From code to optical pulses: compiling QASM on a neutral atom quantum computer

Final Thesis

Author: Xavier Spronken (i6225376)

x.spronken@student.maastrichtuniversity.nl

Supervisor: Claire Blackman

Total Word Count: 3251

Abstract

OpenQASM is a low-level programming language for quantum computing, which can be adapted to any quantum computer. This paper is a preliminary study on the feasibility of writing an OpenQASM compiler for a neutral atom quantum computer. The code base required to parse QASM code was defined. Following this, the ability to translate both single-qubit and multi-qubit quantum logic gates to physical optical pulses within the computer was investigated using Pascal's simulator library: Pulser. Finally, the constraints imposed by multi-qubit gates on the creation of a qubit regirster in Pulser were also identified. Overall this study determines that creating a compiler for OpenQASM is feasible, as long as a way to implement the Phase gate is found.

Contents

1 Introduction

1.1 Quantum Computers

While quantum computers can be used for analog computation or simulating quantum systems[1], it is also possible to make them perform quantum information processing (QIP). The carriers of information in QIP are qubits. These differ from classical qubits in that they make use of quantum superposition. Thus a qubit's state (either $|1\rangle$ or $|0\rangle$) will remain in superposition until measurement. Prior to this, the qubit is said to be in a superposition of both states, where the probability of measuring either one of those states is $\mathrm{P}(|1\rangle) + \mathrm{P}(|0\rangle) = 1$[2]. Mathematicaly a qubit is represented as:

$$|a\rangle = \alpha|1\rangle + \beta|0\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}, \text{ where } |\alpha|^2 + |\beta|^2 = P(|1\rangle) + P(|0\rangle) = 1$$

A method to view the state of a qubit before measuring is the Bloch sphere. In this representation, the $|0\rangle$ state and $|1\rangle$ state are set to the north and south poles of the sphere respectively[3]. In addition to the probability of getting either state upon measurement, the Bloch sphere shows the phase of the qubit due to the latitudinal direction of the arrow in the sphere[4].

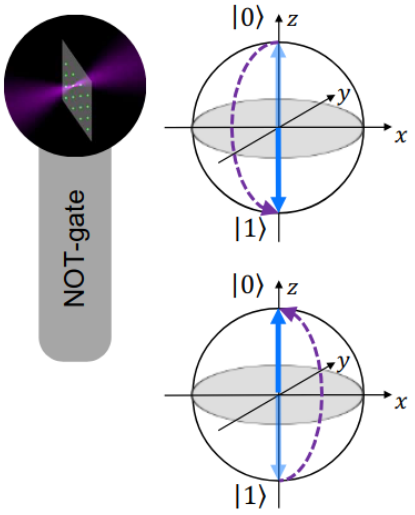

Figure 1: Bloch sphere representation of the effect of a not (or Pauli x) gate.[1]

Before qubit measurement, one can change the state of a qubit using quantum logic gates, which are the quantum computing equivalents of logic gates that make up classical electronic circuits. When applied to a qubit, a quantum gate will set it to a specific superposition of states. This corresponds to a specific rotation of the arrow on the Bloch sphere[5]. Since most gates correspond to a rotation and do not place the qubit directly in a specific state, the resulting superposition depends both on the gate and the state of the qubit before the gate was applied. Some well-known quantum gates are the Pauli x y and z gates, each corresponding to a rotation of 180 degrees around the x, y and z axes of the Bloch sphere. Gates are mathematically represented as matrices that can be applied to the qubit state vector. The Pauli x gate for example would be $\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$.

Quantum gates are applied to single or multiple qubits at the same time, and in general, these are control gates, which will only affect the target qubit if the control qubit is already in state $|1\rangle$. While there are multiple different types of gates, it is possible to create a universal set. Gates in this set can be combined to create any other gate. The set of arbitrary rotation gates $R_x(\theta), R_y(\theta), R_z(\theta)$, combined with a phase shift gate $P(\varphi)$ and a control-X gate are such a set[6].

