλαια με μερικές τροποποιήσεις ανάλογα την περίπτωση. Σε κάθε παράδειγμα θα παρουσιάζεται μόνο το μέρος του κώδικα που τροποποιείται. Στο κεφάλαιο [8.1](#) θα παρουσιαστεί συνολικά ο κώδικας έχοντας σε σχόλια τμήματα του που θα τροποποιηθούν στις ανάλογες περιπτώσεις.

8.1. Πλήρες κώδικας

Αρχικά, θα γίνει η προσθήκη των απαιτούμενων βιβλιοθηκών. Η προσθήκη γίνεται με τον εξής τρόπο:

```
1 # import libraries
2
3 #initialization
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import math
7
8 # importing Qiskit
9 from qiskit import IBMQ, Aer, transpile, assemble
10 from qiskit import QuantumCircuit, ClassicalRegister,
    QuantumRegister
11 from qiskit.providers.ibmq import least_busy
12
13 # import basic plot tools
14 from qiskit.visualization import plot_histogram
```

Έπειτα, θα γίνει προσθήκη των συναρτήσεων για τον Κβαντικό μετασχηματισμό Fourier και τον Αντίστροφο μετασχηματισμό Fourier, όπως περιγράφηκε στην ενότητα [4.8](#).

```

1 # Reverse QFT
2 def reverse_QFT(circuit, number_of_qubits):
3     # add the swaps!
4     for qubit in range(number_of_qubits//2):
5         circuit.swap(qubit, number_of_qubits-qubit-1)
6     for j in range(number_of_qubits):
7         for k in range(j):
8             circuit.cp(-math.pi/float(2**(j-k)), k, j)
9         circuit.h(j)

```

```

1 # Basic quantum fourier transform function
2 def apply_gft(number_of_qubits,circuit):
3
4     # number_of_qubits = number_of_qubits + 1
5     # apply an X gate to the eigenstate quibit
6     qtf_circuit = circuit
7     qtf_circuit.x(number_of_qubits-1)
8
9     # apply Hadamard gates to the counting qubits
10    for qubit in range(number_of_qubits-1):
11        qtf_circuit.h(qubit)
12
13    # repetitions is set to 1
14    repetitions = 1
15
16    # apply all the nesenary gates as many times as the repetitions
17    for counting_qubit in range(number_of_qubits-1):
18        for i in range(repetitions):
19            qtf_circuit.cp(math.pi/4, counting_qubit,
20                number_of_qubits-1); # This is CU
21            repetitions *= 2
22        # qtf_circuit.draw()
23
24    qtf_circuit.barrier()
25    # Apply inverse QFT
26    qft_dagger(qtf_circuit, number_of_qubits-1)
27
28    # return the circuit
29    return qtf_circuit

```

Έπειτα γίνεται η δημιουργία των της βασικής συνάρτησης για τον αλγόριθμο των Bernstein Vazirani:

```

1 def circuit_creation(s):

```

```

2  number_of_qubits = len(s)
3  # We need a circuit with n qubits, plus one auxiliary qubit
4  # Also need n classical bits to write the output to
5  circuit = QuantumCircuit(number_of_qubits+1, number_of_qubits)
6
7  # put auxiliary in state |->
8  circuit.h(number_of_qubits)
9  circuit.z(number_of_qubits)
10
11 # Apply Hadamard gates before querying the oracle
12 for i in range(number_of_qubits):
13     circuit.h(i)
14
15 # apply_gft(number_of_qubits ,circuit)
16
17 # Apply barrier
18 circuit.barrier()
19
20 # Apply the inner-product oracle
21 # reverse s to fit qiskit's qubit ordering
22 s = s[::-1]
23 for q in range(number_of_qubits):
24     if s[q] == '0':
25         circuit.i(q)
26     else:
27         circuit.cx(q, number_of_qubits)
28
29 # Apply barrier
30 circuit.barrier()
31
32 #Apply Hadamard gates after querying the oracle
33 for i in range(number_of_qubits):
34     circuit.h(i)
35
36 # apply_gft(number_of_qubits ,circuit)
37
38 # Measurement
39 for i in range(number_of_qubits):
40     circuit.measure(i, i)
41
42 return circuit

```

Έπειτα, καλούνται οι συναρτήσεις, εντυπώνονται τα κυκλώματα και εκτε-

λούνται σε προσομοιωτές κβαντικών υπολογιστών. Για την περιγραφή του κυκλώματος βάζουμε να εκτελεστούν τα κυκλώματα τυχαίας συμβολοσειράς s . Στην πορεία των κεφαλαίων θα σχολιαστεί αναλυτικά η απόφαση για την εκάστοτε συμβολοσειρά.

```
1 s = '111' # the hidden binary string
2 circuit = circuit_creation(s)
3 circuit.draw(output="mpl")

1 # run simulations on quantum system
2 aer_sim = Aer.get_backend('qasm_simulator')
3
4 transpiled_dj_circuit = transpile(circuit, aer_sim)
5 qobj = assemble(transpiled_dj_circuit)
6
7 # get results and create histogram
8 results = aer_sim.run(qobj).result()
9 answer = results.get_counts()
10 plot_histogram(answer)
```

Στα επόμενα κεφάλαια, για κάθε μια από τις περιπτώσεις που θα σχολιαστούν θα γίνει η περιγραφή των αποτελεσμάτων για τις διαφορετικές τιμές και για τα διαφορετικά μήκη της συμβολοσειράς s . Συνολικά, θα παρουσιαστούν 3 περιπτώσεις:

- Το μήκος της συμβολοσειράς s έχει μήκος 2 qubits
- Το μήκος της συμβολοσειράς s έχει μήκος 3 qubits
- Το μήκος της συμβολοσειράς s έχει μήκος 4+ qubits

8.2. Περιγραφή κώδικα για τις τρεις διαφορετικές περιπτώσεις

Στην ενότητα αυτή, θα παρουσιαστούν 3 εκδοχές του κώδικα, μία για κάθε πιθανή περίπτωση της θέσης του κβαντικού μετασχηματισμού Fourier.

