ConvertQC: Mapping Quantum Computing Circuits Between Different Simulation Environments

CS39440 Major Project Report

Author: Harry Adams (haa14@aber.ac.uk)
Supervisor: Dr. Dimitris Tsakiris (dit5@aber.ac.uk)

3rd May 2019
Version: 1.0 (Release)

This report was submitted as partial fulfilment of a BSc degree in Computer Science (with integrated year in industry) (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, U.K.

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Harry Adams

Date: 3rd May 2019

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Harry Adams

Date: 3rd May 2019

Acknowledgements

I'd like to thank the Major Project Coordinator, Mr. Neil Taylor, for working hard to enable this project to go ahead. His coordination between departments was critical to getting this project off the ground. Additionally, he was supportive at a time when I needed it most, for which I am grateful.

In addition, I'd like to thank my supervisor, Dr. Dimitris Tsakiris, for agreeing to supervise in the first place. He took a project completely out of his comfort zone and agreed to supervise allowing me to chase the research area I wanted.

Immense thanks go to Mr. Alexander Pitchford for being a second supervisor, coming in with a range of knowledge relating to quantum information and QuTiP, without who I know I would not have been able to explain the quantum world sufficiently.

My final single-paragraph acknowledgement goes to my fiancée, Sarah Cole. Without her, I truly believe I wouldn't still be here to have completed my dissertation and degree, her immense support has kept us together and kept me going along through all the good and the bad. Thank you. I hope I can be $\frac{1}{n}$th of the partner you've been, my entangled qubit.

Other gratitudes include:

- My family, particularly mum Caroline and dad Andy, for continuous support and pushing me to the best I can be.
- My sister, Rachel, for helping explain quantum mechanics to those who may not have the background knowledge. She's studying engineering at Sheffield and is much smarter than me;
- My brother, Sam, for always being there when I need to rant and relax;
- Aberystwyth Community of Gamers, the society that has been my home for the last four years and I had the honour to run in my last;
- Morgan, Aaron, Phillip, Matthew and Josh - the little group which claimed PJM Lounge for long days of working and actually made report writing an enjoyable activity;
- Stephen Fearnley and Matt Maufe, for excellent proofreading and keeping me sane throughout the project;
- PepsiCo. and my regular supply of anti-depressants for giving the mental chemistry required to produce this work.

This report is written in memory of my Grandad Terry (1937 - 2019) who sadly passed away during the process of this major project.

Abstract

Quantum computers could potentially be the future. They have the ability to solve certain problems, such as breaking internet security via factorising numbers, but it is unknown if they will become viable. To develop these algorithms, they need to be tested using simulators while the technology is developed. To do this there are a number of toolkits available which are either libraries for pre-existing programming languages, or languages within themselves.

This tool, ConvertQC, is designed to convert a script written in one of these toolkits into another toolkit's format. It takes the user's script as an input and then outputs a converted file. It is written in Python, operates on the Linux Command Line, and currently translates between ProjectQ and QuTiP, both Python-based libraries.

For this, I have written code to parse each line of input code, determine its function, and output the same function in the desired output format. It has achieved relative success, but has some areas with which it struggles, such as measurement. Overall, however, it has been a success.

Contents

List of Figures

List of Tables

Chapter 1

Background & Objectives

1.1 The Future?

We stand in the ongoing turmoil of a second quantum revolution. The first came in the early 20th Century, bringing about the beginnings of quantum mechanics and Schrödinger's infamous cat [9]. This brought about the wave-particle duality [10] - the idea that matter particles sometimes behaved like waves, and light waves sometimes acted like particles. With this, we can understand chemical interactions and wave functions that underpin semiconductor physics, the technology behind computer chips and much of the Information Age [11]. The second revolution will take these quantum rules of reality to develop new technologies [12].

One of the more famous examples of how quantum computers will impact us is internet security. A common current-day systems is asymmetric encryption, or public-key encryption, one of the first examples being RSA[1]. One side of a communication has a private key made of two very large primes (p and q), and a public key which is the product of these ($n = pq$). The public key can be given freely, and is used to encrypt a message which can only be decrypted by having the components of the public key (the private key) [13].

A classical computer could, in theory, crack a private key. The RSA-768 algorithm (utilising a 768 bit number) was factorised in an effort that took approximately 1500 CPU years or two years of real time when spread across many hundreds of computers [14]. However, many RSA keys in everyday use range from 1024 to 4096 bits, providing a large amount of security. No larger key is known to have been cracked, and no algorithm is known to factorise an integer in polynomial time. Shor's algorithm [15] is a quantum algorithm which does run in polynomial time[2] which could lead the way to breaking modern day internet security, from an individual sending payments to banks all the way to governments communicating classified information.

Another high-profile area in quantum computing is healthcare. Radiation plans are devised by simulating numerous possibilities. Quantum computers can allow for more possibilities per simulation and reach a plan faster. Additionally, one part of machine learning

[1]Rivest-Shamir-Adleman
[2]more specifically, pseudo-polynomial time

focuses on pattern recognition, and artificial intelligence algorithms are being developed to search through thousands, if not millions, of patient data sets to identify anomalous data, draw conclusions, and form a diagnosis. This is another area that can be improved by quantum computers with their abilities to process multiple data points simultaneously. Healthcare has a range of broad uses beyond these, including imaging, drug research, etc. [16]

Other potential uses include climatology [17], construction & infrastructure [18], or transport management [19] [20]. The benefits to quantum computers are wide and varied. Quantum computers won't replace classical computers, as they are equal or inferior in a range of areas [21], but are much more powerful in others (as described above). However all of these benefits currently exist at a theoretical level. Large companies such as IBM [22] and Google [23] are releasing chips in the 50-qubit range, but this is not the only factor - quantum noise (the length of time a quantum system is stable and useful) is still in the microseconds, and some believe this noise problem may be insurmountable [24]. Quantum Supremacy is the ability for quantum computers to solve problems that classical computers cannot. In terms of complexity theory, this would be achieved by providing a superpolynomial speedup over the best classical algorithm known or available. We won't know if this is achievable until more research and engineering is done.

1.2 Background Knowledge

Information in this section is primarily drawn from Quantum Computation and Quantum Information by Nielsen & Chuang [21]. For a more casual overview in a beautiful animation style, I recommend "Quantum Computers Explained - Limits of Human Technology" by Kurzgesagt on YouTube[3].

1.2.1 Classical Computing

In a classical computer, the smallest piece of information is the bit. This is represented as 0 or 1, off or on, and typically represents a voltage across a circuit. With bits, the amount of information that can be stored increases with the number of bits - 2^n values where n is the number of bits. For instance, 8 bits can store $2^8 = 256$ values.

These bits can be manipulated via logic gates. These take a certain number of inputs, perform operations, and then return a separate output. For example, a NOT gate maps 0 to 1, and 1 to 0, while an AND gate will give an output of 1 if both inputs are 1, and a 0 otherwise.

1.2.2 Qubits

Qubits (a portmanteau of Quantum Bits) are the quantum computing equivalent of classical bits. Qubits are any two-state quantum-mechanical systems, which can include the

[3]`https://www.youtube.com/watch?v=JhHMJCUmq28`

spin of an electron (spin-up or spin-down) or the polarisation of a photon[4] (horizontally or vertically polarised)

Qubits typically use the spin of an electron - as such the rest of this report will refer to qubits in terms of their spin.

1.2.3 Introduction to Quantum Mechanics Concepts

There are a few terms and ideas of quantum mechanics that must first be understood to go on to understand the quantum aspects of computing.

1.2.3.1 Representation

The standard representation of quantum states is Dirac Notation, using Bras ($\langle x|$) and Kets ($|x\rangle$) to represent states. For instance, a bit can be in the states 0 or 1, while a qubit can be in states $|0\rangle$ or $|1\rangle$ (and more besides).

A qubit can also be represented using a Bloch sphere, shown in Figure 1.1. If we were to represent a classical bit using this representation, it could exclusively be at the North Pole or the South Pole ($|0\rangle$ or $|1\rangle$ on the diagram). However, a qubit in superposition ($|\psi\rangle$) can be at any point on the sphere, of which there are infinitely many.

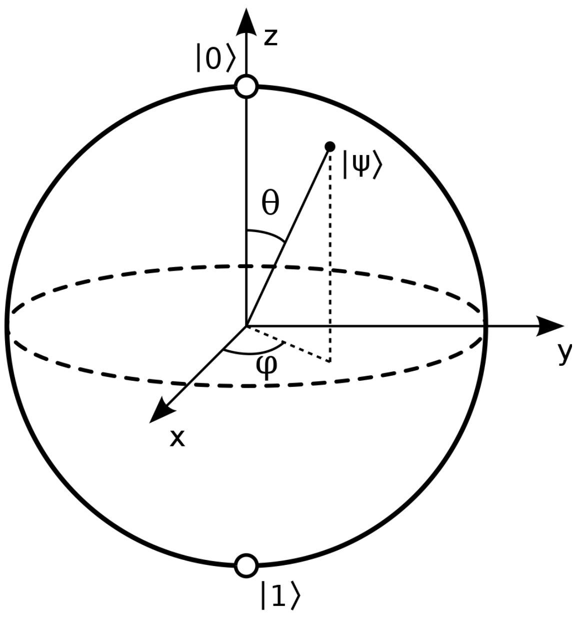


Figure 1.1: Bloch Sphere [1]

Additionally, a qubit can be represented using vectors, where $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ maps to

[4]A photon is an electromagnetic quantum responsible for light. Polarisation describes the plane on which the light oscillates, such as horizontal or vertical polarisation. In this example, a vertically polarised photon can represent 0, and a horizontally polarised photon can represent 1.

$\left[\begin{smallmatrix}\alpha\\\beta\end{smallmatrix}\right]$. The $|0\rangle$ state is mapped as $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ and the $|1\rangle$ state is mapped as $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. This will be useful notation when covering quantum gates in Section 1.3 which can also be represented as unitary matrices, and multiplying them together will produce the new qubit's states and probability amplitudes.

1.2.3.2 Superposition

Qubits can be spin-up to represent $|0\rangle$, and spin-down to represent $|1\rangle$. However, one property of quantum systems is they can be in superposition - that is, they can be at a combination of $|0\rangle$ and $|1\rangle$. This is represented as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where $|\psi\rangle$ represents a qubit in superposition, and α & β are probability amplitudes that can be complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. The probability amplitudes can be calculated from the Bloch sphere representation using

$$\alpha = \cos(\tfrac{\theta}{2}) \quad \beta = e^{i\phi}\sin(\tfrac{\theta}{2})$$

1.2.3.3 Measurements

While qubits (and other quantum systems) are in superposition, they cannot be directly observed. Doing so will cause the superposition to collapse, and for the qubit to permanently fall into $|0\rangle$ with probability $|\alpha|^2$ or $|1\rangle$ with probability $|\beta|^2$. This seems like qubits would be no different to classical bits, until we consider entanglement.

1.2.3.4 Entanglement

Quantum entanglement describes a correlation between two systems so close that they cannot be described separately. In a common example with two entangled qubits, if one were in state $|0\rangle$ the probability of the entangled qubit being in state $|1\rangle$ approaches one. This probability can be relied on for purposes of quantum computation.

1.2.4 Probability Amplitudes

In classical probability theory, a probability lies between 0 and 1, and all probabilities for a system will sum to a value of 1. For instance, a six-sided dice is rolled with six possible outcomes, all with probability $\frac{1}{6}$ on a fair dice. However, in a quantum system, each event has an associated complex value (any number that can be expressed in the form $z = a+bi$ where a and b are real numbers, and i is $\sqrt{-1}$, a solution to the equation $x^2 = -1$). This probability system allows for quantum systems to be affected by interference.

The magnitude - a measure of the size of a mathematical object - is represented by $|x|$ [25]. For real numbers the absolute value is defined by

$$\begin{array}{cc} |r| = r & |-r| = r \\ (r \geq 0) & (r < 0) \end{array}$$

which can be visualised by a number's distance from 0 on a number line - the magnitude of both -5 and 5 is equal to 5. Complex numbers are instead defined by

$$|z| = \sqrt{a^2 + b^2}$$

which could instead be visualised as a distance on a plane from an origin point - this equation is simply Pythagoras' Theorem for calculating the longest length of a right-angle triangle. Within these quantum systems, the squares of the probability amplitudes will sum to 1, similar to classical probability.

As described in Section 1.2.3.1, the probability amplitudes represent the likelihood of different states. For example, take the Hadamard gate as explained in the following section. It will map states $|0\rangle$ to $\frac{1}{\sqrt{2}}|0\rangle$ and $|1\rangle$ to $\frac{-1}{\sqrt{2}}|1\rangle$[5]. A qubit, $|\psi\rangle$, is now in a superposition of these states such that

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{-1}{\sqrt{2}}|1\rangle$$

The probability amplitudes are $\frac{1}{\sqrt{2}}$ and $\frac{-1}{\sqrt{2}}$, therefore

$$\left|\tfrac{1}{\sqrt{2}}\right|^2 + \left|\tfrac{-1}{\sqrt{2}}\right|^2 \quad = \quad (\tfrac{1}{\sqrt{2}})^2 + (\tfrac{1}{\sqrt{2}})^2 \quad = \quad \tfrac{1}{2} + \tfrac{1}{2} \quad = \quad 1$$

Additionally take the Pauli-Y gate operating on the $|0\rangle$ state, which will also be further explained in the Quantum Gates section:

$$|\psi\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix}$$

This can then be represented as

$$|\psi\rangle = 0\,|0\rangle + i\,|1\rangle = i\,|1\rangle$$

The probability amplitudes are 0 and i, and $|i|^2 = 1$.