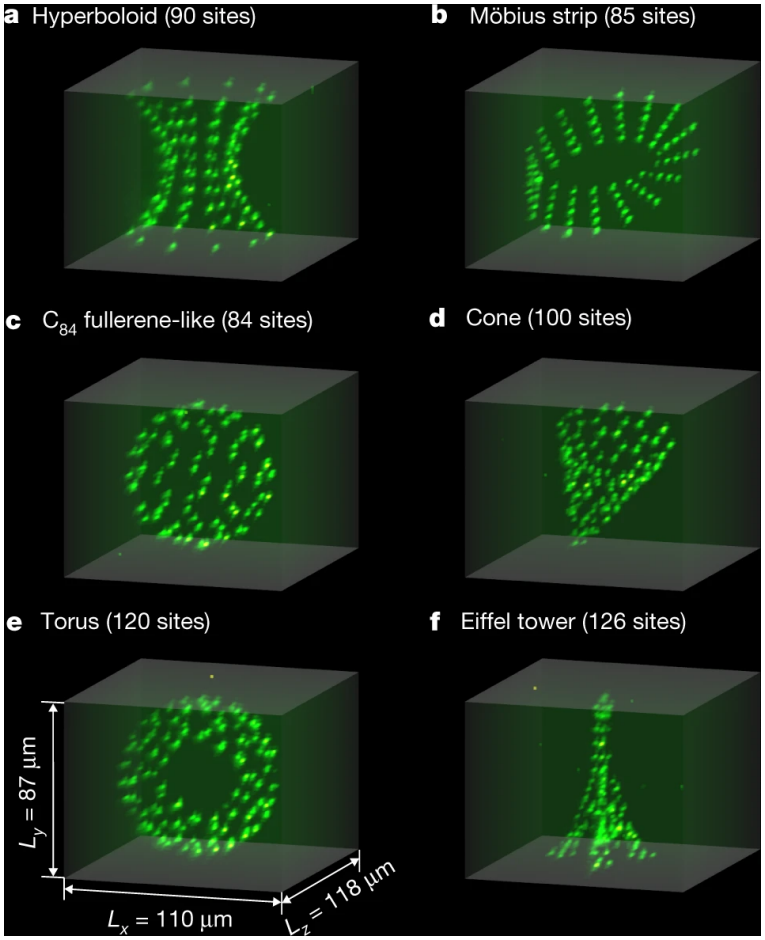

Figure 2: Different qubit configurations in a 3D neutral atom register.[7]

Finally, these qubits are stored in a register. Depending on the type of quantum computer the registry can be fixed, such as for superconducting qubits, where all the qubits are pre-engraved onto a chip and used as needed[8]. Other quantum computers such as the Neutral atom quantum computer built by Pasqal, have a dynamic registry, where a chosen number of qubits can be placed, at the user's will, into a 2D or 3D grid. This extends the ability of the device to be able to perform simulations of quantum systems and analog computations that cannot be represented in a traditional quantum circuit.

1.2 QASM

OpenQASM (Open Quantum Assembly Language) is a programming language for quantum circuits that allows the user to create a register of qubits and comes with a preselection of gates, as well as the universal set of arbitrary rotation, phase shift and CX gate. OpenQASM has become the standard for programming quantum circuits, in part due to it being machine-independent, meaning it can be compiled into any quantum computer[9]. Along with the preset gates, custom user-created gates can be defined and saved to the circuit before compilation and execution. OpenQASM 3 also incorporates timing statements as well as classical computing interactions within a circuit, allowing the user to precisely define when gates or measurements need to be applied, but also allowing the quantum circuit to perform classical operations and

interact with a regular processor during the execution of the circuit[9].

1.2.1 stdgates.inc

stdgates.inc is the file that one can include at the beginning of their QASM code in order to load the preset gates (these can be found in the annex). All the preset gates in this file are defined based on a single built-in gate:
the unitary gate $U(\theta, \phi, \lambda) = \begin{bmatrix} cos(\theta/2) & -e^{i\lambda}sin(\theta/2) \\ -e^{i\phi}sin(\theta/2) & -e^{i(\phi+\lambda)}cos(\theta/2) \end{bmatrix} = e^{\phi+\lambda}Rz(\lambda)Rx(\theta)Rz(\phi)$[9]

This gate is combined with modifiers to allow the creation of multi-qubit gates the global phase $gphase(\gamma)$ which adds a global phase of $e^{i\gamma}$ to an operation. The *ctrl* and *negctrl* modifiers are used to test the value of a control qubit before applying the gate which is being modified, in the case of *ctrl* the control qubit needs a value of 1 for the operation to proceed and 0 in the case of *negctrl*. The *inv* modifier is used to create the inverse $g^\dagger$ of a gate g and finally, the $pow(r)$ modifier raises a gate g to the power g^r.[9]

1.3 Pasqal's Neutral Atom Quantum Computer

1.3.1 Register

This device uses optical beams to create multiple $1\,\mu m^3$ sized traps[10]. Due to their size, these traps can only contain a single atom. The optical beams are then pointed at a vacuum chamber containing atomic vapor and used to single out and reposition atoms into a grid.