Η πρώτη περίπτωση, όπως και στην προηγούμενη ενότητα, είναι αρχικά η προσθήκη των πυλών Hadamard και έπειτα, μετά την Oracle, η προσθήκη του κβαντικού μετασχηματισμού Fourier. Παρακάτω φαίνεται ο κώδικας και το κύκλωμα που παράγεται από την συνάρτηση "circuit_creation":

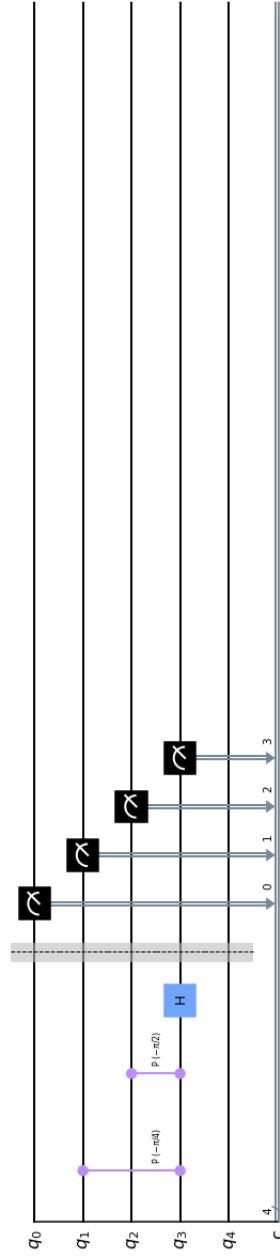
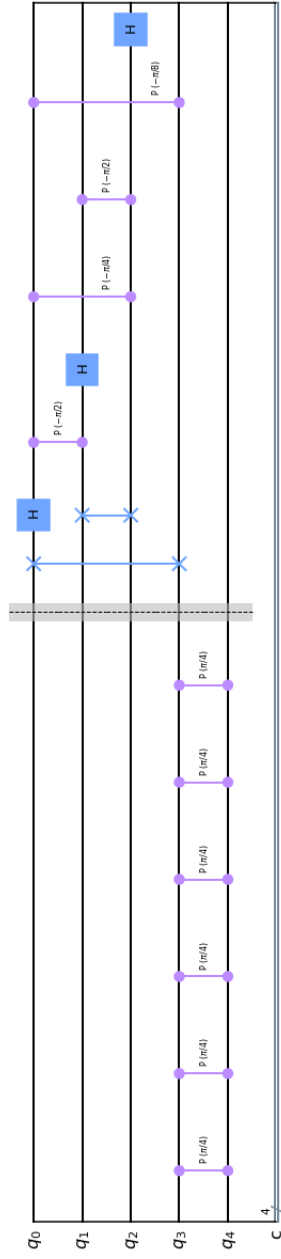
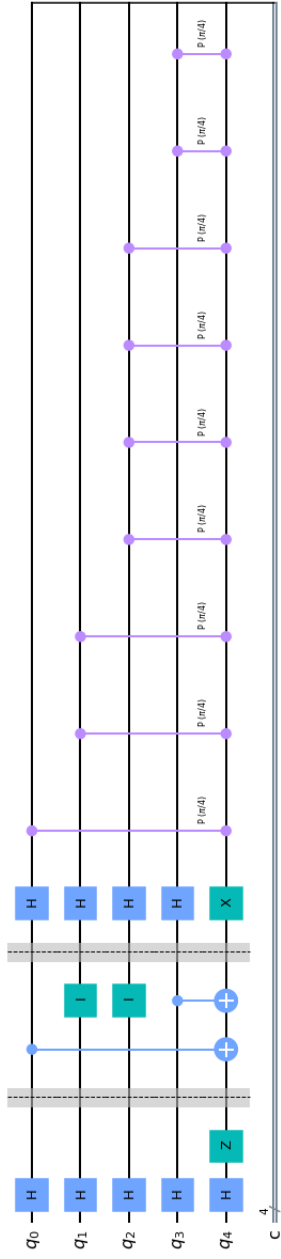
```
1 def circuit_creation(s):
2     number_of_qubits = len(s)
3     # We need a circuit with n qubits, plus one auxiliary qubit
```

```

4  # Also need n classical bits to write the output to
5  circuit = QuantumCircuit(number_of_qubits+1, number_of_qubits)
6
7  # put auxiliary in state |->
8  circuit.h(number_of_qubits)
9  circuit.z(number_of_qubits)
10
11 # Apply Hadamard gates before querying the oracle
12 for i in range(number_of_qubits):
13     circuit.h(i)
14
15 # apply_gft(number_of_qubits ,circuit)
16
17 # Apply barrier
18 circuit.barrier()
19
20 # Apply the inner-product oracle
21 # reverse s to fit qiskit's qubit ordering
22 s = s[::-1]
23 for q in range(number_of_qubits):
24     if s[q] == '0':
25         circuit.i(q)
26     else:
27         circuit.cx(q, number_of_qubits)
28
29 # Apply barrier
30 circuit.barrier()
31
32 #Apply Hadamard gates after querying the oracle
33 # for i in range(number_of_qubits):
34 #     circuit.h(i)
35
36 apply_gft(number_of_qubits ,circuit)
37
38 # Measurement
39 for i in range(number_of_qubits):
40     circuit.measure(i, i)
41
42 return circuit

```

Το κύκλωμα το οποίο προκύπτει είναι στο σχήμα [34](#).



Σχήμα 34: Κύκλωμα για την πρώτη περίπτωση με το μήκος της συμβολοσειράς s να ισούται με 4

Η δεύτερη περίπτωση, είναι αρχικά η προσθήκη του κβαντικού μετασχηματισμού Fourier και έπειτα, μετά την Oracle, η προσθήκη των πυλών Hadamard. Παρακάτω φαίνεται ο κώδικας και το κύκλωμα που παράγεται από την συνάρτηση "circuit_creation":

```
1 def circuit_creation(s):
2     number_of_qubits = len(s)
3     # We need a circuit with n qubits, plus one auxiliary qubit
4     # Also need n classical bits to write the output to
5     circuit = QuantumCircuit(number_of_qubits+1, number_of_qubits)
6
7     # put auxiliary in state |->
8     # circuit.h(number_of_qubits)
9     # circuit.z(number_of_qubits)
10
11     # Apply Hadamard gates before querying the oracle
12     # for i in range(number_of_qubits):
13     #     circuit.h(i)
14
15     apply_gft(number_of_qubits ,circuit)
16
17     # Apply barrier
18     circuit.barrier()
19
20     # Apply the inner-product oracle
21     # reverse s to fit qiskit's qubit ordering
22     s = s[::-1]
23     for q in range(number_of_qubits):
24         if s[q] == '0':
25             circuit.i(q)
26         else:
27             circuit.cx(q, number_of_qubits)
28
29     # Apply barrier
30     circuit.barrier()
31
32     #Apply Hadamard gates after querying the oracle
33     for i in range(number_of_qubits):
34         circuit.h(i)
35
36     # apply_gft(number_of_qubits ,circuit)
37
38     # Measurement
```

```

39     for i in range(number_of_qubits):
40         circuit.measure(i, i)
41
42     return circuit

