VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI - 590018

Report on Internship/Professional Practice (18EC84)

MAPPING THE LANDSCAPE OF QUANTUM EDUCATION

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF ENGINEERING
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

SHISHEER S KAUSHIK
1BG18EC127

Internship Carried Out
at
ELITE TECHNO GROUPS

Internal Guide
Dr. Rekha P
Assistant Professor,
Dept. of ECE, BNMIT

External Guide 1
Mr. Zeki Can Seskir
PhD-[Quantum Technologies],
Karlsruhe Institute of Technical
(KIT)

External Guide 2
Mr. Adam Glos
Program Head &
Coordinator
[Qntern-21]

B.N.M. Institute of Technology

An Autonomous Institution under VTU, Approved by AICTE, Accredited as Grade An Institution by NAAC.
All Eligible UG branches – CSE, ECE, EEE, ISE & Mech.E Accredited by NBA for academic years 2018-19 to 2021-22 & valid up to 30.06.2022
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph.: 91-80- 26711780/81/82 Email: principal@bnmit.in, www.bnmit.org

Department of Electronics & Communication Engineering

2021 – 2022

B.N.M. Institute of Technology

An Autonomous Institution under VTU, Approved by AICTE, Accredited as Grade An Institution by NAAC.
All Eligible UG branches – CSE, ECE, EEE, ISE & Mech.E Accredited by NBA for academic years 2018-19 to 2021-22 & valid up to 30.06.2022
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph.: 91-80- 26711780/81/82 Email: principal@bnmit.in, www.bnmit.org

DEPARTMENT: ELECTRONICS & COMMUNICATION ENGINEERING

Vidyayāmruthamashnuthe

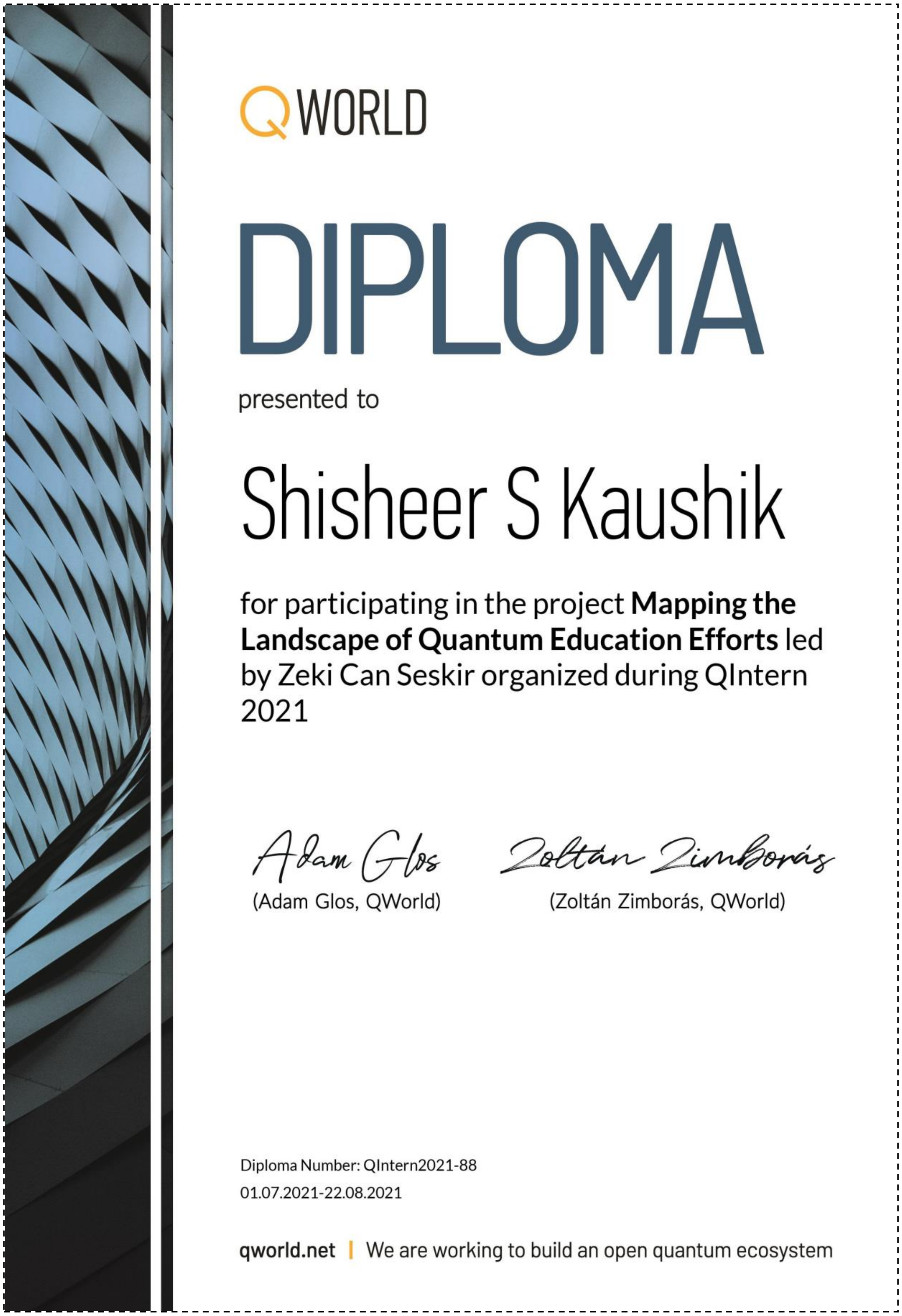
CERTIFICATE

This is to certify that the Internship entitled **Mapping the Landscape of Quantum Education Efforts** carried out by **Mr. Shisheer S Kaushik [1BG18EC127]** a bonafide student of VIII Semester, in partial fulfillment for the Bachelor of Engineering in Electronics & Communication Engineering of the **Visvesvaraya Technological University**, Belagavi during the year 2021-22. It is certified that all corrections / suggestions indicated have been incorporated in the report. The internship report has been approved as it satisfies the academic requirements in respect of internship work prescribed for the said degree.

Dr. Rekha P	**Dr. P.A. Vijaya**	**Dr. Krishnamurthy G. N**
Asst. Professor, Dept. of ECE	**HOD, Dept. of ECE**	**Principal**
BNMIT, Bengaluru	**BNMIT, Bengaluru**	**BNMIT, Bengaluru**

External Viva Voce

Name of the Examiner	**Signature with Date**
1) ______________________	____________________
2) ______________________	____________________



QWORLD

DIPLOMA

presented to

Shisheer S Kaushik

for participating in the project **Mapping the Landscape of Quantum Education Efforts** led by Zeki Can Seskir organized during QIntern 2021

Adam Glos — Zoltán Zimborás

(Adam Glos, QWorld) — (Zoltán Zimborás, QWorld)

Diploma Number: QIntern2021-88
01.07.2021-22.08.2021

qworld.net | We are working to build an open quantum ecosystem

DECLARATION

I, the undersigned solemnly declare that the report of the internship work entitled **“Mapping the Landscape of Quantum Education Efforts”** is based on my work carried out during the course study under the supervision of **Dr. Rekha P, Assistant Professor, Department of ECE,** BNM Institute of Technology, Bangalore and **Mr. Adam Glos, Project Director** along with **Mr. Zeki Can Seskir, Mentor, Post Doc,** Institut für Technikfolgenabschätzung und Systemanalyse (ITAS)**,** Turkey. This Remote internship work was carried out at QIntern-21 organized by QWord Association, Global Network.

I assert that the statements made and conclusions drawn are an outcome of the internship work. I further declare that, to the best of my knowledge and belief, that this report does not contain any work which has been submitted for the award of the degree or any other degree in this university or any other university.