[5]These are represented by $|\bigoplus\rangle$ and $|-\rangle$ respectively

1.3 Quantum Gates

Just like in classical computing there are a number of gates which can influence qubits as they pass through a circuit. Some of the more common gates include:

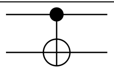
Name	Short	Diagram	Functionality
Hadamard	H	$-\boxed{H}-$	The gate used to put qubits into a superposition. Represented by the matrix: $$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$ Maps $\lvert 0\rangle$ to $\frac{1}{\sqrt{2}}$ and $\lvert 1\rangle$ to $\frac{-1}{\sqrt{2}}$. In layman's terms, means a measurement will have equal probabilities of being 0 or 1 - a superposition.
Pauli-X	X	$-\boxed{X}-$	A special set of phase shift gates which rotate around their respective axes by π radians (180°). The X gate is particularly important as it is the quantum equivalent of a classical NOT gate, with respect to the standard $\lvert 0\rangle$ and $\lvert 1\rangle$. These gates have the following mapping $$\begin{array}{ccc} X & Y & Z \\ \lvert 0\rangle \rightarrow \lvert 1\rangle & \lvert 0\rangle \rightarrow i\lvert 1\rangle & \lvert 0\rangle \rightarrow \lvert 0\rangle \\ \lvert 1\rangle \rightarrow \lvert 0\rangle & \lvert 1\rangle \rightarrow -i\lvert 0\rangle & \lvert 1\rangle \rightarrow -\lvert 1\rangle \end{array}$$ and are represented by the following matrices $$X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \quad Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} \quad Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$
Pauli-Y	Y	$-\boxed{Y}-$	
Pauli-Z	Z	$-\boxed{Z}-$	
Controlled NOT	CNOT		A two-qubit gate that operates a NOT (Pauli-X) gate on the second qubit based on the state of the first qubit. If the first is $\lvert 0\rangle$ then no change is operated, whilst if the first is $\lvert 1\rangle$ then the second qubit will be reversed ($\lvert 0\rangle \rightarrow \lvert 1\rangle$ and $\lvert 1\rangle \rightarrow \lvert 0\rangle$). It is represented by the matrix $$H = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

Table 1.1: Common Quantum Gates [3] [4] [5] [6] [7]

Using these gates we can create circuits to modify the qubits as they pass through. Consider the example for quantum teleportation[6] shown in Figure 1.2. In this instance, a single line represents qubits (in any state) and a double line represents a classical bit once a qubit has been measured. Using this process, the information stored by qubit $|\psi\rangle$ has been transferred, or teleported, to qubit $|0\rangle_B$.

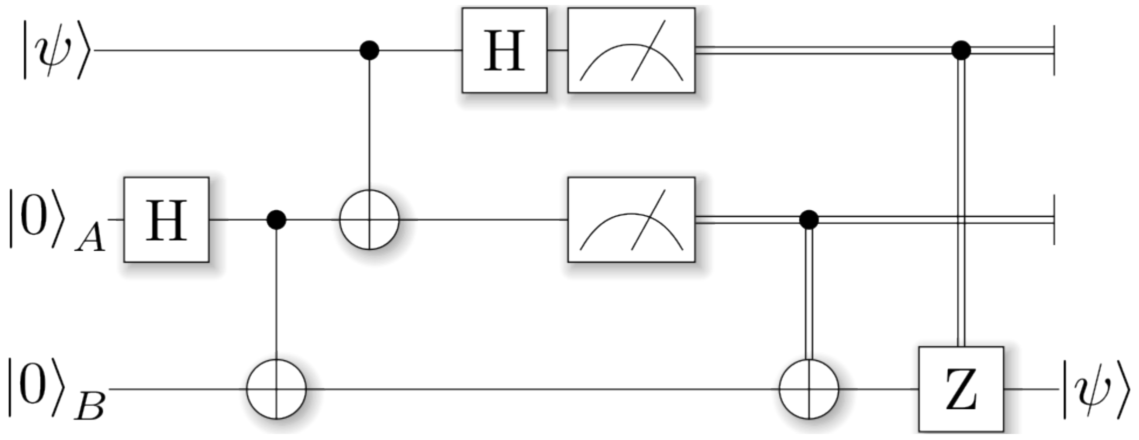


Figure 1.2: Circuit for quantum teleportation [2]

1.4 Classical vs Quantum Computers

Even if Quantum Supremacy is ever achieved then quantum computers likely won't replace everyday classical computers. The most common methods - superconducting circuits or quantum annealing - need cryogenic temperatures to function. Quantum computers show the potential to be vast improvements on a certain subsection of problems, but not all. For example, there are a number of NP complete problems that can be verified in polynomial time, but cannot find a solution to begin with. These problems may not or cannot be efficiently solved either by quantum or classical. The size and cost of quantum computers, in terms of manufacturing and energy, mean that classical computing will still be available for the majority, with quantum computers available for specialised projects.

1.5 Aim of the Project

The aim of this project is to develop a software tool, ConvertQC, which will convert code written in various quantum simulation toolkits into code readable by other toolkits. Each toolkit has their own benefit - for instance, QuTiP [26] [27] has tools for simulating open quantum systems as well as quantum circuits, while ProjectQ [28] has methods to upload quantum circuits directly to IBM's Quantum Experience for simulation or to be run on an

[6]The term teleportation is a bit of a misnomer - it is not a form of transportation but communication. This is a process where quantum information, such as the state of a qubit, can be transmitted via classical communication and previously entangled qubits. It does not require physical items to move in the same way communication over the internet doesn't.

actual 5-qubit quantum computer. This tool allows developers to work in an environment they are more comfortable with and port their code to another toolkit.

1.6 Why This Project

Quantum computing has been a personal interest for a significant period of time. This interest was first developed over the industrial year placement where there was some flexibility to research personal projects, and eventually present them as a learning seminar to the company. Through this an interest in quantum information was developed, and further implemented with the MP35610 Quantum Information Theory 1 module at university. This provided me with a fundamental knowledge of classical information theory, quantum information & encryption, and quantum computation.

With this foundation it was important to have a practical application of this knowledge. This was made clear to the project coordinator in hopes of finding a suitable project. It took two to three months of back and forth between departments, but was eventually confirmed towards the start of January. It is hoped that this project will give me much needed experience to achieve my future career goals, to study a Master's degree and Doctorate in the topic and progress onto research in industry or academia.

1.6.1 Degree Relevance

This project has utilised a wide range of skills and practices from modules undertaken whilst studying at Aberystwyth University. The most important of these was MP35610 Quantum Information Theory 1, taught by Daniel Burgarth, which provided the majority of background knowledge on the quantum computing side. The work methodologies discussed in Section 1.8 are drawn from CS22120 Software Engineering by Chris Price for the Waterfall method and CS31310 Agile Methodologies by Neil Taylor for the Scrum & Agile parts of the development.

Additional skills have been drawn from the Industrial Year placement undertaken during the 2017/18 academic year. These include coding practices such as version control and code standards, as well as practical use of agile methodologies to ensure they are followed correctly. This placement also developed presentation skills, allowing the Mid-Project Demonstration and Final Demonstration to be of higher quality than it would otherwise be.

1.7 Analysis

Before the project title and plan was decided, a meeting was arranged with Neil Taylor, the MMP[7] coordinator; supervisor Dimitris Tsakiris; and Alex Pitchford, a postgraduate

[7]Major & Minor Project

working with quantum information and particularly QuTiP, a quantum toolkit in Python. During this meeting there were two main aims

- Provide a background knowledge of quantum computation & information to those who did not possess it
- Decide a potential area and title for this project

The initial line of intrigue was in a quantum computing simulator, a fully fledged program that could simulate any number of qubits. However it was quickly realised that this would be outside of the scope of a third year dissertation and too difficult. After some further discussion we looked more towards existing tools and simulators such as QuTiP, the system that Alex is involved with, and decide upon a conversion tool to operate between different simulation environments.

1.8 Process

The overall project could be said to follow a Waterfall methodology for its design. Analysis and planning were completed first, including discussions of what the project would entail, the toolkits that would be translated between, and rough timescales of when each piece would be done.

The development process was centred around Scrum, an Agile method, but adapted toward a solo project. For instance, many of the roles have been combined into one - The Scrum Master and Development "Team" were all myself. To a certain extent I was also the Project Owner, as I was familiar with what the tool is designed to do. These roles consist of:

- Project Owner: Determines the tasks to be done, both over the course of the project and in the upcoming sprint.
- Scrum Master: Ensure the project is following the principles of Scrum.
- Development Team: Takes on pieces of work to produce a final product.

Using this, each iteration of work could be planned and evaluated before progressing to the next. Not all features of Scrum were used, such as daily stand-up meetings, due to the single developer nature of this project. Following this methodology also meant following the Scrum Values [29], which is explained more fully in Section 5.3.1 in the Evaluation.

1.9 Deliverables

The deliverables for this project include:

- The source code for ConvertQC including all additional files used to separate code into logical sections
- A clear README file which includes a project description, dependencies, and a user guide.
- The technical report which details the project analysis, implementation, and critical evaluation.

Chapter 2

Design

2.1 Overall Architecture

The software solution is ConvertQC (short for Convert Quantum Circuits), a Python tool to convert between some of the popular tools for quantum computing simulation. This tool works as follows:

- *Take user input and arguments.*
  The tool is run from the command line with arguments that the user specifies, but at minimum an input file, input format, and output format.
- *Process the arguments.*
  Based on the input format, different functions need to be called to correctly parse the input.
- *Convert the input file into the desired format.*
  Parse the input file and identify functions, gates, and qubits used to correctly output into the desired format.
- *Output conversion to file.*
  Once the output has been written, write this to a file for the user.

2.2 Usage

Each toolkit used has a different method of implementing quantum circuits. Let's use a simple quantum random number generator as an example - a qubit in a superposition of $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ which as described previously is a 50% chance of measuring 0, and a 50% chance of measuring 1. To visualise this, I created an account on the IBM Q Experience site [30] which allows for quantum circuits to be tested on their IBMQX4 chips, a 5-qubit quantum computer. Doing so created the circuit shown in Figure 2.1:

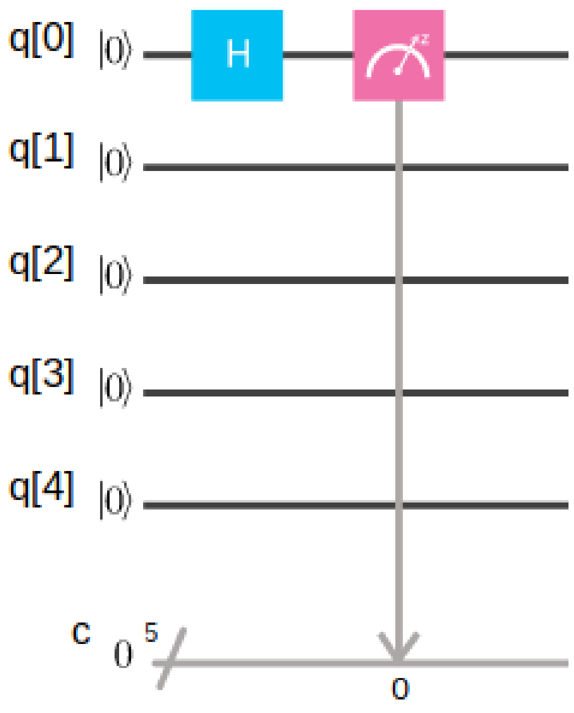
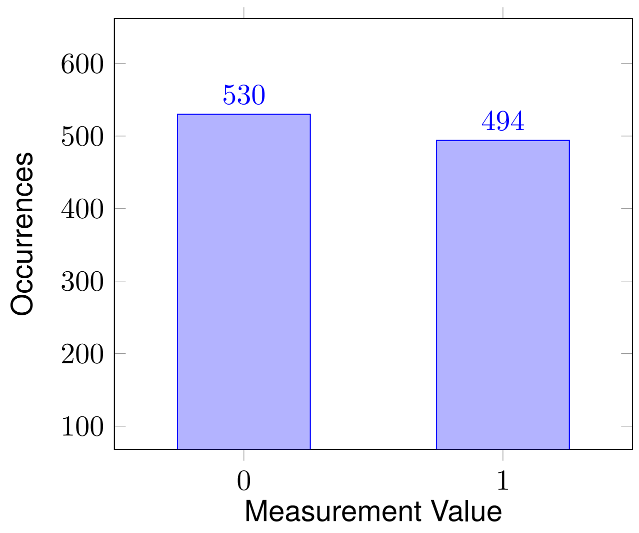


Figure 2.1: Random Number Generator Circuit Setup on IBM Q Experience

This was uploaded to the IBM quantum computer to be run 1024 times, producing the following results:



Within a small margin of error this shows a near-equal chance of being either 0 or 1. This sort of quantum circuit can be written in ProjectQ, and is shown in Section C.1:

- A simulation engine is created on line 5. ProjectQ contains a number of engines, such as one which uploads directly to the IBM Q Experience simulator.
- A qubit, q1, is allocated on line 8.
- This qubit has a Hadamard gate applied to put into a superposition of $|q1\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{-1}{\sqrt{2}}|1\rangle$ such that there is a 50% chance of measuring 0 and 50% chance of measuring 1.
- The qubit is measured on line 14, causing the superposition to collapse and the qubit to become a classical 1 or 0 bit, the state of which is then output on line 18.

Despite the simplicity it shows what format certain functions of ProjectQ will take and what can be done with it. The same function can (almost) be performed using QuTiP, as shown in Section C.2:

- Line 4 creates a quantum circuit as an object, with the parameter specifying how many qubits to include.
- The Hadamard gate is applied on line 7 (snot is the QuTiP abbreviation for it) which produces the superposition described above.
- I was unable to find methods in QuTiP for measuring a qubit as part of the quantum circuits, and am not sure it exists after discussions with Alex.

Each function performs the same task (without measurement) but in different toolkits. The aim of ConvertQC is to be able to take one toolkit as an input, and then output this into the second toolkit. This can currently only work between ProjectQ and QuTiP but future development will allow other toolkits to work as input and output formats, such as those listed in Table 2.1.

2.3 Research

The beginning phase of the development was focused on research and planning on a more general sense, before moving into the iterations of development. Different toolkits were evaluated and considered before deciding which to focus on, as seen in Table 2.1. It was not simply a case of looking at which toolkits had the most benefits, but a wide range of factors. The eventual decision was to use QuTiP due to the availability of the supervisor for assistance, and ProjectQ for its simple syntax and beginner friendly nature.

2.4 User Stories

As part of the Agile methodology a list of stories is produced. These are promises of a future conversation about the tasks with the customer. To follow this procedure, a meeting

Name	Advantages	Disadvantages
QuTiP (Python)	• Supervisor is one of the lead developers on the project, so can ask questions and get advice as needed. • Widely used, with 371 citations at time of writing, meaning tool may have more value.	• Fairly complex and includes a lot of features not required as part of this tool. • Documentation not particularly newcomer-friendly.
ProjectQ (Python)	• Very beginner friendly and simple syntax. • Can upload scripts directly to IBM's Q Experience Simulator.	• Still in early development, missing features and particularly tests.
Qiskit (Python)	• Directly interfaces into IBM Q Experience; only toolkit with dedicated quantum hardware (IBMQX series). • Circuit drawing tools available. Lots of good quality documentation and examples including newcomers to the field.	• Not full range of gates, so possibilities are limited.
Quantum++ (C++)	• Simulation environment only restricted to available memory. • Entirely based in header files with low dependencies.	• No longer regularly developed. • Unfamiliar with C++, meaning more difficult development.
Q#	• Access to Azure Cloud for more qubits in simulation. • Large library support (existing algorithm functionality).	• Azure Cloud access requires paid subscription. • Independent programming language, not a toolkit, so more learning required before implementation.