```

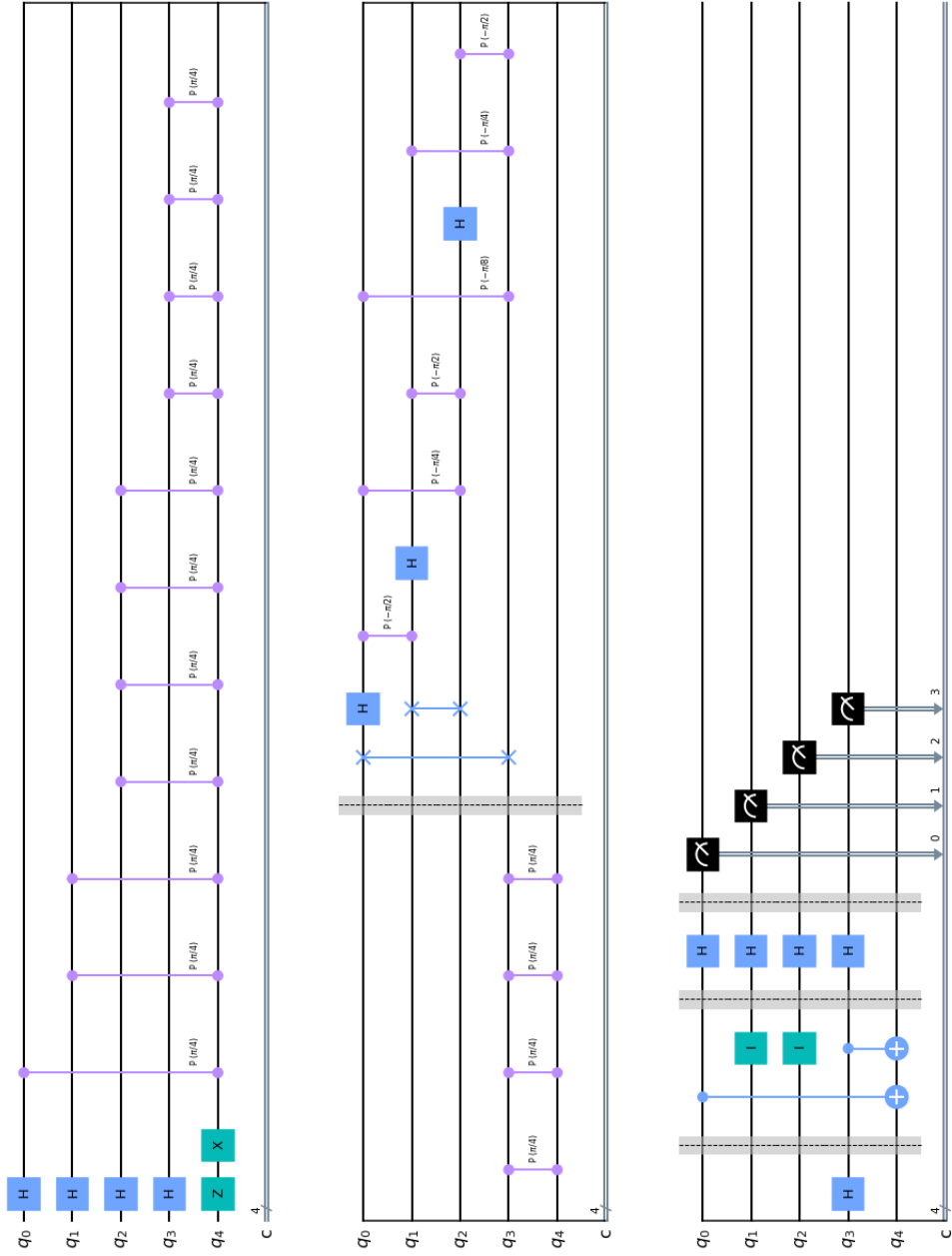
Το κύκλωμα το οποίο προκύπτει είναι στο σχήμα 35.

Η τελευταία περίπτωση, είναι αρχικά η προσθήκη του κβαντικού μετασχηματισμού Fourier και έπειτα, μετά την Oracle, η προσθήκη για δεύτερη φορά, του κβαντικού μετασχηματισμού Fourier. Παρακάτω φαίνεται ο κώδικας και το κύκλωμα που παράγεται από την συνάρτηση "circuit_creation":

```

1 def circuit_creation(s):
2     number_of_qubits = len(s)
3     # We need a circuit with n qubits, plus one auxiliary qubit
4     # Also need n classical bits to write the output to
5     circuit = QuantumCircuit(number_of_qubits+1, number_of_qubits)
6
7     # put auxiliary in state |->
8     # circuit.h(number_of_qubits)
9     # circuit.z(number_of_qubits)
10
11     # Apply Hadamard gates before querying the oracle
12     # for i in range(number_of_qubits):
13     #     circuit.h(i)
14
15     apply_gft(number_of_qubits ,circuit)
16
17     # Apply barrier
18     circuit.barrier()
19
20     # Apply the inner-product oracle
21     # reverse s to fit qiskit's qubit ordering
22     s = s[::-1]
23     for q in range(number_of_qubits):
24         if s[q] == '0':
25             circuit.i(q)
26         else:
27             circuit.cx(q, number_of_qubits)
28
29     # Apply barrier
30     circuit.barrier()
31