SHISHEER S KAUSHIK
1BG18EC127

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of the internship would be incomplete without mentioning the names of the people who made it possible.

I am greatly indebted to our **Director**, **Prof. T.J. Rama Murthy**, **Additional Director, Dr. S.Y.Kulkarni, Dean**, **Prof. Eishwar N. Maanay, Principal**, **Dr. Krishnamurthy G.N**, B.N.M Institute of Technology, Bangalore and **Dr. P.A. Vijaya, Head of Department, Electronics and Communication**, for giving me an opportunity to carry out the Internship. I am really indebted to our great institute BNMIT, which has provided me an opportunity to get a better degree in B.E. with the state- of-the-art facilities, and an impeccable environment.

It is with a deep sense of gratitude and great respect, I owe my indebtedness to my guide **Mr. Zeki Can Seskir.** I thank him for his constant encouragement during the execution of this Internship and also I thank my Internship coordinator **Smt. Keerti Kulkarni** for making the conduction of this Internship successful.

I express my heartfelt thanks to my External Guides at **Qworld Association., Mr. Adam Glos & their Team** for their guidance and unconditional support, despite of their busy work schedule. I thank my **parents**, without whom none of this would have been possible. Their patience and blessings have been with me at every step of this Internship. I would express my thanks to all my friends and all those who have helped me directly or indirectly for the successful completion of the Internship.

SHISHEER S KAUSHIK

1BG18EC127

EXECUTIVE SUMMARY

This report is prepared based on four weeks of internship program that I have done at QWorld Global Association as a requirement of my B.E., program in Visvesvaraya Technological University (VTU).

As this Internship was a part of QIntern-21 program, which is organized annually. I got short listed through the online Interview and Technical Background Test. During these four weeks of internship, I was allotted to one respective group under a mentor/supervisor to work on the Major Computational Issue faced during Quantum Application Programming, which was already in progress, in this group I had a chance to learn a lot about Quantum Error Detection. It was a very effective learning experience. Throughout the Course I had indulged in the development of the project named as 'Mapping the Landscape of Quantum Education Efforts' under that, based on our interest we were allotted to specific research lab with respective mentors. I opted to work in 'Quantum Error Correction Labs'. This internship has enhanced my skills and knowledge in this particular field via a hands on experience and also helped me to improve my network by meeting many reputed person working in this domain.

The work of My Project is divide as follows:

Part 1: gives an insight about how the different types of error affect the computation efficiency of the circuit during the processing. Hence I have designed a specific Quantum Error Detection model using the repetition code and syndrome measurement technique. This model detects the type as well as the position of the error inside a circuit with 100 % precision in Simulator & 48% - 50% precision in physical hardware.

Part 2: deals with an optimization of the circuit. As most of the Quantum Gates requires large data storage and RAM, it was part of my project to find most efficient and optimized circuit which requires less cost to perform Quantum error correction operation. In this project I get to exploit on May optimization techniques used in especially for circuit gates optimization.

Both the Parts were implemented using Jupyter notebook, Py3, and Qiskit SDK with some added libraries {Aer, BasicAer, Ignis, IBMQ provider}for building gates and graphs to showcase the results and observations.

And the source link for my project can be found in this GitHub repo: ***https://github.com/ShisheerKaushik24/IBM-Quantum-challenge-2021-/blob/master/ex3.ipynb***

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

CHAPTER 1

ABOUT THE ORGANISATION

CHAPTER 1

ABOUT THE ORGANISATION

1.1 COMPANY OVERVIEW

QWorld (Association) is a non-profit global organization that brings quantum computing researchers & enthusiasts together. Our main goal is to popularize quantum technologies and software. Also, through education and skill development opportunities, QWorld is training the next generation of quantum scientists.

1.2 CORPORATE PHILOSOPHY

1.2.1 VISION

Our main goal is to popularize quantum technologies and software. We are looking for enthusiastic individuals, groups, institutions, organizations, and companies to work and operate together to develop open quantum ecosystem.

1.2.2 MISSION

QWorld is a global network of individuals, groups, and communities collaborating on education and implementation of quantum technologies and research activities.

1.2.3 OBJECTIVE

We conduct our activities within five departments:

QCousin's implements one of the critical strategies of QWorld is to establish active and collaborative local quantum groups. These groups organize events in their regions, often communicating in the native language. QResearch's goal is to bring together researchers to foster collaboration and match students with mentors to develop a short-term research project. QWorld is also committed to increasing women's representation in the STEM field, and for this reason, QWomen was created. Finally, the QJunior program is focused on high school students.

1.3 SOLUTIONS

1.3.1 INDUSTRIAL SOLUTIONS

QWorld provides a platform for many industries for discreet applications such as:

[1] Design:

Many products are designed and pre-tested using computer simulation. Automotive and aerospace hardware components and subcomponents are 3D-modeled with individual engineering safety margins. These margins can accumulate, culminating in products that are over-engineered, overweight, or higher cost than necessary, which can stifle their commercial viability.

[2] Control:

Modern control processes in manufacturing test the limits of advanced analytics, especially when employing machine learning and analysing multiple variables.

Quantum computing might help find new correlations in data, enhance pattern recognition, and advance classification beyond the capabilities of classical computing. The combination of quantum computing and machine learning, as well as its application to optimization,

[3] Supply:

Supply chains are shifting from a linear model with discrete, sequential, event-driven processes to a more responsive organic model based on evolving real-time market demands and up-to-the-minute availability of key components. Adding to the digital supply chain toolbox of Industry 4.0, quantum computing potentially could accelerate decision-making and enhance risk management to lower operational costs, as well as reduce lost sales because of out-of-stock or discontinued products. Enhancing competitive agility, quantum computing might completely transform the supply chain over time, adaptively redesigning it to optimize vendor orders and accompanying logistics using dynamic near-real-time decision-making based on changing market demands.

CHAPTER 2

ABOUT THE DEPARTMENT

CHAPTER 2

ABOUT THE DEPARTMENT

2.1 INTRODUCTION QUANTUM GATES

The qubits or so called Quantum Bits could be represented by 2D vectors, and that their states are limited to the form:

$$|q\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$$

Wh θ an ϕ are real numbers. In this section we will cover gates, the operations that change a qubit between these states. Due to the number of gates and the similarities between them, this chapter is at risk of becoming a list. To counter this, we have included a few digressions to introduce important ideas at appropriate places throughout the chapter.

The different Types of Gates Used are:

[1] Pauli X gate:

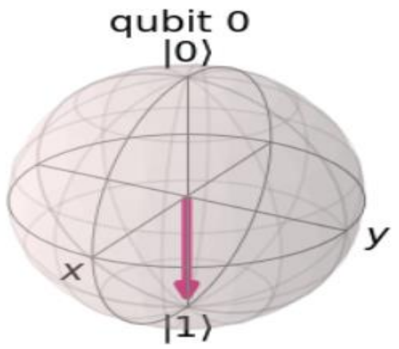
The X-gate is represented by the Pauli-X matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0|$$

To see the effect a gate has on a qubit, we simply multiply the qubit's statevector by the gate. We can see that the X-gate switches the amplitudes of the states $|0\rangle$ and $|1\rangle$:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

Bloch Representation:



[2] Pauli Y & Z Gate:

Similarly to the X-gate, the Y & Z Pauli matrices also act as the Y & Z-gates in our quantum circuits:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$Y = -i|0\rangle\langle 1| + i|1\rangle\langle 0| \qquad Z = |0\rangle\langle 0| - |1\rangle\langle 1|$$

And, unsurprisingly, they also respectively perform rotations by π around the y and z-axis of the Bloch sphere.