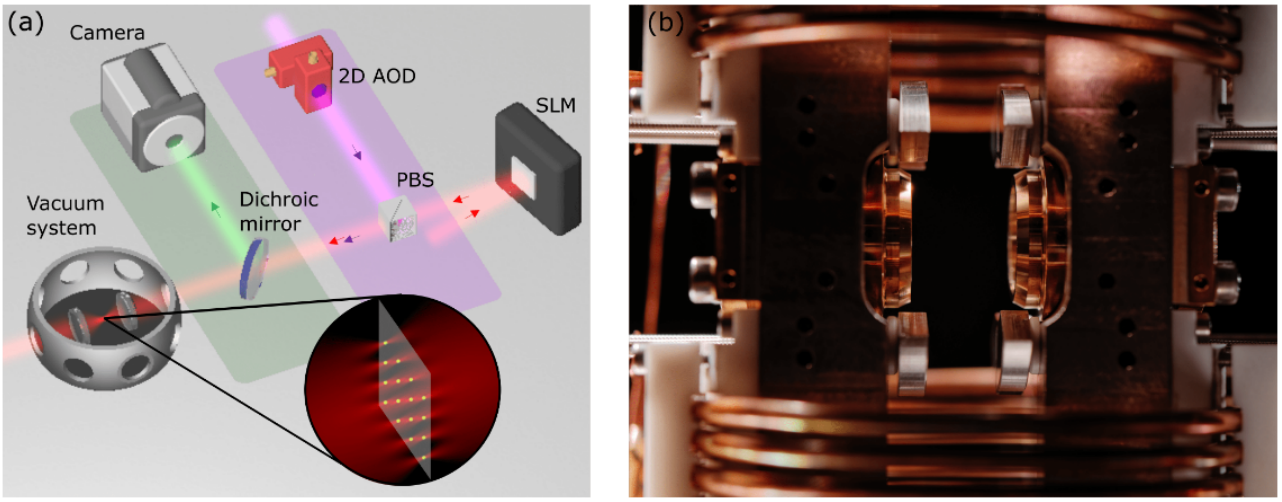

Figure 3: "(a) Overview of the main hardware components constituting a quantum processor. The trapping laser light (in red) is shaped by the spatial light modulator (SLM) to produce multiple microtraps at the focal plane of the lens (see inset). The moving tweezers (in purple), dedicated to rearranging the atoms in the register, are controlled by a 2D acousto-optic laser beam deflector (AOD) and superimposed on the main trapping beam with a polarizing beam-splitter (PBS). The fluorescence light (in green) emitted by the atoms is split from the trapping laser light by a dichroic mirror and collected onto a camera. (b) Photography of the heart of a neutral-atom quantum co-processor. The register is prepared at the center of this setup".[1]

1.3.2 Gates

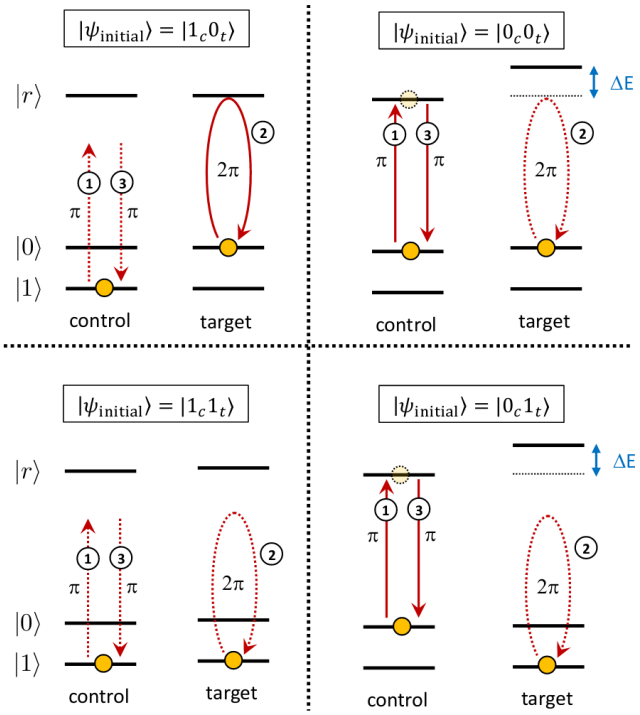

Figure 4: Sequence of optical pulses needed for a Control Z gate[11]

Gates are applied using optical pulses from two different channels: the Raman and Rydberg channels, which can target a single qubit (local) or all of them (global). These pulses are used to attain two different excitation states from the ground state $|g\rangle \equiv |0\rangle$. The Raman pulses are used in single qubit gates to access the hyperfine state $|h\rangle \equiv |1\rangle$[12]. Multi-qubit gates on the other hand require the use of quantum coherence, which is achieved using a phenomenon called the Rydberg blockade: If an atom is excited to the Rydberg state $|r\rangle$[13](using the Rydberg channel of the quantum device), any nearby atom situated within a certain radius will be blocked from achieving the Rydberg state[14]. This interaction is used, for example in the implementation of a CZ gate[15].