```



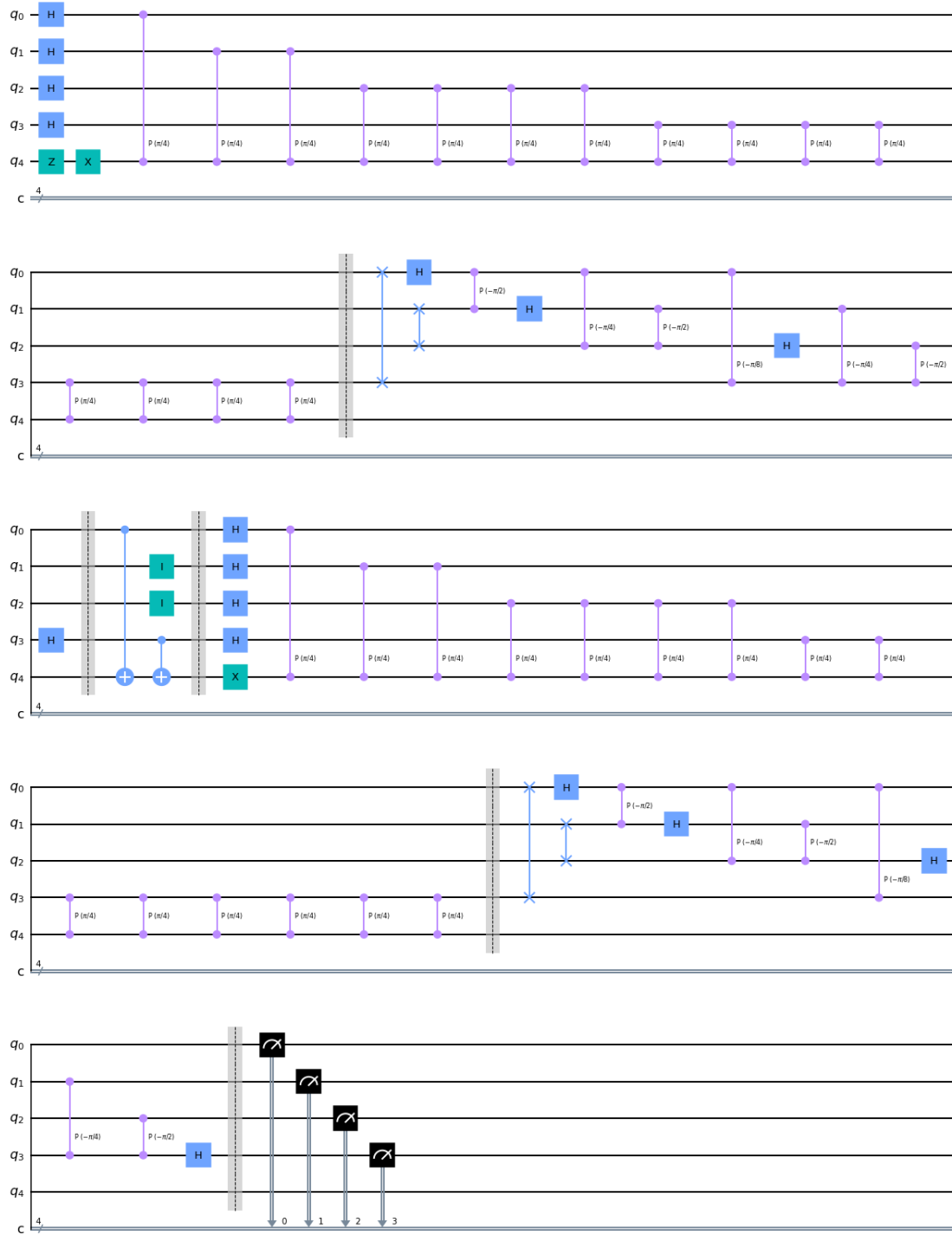
Σχήμα 35: Κύκλωμα για την δεύτερη περίπτωση με το μήκος της συμβολοσειράς s να ισούται με 4

```

32 #Apply Hadamard gates after querying the oracle
33 # for i in range(number_of_qubits):
34 #     circuit.h(i)
35
36 apply_gft(number_of_qubits ,circuit)
37
38 # Measurement
39 for i in range(number_of_qubits):
40     circuit.measure(i, i)
41
42 return circuit

```

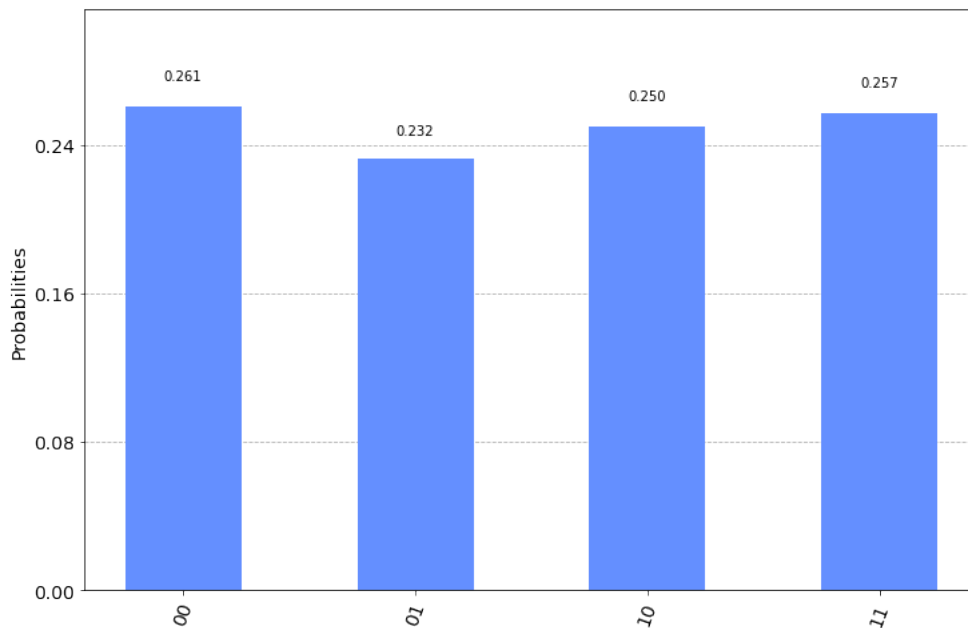
Το κύκλωμα το οποίο προκύπτει είναι στο σχήμα [36](#).



Σχήμα 36: Κύκλωμα για την τρίτη περίπτωση με το μήκος της συμβολοσειράς s να ισούται με 4

8.3. Περίπτωση 1: Το μήκος της συμβολοσειράς s έχει μήκος 2 qubits

Σε κάθε κεφάλαιο, για κάθε διαφορετικό μήκος της συμβολοσειράς s θα γίνεται περιγραφή των αποτελεσμάτων για κάθε μια από τις περιπτώσεις που σχολιάστηκαν στην ενότητα 8.2. Σε κάθε περίπτωση, τα διαγράμματα που προκύπτουν είναι για την συμβολοσειρά $s = "11"$. Αρχικά, θα σχολιαστεί η περίπτωση του κυκλώματος 34. Για την περίπτωση αυτή, το αποτέλεσμα που προκύπτει για την συμβολοσειρά s φαίνεται στην παρακάτω εικόνα:

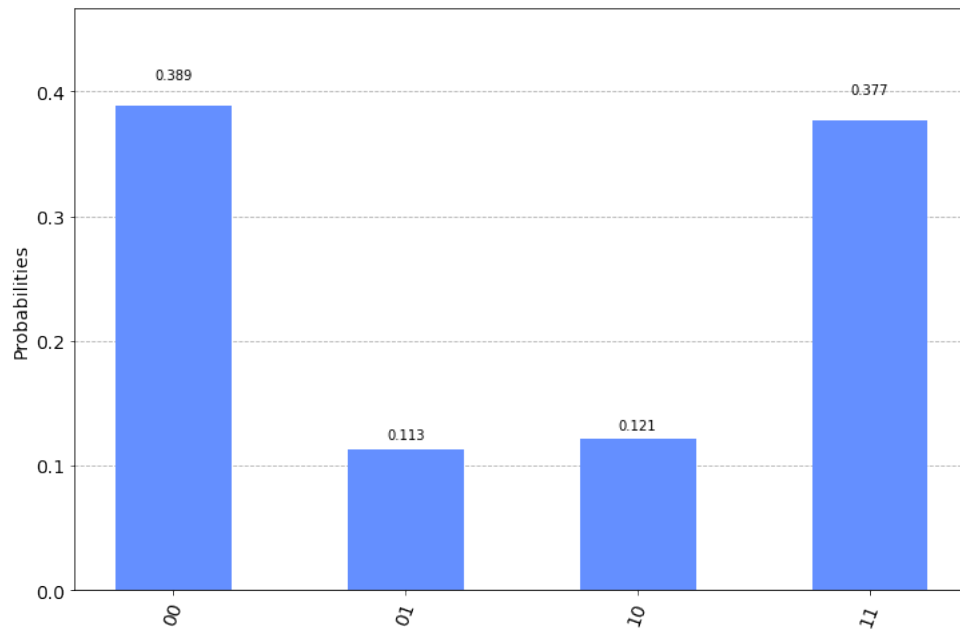


Σχήμα 37: Αποτέλεσμα για την πρώτη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 2