[3] Hadamard Gate [H Gate]

The Hadamard gate (H-gate) is a fundamental quantum gate. It allows us to move away from the poles of the Bloch sphere and create a superposition of $|0\rangle$ and $|1\rangle$. It has the matrix:

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

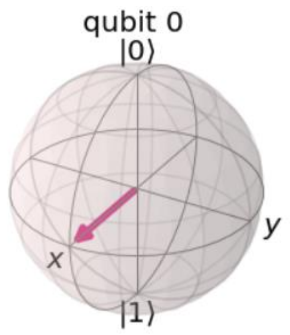
We can see that this performs the transformations below:

$$H|0\rangle = |+\rangle$$

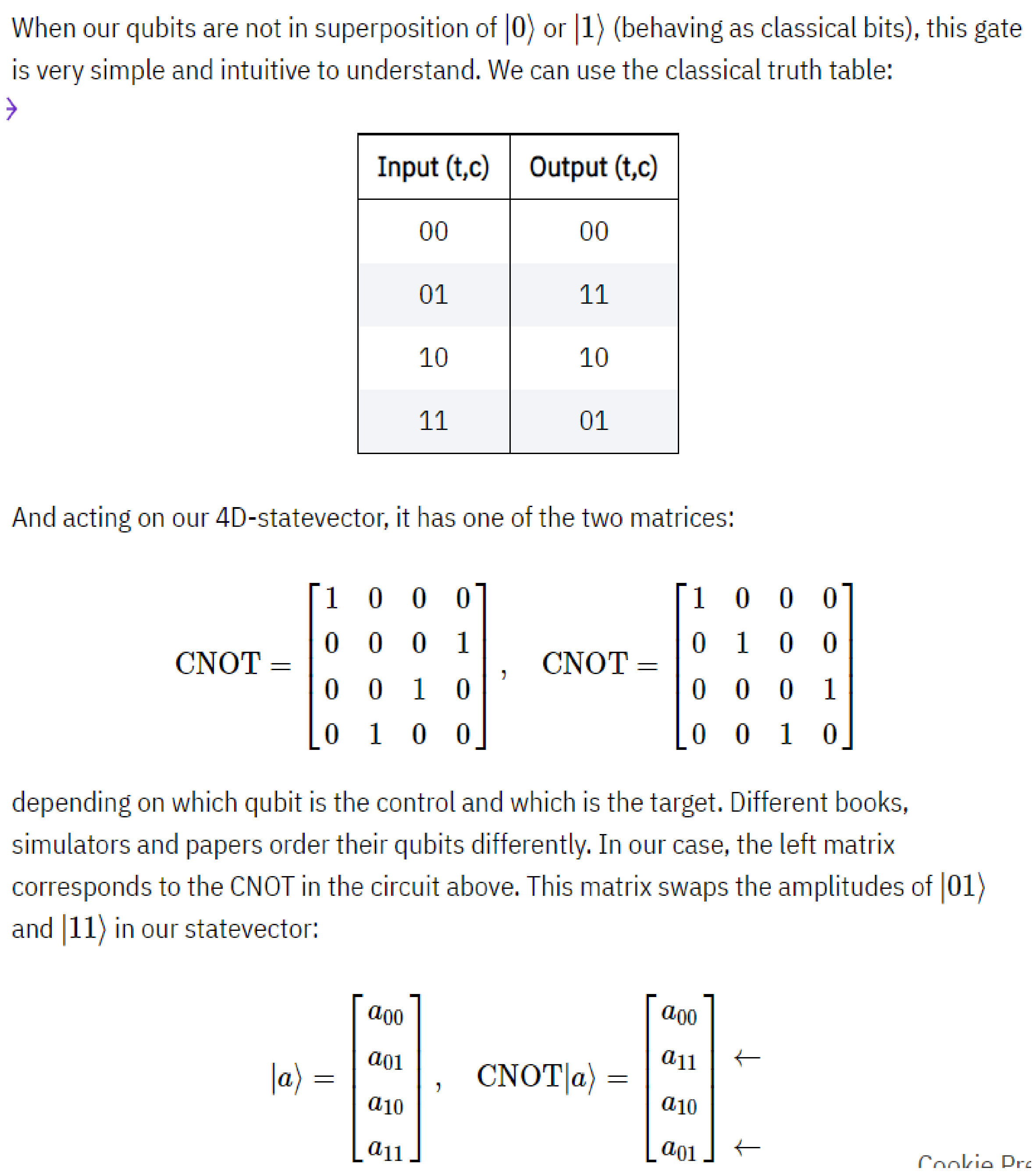
$$H|1\rangle = |-\rangle$$

This can be thought of as a rotation around the Bloch vector `[1,0,1]` (the line between the x & z-axis), or as transforming the state of the qubit between the X and Z bases.

Bloch Representation:



[4] The CNOT-Gate [CX-gate]:



When our qubits are not in superposition of $|0\rangle$ or $|1\rangle$ (behaving as classical bits), this gate is very simple and intuitive to understand. We can use the classical truth table:

Input (t,c)	Output (t,c)
00	00
01	11
10	10
11	01

And acting on our 4D-statevector, it has one of the two matrices:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

depending on which qubit is the control and which is the target. Different books, simulators and papers order their qubits differently. In our case, the left matrix corresponds to the CNOT in the circuit above. This matrix swaps the amplitudes of $|01\rangle$ and $|11\rangle$ in our statevector:

$$|a\rangle = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix}, \quad \text{CNOT}|a\rangle = \begin{bmatrix} a_{00} \\ a_{11} \\ a_{10} \\ a_{01} \end{bmatrix} \begin{array}{c} \\ \leftarrow \\ \\ \leftarrow \end{array}$$

Cookie Pr

Table 2.1: CNOT-Gate Truth Table

2.2 INTRODUCTION TO QUANTUM ERROR CORRECTION

Because of the vast amount of effort required for this process, most operations performed in fault-tolerant quantum computers will be done to serve the purpose of error detection and correction. So when benchmarking our progress towards fault-tolerant quantum computation, we must keep track of how well our devices perform error correction.

In this chapter we will look at a particular example of error correction: the repetition code. Though not a true example of quantum error correction — it uses physical qubits to encode a logical bit, rather than a qubit — it serves as a simple guide to all the basic concepts in any quantum error correcting code. We will also see how it can be run on current prototype devices.

The basic ideas behind error correction are the same for quantum information as for classical information. This allows us to begin by considering a very straightforward example: speaking on the phone. If someone asks you a question to which the answer is 'yes' or 'no', the way you give your response will depend on two factors:

- How important is it that you are understood correctly?
- How good is your connection?

Both of these can be parameterized with probabilities. For the first, we can use PaPa, the maximum acceptable probability of being misunderstood. If you are being asked to confirm a preference for ice cream flavours, and don't mind too much if you get vanilla rather than chocolate, PaPa might be quite high. If you are being asked a question on which someone's life depends, however, PaPa will be much lower. For the second we can use pp, the probability that your answer is garbled by a bad connection. For simplicity, let's imagine a case where a garbled 'yes' doesn't simply sound like nonsense, but sounds like a 'no'. And similarly a 'no' is transformed into 'yes'. Then pp is the probability that you are completely misunderstood. A good connection or a relatively unimportant question will result in p<Pap<Pa. In this case it is fine to simply answer in the most direct way possible: you just say 'yes' or 'no'. If, however, your connection is poor and your answer is important, we will have p>Pap>Pa. A single 'yes' or 'no' is not enough in this case.

The probability of being misunderstood would be too high. Instead we must encode our answer in a more complex structure, allowing the receiver to decode our meaning despite the possibility of the message being disrupted. The simplest method is the one that many would do without thinking: simply repeat the answer many times. For example say 'yes, yes, yes' instead of 'yes' or 'no, no no' instead of 'no'.