Table 2.1: Non-Exhaustive Comparison of Different Quantum Toolkits [8]

was arranged with the project supervisor acting as a customer, allowing some back-and-forth to develop a list of stories:

- Take the user's input to convert.
- Translate the set of quantum gates.
- Translate control functions, such as Dagger.
- Transfer comments and functions appropriately.
- Ensure function definitions are copied appropriately.
- Indicate where code has failed to translate appropriately.
- Output the converted script to a file of the appropriate format.
- Run tests on the output to ensure the conversion has worked.

This set of user stories would be required for every toolkit that ConvertQC would translate between. Additionally, each of these could be broken down into a further divided set of tasks for the development team. For instance, rather than the broad task of "Translate the gates", there would be individual tasks to "Correctly translate the Pauli-X gate", Correctly translate the CNOT gate", etc. with a set of requirements to include testing before marking the task as complete.

2.5 Development Choices

Before beginning development a number of decisions had to be made about the technical environment to be used. The first of these was version control, the process of monitoring development on a project and allowing easy reversion of code issues that may crop up. This was an easy decision as a GitLab suite [31] was provided by the department. It provided project security by requiring my personal university credentials to log in and is automatically private. The other primary alternative, a personal GitHub repository, would require the repository to be public and could leave me open to accusations of plagiarism. For the technical report, it was written in Overleaf [32], an online LaTeXeditor that comes with built-in version control. It is also stored on the cloud for an extra backup.

The language used was primarily decided by the toolkits that the tool was trying to convert between. Both QuTiP and ProjectQ are Python-based toolkits, and as there were the primary focuses of this dissertation, it was deemed appropriate to also write this tool using Python. Additionally it was simple to use, and offered numerous benefits such as large standard libraries for a range of uses, a clean design, and the ability to easily integrate other languages as needed. However, it must be noted that there are disadvantages such as slow run speed and higher potential for runtime errors. The advantages outweighed the disadvantages for this use case.

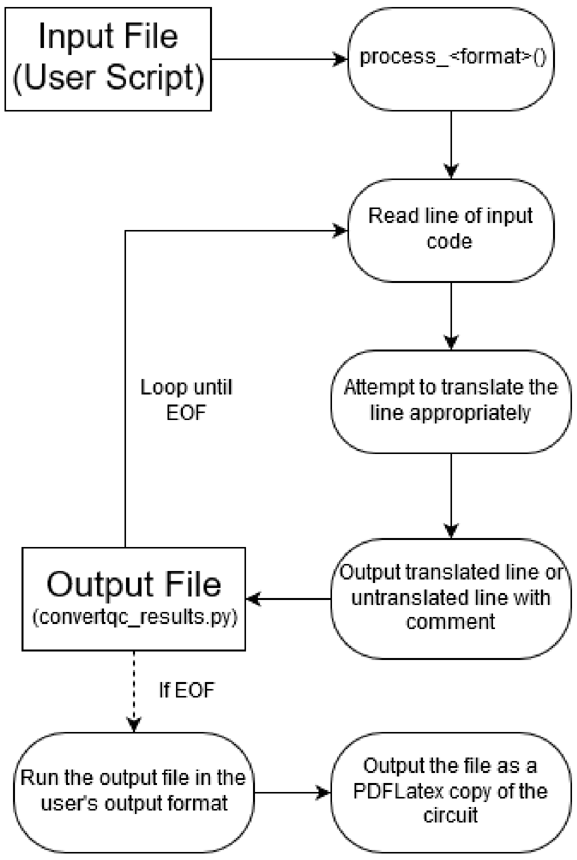

Input File
(User Script)
process_<format>()
Read line of input code
Attempt to translate the line appropriately
Loop until EOF
Output File
(convertqc_results.py)
Output translated line or untranslated line with comment
If EOF
Run the output file in the user's output format
Output the file as a PDFLatex copy of the circuit

Figure 2.2: Data Flow Diagram for ConvertQC

The final development choice that had to be made was which IDE[1] to use. This came down more to personal preference, and a recommendation for PyCharm [33] led to development using it, primarily for its built-in Python compiler, aesthetically appealing UI, and automatic tracking of Python standards deviations.

2.6 Data Flow

As part of my design, a Data Flow Diagram was created to clearly illustrate which route the data takes through my program. As shown in Figure 2.2, this was a relatively simple diagram. It shows the data enters from the user's input file, before running through the steps listed earlier in the Design section to translate the input code. Following the translation, the line is output to the output file before moving on to the next line of the input.

Following this, the program will have an option to run the script in the desired output. By doing this, the user can check the translated script works, and if not, see the first place where the code fails to run. In addition to this, there will be an option to output the script to a PDF diagram of the circuit by translating to a TEXfile, which is then converted to a PDF file via the PDFLatex package.

[1]Integrated Development Environment

2.7 Licensing

It was decided to produce this tool under the GNU GPL v3.0 License [34]. This is due to the open source nature of the toolkits it involves, and due to the author's personal preference for keeping software open source for members of the community to enjoy and develop. This was the preference over a less restrictive license, such as MIT, due to any resulting code produced also being under the GNU GPL v3.0 license, keeping the project open source and free for anybody to use.

2.8 User Interface

User Interface (or UI) was one of the easier parts to plan, but a lot harder to implement due to inexperience with any Python UI toolkits. To begin with, elements that users should be able to do were listed:

- Run the script
- Choose an input format
- Choose an output format
- Compare the input script and the output script

Following this I took two simultaneous approaches - I worked out how to make a useful help window, and drafted what I wanted the UI to look like if I had the ability.

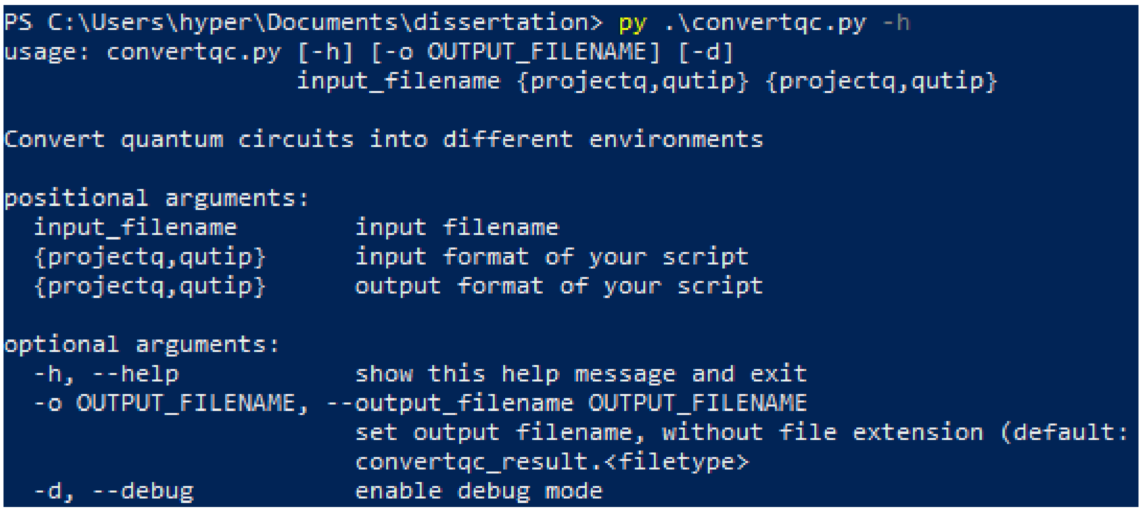
2.8.1 Command Line

I was initially unfamiliar with how Python and the command line worked together. My initial attempts focused on using sys.argv, which is a list of all the arguments included when the program is run. I was then parsing this list, ensuring they matched correctly, that the expected number of arguments was presented, etc. I felt there had to be a better way.

My research brought me to ArgParse, a part of Python[2]. This works by creating an ArgumentParser which handles parsing in the background (line 3). To this, required arguments can be added - in this case an input file and currently an input/output format - as well as optional flags. At the time of writing, these included a debug mode and specifying the name of an output file. In the script, the parser is stored as a global variable. This allowed it to be easily accessed in any subsequent functions without having to pass it as a local variable.

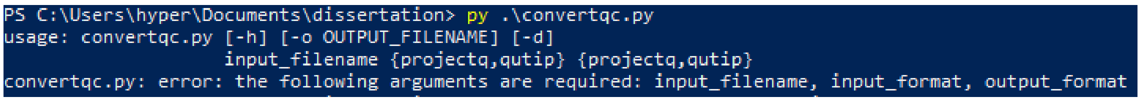
ArgParse was able to handle all of the formatting for me, giving some example outputs:

[2]Function included in Section C.3



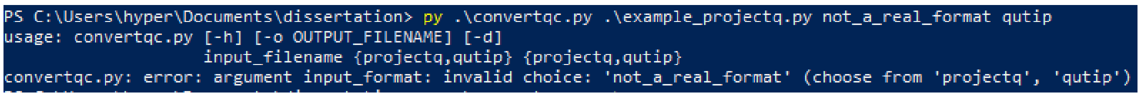
```
PS C:\Users\hyper\Documents\dissertation> py .\convertqc.py -h
usage: convertqc.py [-h] [-o OUTPUT_FILENAME] [-d]
                    input_filename {projectq,qutip} {projectq,qutip}

Convert quantum circuits into different environments

positional arguments:
  input_filename        input filename
  {projectq,qutip}      input format of your script
  {projectq,qutip}      output format of your script

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT_FILENAME, --output_filename OUTPUT_FILENAME
                        set output filename, without file extension (default:
                        convertqc_result.<filetype>
  -d, --debug           enable debug mode
```

Figure 2.3: ConvertQC run with -h flag



```
PS C:\Users\hyper\Documents\dissertation> py .\convertqc.py
usage: convertqc.py [-h] [-o OUTPUT_FILENAME] [-d]
                    input_filename {projectq,qutip} {projectq,qutip}
convertqc.py: error: the following arguments are required: input_filename, input_format, output_format
```

Figure 2.4: ConvertQC run with no arguments



```
PS C:\Users\hyper\Documents\dissertation> py .\convertqc.py .\example_projectq.py not_a_real_format qutip
usage: convertqc.py [-h] [-o OUTPUT_FILENAME] [-d]
                    input_filename {projectq,qutip} {projectq,qutip}
convertqc.py: error: argument input_format: invalid choice: 'not_a_real_format' (choose from 'projectq', 'qutip')
```

Figure 2.5: ConvertQC run with invalid arguments

- Figure 2.3: ConvertQC run with the -h flag. Displays a help window, including a description of the program, and a concise list of required (positional) and optional arguments.
- Figure 2.4: ConvertQC run with no arguments. Displays intended usage, as well as the required arguments.
- Figure 2.5: ConvertQC run with invalid arguments. Again shows the intended usage, but this time specifying the invalid argument and the valid options.

2.8.2 GUI

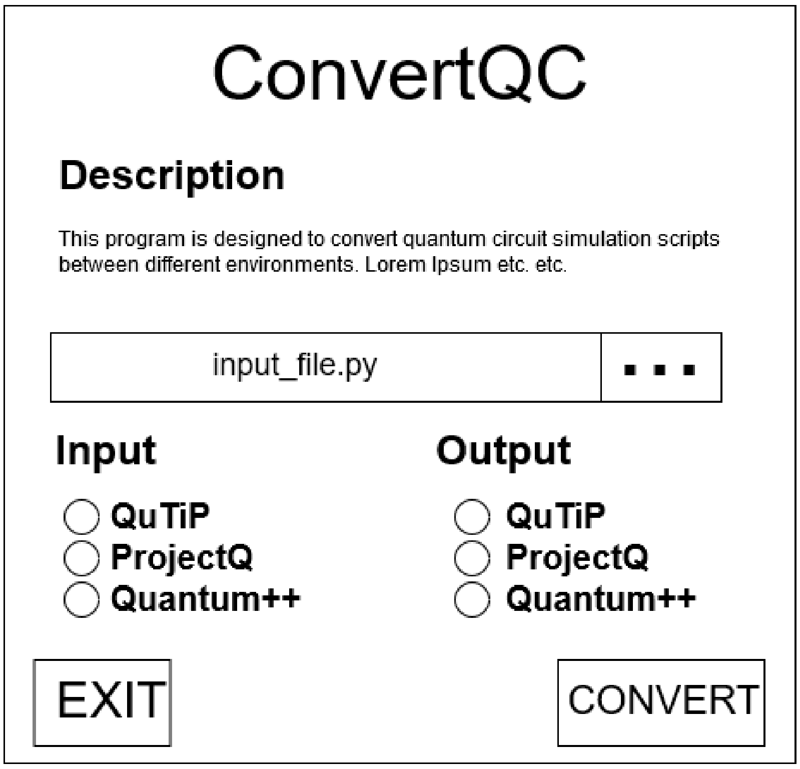
Using draw.io [35], a mock-up of a potential graphical user interface (GUI) was designed to give an idea of what would have been developed with more time and ability, as shown in Figures 2.6 and 2.7.

Figure 2.6 shows the window that would be shown on running the program. This contains a dialog box allowing the user to select their input file. Below this are two columns of choices for input and output format choice. As the tool develops and includes more languages, these could be transformed into scrolling sections sorted either alphabetically or with the most common at the top.

Figure 2.7 is the window after translation has been completed. The two sections allow the user to compare their input script and the translated output script. This is also optionally colour-coded in a similar fashion to some diff tools for git. Green lines highlight the translated lines, white represents unchanged lines, whilst red shows lines that the tool was unable to convert. This allows the user to clearly see and compare the changes, and alerts the user to failed lines so they can change them manually. Missing from the diagram, but features to include, are saving an error log and being able to edit the output script in this window before saving it (and perhaps highlight this in another colour to indicate lines changed by the user).