Όπως φαίνεται από το παραπάνω σχήμα, το αποτέλεσμα που προκύπτει είναι οι αριθμοί από "00" έως "11" στο δυαδικό σύστημα, με ίση πιθανότητα. Επομένως προκύπτουν όλες οι πιθανές τιμές με ίση πιθανότητα εμφάνισης, που αποδικνύει πως ο κβαντικός μετασχηματισμός Fourier εξουδετερώνει την δράση των πυλών Hadamard.

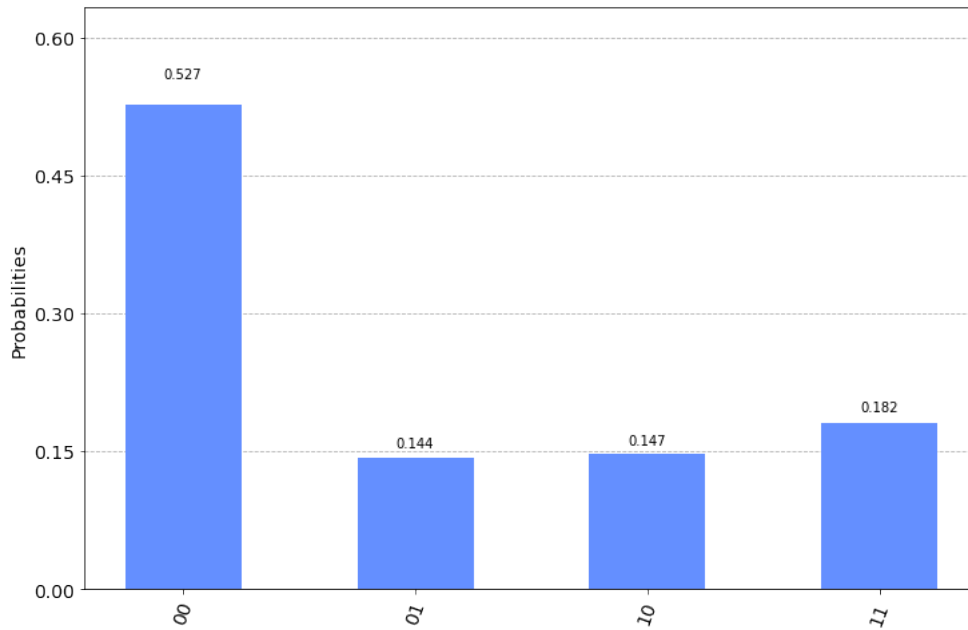
Η δεύτερη περίπτωση είναι η περίπτωση κατά την οποία αρχικά, στο κύκλωμα εφαρμόζεται ο κβαντικός μετασχηματισμός Fourier και έπειτα η πύλη Hadamard. Το κύκλωμα που προκύπτει είναι παρόμοιο με αυτό στο σχήμα

35 αλλά δεν έχει την ίδια πιθανότητα εμφάνισης όλων των αριθμών. Από τα αποτελέσματα που προκύπτουν, δεν μπορεί να ερμηνευτεί με σιγουριά πως ο κβαντικός μετασχηματισμός Fourier εξουδετερώνει τις πύλες Hadamard εφόσον τα αποτελέσματα είναι διατεταγμένοι οι αριθμοί "00" έως "11" με διαφορετική πιθανότητα εμφάνισης, όπως φανερώνει και στο σχήμα 38.



Σχήμα 38: Αποτέλεσμα για την δεύτερη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 2

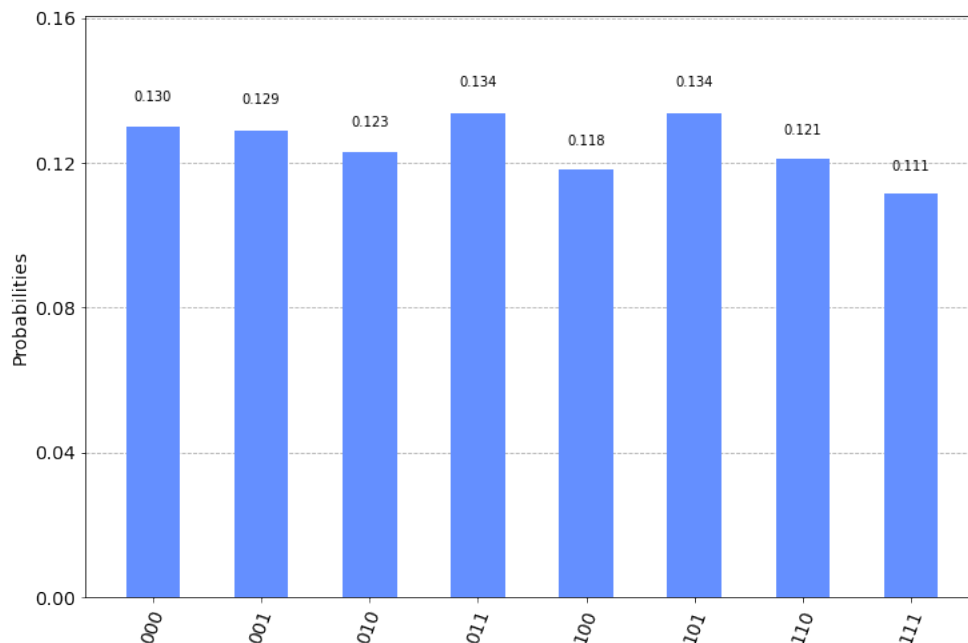
Τέλος, η τρίτη περίπτωση είναι η περίπτωση κατά την οποία αρχικά, στο κύκλωμα εφαρμόζεται ο κβαντικός μετασχηματισμός Fourier και έπειτα εφαρμόζεται, για δεύτερη φορά, ο κβαντικός μετασχηματισμός Fourier. Στην περίπτωση αυτή, τα αποτελέσματα δεν είναι εύκολα ερμηνεύσιμα, όπως και στην δεύτερη περίπτωση, εφόσον η πιθανότητα εμφάνισης του κάθε αποτελέσματος δεν είναι ίδια. Μάλιστα, η κρυφή συμβολοσειρά s έχει από τα μικρότερα ποσοστά εμφάνισης, όπως φανερώνει και στο σχήμα 39.