If the receiver hears -'Yes, yes, yes' in this case, they will of course conclude that the sender meant

' Yes'. If they hear 'no, yes, yes', 'yes, no, yes' or 'yes, yes, no', they will probably conclude the same thing, since there is more positivity than negativity in the answer. To be misunderstood in this case, at least two of the replies need to be garbled. The probability for this, PP, will be less than pp. When encoded in this way, the message therefore becomes more likely to be understood.

2.3 QISKIT SDK

Qiskit is an open-source software development kit (SDK) for working with quantum computers at the level of circuits, pulses, and algorithms. It provides tools for creating and manipulating quantum programs and running them on prototype quantum devices on IBM Quantum Experience or on simulators on a local computer. It follows the circuit model for universal quantum computation, and can be used for any quantum hardware (currently supports superconducting qubits and trapped ions) that follows this model.

Qiskit was founded by IBM Research to allow software development for their cloud quantum computing service, IBM Quantum Experience. Contributions are also made by external supporters, typically from academic institutions.

The primary version of Qiskit uses the Python programming language. Versions for Swift and JavaScript were initially explored, though the development for these versions have halted. Instead, a minimal re-implementation of basic features is available as MicroQiskit, which is made to be easy to port to alternative platforms.

CHAPTER 3

QUANTUM ERROR DETECTION CODE

CHAPTER 3

QUANTUM CODE ERROR DETECTION

3.1 THE PROBLEM OF ERRORS

Errors occur when some spurious operation acts on our qubits. Their effects cause things to go wrong in our circuits. The strange results you may have seen when running on real devices is all due to these errors.

There are many spurious operations that can occur, but it turns out that we can pretend that there are only two types of error: bit flips and phase flips.

Bit flips have the same effect as the **X** gate. They flip the **|0⟩** state of a single qubit to **|1⟩** and vice-versa. Phase flips have the same effect as the z gate, introducing a phase of **−1** into superposition. Put simply, they flip the **|+⟩** state of a single qubit to **|−⟩** and vice-versa.

The reason we can think of any error in terms of just these two is because any error can be represented by some matrix, and any matrix can be written in terms of the matrices **X** and **Z**. Specifically, for any single qubit matrix **M**,

$$M = \alpha I + \beta X + \gamma XZ + \delta Z,$$

For some suitably chosen values α, β, γ and δ

So whenever we apply this matrix to some single qubit state |ψ⟩ we get

$$M|\psi\rangle = \alpha|\psi\rangle + \beta X|\psi\rangle + \gamma XZ|\psi\rangle + \delta Z|\psi\rangle$$

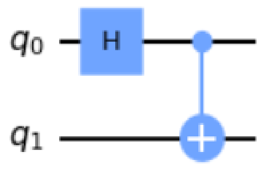
The resulting superposition is composed of the original state, the state we'd have if the error was just a bit flip, the state for just a phase flip and the state for both. If we had some way to measure whether a bit or phase flip happened, the state would then collapse to just one possibility. And our complex error would become just a simple bit or phase flip.

So how do we detect whether we have a bit flip or a phase flip (or both)? And what do we do about it once we know? Answering these questions is what quantum error correction is all about.

3.2 An Overly Simplified Entangled Circuit

One of the first quantum circuits that most people ever write is to create a pair of entangled qubits. In this journey into quantum error correction, we'll start the same way.

```
from qiskit import QuantumCircuit, Aer

# Make an entangled pair
qc_init = QuantumCircuit(2)
qc_init.h(0)
qc_init.cx(0,1)

# Draw the circuit
display(qc_init.draw('mpl'))

# Get an output
qc = qc_init.copy()
qc.measure_all()
job = Aer.get_backend('qasm_simulator').run(qc)
job.result().get_counts()
```



```
{'00': 498, '11': 526}
```

Figure 3.2: {Code}: Creating Entanglement State

Here we see the expected result when we run the circuit: the results **00** and **11** occurring with equal probability.

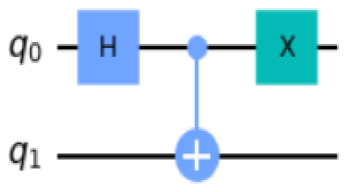
3.3 Bit Flip 'Error'

A bit flip error is specific type of error where the qubits computational state flips from 1 to 0 or vice versa. However a bit flip can be corrected using the bit flip code. This is a 3 qubit circuit that makes use of 2 ancillary qubits to correct 1 qubit.

A bit flip occurs when you're copying data and one of the bits changes so that it's incorrect. A value of 1 incorrectly becomes a zero, or vice versa. Bit flips that lead to bug checks are a common way that Windows detects a hardware problem (e.g., bad memory, an overheating CPU).

But what happens when we have the same circuit, but with a bit flip 'error' inserted manually.

```
# Make bit flip error
qc_insert = QuantumCircuit(2)
qc_insert.x(0)

# Add it to our original circuit
qc = qc_init.copy()
qc = qc.compose(qc_insert)

# Draw the circuit
display(qc.draw('mpl'))

# Get an output
qc.measure_all()
job = Aer.get_backend('qasm_simulator').run(qc)
job.result().get_counts()
```



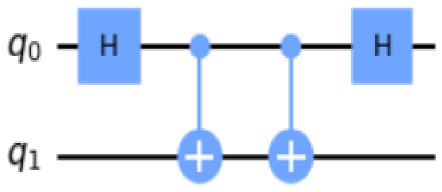
```
{'01': 521, '10': 503}
```

Figure 3.3.1: {Code}: Creating Bit Flip 'Error'

Now the results are different: **01** and **10**. The two bit values have gone from always agreeing to always disagreeing. In this way, we detect the effect of the error.

Another way we can detect it is to undo the entanglement with a few more gates. If there are

```
# Undo entanglement
qc_syn = QuantumCircuit(2)
qc_syn.cx(0,1)
qc_syn.h(0)

# Add this after the error
qc = qc_init.copy()
qc = qc.compose(qc_syn)

# Draw the circuit
display(qc.draw('mpl'))

# Get an output
qc.measure_all()
job = Aer.get_backend('qasm_simulator').run(qc)
job.result().get_counts()
```



```
{'00': 1024}
```

no errors, we return to the initial **|00⟩** state.

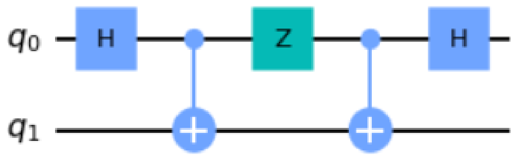
Figure 3.3.2: {Code}: Detection of Bit Flip 'Error'

\But what happens if there are errors one of the qubits? Try inserting different errors to find out.

Here's a circuit with all the components we've introduced so far: the initialization **qc_init,** the inserted error in **qc_insert** and the final **qc_syn** which ensures that the final measurement gives a nice definite answer.