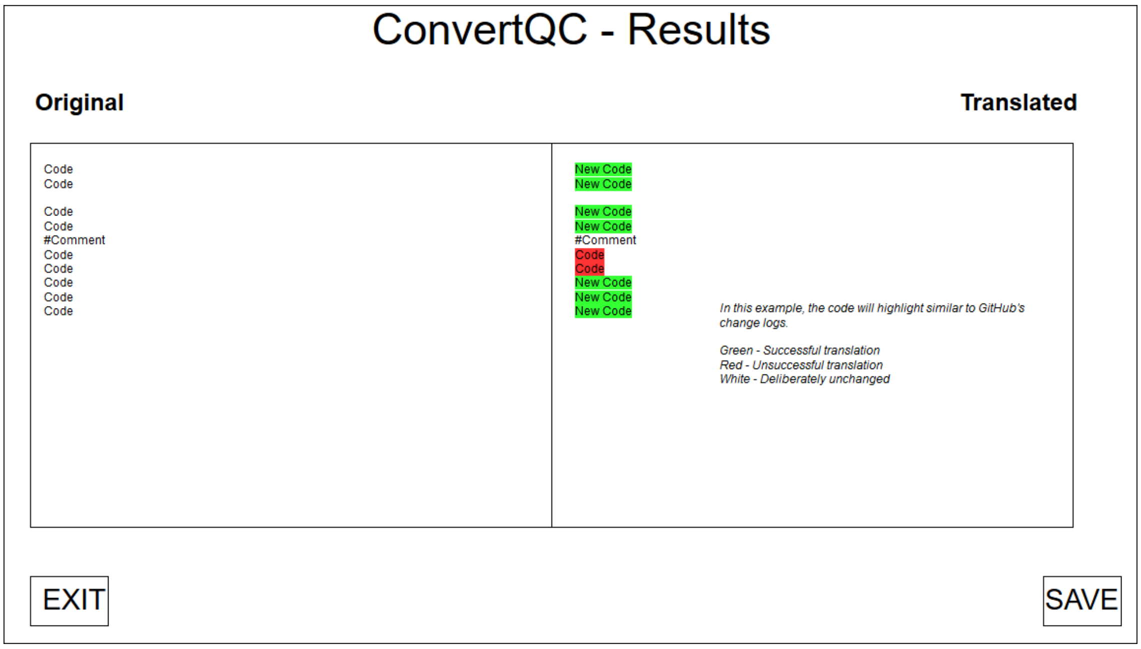
2.9 Foreseen Challenges

A number of potential challenges were foreseen in advance of development beginning. The first of these was being able to correctly parse code and process how different people would write their code. Not everybody will have followed coding guidelines and standards, with issues such as incorrect spacing or improperly formatted strings, meaning the code is incorrectly parsed. My intention for working around this was to implement some auto-formatting to ensure the input code is in PEP8 [36] format.



This program is designed to convert quantum circuit simulation scripts between different environments. Lorem Ipsum etc. etc.

Figure 2.6: First GUI Window



```
Code
Code

Code
Code
#Comment
Code
Code
Code
Code
Code
```

```
New Code
New Code

New Code
New Code
#Comment
Code
Code
New Code
New Code
New Code
```

In this example, the code will highlight similar to GitHub's change logs.

Green - Successful translation
Red - Unsuccessful translation
White - Deliberately unchanged

Figure 2.7: Second GUI Window

Chapter 3

Implementation

3.1 Sprints

Sprints were divided into two-week periods with specific tasks to complete during these sprints. It was decided early on that a two week iteration, provided it contained appropriate sized stories and tasks, would allow for continuous development without requiring a re-evaluation of the Scrum principles.

3.1.1 Sprint 0: Thursday 24th Jan - Wednesday 30th Jan

This sprint involved the work done before the project officially began. As discussed in Section 1.7 before development could begin it needed to be finalised at a departmental level and find an appropriate supervisor. This was officially confirmed on Thursday 24th January. Following the confirmation, discussion took place with Dimitris and Alex to establish a meeting date and what would be required prior to this.

A meeting was arranged for Thursday 31st January. Prior to this I was tasked to create an overall plan for the project. This would include rough time estimates for how long different sections would take and what the overall steps would be.

3.1.2 Sprint 1: Thursday 31st Jan - Wednesday 13th Feb

The first meeting between myself, Dimitris, and Alex started this iteration. We discussed the suitability of the plan, and adapted this. Initial discussions suggested this tool could be a feature of QuTiP to translate between QuTiP and other languages and vice versa. After further discussion, however, it was decided to create a standalone tool that would have QuTiP as one of its input & output formats.

During this iteration the focus was on researching the toolkits and environments that ConvertQC would operate on. From this a list was developed:

- QuTiP - this would be the primary focus due to it being the area Alex is involved with and most feedback would be available quickly if needed [26] [27].
- Qiskit - quickly became a popular option to include, Qiskit is IBM's framework. It could be used to directly tie in to their public quantum simulator [37].
- Q# - Microsoft's entry into quantum computing, with their bundled Quantum Development Kit. This is its own programming language as opposed to a toolkit for a language such as C#. As such, with a short timeframe for the project, it was decided not to focus on this [38].
- Q++ - A C++ header that works in a similar manner to QuTiP. It was decided not to focus on this one due to unfamiliarity with the C++ language, but would likely be a strong contender for future work [39].

In the second week of the iteration Alex advised toward ProjectQ [28], another Python based framework that had a lot of the simulation backend provided. Additionally it provided functionality to upload directly to IBMs simulator, the IBM Quantum Experience [30]. It was decided to focus on this framework due to its ability to directly upload to the IBM computers.

It was also during this sprint that the research described in 2.3 and choices made in 2.5 took place. The repository on the university's GitLab server was set up to allow for version control by pushing an initial README file, and PyCharm was installed and configured to develop the tool.

The final aim of this iteration was to re-familiarise with Python, in terms of its syntax and tools available. I was not hugely experienced in it to start, having learnt the language during self learning at my industrial year. As such, this iteration was spent going over old Python programs, running through some online training courses, and generally familiarising with Python.

3.1.3 Sprint 2: Thursday 14th Feb - Wednesday 27th Feb

Development formally began during this sprint after another meeting with supervisors. The first step was to implement the entry program convertqc.py. By doing this, the user is able to call one command and specify arguments to process their file, rather than have a variety of files that the user needs to sift through to find the one they need.

The first thing to be developed was the help window, as shown in the User Interface section. By doing this first the required arguments and optional arguments could be used as guidelines for future work. The optional arguments included a Debug mode, which outputs everything required for debug; a Verbose mode, which outputs in more general terms what is happening; and the option to specify an output filename (else the default convertqc_output.py will be used).

Following this, defensive programming features were implemented related to the arguments, the code for which is shown in Appendix C. The top of the function contains a list of error codes stored as constants. This could eventually be stored in its own file, for clarity, but as the tool is still fairly small it was decided to leave the error handling and codes together. This file would be imported by all other files, and provide a common function for reporting errors by providing the error code, and whether or not the error is fatal (causing the application to stop). Fatal errors might include being unable to find the input file, or the desired output filename containing invalid characters.

Once the arguments have been checked and the input format identified, the arguments are passed to a separate script based on which toolkit the input format is. Currently there are two, process_projectq.py and process_qutip.py. As more languages are added, the specifics for processing these input languages would also be put into their own files for the sake of clarity (process_qiskit.py, etc.). By doing this, development is separated into the related inputs without chance for confusion between languages.

It was decided to focus on ProjectQ to QuTiP as the first direction, as it seemed the simpler language to begin with and had some very clear examples to use as inputs. The first step was to process the entry point for this script, which is called from convert_qc.py and the code for which is added in Appendix C, Section C.4. This was a self-titled function, and takes the arguments the script was run with (as an ArgParser object) as well as the desired output filename. Whilst the output filename is taken as an optional argument, it is taken without a file extension. This is because plans were initially in place to handle formats such as Quantum++, which would require a .h or .cpp file extension as opposed to .py.

From here, the required arguments are removed and stored as global variables to allow easy access in other functions. This appeared to be the preferable option, as opposed to passing the whole ArgParsed object to subsequent functions, so the other functions only acquire variables they need. Following this, the output file is opened to allow ConvertQC to write to it. This will later be closed at the end of this function call. The input file is read into a list object, to allow later functions to work on the file as a whole. write_new_function() is then called, which handles all of the code to read through the file and convert each line as appropriate. The final function call, conversion.output_error_log, will output to a separate error_log.txt file unless disabled in the arguments.. The parameter error_lines is a dictionary which maps a line number to lines which could not be translated to the intended output language.

3.1.4 Sprint 3: Thursday 28th Feb - Wednesday 13th Mar

Unfortunately the development on this sprint was impacted due to personal circumstances. At the start of this previous sprint I learned my Grandfather had passed away, and had to return home during this sprint to attend the funeral. Due to this unplanned trip home, the work done during sprint 3 and 4 were rather incomplete, causing me to fall behind on my plans.

Work was started on the conversion functions to go from ProjectQ to QuTiP. This began by listing out the different gates available, of which there are many [40]. These were then matched to their QuTiP equivalents. Other common functions were then found from examples, such as Control which operates similarly to a controlled gate, but based on which qubit is passed to it as a parameter. Take the following example:

```python
with Control(eng, b1):
        X | b2
```

The parameter "eng" is the engine that ProjectQ runs with, and is not relevant in this example. The second parameter, "b1", is a qubit which is used as the control point: if the qubit is measured to be a classical 1, then the following code will be run, and otherwise will not. It operates similarly to an "if" statement, but will only operate if the qubit has been previously measured. If this was used in an "if" statement then the qubit may be measured prematurely.

Development began by identifying the line that was being read in. For every line of the input file, it is initially stripped to remove leading whitespace and then split using a space character as a delimiter and storing each separate word as an entry in a list. Then, these lists were searched for keywords:

- "#": A comment. This had to be carefully checked for variables or parameters that may include the "#" symbol. This was primarily done by only checking the first item in the list. However, if a comment appears without a space after the "#" symbol, such as "#comment" then this would not be detected. There is a secondary check to see if the first character is also a "#". Upon further reflection it was realised that only the second check was required; this could later be refactored to improve efficiency.
- "def": A function definition. It was decided to leave function definitions exactly as they were. This is so that the user can more easily compare the ConvertQC output to their input. Additionally, all lines under this function will use parameters that have the same names, so to avoid any confusion, the entire function definition was added as one line.
- Empty Lines: Rather than checking for an item in the list, check to see if the list is empty, meaning the input line had no content. Printed out another empty line.
- "|": Whilst technically a Bitwise OR operator, this is used in ProjectQ to indicate a gate being applied to a qubit, such as in the line "H | qubit_one". The H gate is being applied to qubit_one so by looking for the pipe character the gates can be identified.
- "with": This was paired with another keyword from a set list, such as "Control" or "Dagger", both meta processes as part of ProjectQ. Once both had been identified, they were passed to another function to handle meta conversion.

Once these key terms had been identified, a range of functions were developed to begin to process. The first three items of the above list were copied verbatim into the output file as they needed no modification.

The conversion scripts were fairly simple once the lines had been identified. One function was used to convert rotation gates:

```
def rotation_gate(whitespace, orientation, qubit, angle):
    if orientation == "X":
        print(whitespace + "quantum_circuit.add_gate(\"RX\", " +
            ↪  str(qubit) + ", " + str(angle) + ")",
            ↪  file=output_file)
```

Separate "if" statements were used for the three orientation axes, X Y & Z. The whitespace parameter is simply the amount of whitespace on the original line - as Python does not use semicolons or curled brackets to define what code falls into control statements, each line and its placement is determined only by preceding whitespace. By saving the original whitespace this has been fairly well maintained. Following this is which qubit is being operated on, and the angle used. In quantum computing there are variants of a rotation gate - there are Pauli-X/Y/Z gates, which are a complete reversal where the angle given is π radians or 180°. However, there are also more specific rotations, which can rotate by any amount of radians specified. This parameter allows the same code to process all kinds of axis rotation gates.

All other conversion was fairly simple, and involved matching the input format for a particular sequence to the output format.

3.1.5 Sprint 4: Thursday 14th Mar - Wednesday 27th Mar

During the first week of this dissertation work was remarkably slow. I was still visiting family following the funeral so work was slowed. However I was content with the progress that had been made so far, but just had to make some adjustments to the scale of the project, primarily only translating between two toolkits instead of three or more.

On Tuesday 19th March, the Mid-Project Demonstration was held. This was a chance to demonstrate the progress of the project to a second supervisor who has had no involvement with project development and has no background knowledge to the project. The second supervisor was Patricia Shaw who gave some constructive feedback following the demonstration, which will be used for valuable improvements to give a better final demonstration after the project is submitted:

- *Project Description - able to explain what the project is about and general purpose?* Rated 4/5, suggesting a very good overall understanding but one or two areas which were less specific.
- *Technical Work - how advanced is the technical work?* Rated 3/5, suggesting reasonable progress but several areas are underdeveloped. This was a measure of the progress of the dissertation project and the demonstration was just following the aforementioned personal circumstances. As a result, this was the expected mark for this section.
- *Technical Issues - to what extent are the technologies in the project understood?* Rated 4/5, suggesting a very good understanding of the technology and technical issues but with one or two areas which are less specific.
- It was unclear what the aim of the project was when initially described. Specifically it was unclear the tool was designed to parse code and output to another format, or what the benefits of this were. This can be improved during the final demonstration.

- A suggestion was made to use a common descriptor format (similar to XML) to allow for easier expansion. If the project were developed further this would be a route to consider, but due to only featuring two languages and a direct translation, the decision was made to continue with the current direction.

Following this demonstration, work began on translating code in the other direction, from QuTiP into ProjectQ. Before this began, development stopped to evaluate the progress on the code created so far. This pause allowed the team to identify potential issues that could carry over to translation in the other direction. For instance, it was realised that some code was too broad in its evaluation, and was duplicating lines due to a missing "elif" statement. Additionally there were multiple instances of extra blank lines being included or incorrect indentation causing lines to fall outside of a control statement. By identifying these issues early, they could be patched and used again in the reverse translation

It was also toward the end of this sprint that the refactoring described in Section 3.3.1. Doing this allowed for the repetitive aspects of the code, such as reading files, to be separated out and used again.

By this point there were only a few days left of the sprint, so the basic requirements were added to allow ConvertQC to call the entry point of the process_qutip.py scripts. These basics would read the input file and open the output file ready for the conversions to enter.

3.1.6 Sprint 5: Thursday 28th Mar - Friday 5th Apr

This sprint was extended until the end of the week, due to it being the final teaching week of the semester and meant the next sprint could begin cleanly at the start of the Easter holiday.

This sprint continued development on translating in the other direction from ProjectQ to QuTiP. Converting in this direction was more difficult, due to having to use regular expressions (regex) to try and pick out the arguments used in function calls. This being considered, progress was still made but more slowly. Additionally these sprints are largely repeats of the sprints described in much more detail earlier on, just in the opposite direction, so will instead focus on the differences rather than the process.

One of the biggest differences is in readability and "friendliness" for lack of a better term - the ProjectQ code had easy to read, beginner friendly documentation compared to QuTiP which assumes a lot of background academic level knowledge[1] and made it slightly harder to identify exactly what some of the code was doing and find examples of it working. In terms of readability, the prime difference is that ProjectQ has a gate which operates on a qubit as two separate objects, whereas QuTiP has a Circuit object to which gates can be added. It seems a small difference but a user can clearly go through a ProjectQ example and see exactly what each gate is doing without it seeming like reading lines of code.

By the end of this sprint, ConvertQC was able to translate in both directions with a fair degree of accuracy. There are still some untranslateable lines, but these were primarily due to certain functions not being translateable. For example, the aforementioned

[1]Which in itself is not a criticism of QuTiP - it's designed to be an academic level quantum toolkit

"with Control:" function requires a qubit to fall into a classical bit to act as a control bit. However with QuTiP there does not appear to be a method of measuring a qubit, so this could not be done.

3.1.7 Sprint 6: Monday 8th Apr - Sunday 14st Apr

At the beginning of the Easter holiday, it was decided to shorten the sprints to one week. It was felt that by doing this outstanding tasks could be prioritised so that additional features would be included only if necessary for the project.