Σχήμα 39: Αποτέλεσμα για την τρίτη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 2

8.4. Περίπτωση 2: Το μήκος της συμβολοσειράς s έχει μήκος 3 qubits

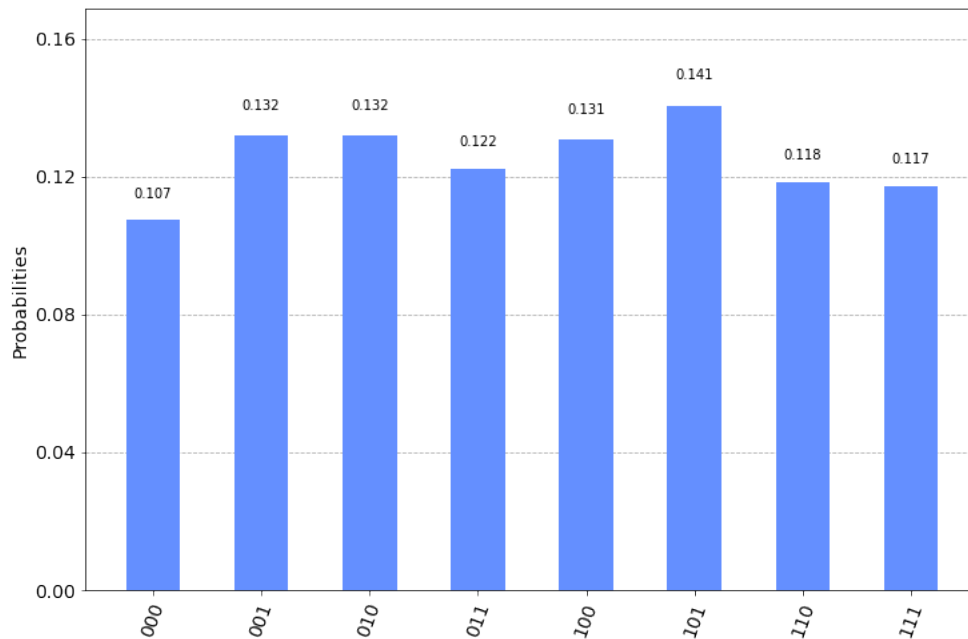
Σε κάθε κεφάλαιο, για κάθε διαφορετικό μήκος της συμβολοσειράς s θα γίνεται περιγραφή των αποτελεσμάτων για κάθε μια από τις περιπτώσεις που σχολιάστηκαν στην ενότητα 8.2. Σε κάθε περίπτωση, τα διαγράμματα που προκύπτουν είναι για την συμβολοσειρά $s = "101"$. Αρχικά, θα σχολιαστεί η περίπτωση του κυκλώματος 35. Για την περίπτωση αυτή, το αποτέλεσμα που προκύπτει για την συμβολοσειρά s φαίνεται στην παρακάτω εικόνα:



Σχήμα 40: Αποτέλεσμα για την πρώτη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 3

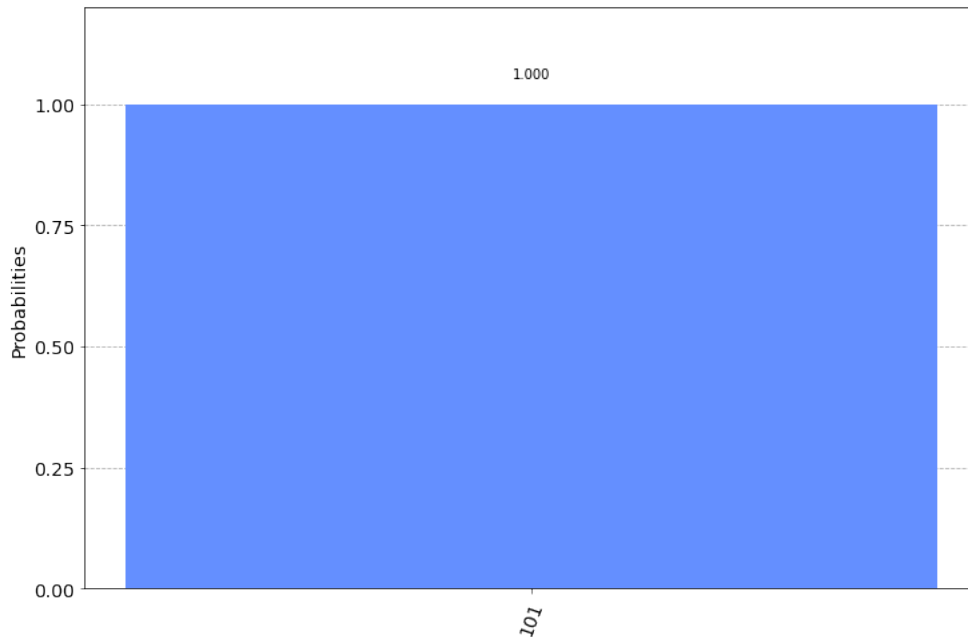
Όπως φαίνεται από το παραπάνω σχήμα, τα αποτελέσματα που προκύπτουν είναι διατεταγμένοι οι αριθμοί σε δυαδικό σύστημα από το "000" έως το "111" και σε δεκαδικό από το 0 έως στο 7. Όλα τα αποτελέσματα έχουν ίση πιθανότητα εμφάνισης με τα υπόλοιπα.

Η δεύτερη περίπτωση είναι η περίπτωση κατά την οποία αρχικά, στο κύκλωμα εφαρμόζεται ο κβαντικός μετασχηματισμός Fourier και έπειτα η πύλη Hadamard, όπως στο κύκλωμα του σχήματος 35. Ομοίως, όπως και στην προηγούμενη περίπτωση, τα αποτελέσματα προκύπτουν είναι διατεταγμένοι οι αριθμοί από το 0 έως το 7 σε δεκαδικό σύστημα όπως φαίνεται στο σχήμα 41.