1.3.3 Pulser

Pasqal has published a Python library named Pulser that simulates their quantum device. It can create a registry with any kind of pattern as well as the required optical pulses. The pulses have a configurable waveform with amplitude area Ω, detuning δ, phase shifting φ and the duration τ of the pulse. Manipulating these parameters, it is possible to recreate any single-gate rotation on the Bloch sphere with angles :

$$(\Omega\tau\cos\phi, \Omega\tau\sin\phi, \delta\tau)\,[1]$$

1.3.4 Pulser's Unitary gate and virtual phase shifts

Using the above definition, one can create an optical pulse that corresponds directly to a parameterizable unitary gate. The optical pulse of area θ, detuning $\delta = 0$ and phase ϕ corresponds to the Bloch sphere rotation:

$$R_{Bloch} = R_z(-\phi)R_x(\theta)Rz(\phi).\,[1]$$

While this particular pulse for rotations around the x axis when $\phi = 0$ or is a multiple of π and y axis when ϕ is a multiple of $\pi/2$. Rotations around the z axis are impossible and need to be implemented via so-called Virtual Z-gates. Since Z gates affect the phase of a qubit and not the probability to measure 0 or 1, virtual gates are achieved by applying a phase shift to

all the following x and y gates. In Pulser, this is known as a post-phase shift. Combining this phase shift and the resonant pulses one can create the gate:

$$U(\gamma, \theta, \phi) = R_z(\gamma + \phi)R_{Bloch} = R_z(\gamma + \phi)R_z(-\phi)R_x(\theta)R_z(\phi)^{1}$$

This actually corresponds to the unitary gate:

$$U(\gamma, \theta, \phi) = R_z(\gamma)R_x(\theta)R_z(\phi)^{1}$$

It is important to note that this unitary gate differs from the unitary gate defined by QASM in that the middle rotation is an R_x rotation and there is a difference in the global phase.

1.4 Aim

There are currently no QASM compilers specific to neutral atom quantum computers, thus this study aims to determine the feasibility of such a project. This will be done by first establishing whether the required gates for such a task can be implemented on a physical level. Second, the constraints specifically imposed on register creation need to be identified. Finally, the work on the compiler will start by starting to parse information from some of the main commands of QASM.

2 Methodology

2.1 Quantum Logic Gates

The *pulser.Pulse* class was used to create the optical pulses corresponding to each quantum gate, using $pulser.waveforms.BlackmanWaveform$ as a preset. Below is the function used to create a Raman pulse, this is the pulse used for single-qubit operations:

```
def raman_pulse(area, detuning, phase, postphase, duration, sequence, target
    ):
    pulse = Pulse.ConstantDetuning(
        BlackmanWaveform(duration, area), detuning, phase, post_phase_shift=
    postphase)

    if "raman" not in sequence.declared_channels:
        sequence.declare_channel("raman", "raman_local", target)
    else:
        sequence.target(target, "raman")

    sequence.add(pulse, "raman", "wait-for-all")

```

Other waveform presets are available but have not yet been tested, as they would require different phase parameters. Since the waveforms, are defined directly by the area under the curve there is no need to adjust the amplitude or the duration of the pulse. Thus the duration is set to 250ns, but this can be changed to a shorter duration depending on the specifications of the device. The detuning is also constant and set to 0 for every gate to produce a resonant pulse.

2.1.1 Translating Single Qubit Gates

When it comes to translating single qubit gates, there are two options, one can either try and recreate, the QASM unitary gate and modifiers using native pulses to the Pascal computer or reproduce every gate in the stdgates.inc. Since the gates in this file are a universal set, one would then have to make use of a transpiler to implement user-defined gates, whereas the first solution would require finding adequate parameters for the unitary gate. The parameters used for all the gates are provided in the annex.

2.1.2 Translating Two Qubit Gates

The only native two-qubit gate is the CZ gate, achieved by producing a sequence of three pulses using the Rydberg channel. As shown in Figure 4, these pulses are simply a pulse of area π followed by a 2π pulse and another π pulse after that. All of these pulses have a detuning and phase of 0. One can then combine this gate with two Hadammard gates to form a CNOT gate, which combined with the Rx, Ry and Rz gates is considered a Universal set. When running a pulser simulation with these gates, the qubits temporarily become qutrits since a third state is reached, thus the initial state will also have to be defined using qutrits. The results of the simulation will then need to be reduced back to base 2.

2.1.3 Testing Gates

Since pulser offers the possibility to simulate with a user-defined arbitrary initial state, multiple simulations can be run consecutively for the same gate, but with different initial state vectors. This way a gate can be comprehensively tested against the mathematical theory. The initial state is defined randomly using the *rand_ket*() method from the Qutip library. The theoretical results are then computed by applying the gate operator to the initial state vector. These theoretical results can then be compared to the results produced by the corresponding sequence of pulses. One should note that the results should be compared to an accuracy of 10^{-4} as the fidelity of the gates starts breaking down at smaller accuracies. The comparison is then repeated multiple times, with different initial states, if 100% of the test succeeds, then the gate is considered correct. Two different types of comparisons are made, first comparing the absolute value of the eigenvalues, we can determine whether the probabilities of measuring a certain state after applying the tested gates are correct. If this first comparison succeeds we can then compare the eigenvalues directly. If the first comparison succeeds and the second fails, this means there is an issue with the phase of the qubit after the gate was applied.