3.4 Phase Flip ‘Error’

```
# Define an error
qc_insert = QuantumCircuit(2)
qc_insert.z(0)

# Undo entanglement
qc_syn = QuantumCircuit(2)
qc_syn.cx(0,1)
qc_syn.h(0)

# Add this after the error
qc = qc_init.copy()
qc = qc.compose(qc_insert)
qc = qc.compose(qc_syn)

# Draw the circuit
display(qc.draw('mpl'))

# Get an output
qc.measure_all()
job = Aer.get_backend('qasm_simulator').run(qc)
job.result().get_counts()
```

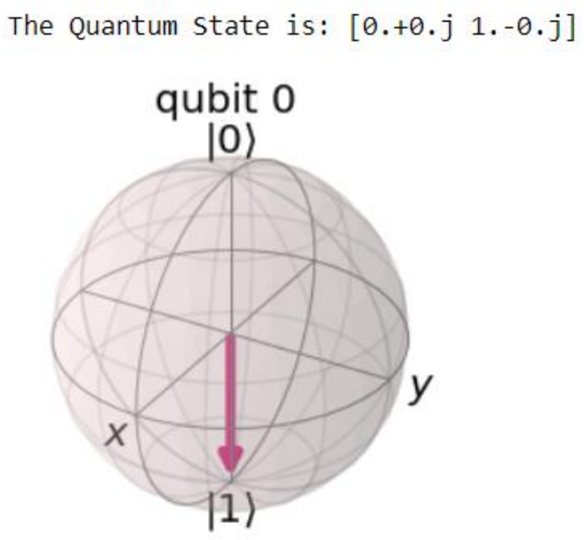


```
{'01': 1024}
```

Figure 3.4.1: {Code}: Creating a Phase Flip ‘Error’

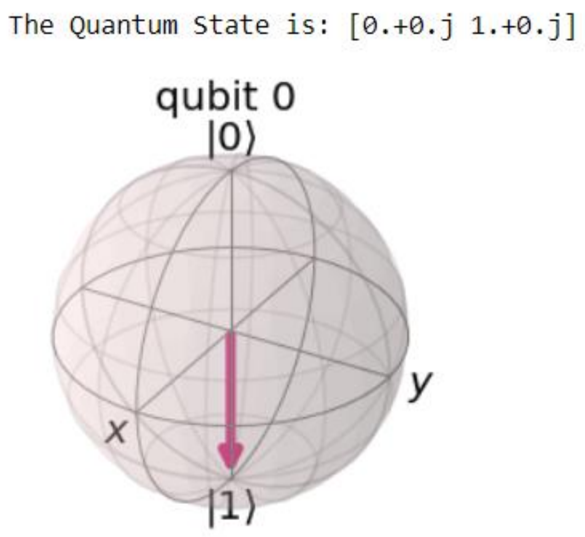
You'll find that the output tells us exactly what is going on with the errors. Both the bit and phase flips can be detected. The bit value on the left is 1 only if there is a bit flip (*and so if we have inserted an* ***x(0)*** *or* ***x(1)).*** The bit on the right similarly tells us there is a phase flip (*an inserted* ***z(0)*** *or* ***z(1)).*** This ability to detect and distinguish bit and phase flips is very useful. But it is not quite useful enough. We can only tell ***what type*** of errors are happening, but not ***where***. Without more detail, it is not possible to figure out how to remove the effects of these operations from our computations. For quantum error correction we therefore need something better.

<Figure 3.4.2>: **The Expected Result {Quantum State}:**



```
The Quantum State is: [0.+0.j 1.-0.j]
```

<Figure 3.4.3>: **The Obtained Result {With Error Induced}:**



```
The Quantum State is: [0.+0.j 1.+0.j]
```

This Representation shows that, only the phase has been flipped but not the Bit. As seen above the state has remained at pointing towards **|1⟩** even after the circuit changed**.**

CHAPTER 4

TASKS PERFORMED

CHAPTER 4

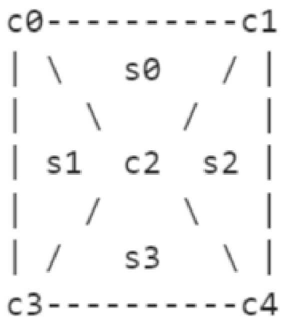
TASKS PERFORMED

Goal: Create circuits which can detect `x` and `z` errors on two qubits, also with the lowest possible Cost Estimation.

4.1 Implement a Cost Efficient Circuit for Error Detection

Mapping the Circuit using *STABILIZER OPERATOR* on 3 Qubits:

We consider the qubits to be laid out as follows, with the code qubits forming the corners of four triangles, and the syndrome qubits living inside each triangle.



```
c0----------c1
| \   s0   / |
|   \    /   |
| s1  c2  s2 |
|   /    \   |
| /   s3   \ |
c3----------c4
```

Figure 4.1.1: Circuit Layout Graph

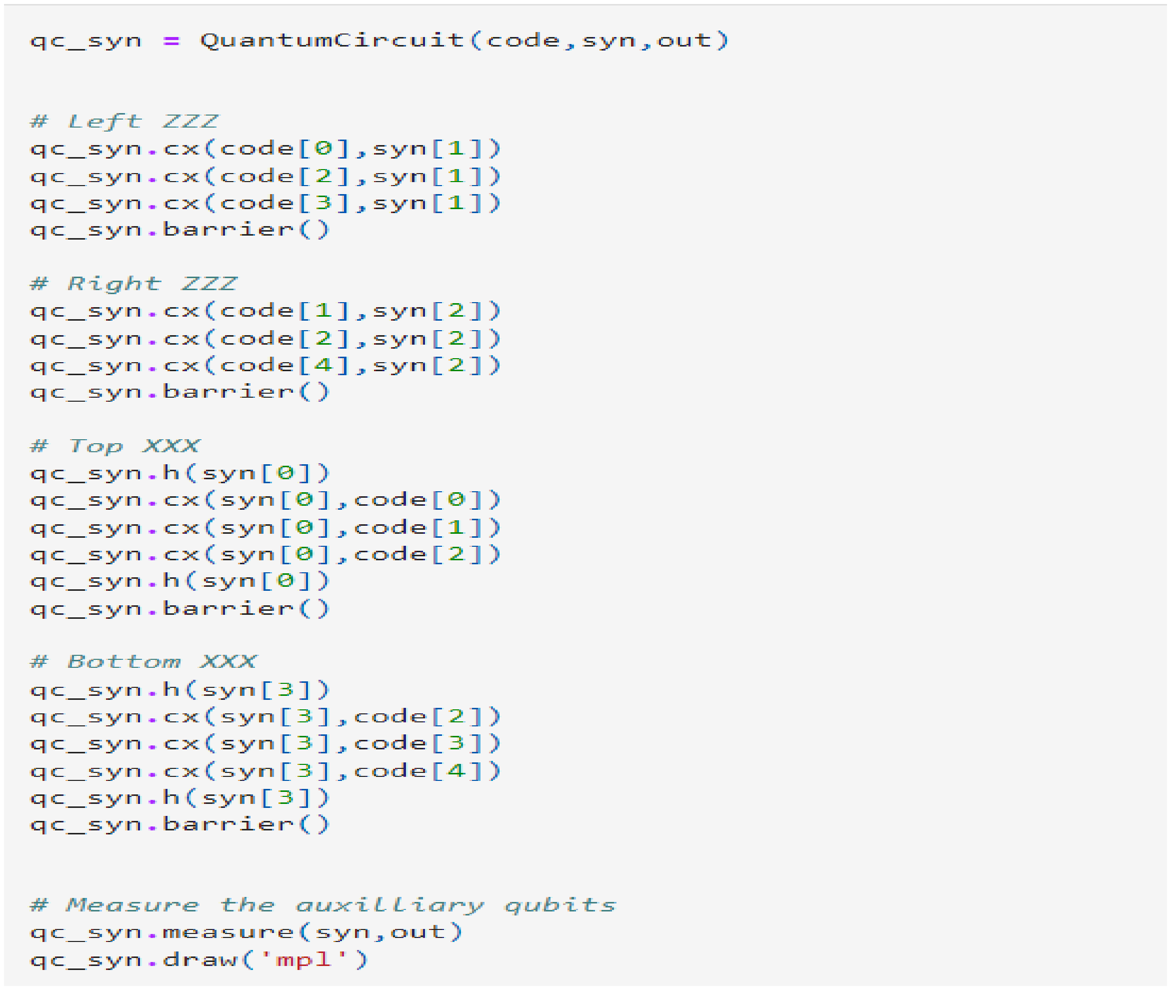
For each triangle we associate a stabilizer operation on its three qubits. For the qubits on the sides, the stabilizers are **ZZZ**. For the top and bottom ones, they are **XXX**.