This sprint was spent writing the technical report so development was paused. By the end of the sprint the report was mostly complete, and needed feedback from supervisors as well as expansion in some areas, particularly Testing and Evaluation.

3.1.8 Sprint 7: Monday 15th Apr - Sunday 21st Apr

By this sprint most of the code was nearing completion. To become a more complete tool it would require the ability to be uploaded to the Python Package Index (PyPI) where it could be easily installed by other users through the "pip" command. An online guide was followed which gave an overview [41].

First was to create the additional documents required. This includes a License, which as described in Section 2.7 is the GNU GPL v3.0 License. Additionally is the README file, which is written in reStructuredText (reST) [42] as this is the conventional way within the Python community. Finally was requirements.txt which could be parsed by the setup script to list the dependencies of ConvertQC, such as autopep8, QuTiP, and ProjectQ. Following this was creating the setup script, setup.py. This includes the details of the project such as description and author details, as well as directing to the packages and source files required.

Once all of these were created, the package could be submitted to PyPI and allow users to install it. Using this method, the script will be added to the user's PATH in a terminal, so rather than run the command using "python convertqc.py" and have to navigate to where the source code is stored, the user can run convertqc from any location.

Near the end of this sprint the python module ast was discovered. This handles abstract syntax trees (AST) which are a tree representation of the structure of source code, an example of which is shown in Figure 3.1. With this method, the ast library is able to easily identify which lines of code are associated with particular conditional statements, which parameters are associated with which variables, etc.

For example, in the aforementioned diagram, it is initially split into a while statement and return statement. The return node is simple and only links to the variable to return as its child note. The while is split between its comparison node, and the rest of the child notes which represent statements that fall under the control of the while loop. The clear link and node system shows clear assignment between different parts of a line of code.

This made parsing the QuTiP input formats easier as the parameters are each parsed

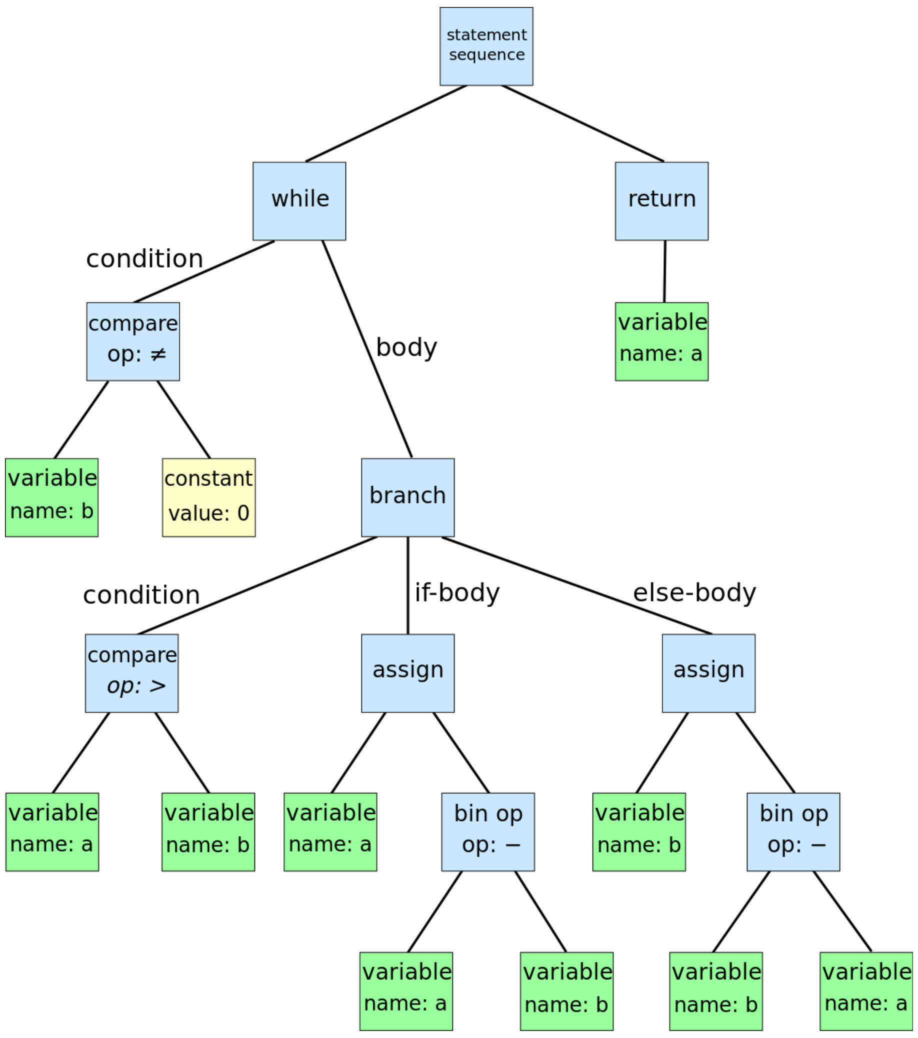


Figure 3.1: Example of Abstract Syntax Tree for code shown in Section C.7

and processed. Due to Python being a dynamically typed language, each variable does not have an implicit type, so when parsing lines this could not be relied on. However, ast has methods for determining a variable type, allowing for an extra level of defensive programming

Unfortunately, there was not enough time left within the project to learn the ast library and use it effectively. However, some basic tools were learnt and applied to ConvertQC. Consider the function call listed in Section C.8:

- To begin with, it checks every line of the input file for the string which creates the quantum circuit.
- If one is found, the line is parsed by the ast library.
- Once parsed, the tree is traversed by the Visitor class, as shown on line 12 and below. This is an override of the NodeVisitor class which allows particular tasks to happen when encountering a node of a particular type. In this instance the Call

function has been overridden to get the qubit count based on the first parameter. The first parameter is a Boolean choice but is optional so won't always be present.

- The first parameter is checked to see if it is a Number, as opposed to a Boolean. If so, this is the qubit count, and if not, the second parameter will be.
- Future development will include handling if neither is - this would be incorrectly written QuTiP code. There will likely be a user choice if they wish the line omitted or to cancel entirely.

There are issues with this method; primarily there are currently no checks to see if the qubit count had already been set. This would be most prominent in an instance where circuit definition is set as part of a set of "if" statements, where the number of qubits changes based on a condition. Additionally, there is no chance to expand the visit_Call override if there are other uses in the future, and this will require refactoring. However, for limited knowledge and time, this method was quick to implement and effective to ensure the correct qubit count was included.

The primary use of the ast library will be to effectively identify parameters to translate from QuTiP correctly. Additionally, ast can be used to get docstring information about the function it is a part of. This could be used in conjunction with parameters to ensure the correct ones are being extracted.

3.1.9 Sprint 8: Monday 22th Apr - Sunday 28st Apr

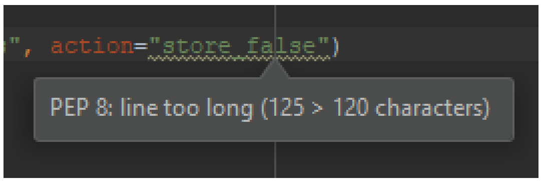
By this point in development the code was mostly complete at a functional level. However, as discussed in the evaluation, there is plenty left to do and a lot of improvement that could be made. Some of the tasks in this sprint included:

One of the main tasks was to clean up the debug printing statements, and insert debugging lines for future work. If work continues on ConvertQC and it were to be released to the public, a developer wouldn't be able to run the tool on their code to determine issues. As such, it's important to make sure the required debugging information is in place before release so the user can send this to be looked at by the development team and fix any issues. Additionally, other print statements were added to the verbose flag. With this, the user can see what is being done by the code as it runs. With this, they can potentially spot lines which have not been translated correctly or are not performing the expected function so they can manually change the output.

The other important task in this final sprint was cleaning up the code. This included refactoring to meet PEP 8 standards and inserting comments into the code to aid future development.

3.2 Standards

Following the standards was perhaps not one of the most important parts of the development processes - there are always tools to do it automatically or done as part of the



```
", action="store_false")
PEP 8: line too long (125 > 120 characters)
```

Figure 3.2: PEP 8 warnings shown in PyCharm

refactoring process - but it is at least something to be aware of. The main ones focused on were the PEP 8 [36] guidelines on code styling and the PEP 257 [43] guidelines on docstrings. These standards would ensure my docstrings would be detected by any automated tools, and that my code meets style standards.

Most of the PEP 8 requirements were already monitored in the IDE, PyCharm. This would flag lines that didn't match requirements, as shown in Figure 3.2. However, for peace of mind, the code was then processed by autopep8 [44], a Python tool to auto-format the code. This is the same tool that is run on input files before being processed by ConvertQC.

3.3 Refactoring

3.3.1 Mid-development

Refactoring was not efficiently done throughout the project. It was felt that refactoring could be done at the end of the project as needed. Had I been following a different agile methodology such as Test Driven Development then this would be a grievous deviation from the method and would likely have resulted in an inferior product.

However, there was a point when development on conversion from ProjectQ to QuTiP was nearing completion, but QuTiP to ProjectQ had not begun, that it became clear that a lot of methods are going to be repeated - for instance, those for reading & writing files or processing lines unable to be translated. As such, before beginning progress on the second conversion direction, development was paused and refactored to put the common methods into their own common Python file which could then be imported. It would have saved time had this been in place from the start, however stopping at this stage to refactor meant that it would save more time in the long run, especially if this project continues development to include other toolkits.

3.3.2 End of Development

Once development had completed, some last checks were done for comments and refactoring opportunities. Following this, the codebase was packaged into a .tar.gz file ready for submission.

Chapter 4

Testing

4.1 During Development

Testing of each function was manually performed at the end of each sprint. This was performed by passing example scripts through ConvertQC to ensure the tool is correctly translating the component that has been worked on. For instance, if functionality to convert a CNOT gate has been added, then that line would be manually checked for syntax, and then run using the appropriate tool to ensure it compiles using that.

4.2 Manual Testing

With this project a lot of functionality was tested manually. To do so, test plans were created which would cover the basic functionality of each script and work to cover each function in a test. These were each given a unique ID and task, as well as the input to test. Following this each test was run and the result recorded with a Pass or Fail mark. These test plans are included in Appendix E.

This testing required a lot of time which could be spent on development. However I was unfamiliar with Python unit tests, and ran into many issues as described in Section 4.3. As a result testing was still going to be required so a manually run test plan was decided to be the best way forward. With future work and development, each of these tests can look to be automated and added as part of a continuous integration package.

4.3 Unit Testing

These tests focus on the functionality of code, each testing a small portion and ensuring it is correct. For these, I used guidelines online [45] to ensure good quality unit tests. They were primarily used to test some inputs which could confuse my interpreter. Take the following statement:

```
circuit.add_gate("RX", 1, 2) # Add Pauli-X gate to Qubit 1
```

There are a number of pieces to this line to check have been translated correctly:

- Has the line itself been identified as a gate, as opposed to a function declaration or similar?
- The parameter "1" represents which qubit is being acted upon, while the parameter "2" shows how many radians of rotation should be applied. Have these been read correctly, and not switched?
- Has the end comment correctly been included? Has its presence caused the line to be copied verbatim, as normally happens with single-line comments, rather than translated?

The desired output of this translation would be

```
RX(2) | 1 # Add Pauli-X gate to Qubit 1
```

with "1" likely being changed to "qubit_one" or similar. For each unit test, I took a sample line as the aim for the test. This was then written to a file "test.py", and the ConvertQC tool called from the command line to translate this one line file. This would then output to the expected convertqc_results.py which would then be parsed by the test suite to ensure correct parameters and variables.

This was a fairly inefficient submission due to the amount of file opening and closing functions. Due to the current small scale of the tool it was deemed unnecessary to alter this. However, before developing to include other toolkits or other functionality, it would be beneficial to refactor these tests in such a way to minimise the amount of file input & output. This could potentially be done with mock classes, a feature of unit testing I was unable to explore fully within the deadline.

Additionally I was unable to test large portions of the code due to the way it had been written. For instance, take the function shown in Appendix **??**. This is a control function to identify the type of gate that is called when the ".add_gate()" method is detected, and then works to call appropriate methods based on which gate has been added. For example, identifying a rotation gate such as Pauli-X, Pauli-Y, or Pauli-Z will call a function to process this named "convert_rotation_gate()".

There are a number of variables used in this function which are not initialised when run by the test suite. In this example it is primarily the arguments which are usually provided by the user when running ConvertQC from the command line. However, these arguments are not set when this function is called by the test suite. To work around this, I attempted to recreate the function which handles the arguments in such a way that the variables are set at a default True or False value. However this still did not work, as the args variable is still registered as NoneType (Python's version of null). The alternative approach was to create a test-only copy of the file with every instance of args variables removed. This however did not seem the appropriate way to do so - the test file will not maintain parity with development, and is duplicating work unnecessarily. Unfortunately within the time constrictions I was unable to work around these issues, leaving little coverage of the code under automated unit testing.

4.4 Integration Testing

Due to ConvertQC essentially being a single script, integration testing was not a part of the test plan. The primary integration was to ensure the separate processing files are correctly imported to call the functions inside. During installation testing, described in 4.5, this proved to be a larger issue than anticipated but simple once resolved.

One minor test to run was the addition of a third fake package. This had the basic ability to be called by the main ConvertQC function, but did no data processing. By doing this I could test the ease of adding new toolkits in the future and see exactly where new parameters would need to be added. At its current state of development, all that was required was to add a new option to the allowed parameters (which is a list, rather than set for each parameter) and add new conditional "if" loops to the section which controls which processing function is called - altogether ten minutes of work.

This testing raised a major issue with the format of the code and how it handles outputs.

4.5 Installation Testing

Toward the end of development, the tool needed to be prepared as a package so it could be easily installed. This was done and sent to a clean install of Ubuntu 18.04 to ensure that my pre-installed modules and packages would not impact the tests.

The first issue arose with using the "pip install" function. Unbeknownst to me, the pip function only installs for Python2. However, the ConvertQC tool is designed to run in Python3, and has functions that are specific to Python3. To install the tool it was required to use pip3 instead. This test caused an update to the README file and required Python version in the setup script.

The second issue came from a lack of dependencies. The first set of bugs was in the setup.py file. Firstly, the project used a requirements.txt file to list the dependencies as this was the first instance that was found. However after more research, it was decided to list the dependencies directly in the setup file due to there being a relatively small number. Additionally, the dependencies are typically automatically installed by pip, including the dependencies required by this project's dependencies, which was also not happening. This issue remains unresolved, and the documentation was updated to alert the user to this problem, and directed them to the list of dependencies with their respective dependencies.