2.2 Testing Rydberg Blockade

This test is conducted to check whether qubits within the Rydberg radius of the target and control qubits of a certain multi-qubit gate would affect results. To test this, 4 qubits are placed in a 1 nm-sided square so each qubit is within the Rydberg radius of the three others. Following this a CNOT gate is first applied to q0 and q1, after which a second CNOT gate is applied to q2 and q3. The initial states in pulser have to be modified into random qutrits with 0 probability of being in the Rydberg state. The Pulser simulation results can then be reduced back to qubits and compared with the mathematical theory.

2.3 Parsing QASM Code

Parsing the QASM code is done using the OpenQASM3.parse() method. This method converts the QASM code into an Abstract Syntax Tree (AST), which can be accessed through node visitor classes. Each node visitor will be launched consecutively with a context parameter that will store the information retrieved. Once a node visitor is called, it may also call other visitors with a specific context variable. Each node visitor will have to contain QASM syntax checking for each of the QASM commands that are available and interrupt the compilation to throw an error to the user. The context parameters are created as lists of tuples that contain all the information retrieved by a visitor in a particular context.

3 Results

3.1 Unitary Gates

Translating Pulser's native unitary gate to QASM's unitary gate was only partially successful. Three of the four comparisons made on each test iteration were successful. the eigenvalues for the $|0\rangle$ are equal, however the eigenvalues of the $|1\rangle$ state is in the wrong phase.

```
init Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
Qobj data =
[[-0.19284728+0.61531422j]
 [-0.64960343-0.40275765j]]
theory Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
Qobj data =
[[-0.1398945 +0.92799712j]
 [-0.27096202+0.2140805j ]]
sim Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
Qobj data =
[[-0.13989332+0.92799979j]
 [-0.3387502 -0.06704144j]]
```

Figure 5: Example resulting states of A QASM_U test

As seen in the figure, the eigenvalues for the $|0\rangle$ are identical up to an accuracy of 10^{-5}, but the $|1\rangle$ states are different. However, if the absolute value is taken:

- $\sqrt{(}-0.27096202^2 + 0.2140805^2) = 0.34532720246$
- $\sqrt{(}-0.3387502^2 + 0.06704144^2) = 0.34532050717$

We notice that the measurement probabilities are indeed correct, thus there is a phase issue around the $|1\rangle$ state.

3.2 Pauli and Hadammard Gates

Similar to the QASM unitary gates the implementation of Pauli gates suffers phase issues. The phase issues result in a sign flip either on the $|0\rangle$ state or the $|1\rangle$ depending on which gate is used. As seen in figure 6, the Pauli gates flip the sign of the $|0\rangle$ state whereas the Hadamard gate flips the sign on the $|1\rangle$ state

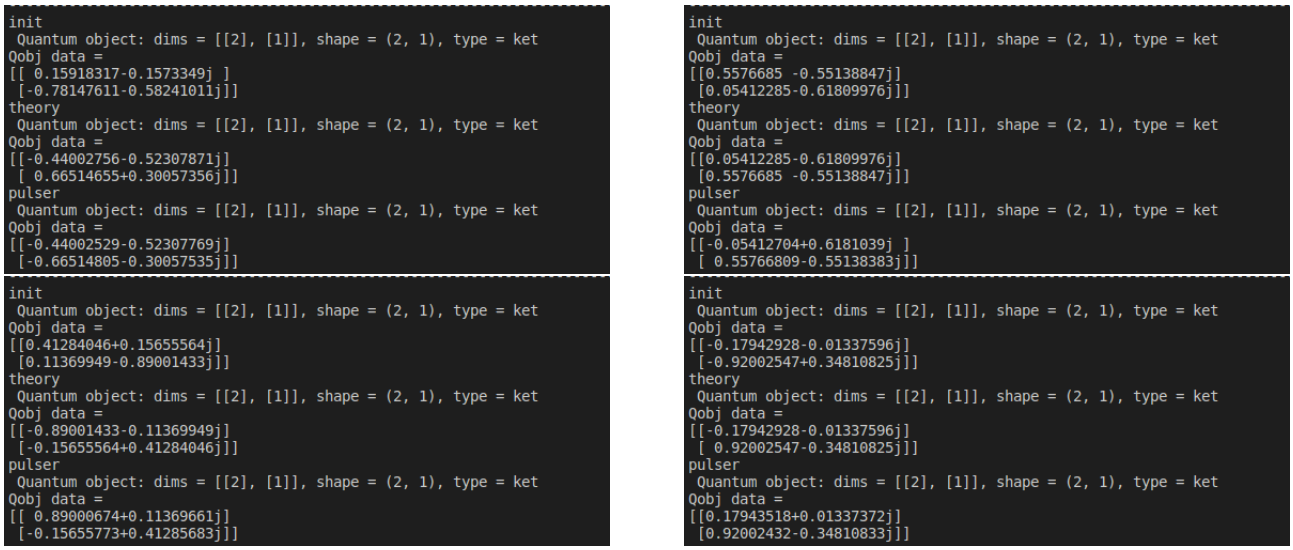

Figure 6: From top-left to bottom-right, an example of the Hadamard, X, Y and Z gates test results