The syndrome measurement circuit corresponds to a measurement of these observables. This is done in a similar way to surface code stabilizers *(in fact, this code is a small version of a surface code).*

Creating the *SYNDROME* Circuit:

Now, we need to work on the syndrome circuit to make sure all CNOT gates are between connected qubits.

Now let's take a look at which qubits need to be connected for our syndrome circuit. The circuit that the problem notebook gives us by default is the following:



```
qc_syn = QuantumCircuit(code,syn,out)


# Left ZZZ
qc_syn.cx(code[0],syn[1])
qc_syn.cx(code[2],syn[1])
qc_syn.cx(code[3],syn[1])
qc_syn.barrier()

# Right ZZZ
qc_syn.cx(code[1],syn[2])
qc_syn.cx(code[2],syn[2])
qc_syn.cx(code[4],syn[2])
qc_syn.barrier()

# Top XXX
qc_syn.h(syn[0])
qc_syn.cx(syn[0],code[0])
qc_syn.cx(syn[0],code[1])
qc_syn.cx(syn[0],code[2])
qc_syn.h(syn[0])
qc_syn.barrier()

# Bottom XXX
qc_syn.h(syn[3])
qc_syn.cx(syn[3],code[2])
qc_syn.cx(syn[3],code[3])
qc_syn.cx(syn[3],code[4])
qc_syn.h(syn[3])
qc_syn.barrier()


# Measure the auxiliary qubits
qc_syn.measure(syn,out)
qc_syn.draw('mpl')
```

Figure 4.1.2: {Code}: Creating a Syndrome Circuit

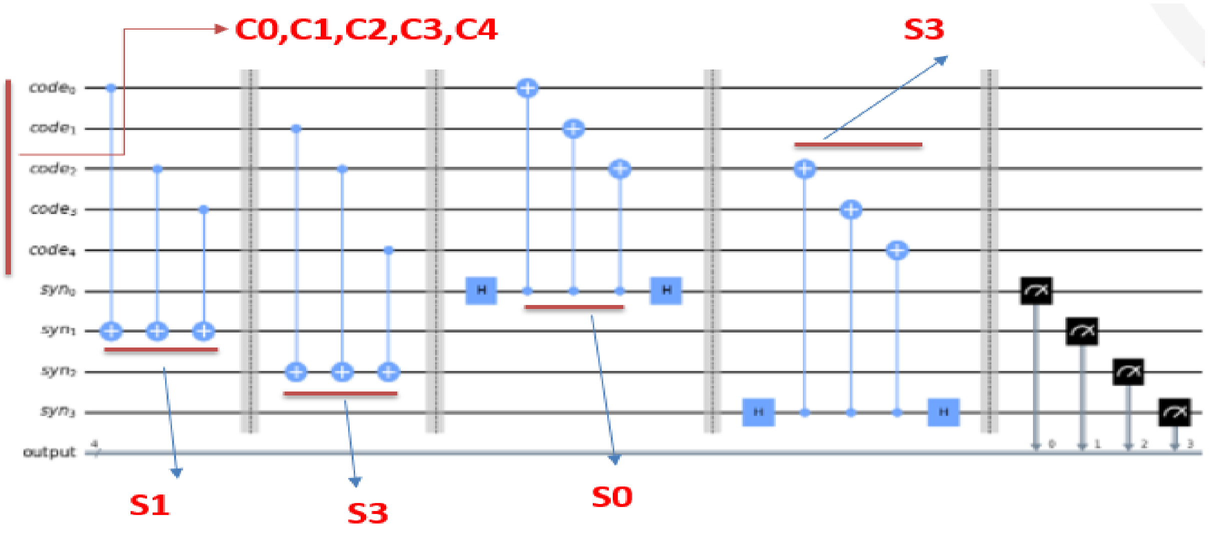


Figure 4.1.3: {Circuit}: Syndrome Circuit

Analyzing this, we find that the following connectivity chart:

```
c0....s0....c1
:     :     :
:     :     :
s1....c2....s2
:     :     :
:     :     :
c3....s3....c4
```

Mapping Logical Qubits from circuit to Hardware:

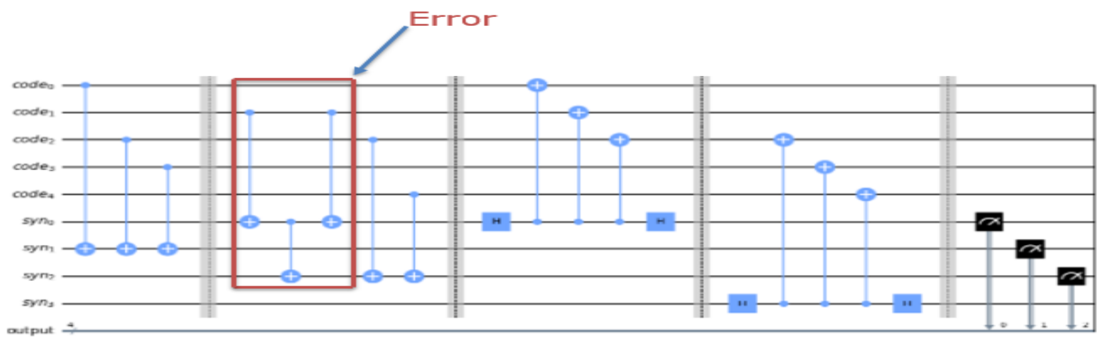
// initializing the Array to ***'initial_layout '***

```
initial_layout = [0,2,6,10,12,1,5,7,11]
```

This means that **code[0]** will map to qubit 0 on the device, **code[1]** to qubit 2, and so on. After looking at this and checking with qubits have a direct connection on the device, we see that the only **CNOT** that will not work on hardware is the first one on the Right **ZZZ** block, i.e. between **code[1]** and **syn[2].**

The easiest thing to do here is to build a series of gates that perform the same operation than a **CNOT** without the two gates being actually connected to each other. And doing this is easy using an **ancilla** qubit. At this point in the circuit, we have two options for **ancilla** qubits, **syn[0]** and **syn[3];** we cannot use **syn[1]** since it has already been used by the first block and it's best to not change its state. 1 chose to use **syn[0].** The code is rather easy. First, we apply a **CNOT** gate with **code[1]** as control and **syn[0].** Since **syn[0]** is still in the **0** state, it will take on the state of code[1] after the **CNOT**. Then, we apply a CNOT gate with **syn[0]** as control and **syn[2]** as target, this will have the same effect as a **CNOT** between code[1] and **syn[2]** directly. Finally, we apply the first **CNOT** again to reverse **syn[0]** to the **0** state.