Σχήμα 41: Αποτέλεσμα για την δεύτερη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 3

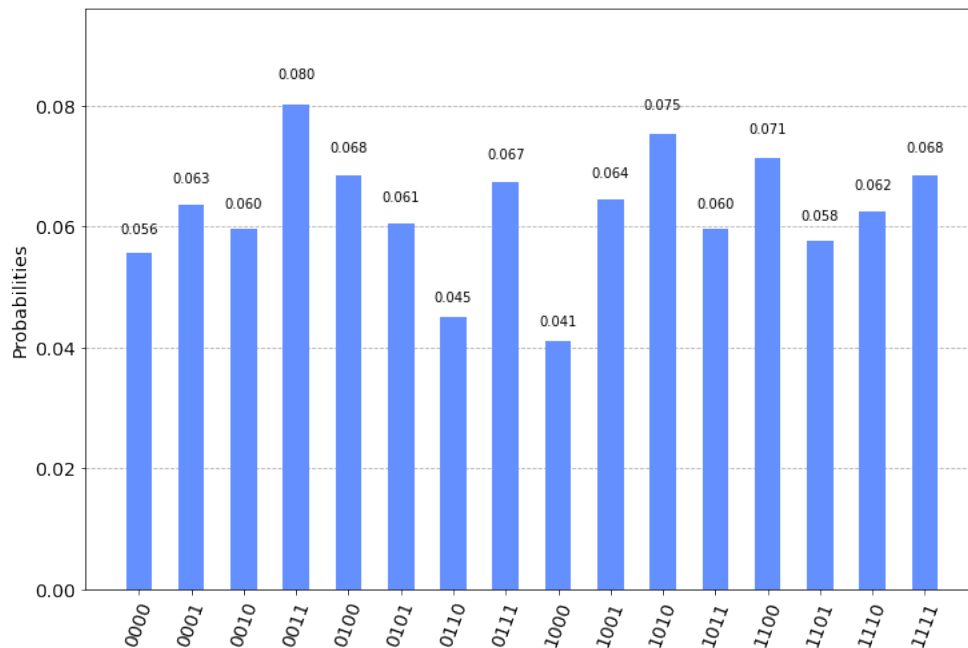
Τέλος, η τρίτη περίπτωση είναι η περίπτωση κατά την οποία αρχικά, στο κύκλωμα εφαρμόζεται ο κβαντικός μετασχηματισμός Fourier και έπειτα εφαρμόζεται, για δεύτερη φορά, ο κβαντικός μετασχηματισμός Fourier, παρόμοιο με αυτό στο σχήμα 36. Στην περίπτωση αυτή, το διάγραμμα που προκύπτει δεν είναι παρόμοιο με κανένα από τα προηγούμενα, εφόσον προκύπτει μόνο ένας αριθμός. Παρόλο που μπορεί να φαίνεται στην εικόνα πως ο αριθμός παράχθηκε σωστά, στην πραγματικότητα είναι τυχαίος. Για την ακρίβεια, στην περίπτωση που η συμβολοσειρά s έχει την τιμή "011" προκύπτει πάλι ο αριθμός "101". Το αποτέλεσμα της υλοποίησης, φαίνεται το σχήμα 42.



Σχήμα 42: Αποτέλεσμα για την τρίτη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 3

8.5. Περίπτωση 3: Το μήκος της συμβολοσειράς s έχει μήκος $4+$ qubits

Σε κάθε κεφάλαιο, για κάθε διαφορετικό μήκος της συμβολοσειράς s θα γίνεται περιγραφή των αποτελεσμάτων για κάθε μια από τις περιπτώσεις που σχολιάστηκαν στην ενότητα 8.2. Σε κάθε περίπτωση, τα διαγράμματα που προκύπτουν είναι για την συμβολοσειρά $s = "1001"$. Αρχικά, θα σχολιαστεί η περίπτωση του κυκλώματος 36. Για την περίπτωση αυτή, το αποτέλεσμα που προκύπτει για την συμβολοσειρά s φαίνεται στην παρακάτω εικόνα:

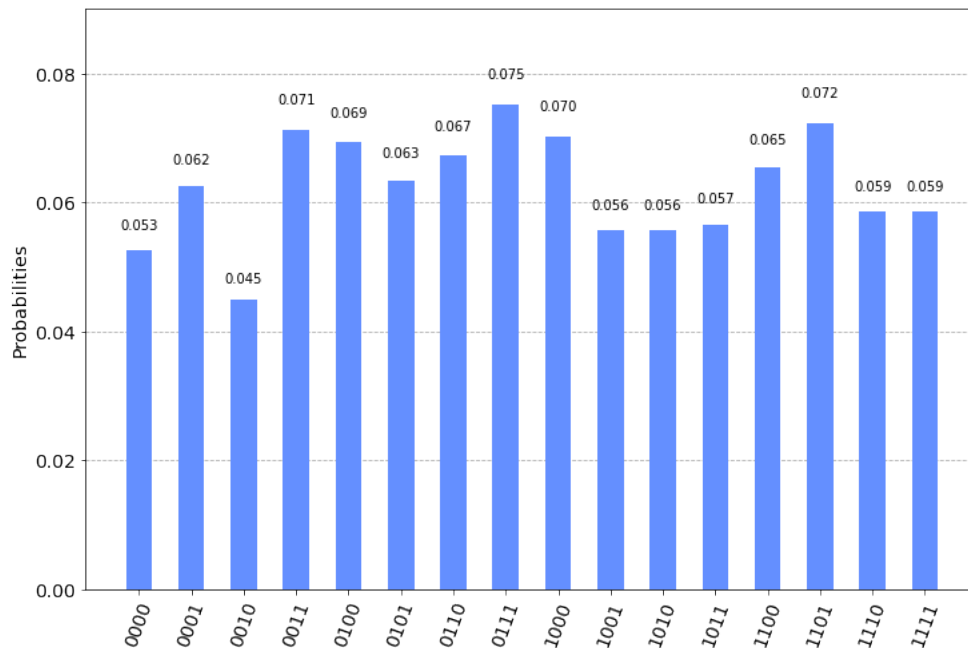


Σχήμα 43: Αποτέλεσμα για την πρώτη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 4

Όπως φαίνεται από το παραπάνω σχήμα, τα αποτελέσματα εμφανίζονται με ίση πιθανότητα από το 0 έως το 15 σε δεκαδικό σύστημα.