This set of testing identified the lack of scalability with the setup script. There is currently just one field which lists all the required packages. If other toolkits are added, it is reasonable to assume that the user may only want to use a subsection of them. In this instance, I would look to implement an optional dependency option which when paired with flags will install the user's choice of toolkits. The code would also need improving, as there are currently no proper checks for some of the dependencies. As a result, it may attempt to run code designed for QuTiP despite the user choosing not to install QuTiP as a dependency. Alternatively, a method of running ConvertQC without testing the converted script runs correctly could be implemented, to allow users to convert scripts without having toolkits installed.

4.6 Dependency Testing

The two primary dependencies for this project are QuTiP and ProjectQ. It would be ideal to ensure these are installed correctly as a part of ConvertQC. One instance of this could be a flag when running ConvertQC which runs a suite of unit tests designed to ensure elements of each toolkit. Additionally, QuTiP comes with its own method

```
import qutip.testing as qt
qt.run()
```

which will run its own unit test packages. How to ensure the user has their desired toolkits installed correctly could be included in the ConvertQC documentation.

4.7 Usability Testing

This wasn't so much a formal level of testing as it was ensuring the user was able to use the software with ease. To do this, some Computer Science and Physics students were given a Linux terminal which had ConvertQC pre-installed and a directory with an example ProjectQ script. They were given the instructions "This file is in a ProjectQ format. I need you to use the tool, ConvertQC, to translate this into QuTiP. The command to run the tool is 'convertqc'". These people were then let loose to figure out how to run the translation. The testers were provided with consent forms which detailled the aim of the project, how their data will be stored, and details of how to withdraw if they desire.

The Computer Science students found this a fairly simple task. On average, they started by running the command without any arguments to see what is returned, which was the screen shown in Figure 2.4. From this they were able to identify the "-h" flag, which is commonly known as help. The second command they ran was "convertqc –h" which then displayed the help window as shown in Figure 2.3. This listed the required arguments and the options allowed for the second and third arguments, meaning on their third attempt they were able to run the tool with no issues.

Physics students, however, found the process slightly more complex. It took several attempts at running the command and identifying the error messages before being able to successfully run the command. As a result I decided to amend the documentation to clearly list the arguments and how they were used when running the tool.

These error messages were unable, to the best of my knowledge, to be changed due to using the ArgParse library and many of them are auto-generated.

4.8 Stress Testing

It is unlikely that this tool was going to require heavy stress testing, due to being a script translator. However it is possible a future user may have long scripts or a large number, each of which need to be translated, which could potentially cause heavy stress. This test revealed that ConvertQC does not scale particularly well. A file was written to test this, which was simply

```
H | one
X | two
```

repeated 50,000 times (a total of 100,000 lines in the file. The tool was then run with the time measurement functionality of bash which measured at 9 minutes 39.599 seconds, a horrifically inefficient time. It is believed this is caused by the amount of memory required by reading the file into memory, causing slow CPU processing. This can be fixed in future development by parsing a file by line as opposed to storing the entirety into memory.

Chapter 5

Evaluation

5.1 Aim

As discussed in Chapter One, the aim of this project was to develop a tool that could convert programs written in different quantum computing toolkits into different output formats. For a one-sentence aim, this has been achieved, but not the scale I had envisioned when beginning this project. It can only convert between two toolkits, but can convert in either direction. The smaller scale of the project I feel has a few causes:

- Unforseen personal circumstances, as discussed during the Implementation section, caused a slowdown and delay in certain features being developed.
- I had definitely underestimated the amount of work that would go into parsing code. At the beginning of the project, concerns were shared with the project supervisor as to whether this tool would meet the technical difficulty requirements for a dissertation. All it took was listing off just some of the things that the code had to deal with for me to go from "This may not be difficult enough" to "My code needs to be incredibly robust and able to deal with unforseen inputs".

5.2 Conversion

One aspect where I feel the tool was not built to standards is on the scaling of output conversion. In its current format, the code is ran with convert_qc.py which then calls one of two scripts, process_projectq.py or process_qutip.py dependant on the input format. In these scripts, the output is directly converted into the other format. For example, if process_projectq.py has read the line "H | qubit_one"[1], this would be converted by the following code:

[1]This line would apply a Hadamard gate to qubit_one

```
    elif gate == "H":
        print(whitespace + "quantum_circuit.add_gate(\"snot\", "
         ↪  + str(get_qubit_from_list(split_line[2])) + ")",
            file=output_file)
```

This is the code that would be called any time the input format is ProjectQ. However, let's assume some future work is done which allows conversion into another language such as Quantum++. This code has not been built to scale - in its current format I would have to add extra if statements to every line which outputs to file to output a different format dependant on what the desired output format language is.

This could be solved with some refactoring in the future. Ideally, the functions to output code could be separated into their own functions, and then passed relevant parameters, such as "def output(format, gate, qubits [])" where the format is the output format set in arguments, the gate being the quantum gate that needs to be output, and qubits[] being a list of qubits being involved (this would only be a single qubit in many cases, and would have to be checked in other circumstances to ensure the qubits are input in the correct order - important for gates such as CNOT where the first qubit is the control). However, this realisation was made too late to perform ready for submission.

5.3 Process

The processes used as a part of this project worked well. By splitting the end goal into smaller stories which would be more manageable, and spreading these across iterations, it made a daunting task much more feasible. Additionally, testing had not been left to the last minute, and was added throughout the project. As a result, it was known that the code produced so far was in a working state, even if it is incomplete.

On the flip side, I feel that having a hybrid approach was incredibly beneficial. It allowed me to have an overall plan before progressing on, a plan which had been influenced by stories and development choices. This plan was much more flexible than in strictly plan-driven approaches such as Waterfall, so leaned more toward an agile approach, but not fully agile.

5.3.1 Scrum Values

The Scrum Values [29] were integral to my work method. Without them the Scrum method would have just been a set of steps, but with these values I felt I was able to work effectively, communicate effectively, and produce a higher quality product.

- *Focus - Everyone focuses on the work of the Sprint and the goals of the Scrum Team*
  My personal work method did not allow me to adhere to this value. I frequently jump between different parts of the work, both on the report and technical work. Whilst these were always tasks that were necessary for the product, they would frequently be tasks that were out of the sprint or in parts that had not started development yet.

Despite this all tasks were relevant to development of ConvertQC, so this value is partially fulfilled.

- *Openness - The Scrum Team and its stakeholders agree to be open about all the work and the challenges with performing the work*
  I was in contact with my supervisors throughout my project, and was able to explain my shortcomings and areas of struggle without fear of judgement or condescension. As such this value was maintained.

- *Respect - Scrum Team members respect each other to be capable, independent people*
  This is more of a value tailored toward group Scrum projects, so was not relevant. However, I maintained respect for my supervisors by meeting targets and holding regular meetings, and they held respect for me by not micromanaging and trusting me to achieve the work expected.

- *Courage - Scrum Team members have courage to do the right thing and work on tough problems*
  Throughout the development tough problems cropped up, primarily related to dealing with edge cases. Throughout these I was able to push through and continue development, sometimes with support and sometimes without, so had the courage to do so.

- *Commitment - People personally commit to achieving the goals of the Scrum Team*
  Commitment is a fairly integral part of a dissertation - a project as large as this requires continuous commitment and dedication to achieve the outlined goals. The commitment stretches to the entire degree, and to future work involving this project in progressing to study a doctorate in Quantum Information. Commitment has been shown both in the short- and long-term.

5.4 Documentation

I am very pleased with documentation as a part of this project. During my year in industry, I worked with a lot of poorly documented and commented code and worked hard to ensure that if somebody else were to take over my code they would not be left in a similar position. This is a principle I have worked hard to adhere to in this project. Each method contains a docstring comment which details its function, parameters, and what is returned. Additionally, the majority of non-clear lines are commented above with their functionality.

I was additionally commended on my technical writing abilities. This gave me the confidence to write a detailed report which included the necessary detail while not being too long-winded or rambling. Additionally it led to a more concise README file and clear help screen descriptions allowing the user to clearly understand how the program is used.

5.5 Testing

This is likely the weakest area of the project. The unit tests included are very basic and definitely do not cover the whole codebase. Additionally, as described in the previous chapter, they are remarkably inefficient with their implementation of file input and output. This is less than ideal, as it is not possible to ensure the code is working as expected without manually converting and checking the results.

Additionally, the tests which have been implemented need to be run manually as part of a test suite. An automatic testing process should have been used to ensure that as code is added it is not causing issues in other unknown areas of the code. With automatic testing, as a new test or function is added and committed to a repository, this set of tests would be run automatic. This is a feature of GitLab, but not one I had looked into when developing.

This is an issue that could potentially be resolved by using Test-Driven Development. With this methodology, the tests are written first, and then code to get the tests to pass is written afterwards. This would ensure the code is being tested throughout the project, rather than added as an afterthought.

5.6 Defensive Programming

I feel I spent too much of my time working on defensive programming. This is a practice that had been a major focus of my year in industry, so is something I naturally wanted to include. The code which has been included does a good job of defending against most issues - the user cannot translate to and from the same language, their custom filename is checked for any illegal characters, most parsed lines are checked thoroughly for potential issues.

However there is a point where implementing defensive methods causes other aspects of the work to fall behind which is what happened here. By implementing this level of checks, some functionality fell short or was not included. For example, ConvertQC can only convert between ProjectQ and QuTiP, whereas if less time had been spent on defensive programming then extra functionality could have been added.

5.7 Development Choices

The choices made for development as listed in Section 2.5 were, I feel, appropriately chosen for this project. Whilst they weren't important to the tool and its functionality, they were important to my ability to create the tool effectively.

The first was using Python as my language of choice. Despite my personal issues with it, such as whitespace requirements and dynamically typed variables, it is compact and easy to run. The toolkits converted between were both written in Python, as was the third choice eventually not developed. This meant it was easy to include these toolkits as Python modules, and easy to install using the pip command. The simple syntax and readability also meant that fewer comments were required and it was self-documenting.

Next is the version control system GitLab. This just worked - setup took five minutes and following this the code was easy to push to the cloud-based repository. My only regret is not further investigating the possibility of automated testing as part of the DevOps suite. This could have ensured my code is fully working as expected during development.

Finally was the PyCharm IDE. There isn't a lot to say, but it worked effectively and kept my code following PEP 8 standards. The debug functionality was also excellent, allowing me to manually insert breakpoints to test particular functions.

5.8 Project Management

Project management was definitely one of the weaker areas of my dissertation project. This included the areas of time management and prioritisation of tasks. These skills were improved over my industrial year, but still remained one of the more difficult aspects of the project.

However, as the project continued, this became easier and became a stronger skill. It became easier with more experience to be able to estimate how long a task will take.

5.9 Personal Reflection

This project has been an excellent experience from the beginning. The magnitude of it is beyond anything done previously in university and taught a number of skills both technical and non-technical. On a technical note, my programming skills have sharpened due to the large-scale nature of the project. Having to solve issues and find solutions myself as opposed to having team members to assist has meant that my research and critical thinking skills have become better. Tools I was inexperienced in previously such as git and LaTeXare also better, parts that are going to be useful when working in industry.

5.9.1 Next Time

There are a number of elements I would look to improve or do differently on projects whilst in employment:

- Testing was perhaps the weakest area, as described above, so I would look to strongly improve my testing methodology. I believe one way of doing this would be to adopt Test-Driven Development, the practice of writing a failing unit test, writing the minimum code to get the test to pass, and then refactoring the code to improve it. By doing this, it would ensure the tests are in place before beginning development and ensure the code is meeting the desired outcomes.

- More background research was required. I knew some of the background quantum information but was not as strong on the programming toolkits available as well as not being particularly strong coding in Python. Due to this there were some features that I was unaware of while developing, such as the ast library, which would have been rather useful during development. With some more research into available tools and the tools my code is interacting with, I feel my code could have been stronger and more robust.
- Working with a hybrid methodology had its benefits, but I feel not to the extent that I had hoped. Due to a large amount of planning done before development, I tended to skip over pre-iteration planning which led to issues down the line with buggy or missing features. Additionally I did not follow a number of the agile processes such as the end of sprint retrospective which I feel led to some of those issues being overlooked. Next time I would look to pick either a plan-driven or agile approach and embrace it fully.

This list is not exhaustive and there are likely a variety of smaller parts of the process which could have been better, but I feel these were the primary areas that could be improved.

5.10 Future Work

The clearest opportunity for future work is in the range of languages that can be supported. Adding a third toolkit to translate between would require a bit of refactoring work, as described earlier, but would be beneficial to allow the toolkit to expand. By separating out functions for outputting in the correct format, it would allow the tool to scale to as many languages as desired.

Additional work could include non-circuit based quantum tools. For example, QuTiP has tools for equation solving, visualising quantum states, etc. I have not heavily researched these areas, due to them not being relevant at the time of development, but feel that incorporating these tools could allow for ConvertQC to reach a wider audience and have greater uses for the community (though may need a re-branding!)

Some further work on parsing code would be beneficial. The tool works fairly well so far by running the input files through an automatic formatting tool to meet PEP 8 standards. This however will not catch every potential issue, so is not the most robust system. Future work can be done to improve the code parsing to be more reliable. With the late discovery of abstract syntax trees and the code available to handle them, they could be a powerful tool for this.

An important improvement would be in the tests. They need to cover a much larger portion of the codebase as well as be much more efficient to allow the tool develop. To do this, a lot more research into the default `unittest` package is required to find extra functionality and utilise it effectively. Alternatively, looking into other Python unit test libraries could be beneficial for offering tools that could be more suitable.

Finally, some more understanding of how each component works in the quantum toolkit would be beneficial to outputting. For instance, there are some functions in ProjectQ such as "with Control:" or "with Dagger:" that I was not able to convert into a QuTiP format due to not understanding their function, so could not find their equivalent in other languages. By understanding their functions, I could write my own code to convert to, meaning a fuller translation of the input file.

Annotated Bibliography

[1] Wikipedia, the free encyclopedia, “Bloch sphere,” 2019, [Online; accessed April 08, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/File:Bloch_sphere.svg

> Diagram of a Bloch sphere. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license [46]

[2] ProjectQ, “Examples.” [Online]. Available: https://projectq.readthedocs.io/en/latest/examples.html

> Diagram for a Quantum Teleportation circuit. Taken from the ProjectQ Examples page, and used under the Apache 2.0 License [47]

[3] Wikipedia, the free encyclopedia, “H gate circuit diagram,” 2019, [Online; accessed April 09, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/File:Hadamard_gate.svg

> H Gate on a quantum circuit diagram. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license [46]

[4] Wikipedia, the free encyclopedia, “X gate circuit diagram,” 2019, [Online; accessed April 09, 2019]. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=41692959

> X Gate on a quantum circuit diagram. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license [46]. Modification - cropped at the sides.

[5] Wikipedia, the free encyclopedia, “Y gate circuit diagram,” 2019, [Online; accessed April 09, 2019]. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=41692987

> Y Gate on a quantum circuit diagram. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license [46]. Modification - cropped at the sides.