Circuit after commits *{with respect to Layout}*:



Fake Tokyo's Architecture:

```
qc_init = QuantumCircuit(code,syn,out)

qc_init.h(syn[0])
qc_init.cx(syn[0],code[0])
qc_init.cx(syn[0],code[1])
qc_init.cx(syn[0],code[2])
qc_init.cx(code[2],syn[0])

qc_init.h(syn[3])
qc_init.cx(syn[3],code[2])
qc_init.cx(syn[3],code[3])
qc_init.cx(syn[3],code[4])
qc_init.cx(code[4],syn[3])

qc_init.barrier()
qc_init.draw('mpl')
```

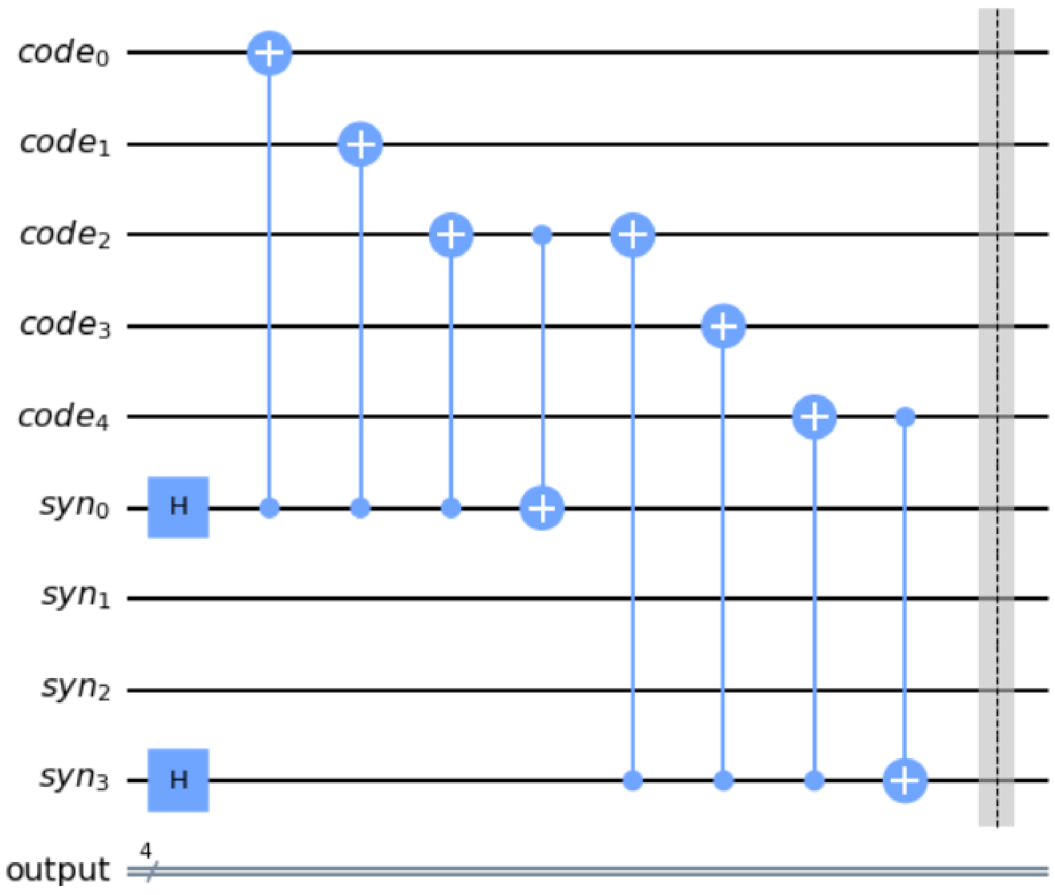
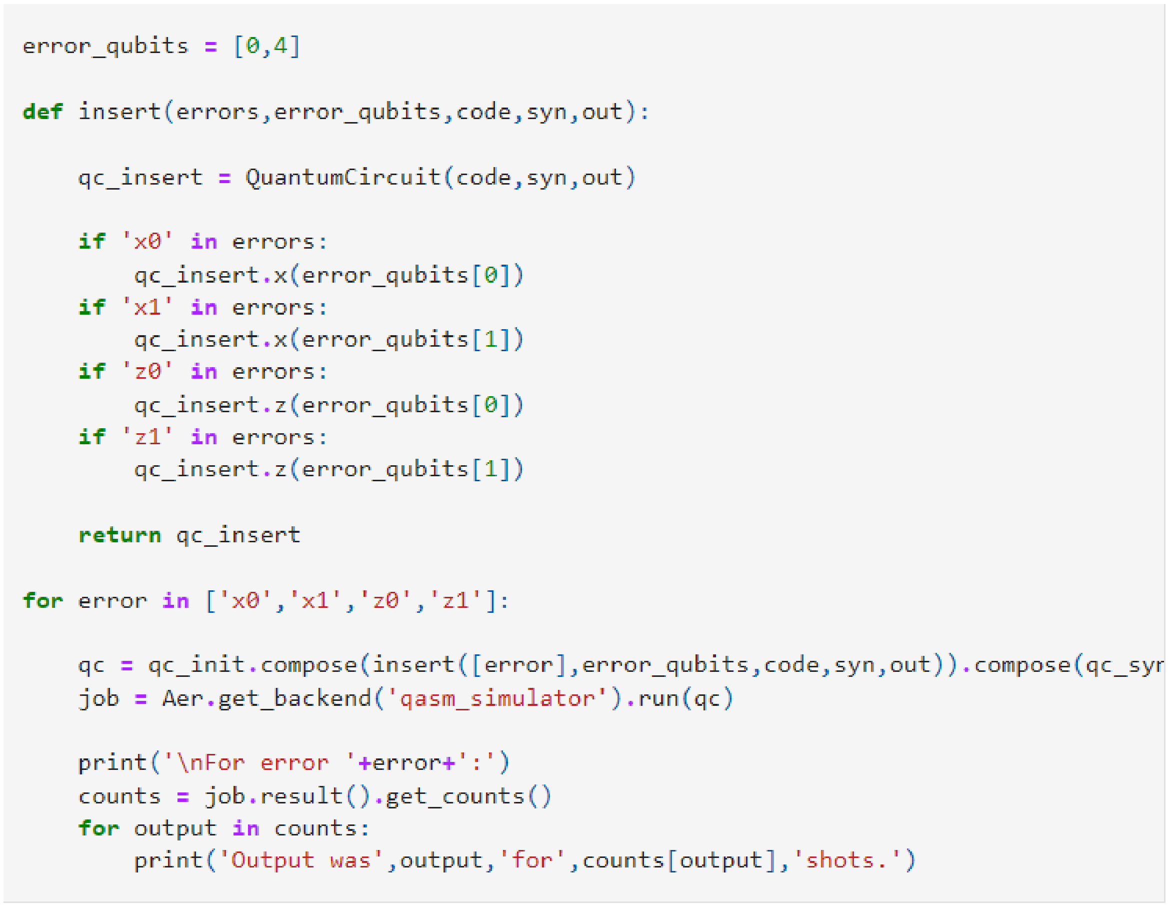


Figure 4.1.4: {Circuit}: Creating a Fake-Tokyo Architecture

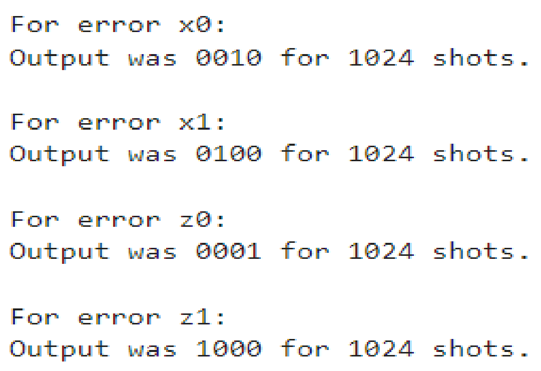
Inducing the *‘Error’* to Circuit Model:

To check that our circuit runs correctly, we define the error qubits and insert some artificial errors to make sure we can detect them:

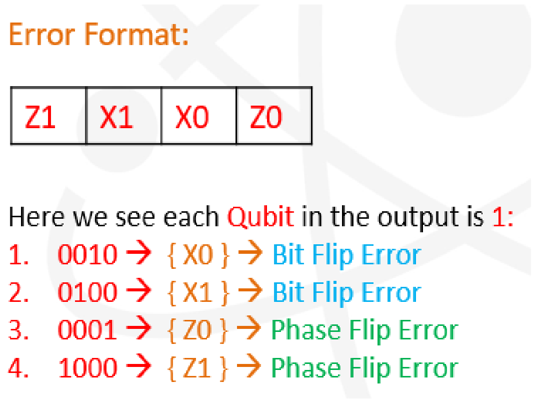


```
error_qubits = [0,4]

def insert(errors,error_qubits,code,syn,out):

    qc_insert = QuantumCircuit(code,syn,out)

    if 'x0' in errors:
        qc_insert.x(error_qubits[0])
    if 'x1' in errors:
        qc_insert.x(error_qubits[1])
    if 'z0' in errors:
        qc_insert.z(error_qubits[0])
    if 'z1' in errors:
        qc_insert.z(error_qubits[1])

    return qc_insert

for error in ['x0','x1','z0','z1']:

    qc = qc_init.compose(insert([error],error_qubits,code,syn,out)).compose(qc_syr
    job = Aer.get_backend('qasm_simulator').run(qc)

    print('\nFor error '+error+':')
    counts = job.result().get_counts()
    for output in counts:
        print('Output was',output,'for',counts[output],'shots.')
```