3.3 Arbitrary Rotation Gates

Figure 7: Rx and Rz test results for 10 random initial states paired with 10 random values of $\theta \in [-2\pi, 2\pi]$

As mentioned in the introduction, the Rx and Ry gates can be implemented relatively easily using, Pulser's unitary gate. For both these gates not only was the testing fully successful, but the gate fidelity was also increased to 10^{-5}. The Rz gate is also fully functional as long as θ remains positive. This is due to the fact Rz is implemented as a virtual gate. The phase shifts pulser implements are always $\lambda \in [0, 2\pi]$ rather than $\lambda \in [-2\pi, 2\pi]$, thus the extra phase shift is not applied correctly when negative.

3.4 CNOT gate

Having been implemented, by combing two H gates and a CZ gate (H CZ H), the CNOT gate is a good way to test whether consecutive gates work correctly, but also makes use of basis changes, since H is a single qubit gate. This allows us to test whether switching from qubits to qutrits and back works correctly. Like the Rx and Ry gates the CNOT gate was a complete success, regardless of which qubit was designated as control, with a fidelity of 10^{-5}.

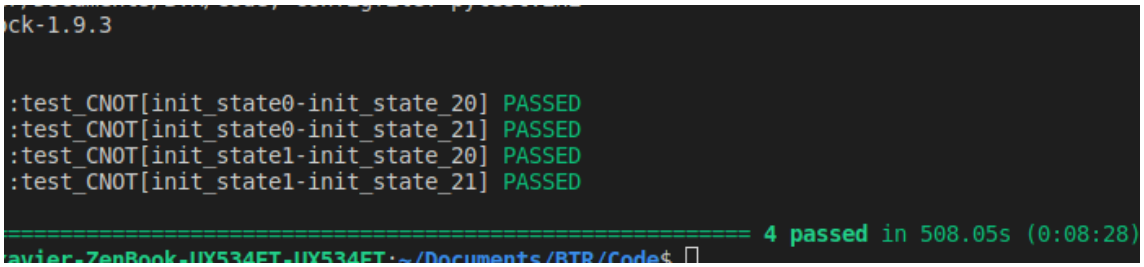

Figure 8: CNOT test results using two different random initial states

3.5 Rydberg Blockade Testing

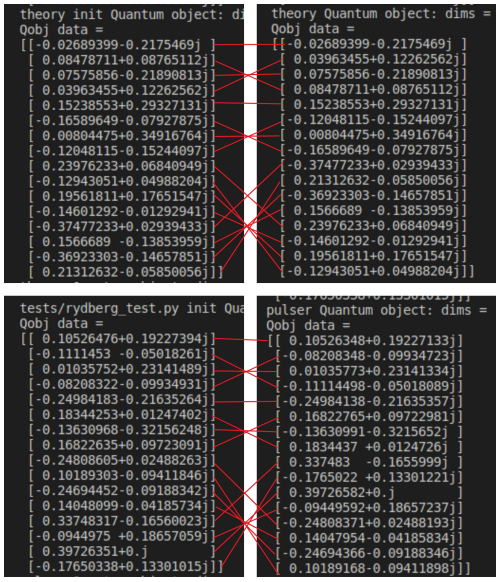

Figure 9: Rydberg Blockade test results

Due to problems with changing bases from qutrits to qubits, the initial state for the theoretical gates and the Pulser simulation does not start with the same initial state. This problem was not solved during the time of the project and thus this test cannot be considered rigorous. However, there is some information that can be taken away, as seen in Figure 9, even though the initial states are different, there is an obvious pattern as to how both the theoretical gate and the Pulser simulation, modify the initial state. Both these patterns being identical, one could assume that the gate works correctly.

3.6 Parsing

While the parsing methods are far from being completed, they are already partially functional and information about the registers, defined gates and used gates can be obtained. For the moment these methods still offer minimal syntax error detection. For example, the code is currently able to detect when the targets and controls of a two-qubit gate are badly set. This can be illustrated in Figure 10.

```
    gate x a { U(pi, 0, pi) a; }
    gate y a { U(pi, pi/2, pi/2) a;
               U(pi, pi/2, pi/2) a;
    }

    qubit[2] q;
    qubit[3] b;

    cnot b, q;
    x q[1];
```