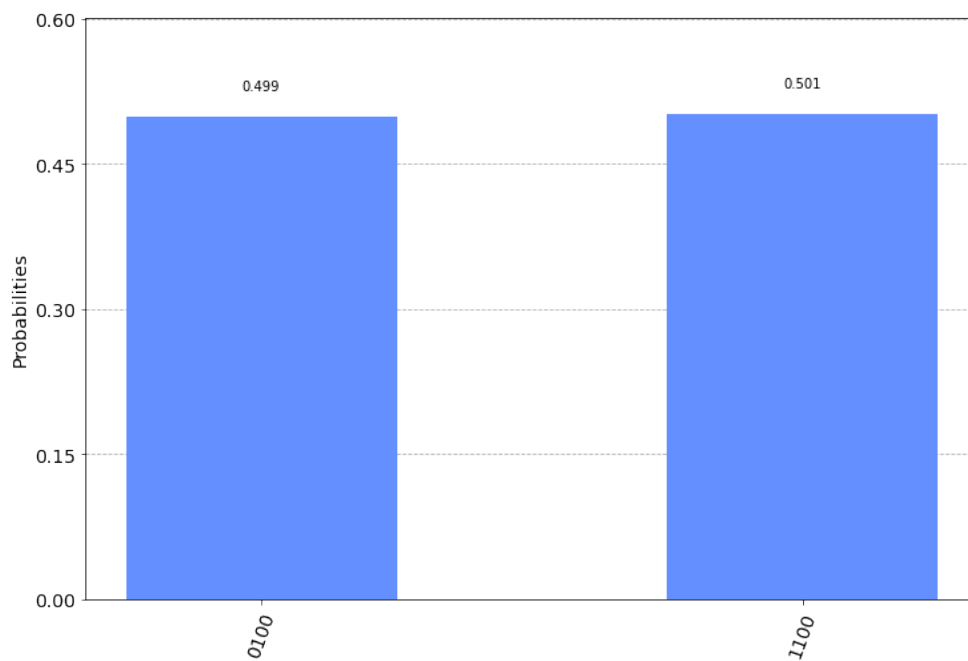
Για κάθε μια από τις περιπτώσεις, όπως έχει παρατηρηθεί σε όλα τα κυκλώματα της πρώτης περίπτωσης δεν εμφανίζεται η συμβολοσειρά s , όπως θα έπρεπε να προκύπτει φυσιολογικά, από τον αλγόριθμο των Bernstein Vazirani, αλλά δίνεται κάθε φορά ένα εύρος τιμών, το οποίο περιλαμβάνει την συμβολοσειρά. Παρόλα αυτά όμως, δεν είναι δυνατό να βρεθεί η συμβολοσειρά, εφόσον τα αποτελέσματα που προκύπτουν έχουν την ίδια πιθανότητα εμφάνισης.

Η δεύτερη περίπτωση είναι η περίπτωση κατά την οποία αρχικά, στο κύκλωμα εφαρμόζεται ο κβαντικός μετασχηματισμός Fourier και έπειτα η πύλη Hadamard, όπως φαίνεται στο σχήμα 35. Ομοίως, όπως και στην προηγούμενη περίπτωση, τα αποτελέσματα είναι διατεταγμένοι οι αριθμοί από το 0 έως το 15 σε δεκαδικό σύστημα, με ίση πιθανότητα εμφάνισης, όπως φανερώνει και στο σχήμα 44.



Σχήμα 44: Αποτέλεσμα για την δεύτερη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 4

Τέλος, η τρίτη περίπτωση είναι η περίπτωση κατά την οποία αρχικά, στο κύκλωμα εφαρμόζεται ο κβαντικός μετασχηματισμός Fourier και έπειτα εφαρμόζεται, για δεύτερη φορά, ο κβαντικός μετασχηματισμός Fourier όπως φαίνεται στο σχήμα 36. Στην περίπτωση αυτή, το διάγραμμα είναι διαφορετικό από τις υπόλοιπες περιπτώσεις εφόσον προκύπτουν 2 διαφορετικές τιμές, οι οποίες όμως παραμένουν ίδιες ανεξαρτήτως της συμβολοσειράς s , δηλαδή, προκύπτει το ίδιο αποτέλεσμα για οποιαδήποτε συμβολοσειρά s με μήκος 4. Παρακάτω, φαίνεται το διάγραμμα με τα αποτελέσματα. 45



Σχήμα 45: Αποτέλεσμα για την τρίτη περίπτωση όπου το μήκος της συμβολοσειράς s να ισούται με 4

Αναφορές

- [1] Michael Ampatzis και Theodore Andronikos. «QKD based on symmetric entangled bernstein-vazirani». Στο: *Entropy* 23.7 (2021), σ. 870.
- [2] Theodore Andronikos και Alla Sirokofskich. «The connection between the PQ penny flip game and the dihedral groups». Στο: *Mathematics* 9.10 (2021), σ. 1115.
- [3] Theodore Andronikos και Michael Stefanidakis. «A two-party quantum parliament». Στο: *arXiv preprint arXiv:2201.08510* (2022).
- [4] Theodore Andronikos κ.ά. «Finite automata capturing winning sequences for all possible variants of the PQ penny flip game». Στο: *Mathematics* 6.2 (2018), σ. 20.
- [5] MD SAJID ANIS κ.ά. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. doi: [10.5281/zenodo.2573505](https://doi.org/10.5281/zenodo.2573505).
- [6] Adriano Barenco κ.ά. «Elementary gates for quantum computation». Στο: *Physical review A* 52.5 (1995), σ. 3457.
- [7] Ethan Bernstein και Umesh Vazirani. «Quantum complexity theory». Στο: *SIAM Journal on computing* 26.5 (1997), σσ. 1411–1473.
- [8] David Deutsch και Richard Jozsa. «Rapid solution of problems by quantum computation». Στο: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), σσ. 553–558.
- [9] Jiangfeng Du κ.ά. «Implementation of a quantum algorithm to solve the Bernstein-Vazirani parity problem without entanglement on an ensemble quantum computer». Στο: *Physical Review A* 64.4 (2001), σ. 042306.
- [10] Konstantinos Giannakis και Theodore Andronikos. «Membrane automata for modeling biomolecular processes». Στο: *Natural Computing* 16.1 (2017), σσ. 151–163.
- [11] Konstantinos Giannakis κ.ά. «Associating ω -automata to path queries on Webs of Linked Data». Στο: *Engineering Applications of Artificial Intelligence* 51 (2016), σσ. 115–123.
- [12] Konstantinos Giannakis κ.ά. «Dominant strategies of quantum games on quantum periodic automata». Στο: *Computation* 3.4 (2015), σσ. 586–599.
- [13] Konstantinos Giannakis κ.ά. «Quantum conditional strategies and automata for Prisoners' Dilemmata under the EWL scheme». Στο: *Applied Sciences* 9.13 (2019), σ. 2635.

- [14] I. Karafyllidis. *Quantum Computing [Undergraduate textbook]*. Kallipos, Open Academic Editions, 2005.
- [15] Phillip Kaye, Raymond Laflamme και Michele Mosca. *An introduction to quantum computing*. OUP Oxford, 2006.
- [16] Christos Papalitsas κ.ά. «A QUBO model for the traveling salesman problem with time windows». Στο: *Algorithms* 12.11 (2019), σ. 224.
- [17] Vivek V Shende και Igor L Markov. «On the CNOT-cost of TOFFOLI gates». Στο: *arXiv preprint arXiv:0803.2316* (2008).
- [18] Daniel R Simon. «On the power of quantum computation». Στο: *SIAM journal on computing* 26.5 (1997), σσ. 1474–1483.
- [19] Evangelos Stogiannos, Christos Papalitsas και Theodore Andronikos. «Experimental analysis of quantum annealers and hybrid solvers using benchmark optimization problems». Στο: *Mathematics* 10.8 (2022), σ. 1294.
- [20] Yaakov S Weinstein κ.ά. «Implementation of the quantum Fourier transform». Στο: *Physical review letters* 86.9 (2001), σ. 1889.