Detection of both {Error Places & Types}:



```
For error x0:
Output was 0010 for 1024 shots.

For error x1:
Output was 0100 for 1024 shots.

For error z0:
Output was 0001 for 1024 shots.

For error z1:
Output was 1000 for 1024 shots.
```



Error Format:

Z1	X1	X0	Z0

Here we see each Qubit in the output is 1:
1. 0010 → { X0 } → Bit Flip Error
2. 0100 → { X1 } → Bit Flip Error
3. 0001 → { Z0 } → Phase Flip Error
4. 1000 → { Z1 } → Phase Flip Error

Figure 4.1.5: Error Location & Type

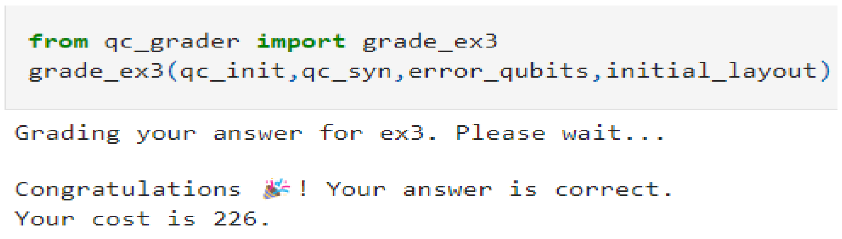
As you can see, we detect all four types of errors correctly and with a different value each.

Expected output:

The initialization circuit prepares an eigenstate of these observables, such that the output of the syndrome measurement will be ***0000*** with certainty.

Hence, the location and type of Error is detected and displayed with 100% precision or accuracy.

4.2 Estimating the cost of Circuit:



```
from qc_grader import grade_ex3
grade_ex3(qc_init,qc_syn,error_qubits,initial_layout)
```

```
Grading your answer for ex3. Please wait...

Congratulations 🎉! Your answer is correct.
Your cost is 226.
```

To calculate the cost:

a. CX-Gate {10} : 22 * 10 = 220

b. H-Gate {1} : 6 * 1 = 6

Total: 226

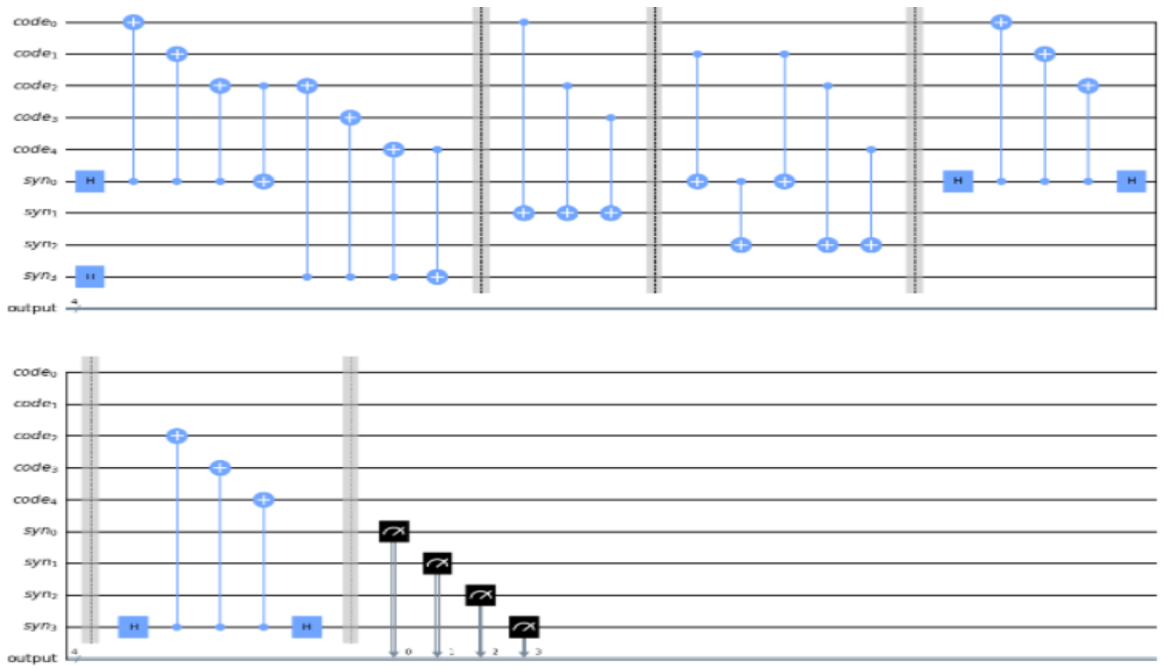


Figure 4.2: {Circuit}: Cost Effective Circuit

CHAPTER 5

CONCLUSION

CHAPTER 5
CONCLUSION

5.1 REFLECTION NOTES

During the period of internship at QIntern-21 organized by QWorld Association, I had a great exposure to Qiskit SDK which they primarily use for designing Quantum application development. My task involved designing the Error Detection Model which is Cost Effective. The design code were written in Python code and simulated on IBMQ. Going through all these procedures I learnt to perform verification Visualization graphs, emulator error model, and functional error due to gate errors.

I got valuable insight into the field of Quantum error Correction & Detection from Industry Professionals and also the different career opportunities for a fresher in the field of Quantum Application developer and also the future growth prospects in this field.

TOOLS USED:

Qiskit is made up of elements that work together to enable quantum computing. The central goal of Qiskit is to build a software stack that makes it easy for anyone to use quantum computers, regardless of their skill level or area of interest; Qiskit allows users to easily design experiments and applications and run them on real quantum computers and/or classical simulators. Qiskit provides the ability to develop quantum software both at the machine code level of OpenQASM, and at abstract levels suitable for end-users without quantum computing expertise. This functionality is provided by the following distinct components.

a. ***Qiskit Terra*** :

The element Terra is the foundation on which the rest of Qiskit is built. Qiskit Terra provides tools to create quantum circuits at or close to the level of quantum machine code.

b. Qiskit Aer:

The element Aer provides high-performance quantum computing simulators with realistic noise models. In the near-term, development of quantum software will depend largely on simulation of small quantum devices.

c. Qiskit Ignis:

The element Ignis provides tools for quantum hardware verification, noise characterization, and error correction. Ignis is a component that contains tools for characterizing noise in near-term devices, as well as allowing computations to be performed in the presence of noise.

\

5.2 REFERENCES

[1] Exponential suppression of bit or phase flip errors with repetitive error correction
https://arxiv.org/pdf/2102.06132.pdf

[2] Introduction to Quantum Error Correction
https://qiskit.org/textbook/ch-quantum-hardware/error-correction-repetition-code.html

[3] Project Source code
https://github.com/ShisheerKaushik24/IBM-Quantum-challenge-2021-
/blob/master/ex3.ipynb

[4] Internship website {Project 17: Mentor: Zeki Seskir}
https://qworld.net/qintern-2021/

[5] Qiskit SDK
https://qiskit.org/documentation/getting_started.html

\