[6] Wikipedia, the free encyclopedia, “Z gate circuit diagram,” 2019, [Online; accessed April 09, 2019]. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=41693041

> Z Gate on a quantum circuit diagram. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license [46]. Modification - cropped at the sides.

[7] Wikipedia, the free encyclopedia, “Cnot gate circuit diagram,” 2019, [Online; accessed April 09, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/File:CNOT_gate.svg

CNOT Gate on a quantum circuit diagram. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license [46]

[8] R. LaRose, “Overview and Comparison of Gate Level Quantum Software Platforms,” *Quantum*, vol. 3, p. 130, Mar. 2019. [Online]. Available: https://doi.org/10.22331/q-2019-03-25-130

Paper comparing four Quantum Computing toolkits, three of which were considered for use in this project

[9] Erwin Schrödinger, translated by John D. Trimmer, “The present situation in quantum mechanics: A translation of schrödinger’s "cat paradox" paper,” *Proceedings of the American Philosophical Society*, vol. 124, no. 5, pp. 323–338, 1980. [Online]. Available: http://www.jstor.org/stable/986572

A translation of the original paper which Schrödinger wrote as a dismissal of quantum mechanics. Not hugely important to this dissertation, but included as an example.

[10] M. Niaz, *Critical appraisal of physical science as a human enterprise dynamics of scientific progress*. Springer, 2009.

Specifically Chapter 12, focuses on the wave-particle duality experiments.

[11] J. S. Kilby, “Turning potential into realities: The invention of the integrated circuit,” *International Journal of Modern Physics B*, vol. 16, no. 05, pp. 699–710, Aug 2001.

A Nobel lecture from Jack Kilby, inventor of the first integrated circuit.

[12] J. P. Dowling and G. J. Milburn, “Quantum technology: the second quantum revolution,” *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 361, no. 1809, pp. 1655–1674, 2003.

A slightly dated article, but one which correctly predicted the current obsession with quantum information and technologies.

[13] T. Beardon, “Public key cryptography,” 2004. [Online]. Available: https://nrich.maths.org/2200

An overview of Public Key Cryptography

[14] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann, “Factorization of a 768-bit rsa modulus,” Cryptology ePrint Archive, Report 2010/006, 2010, https://eprint.iacr.org/2010/006.

Paper about breaking of RSA-768

[15] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, Nov 1994.

Paper detailing Shor’s algorithm on quantum computers to factorise integers in polynomial time.

[16] A. Raudaschl, “Quantum computing and health care,” *British Medical Journal Technology Blog*, Nov 2017. [Online]. Available: https://blogs.bmj.com/technology/2017/11/03/quantum-computing-and-health-care/

A blog article in the British Medical Journal about the advances quantum computing can make in healthcare.

[17] W. Hurley, “Quantum computers could end climate change debate, says expert,” Mar 2018. [Online]. Available: https://www.inverse.com/article/42244-sxsw-2018-quantum-computers-end-climate-change-debate

William Hurley’s keynote speech at SXSW 2018 to talk about quantum simulations of climate change prevention strategies

[18] G. Satell, “Here’s how quantum computing will change the world,” Oct 2016. [Online]. Available: https://www.forbes.com/sites/gregsatell/2016/10/02/heres-how-quantum-computing-will-change-the-world/

Forbes article about quantum computing impacting real world logistics

[19] S. Castellanos, “Vw nears commercialization of ’quantum routing’ technique,” Nov 2018. [Online]. Available: https://blogs.wsj.com/cio/2018/11/21/vw-nears-commercialization-of-quantum-routing-technique/

Wall Street Journal article about Volkswagen experimenting with quantum computers for traffic routing.

[20] ATOS, “How will quantum information benefit the world?” 2018. [Online]. Available: https://atos.net/content/mini-sites/ascent-magazine-2018/quantum.html

A panel inteview mixed with article on benefits of quantum computing and the future.

[21] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th ed. New York, NY, USA: Cambridge University Press, 2011, 1107002176, 9781107002173.

Considered the primary textbook on quantum information and quantum computing. Provides a thorough explanation of all aspects of quantum information from a more theoretical point of view, without delving into the specifics of practicality.

[22] W. Knight, “IBM announces a trailblazing quantum machine,” Nov 2017. [Online]. Available: https://www.technologyreview.com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/

IBM's recent Quantum Computer

[23] S. Curtis, "Google aims for quantum supremacy," Mar 2018. [Online]. Available: https://physicsworld.com/a/google-aims-for-quantum-supremacy/

Google's recent Quantum Computer

[24] T. Greene, "Understanding quantum computers: The noise problem," Oct 2018. [Online]. Available: https://thenextweb.com/artificial-intelligence/2018/10/25/understanding-quantum-computers-the-noise-problem/

A simple and well-written article on the problem of quantum noise

[25] E. Mendelson, *Beginning calculus*. McGraw-Hill, 2010.

Introductory guide to calculus, which covers the probability amplitudes and complex numbers

[26] J. Johansson, P. Nation, and F. Nori, "Qutip: An open-source python framework for the dynamics of open quantum systems," *Computer Physics Communications*, vol. 183, no. 8, pp. 1760–1772, 2012.

The initial paper introducing QuTiP

[27] J. Johansson, P. Nation, and F. Nori, "Qutip 2: A python framework for the dynamics of open quantum systems," *Computer Physics Communications*, vol. 184, no. 4, pp. 1234–1240, 2013.

A follow-up paper to the QuTiP paper ([26]) with developments and improvements

[28] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: an open source software framework for quantum computing," *Quantum*, vol. 2, p. 49, Jan. 2018. [Online]. Available: https://doi.org/10.22331/q-2018-01-31-49

[29] The Scrum Alliance, "Scrum values." [Online]. Available: https://www.scrumalliance.org/learn-about-scrum/scrum-values

The values of Scrum, my development methodology

[30] "Ibm q - quantum computing," Jun 2018. [Online]. Available: https://www.research.ibm.com/ibm-q/

IBM's cloud-based system for quantum simulatio n and running algorithms on quantum chips

[31] "Gitlab - about." [Online]. Available: https://about.gitlab.com/

GitLab, a full DevOps tool but in this instance used only for its version control

[32] "Overleaf, online latex editor." [Online]. Available: https://www.overleaf.com/

Overleaf, an online LaTeX editor with built-in PDF display

[33] "Pycharm: the python ide for professional developers by jetbrains." [Online]. Available: https://www.jetbrains.com/pycharm/

The PyCharm IDE which was used for all of the development work.

[34] "Gnu gpl v3.0 license." [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html

GNU GPL v3.0 License. Allows near complete freedom provided the work produced includes the same license.

[35] "draw.io diagram maker." [Online]. Available: https://www.draw.io/

draw.io, free online tool primarily used for flow control diagrams, but was suitable for a GUI mockup.

[36] "Pep 8 – style guide for python code," Jul 2001. [Online]. Available: https://www.python.org/dev/peps/pep-0008/

PEP 8 Guidelines for Python Code

[37] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernádez, J. Carballo-Franquis, A. Chen, C.-F. Chen, J. M. Chow, A. D. Córcoles-Gonzales, A. J. Cross, A. Cross, J. Cruz-Benito, C. Culver, S. D. L. P. González, E. D. L. Torre, D. Ding, E. Dumitrescu, I. Duran, P. Eendebak, M. Everitt, I. F. Sertage, A. Frisch, A. Fuhrer, J. Gambetta, B. G. Gago, J. Gomez-Mosquera, D. Greenberg, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, H. Horii, S. Hu, T. Imamichi, T. Itoko, A. Javadi-Abhari, N. Kanazawa, A. Karazeev, K. Krsulich, P. Liu, Y. Luh, Y. Maeng, M. Marques, F. J. Martín-Fernández, D. T. McClure, D. McKay, S. Meesala, A. Mezzacapo, N. Moll, D. M. Rodríguez, G. Nannicini, P. Nation, P. Ollitrault, L. J. O'Riordan, H. Paik, J. Pérez, A. Phan, M. Pistoia, V. Prutyanov, M. Reuter, J. Rice, A. R. Davila, R. H. P. Rudy, M. Ryu, N. Sathaye, C. Schnabel, E. Schoute, K. Setia, Y. Shi, A. Silva, Y. Siraichi, S. Sivarajah, J. A. Smolin, M. Soeken, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, K. Trabing, M. Treinish, W. Turner, D. Vogt-Lee, C. Vuillot, J. A. Wildstrom, J. Wilson, E. Winston, C. Wood, S. Wood, S. Wörner, I. Y. Akhalwaya, and C. Zoufal, "Qiskit: An open-source framework for quantum computing," 2019.

Qiskit, IBM's quantum framework. I know it's a lot of authors, but this Bibtex entry is how it is provided by Qiskit so I did not want to edit it.

[38] QuantumWriter, "The q# programming language." [Online]. Available: https://docs.microsoft.com/en-us/quantum/language/

Q#, Microsoft's quantum framework included as part of the Quantum Development Kit

[39] V. Gheorghiu, "Quantum++ : A modern c++ quantum computing library," *Plos One*, vol. 13, no. 12, 2018.

The paper releasing the C++ header-based quantum computing library Quantum++

[40] “Projectq ops.” [Online]. Available: https://projectq.readthedocs.io/en/latest/projectq.ops.html

The documentation page for ProjectQ which lists the available gates

[41] T. Ziad{’e, “Packaging quick start,” 2009. [Online]. Available: https://the-hitchhikers-guide-to-packaging.readthedocs.io/en/latest/quickstart.html

Online quick-start guide to creating packages for Python

[42] “restructuredtext,” May 2016. [Online]. Available: http://docutils.sourceforge.net/rst.html

Guide for writing in reStructuredText.

[43] “Pep 257 – docstring conventions,” May 2001. [Online]. Available: https://www.python.org/dev/peps/pep-0257/

PEP 257 Guidelines for Docstrings

[44] “autopep8.” [Online]. Available: https://pypi.org/project/autopep8/

autopep8, a tool to auto-format Python scripts to follow PEP 8 Guidelines

[45] “Testing your code - the hitchhiker’s guide to python.” [Online]. Available: https://docs.python-guide.org/writing/tests/

A guide to writing tests for Python

[46] C. Commons. [Online]. Available: https://creativecommons.org/licenses/by-sa/3.0/legalcode

The Creative Commons Attribution-Share Alike 3.0 Unported license. In short, I am free to share and adapt the source provided I give appropriate credit, link to the license, and list any changes made. If I reshare it must be under the same license.

[47] Apache, “Apache license,” 2004. [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0.txt

Apache 2.0 License. Allows share, usage including commercial, provided I include copyright, license, and changes.

[48] “The MIT License.” [Online]. Available: https://opensource.org/licenses/MIT

MIT License. Allows complete freedom.

[49] “The 3-clause bsd license.” [Online]. Available: https://opensource.org/licenses/BSD-3-Clause

Modified BSD License. Allows near complete freedom provided the work produced includes the same license.

Appendices

The appendices are for additional content that is useful to support the discussion in the report. It is material that is not necessarily needed in the body of the report, but its inclusion in the appendices makes it easy to access.

For example, if you have developed a Design Specification document as part of a plan-driven approach for the project, then it would be appropriate to include that document as an appendix. In the body of your report you would highlight the most interesting aspects of the design, referring your reader to the full specification for further detail.

If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification. Perhaps you use stories to keep track of the functionality and the 'future conversations'. It might not be relevant to include all of those in the body of your report. Instead, you might include those in an appendix.

There is a balance to be struck between what is relevant to include in the body of your report and whether additional supporting evidence is appropriate in the appendices. Speak to your supervisor or the module coordinator if you have questions about this.

Appendix A

Third-Party Code and Libraries

- autopep8 - a tool to automatically format python scripts to follow PEP 8 guidelines. Version 1.4.4 was used. The library is open source, available for download using 'pip' or cloned from GitHub [44]. This library is released using the MIT License [48]. This library was used without modification.
- ProjectQ - a framework for quantum computing, including simulation of quantum circuits. Version 0.4.2 was used. The library is open source, and is available for download from GitHub. This library is released using the Apache 2.0 License [47]. This library was used without modification.
- QuTiP - a python toolkit for quantum mechanics & computing, including simulation of quantum circuits. Version 4.3.1 was used. The library is open source, available for download using 'pip' or cloned from GitHub [26] [27]. This library is released using the 3-Clause BSD License [49]. This library was used without modification.

Appendix B

Ethics Submission

Below I have directly embedded the email copy of my Ethics Submission.

20/02/2019

For your information, please find below a copy of your recently completed online ethics assessment

Next steps

Please refer to the email accompanying this attachment for details on the correct ethical approval route for this project. You should also review the content below for any ethical issues which have been flagged for your attention

Staff research - if you have completed this assessment for a grant application, you are not required to obtain approval until you have received confirmation that the grant has been awarded.

Please remember that collection must not commence until approval has been confirmed.

In case of any further queries, please visit www.aber.ac.uk/ethics or contact ethics@aber.ac.uk quoting reference number **12218**.