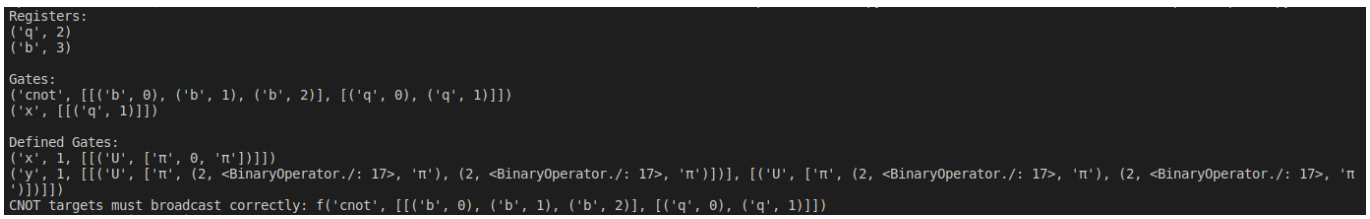

Figure 10: parsing results

4 Discussion

4.1 Test Results

The test results are quite promising, although there are numerous phase and sign problems. The fact that the CNOT gate was a full success while using two Hadamard gates, indicates that the sign flips on the Pauli and Hadamard gates may not be problematic.

4.2 Rydberg Blockade

The results of this test are very encouraging for optimizing the register population. If all Rydberg pulses are applied consecutively and not in parallel, the position of all the qubits in the register does not affect the result of the gate as long as it is applied on two qubits that are within each other's Rydberg radius. This is a major boon, as a very simple solution to this problem is to place all qubits inside a circle of diameter slightly inferior to the Rydberg blockade. This solution, while very simple is also limited by hardware specs, as only a limited amount of qubits could be placed in such a circle. On the other hand, this solution could be extended, by creating multiple overlapping circles and placing the qubits in each circle based on which other qubits they have to interact with.

4.3 Gate Modifiers

Other than the *inv* modifier, which could be implemented quite easily since the Rx, Ry, as well as the Unitary gate, accept negative parameters. gate modifiers are the more difficult command to translate since they are extremely versatile and can be used on any gate. However, creating an algorithm that could implement them would be an extremely difficult task. Every user-defined control gate created would have to be transpiled into a combination of CNOT and single-qubit gates as there is no way of natively implementing the *ctrl* modifiers.

4.4 Parsing

Once the structure needed to create an AST node visitor is understood, retrieving the information and catching syntax errors is fairly easy. However, there is a great number of commands that can be used in the QASM language, and each of these commands has a multitude of possible syntax errors. This makes the task of building an exhaustive parser very time-consuming, although not necessarily excessively difficult.

4.5 Phase Gate and Final Thoughts

The previous results are promising regarding the possibility of compiling QASM on a quantum computer. Some promise an easier approach than initially expected, registry creation for example, and some predict future tasks to be more difficult, such as translating user-defined controlled gates. However, there is one major problem in the lack of a phase gate. Implementing this gate would not only solve the universal set problem by completing the Arbitrary rotation gates but it could also be combined with the current implementation of the QASM unitary gate. Thus making user-defined gates very easy to implement since they are already defined in QASM using the QASM U gate. The problem is that so far no solution to implement this gate

has been found using pulses with constant detuning, and there is no indication of whether a solution exists, which could seriously compromise the feasibility of this project.

5 Critical Reflection

First of all, I would like to thank my supervisors Claire Blackman and Simon Cross, as they were always available to help me when stuck. The regular weekly meetings were also greatly appreciated as they allowed me to set weekly goals, with a short deadline to combat procrastination.

Regarding the project, I feel like I grossly underestimated the amount of work that would be needed to produce a functional compiler as suggested initially in the thesis proposal, even for an extremely simple compiler. Another aspect I did not expect to be a factor was computing time, when running the tests, as the screenshots show, single qubit gates could undergo hundreds of tests in a couple of seconds, however when testing two-qubit gates, compute time could last up to two hours for a single test. This also considerably slowed down the pace at which results could be obtained. These two factors were the most significant as to why I was not able to stick to the initial plan, which, in retrospect, was already quite ambitious. However I am very happy with how far I got, although I wasn't able to produce a functional compiler, I now have a good idea of how feasible the project is.

This was also my first time coding with limited documentation and made me greatly appreciate how important producing good quality documentation along with the code is crucial for anyone to be able to work on the project.

Overall I am very satisfied with how the project ended up playing out, the challenging nature of it forcing me to learn a great deal in the process.

References

[1] Henriet L, Beguin L, Signoles A, Lahaye T, Browaeys A, Reymond GO, et al. Quantum Computing with Neutral Atoms. Quantum. 2020 Sep;4:327.

[2] Wong TG. Introduction to Classical and Quantum Computing. Omaha, Nebraska: Rooted Grove; 2022.

[3] Wiseman HM, Milburn GJ. Interpretation of Quantum Jump and Diffusion Processes Illustrated on the Bloch Sphere. Physical Review A. 1993 Mar;47(3):1652-66.

[4] McKay DC, Wood CJ, Sheldon S, Chow JM, Gambetta JM. Efficient Z Gates for Quantum Computing. Physical Review A. 2017 Aug;96(2):022330.

[5] Mummaneni BC, Liu J, Lefkidis G, Hübner W. Laser-Controlled Implementation of Controlled-NOT, Hadamard, SWAP, and Pauli Gates as Well as Generation of Bell States in a 3d–4f Molecular Magnet. The Journal of Physical Chemistry Letters. 2022 Mar;13(11):2479-85.