Assesment Details

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
haa14@aber.ac.uk

Full Name
Harry Adams

Please enter the name of the person responsible for reviewing your assessment.
Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department

cs
Module code (Only enter if you have been asked to do so)

Proposed Study Title
Mapping Quantum Computing Circuits Between Different Simulation Environments

Proposed Start Date
01/02/19

Proposed Completion Date
03/05/19

Are you conducting a quantitative or qualitative research project?
Qualitative

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
Translating a program to write a circuit between various programming languages

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Yes

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here:

Appendix C

Code Examples

C.1 Example ProjectQ Script

ProjectQ script for a quantum random number generator.

```
from projectq.ops import H, Measure
from projectq import MainEngine

# create a main compiler engine
eng = MainEngine()

# allocate one qubit
q1 = eng.allocate_qubit()

# put it in superposition
H | q1

# measure
Measure | q1

eng.flush()
# print the result:
print("Measured: {}".format(int(q1)))
```

C.2 Example QuTiP Script

QuTiP script for a quantum random number generator.

```python
from qutip import *

# create the circuit object
circuit = QuantumCircuit(1)

# add a Hadamard gate
circuit.add_gate("snot")

# unsure how to measure/print
```

C.3 ArgParse

This code details the argument handling by ArgParse

```python
def process_args():
    global args

    parser = argparse.ArgumentParser(description="Convert quantum circuits into different environments")
    parser.add_argument("input_filename", help="input filename")
    parser.add_argument("input_format", help="input format of your script", choices=valid_program_types)
    parser.add_argument("output_format", help="output format of your script", choices=valid_program_types)
    parser.add_argument("-o", "--output_filename", help="set output filename, without file extension (default: convertqc_result.<filetype>")
    parser.add_argument("-d", "--debug", help="enable debug mode", action="store_true")

    args = parser.parse_args()
```

C.4 Processing Entry Function

This is the overall function for each conversion direction, which is called by ConvertQC. It processes the full input, conversion, and output.

```python
def process_projectq(input_args, output_filename):
    global args
    global filename
    global output_file
    global file_as_list
    args = input_args
    filename = output_filename
    if args.debug:
        print("Processing ProjectQ input file...")

    output_file = conversion.open_output_file(filename)
    file_as_list =
     ↪ conversion.read_input_file(args.input_filename)
    setup_output(check_for_qubits())
    convert_script()
    conversion.close_output_file(output_file)
    if args.output_error_log:
        conversion.output_error_log(error_lines)
```

C.5 Error Handling

This script is the error handling for ConvertQC, and contains functionality for fatal and non-fatal errors as well as a list of error codes.

```python
#ERROR_CODES
INPUT_FILE_NOT_FOUND = 1
OUTPUT_FILENAME_INVALID_CHARACTER = 2
MATCHING_INPUT_OUTPUT = 3

forbidden_filename_chars = ['/', '<', '>', ':', '"', '\\', '|',
 ↪ '.', '?', '*']


def process_error(code, is_fatal):
    switch = {
        1: "Input file not found",
        2: "Output filename contains invalid characters",
        3: "Input file format matches output file format"
    }
    print_error(code, switch.get(code), is_fatal)


def print_error(code, error_msg, is_fatal):
    if is_fatal:
        print ("FATAL ", end='')
    print("ERROR:")
    print("    Code: " + str(code))
    print("    Message: " + str(error_msg))
    if is_fatal:
        print("\nExiting...")
        exit(code)
```

C.6 Process ProjectQ - Gate Output

```python
def rotation_gate(whitespace, orientation, qubit, angle):
    if orientation == "X":
        print(whitespace + "quantum_circuit.add_gate(\"RX\", " +
         ↪ str(qubit) + ", " + str(angle) + ")",
         ↪ file=output_file)
    elif orientation == "Y":
        print(whitespace + "quantum_circuit.add_gate(\"RY\", " +
         ↪ str(qubit) + ", " + str(angle) + ")",
         ↪ file=output_file)
    elif orientation == "Z":
        print(whitespace + "quantum_circuit.add_gate(\"RZ\", " +
         ↪ str(qubit) + ", " + str(angle) + ")",
         ↪ file=output_file)
    else:
        print("Failed to print rotation gate")
```

C.7 Abstract Syntax Tree Diagram Code

The code represented by the Abstract Syntax Tree diagram in Figure 3.1. Not a hugely useful function, but works as an example.

```
while b != 0
  if a > b
    a = a - b
  else
    b = b - a
return a
```

C.8 Identifying Argument Parameters

**TODO - Explain

```
for line in file_as_list:
    if " = QubitCircuit(" in line:
        new_line = line.split()
        circuit_name = new_line[0]

        visit = Visitor()
        tree = ast.parse(line)
        visit.visit(tree)

    ...

class Visitor(ast.NodeVisitor):
    def visit_Call(self, node):
        first_arg =
        if isinstance(node.args[0], ast.Num):
            set_qubit_count(node.args[0])
        elif isinstance(node.args[1], ast.Num):
            set_qubit_count(node.args[1])
        else:
            # Incorrectly formatted script
            # TODO
```

C.9 Convert Gate Function (Testing Evaluation)

This section of code is critically analysed during the Testing section, so has been provided in its entirity for clarity.

```python
def process_gate(line):
    """
    Translates gates into ProjectQ format
    TODO
    :param line: Line to translate
    :return: None
    """

    # Extract gate from parameters
    gate = line[line.find("(\"")+2:line.find("\",")]
    if args.debug:
        conversion.debug_print(current_line_no, "Gate found: " +
        ↪  gate)

    # CNOT Gate
    if gate == "CNOT":
        return convert_cnot_gate(line)
    # Rotation Gate
    elif gate[0] == "R" and (gate[1] == "X" or "Y" or "Z"):
        return convert_rotation_gate(line, gate[1])
    # Hadamard Gate
    elif gate == "SNOT":
        if args.verbose:
            conversion.verbose_print(current_line_no,
            ↪  "Converting to Hadamard gate")
        qubit = line[line.find(",")+1:line.find(")")].strip()
        print(conversion.get_leading_spaces(line) +  "H | " +
        ↪  get_named_qubit_from_position(qubit),
        ↪  file=output_file)
        return True
    # SWAP Gate
    elif gate == "SWAP":
        return convert_swap_gate(line)
    elif gate == "SQRTSWAP":
        return convert_swap_gate(line, sqrt=True)
    # ISWAP Gate
    elif gate == "ISWAP":
        if args.verbose:
            conversion.verbose_print(current_line_no, "Could not
            ↪  translate ISWAP Gate - No Known ProjectQ
            ↪  Version")
        if args.mark:
            print("# Could not translate ISWAP Gate - No Known
             ↪  ProjectQ Version", file=output_file)
            return True
        else:
            return False
    elif gate == "CPHASE":
        return convert_cphase_gate(line)
    else:
        if args.verbose:
            conversion.verbose_print(current_line_no, "Unsure
             ↪  how to translate line. Copying verbatim")
        return False
```

Appendix D

ConvertQC README

Below is embedded a PDF copy of the README file for ConvertQC, followed by the raw ReStructuredText format

ConvertQC

A Python command-line tool to convert quantum computing scripts between different toolkits (currently only supporting QuTiP and ProjectQ).

Author - Harry Adams

Date - 03/05/19

Email - convertqc@gmail.com

Dependencies

- qutip (Dependencies - http://qutip.org/docs/4.1/installation.html)
- projectq
- autopep8

Installation

- Download the source repository (GitHub link to add once released).
- Navigate to the root directory of the project:

  `cd convertqc`
- Run `pip3 install -e .` to install

Running

- `convertqc <input_filename> <input_format> <output_format>`
- Formats: projectq, qutip

Examples

- To display the help window

  ```
  convertqc -h
  ```

- To translate a file (`example.py`) from ProjectQ to QuTiP

  ```
  convertqc example.py projectq qutip
  ```

- To translate a file (`example.py`) from QuTiP to ProjectQ

  ```
  convertqc examply.py qutip projectq
  ```

- To specify an output filename, use the -f flag (without the file extension)

  ```
  convertqc example.py -f output_file projectq qutip
  ```

- To disable comments in code which identify untranslated lines, use the -m flag

  ```
  convertqc example.py -m projectq qutip
  ```

File Structure

- Current directory: Root directory which contains the license, readme, and setup script.
- /convertqc/: Contains the scripts for running ConvertQC. Copied to user directory when installed using above method.
- /stress/: Contains files with large numbers of repetitive lines to be used as part of stress testing
- /test/: Contains test scripts which can be run using the command

  ```
  python3 -m unittest
  ```

License

This software is shared under the GNU GPL v3.0 License. Please view the LICENSE.txt file to view.

D.1 ReStructuredText

```
=========
ConvertQC
=========

A Python command-line tool to convert quantum computing scripts
 ↪ between different toolkits (currently only supporting QuTiP
 ↪ and ProjectQ).

Author - Harry Adams

Date - 03/05/19

Email - convertqc@gmail.com

Dependencies
------------

* qutip (Dependencies -
 ↪ http://qutip.org/docs/4.1/installation.html)
* projectq
* autopep8

Installation
------------

* Download the source repository (GitHub link to add once
 ↪ released).
* Navigate to the root directory of the project:

  ``cd convertqc``
* Run ``pip3 install -e .`` to install

Running
-------

* ``convertqc <input_filename> <input_format> <output_format>``

* Formats: projectq, qutip

Examples
--------

* To display the help window

    ``convertqc -h``
```

```

* To translate a file (``example.py``) from ProjectQ to QuTiP

    ``convertqc example.py projectq qutip``

* To translate a file (``example.py``) from QuTiP to ProjectQ

    ``convertqc examply.py qutip projectq``

* To specify an output filename, use the -f flag (without the
  ↪ file extension)

    ``convertqc example.py -f output_file projectq qutip``

* To disable comments in code which identify untranslated lines,
  ↪ use the -m flag

    ``convertqc example.py -m projectq qutip``


File Structure
--------------

* Current directory: Root directory which contains the license,
  ↪ readme, and setup script.

* /convertqc/: Contains the scripts for running ConvertQC.
  ↪ Copied to user directory when installed using above method.

* /stress/: Contains files with large numbers of repetitive
  ↪ lines to be used as part of stress testing

* /test/: Contains test scripts which can be run using the
  ↪ command

    ``python3 -m unittest``

License
-------

This software is shared under the GNU GPL v3.0 License. Please
  ↪ view the LICENSE.txt file to view.
```

Appendix E

Manual Test Plans

Below are the manual test plans which were created as part of the manual testing of this project. This is not a complete list of possible tests.

ID	Test	Input	Expected Output	Success?
1.1	Input Filename which cannot be found causes a fatal error	convertqc not_a_real_file.py projectq qutip	Fatal Error code 1	Pass
1.1	Output Filename with invalid characters causes a fatal error	conversion.py -f example/fake.file projectq qutip	Fatal Error code 2	Pass
1.2	Input Format matching Output Format causes a fatal error	convertqc conversion.py projectq projectq	Fatal Error code 3	Pass
1.2	Input Format matching Output Format causes a fatal error	convertqc conversion.py qutip qutip	Fatal Error code 3	Pass

Table E.1: Test Plan - error_cqc.py

ID	Test	Input	Expected Output	Success?
2.1.1	Can convert Pauli-X gate	X | qubit_one	circuit.add_gate("RX", 1, pi)	Pass
2.1.2	Can convert X Rotation gate	Rx(1.21) | qubit_one	circuit.add_gate("RX", 1, 1.21)	Pass
2.2.1	Can convert Pauli-y gate	Y | qubit_one	circuit.add_gate("RY", 1, pi)	Pass
2.2.2	Can convert Y Rotation gate	Ry(1.21) | qubit_one	circuit.add_gate("RY", 1, 1.21)	Pass
2.3.1	Can convert Pauli-Z gate	Z | qubit_one	circuit.add_gate("RZ", 1, pi)	Pass
2.3.2	Can convert Z Rotation gate	Rz(1.21) | qubit_one	circuit.add_gate("RZ", 1, 1.21)	Pass
2.4	Can convert Hadamard gate	H | qubit_one	circuit.add_gate("SNOT", 1)	Pass
2.5.1	Can convert CNOT gate on two adjacent qubits, lowest first	CNOT | (qubit_one, qubit_two)	circuit.add_gate("CNOT", 1, 2)	Pass
2.5.2	Can convert CNOT gate on two adjacent qubits, highest first	CNOT | (qubit_two, qubit_one)	circuit.add_gate("CNOT", 2, 1)	Pass
2.5.3	Can convert CNOT gate on two non-adjacent qubits, highest first	CNOT | (qubit_zero, qubit_two)	circuit.add_gate("CNOT", 0, 2)	Pass
2.5.4	Can convert CNOT gate on two non-adjacent qubits, highest first	CNOT | (qubit_two, qubit_zero)	circuit.add_gate("CNOT", 2, 0)	Pass

Table E.2: Test Plan - process_projectq.py

ID	Test	Input	Expected Output	Success?
3.1.1	Can convert Pauli-X gate	circuit.add_gate("RX", 1, pi)	X | qubit_one	Pass
3.1.2	Can convert X Rotation gate	circuit.add_gate("RX", 1, 1.21)	Rx(1.21) | qubit_one	Pass
3.2.1	Can convert Pauli-y gate	circuit.add_gate("RY", 1, pi)	Y | qubit_one	Pass
3.2.2	Can convert Y Rotation gate	circuit.add_gate("RY", 1, 1.21)	Ry(1.21) | qubit_one	Pass
3.3.1	Can convert Pauli-Z gate	circuit.add_gate("RZ", 1, pi)	Z | qubit_one	Pass
3.3.2	Can convert Z Rotation gate	circuit.add_gate("RZ", 1, 1.21)	Rz(1.21) | qubit_one	Pass
3.4	Can convert Hadamard gate	circuit.add_gate("SNOT", 1)	H | qubit_one	Pass
3.5.1	Can convert CNOT gate on two adjacent qubits, lowest first	circuit.add_gate("CNOT", 1, 2)	CNOT | (qubit_one, qubit_two)	Pass
3.5.2	Can convert CNOT gate on two adjacent qubits, highest first	circuit.add_gate("CNOT", 2, 1)	CNOT | (qubit_two, qubit_one)	Pass
3.5.3	Can convert CNOT gate on two non-adjacent qubits, highest first	circuit.add_gate("CNOT", 0, 2)	CNOT | (qubit_zero, qubit_two)	Pass
3.5.4	Can convert CNOT gate on two non-adjacent qubits, highest first	circuit.add_gate("CNOT", 2, 0)	CNOT | (qubit_two, qubit_zero)	Pass

Table E.3: Test Plan - process_qutip.py

ID	Test	Input	Expected Output	Success?
4.1.1	ConvertQC can translate a valid ProjectQ file	convertqc example_projectq.py projectq qutip	Syntactically identical QuTiP file	Pass
4.1.2	ConvertQC can translate a valid QuTiP file	convertqc example_qutip.py qutip projectq	Syntactically identical ProjectQ file	Pass
4.2.1	No text is output to the command line when run without the debug or verbose flag	convertqc conversion.py projectq qutip	No text output on command line	Pass
4.2.2	Only verbose output (no debug) is output to command line when run with the verbsoe flag	convertqc conversion.py -v projectq qutip	Line number, followed by a verbose comment for each non-empty line in the file	Pass
4.2.3	Verbose and Debug output on command line when run with the debug flag	convertqc conversion.py -d projectq qutip	Line number, followed by a verbose or debug comment for each non-empty line in the file	Pass
4.2.4	Empty lines in input file do not create any command line output with verbose flag	convertqc conversion.py -v projectq qutip	Only non-empty lines appear in verbose output	Pass
4.2.5	Empty lines in input file do not create any command line output with debug flag	convertqc conversion.py -d projectq qutip	Non-empty lines only appear in debug or verbose output	Pass

Table E.4: Test Plan - convertqc.py

Appendix F

Version History

Version	Type	Changes
0.1	Draft	Initial unfinished draft. Includes: • Completed all front matter, except Acknowledgements • Started: Chapter 1 - Background knowledge and process • Started: Chapter 2 - User Interface
0.2	Draft	• Finished Chapter 1 • More progress on Chapter 2 • Chapter 3 began, sprints until this point fairly comprehensive • Chapter 5 approximately mid-way complete
0.21	Draft	Various small changes, version sent to other people for grammar checks.
0.3	Draft	Completed report, send to supervisor for checks.
0.31	Draft	Minor syntax / content changes in response to feedback.
0.4	Draft	Manual Test Plans added
1.0	Release	Initial Release

Table F.1: Version History