[6] Williams CP. Quantum Gates. In: Williams CP, editor. Explorations in Quantum Computing. Texts in Computer Science. London: Springer; 2011. p. 51-122.

[7] Barredo D, Lienhard V, de Léséleuc S, Lahaye T, Browaeys A. Synthetic Three-Dimensional Atomic Structures Assembled Atom by Atom. Nature. 2018 Sep;561(7721):79-82.

[8] Ladd TD, Jelezko F, Laflamme R, Nakamura Y, Monroe C, O'Brien JL. Quantum Computers. Nature. 2010 Mar;464(7285):45-53.

[9] Cross A, Javadi-Abhari A, Alexander T, De Beaudrap N, Bishop LS, Heidel S, et al. OpenQASM 3: A Broader and Deeper Quantum Assembly Language. ACM Transactions on Quantum Computing. 2022 Sep;3(3):12:1-12:50.

[10] Schlosser N, Reymond G, Protsenko I, Grangier P. Sub-Poissonian Loading of Single Atoms in a Microscopic Dipole Trap. Nature. 2001 Jun;411(6841):1024-7.

[11] Silvério H, Grijalva S, Dalyac C, Leclerc L, Karalekas PJ, Shammah N, et al. Pulser: An Open-Source Package for the Design of Pulse Sequences in Programmable Neutral-Atom Arrays. Quantum. 2022 Jan;6:629.

[12] Tsai RBS, Silvério H, Henriet L. Pulse-Level Scheduling of Quantum Circuits for Neutral-Atom Devices. arXiv; 2022.

[13] Cohen SR, Thompson JD. Quantum Computing with Circular Rydberg Atoms. PRX Quantum. 2021 Aug;2(3):030322.

[14] Saffman M. Quantum Computing with Atomic Qubits and Rydberg Interactions: Progress and Challenges. Journal of Physics B: Atomic, Molecular and Optical Physics. 2016 Oct;49(20):202001.

[15] Saffman M, Walker TG, Mølmer K. Quantum Information with Rydberg Atoms. Reviews of Modern Physics. 2010 Aug;82(3):2313-63.

Appendix: Gates

```
#Unitary gate
def U(gamma, theta, phi, target,seq):
    target = "q" + str(target)
    raman_pulse(theta,0,phi,gamma+phi,raman_duration,seq,target)


#QASM Unitary
def QASM_U(theta, phi, lambda, target, seq):
    if theta >= 0:
        U(phi-np.pi/2,theta,lambda-np.pi/2, target, seq)
    else:
        U(phi-np.pi/2,-theta,lambda+np.pi/2, target, seq)


#Pauli Gates
def Y(target, seq):
    U(0,np.pi,np.pi,target, seq)

def X( target, seq):
    U(-np.pi/2,np.pi,np.pi/2,target, seq)

def Z(target, seq):
    U(-np.pi/2,2*np.pi,np.pi/2, target, seq)


# Hadamard gate
def H(target, seq):
    U(np.pi/2,np.pi/2,np.pi/2,target,seq)


def Rx(theta, target, seq):
    if theta >= 0:
        U(0,theta,0,target, seq)
    else:
        U(-np.pi,-theta,np.pi,target, seq)

def Ry(theta, target, seq):
    if theta >= 0:
        U(np.pi/2,theta,-np.pi/2,target, seq)
    else:
        U(-np.pi/2,-theta,np.pi/2,target, seq)

def Rz(theta, target, seq):
    U(theta,0,0, target, seq)

# Identity gate
def I(target, seq):
    X(target, seq)
    X(target, seq)
    X(target, seq)
    X(target, seq)

# Control-Z gate
def CZ(control, target, seq):
    target = "q" + str(target)
```

```
    control = "q" + str(control)

    # pulses
    pi_pulse = Pulse.ConstantDetuning(
        BlackmanWaveform(rydberg_duration, np.pi), 0.0, 0
    )
    twopi_pulse = Pulse.ConstantDetuning(
        BlackmanWaveform(rydberg_duration, 2 * np.pi), 0.0, 0
    )

    # Perform a single-qubit Identity gate in order to get the right state
    basis in the simulation
    if seq.declared_channels == {}:
        I(0, seq)

    if "ryd" not in seq.declared_channels:
        seq.declare_channel("ryd", "rydberg_local", control)
    else:
        seq.target(control, "ryd")

    # add pulses to sequence
    seq.add(
        pi_pulse, "ryd", "wait-for-all"
    )  # Wait for state preparation to finish.
    seq.target(target, "ryd")  # Changes to target qubit
    seq.add(twopi_pulse, "ryd")
    seq.target(control, "ryd")  # Changes back to control qubit
    seq.add(pi_pulse, "ryd")

# Control-NOT gate
def CNOT(control, target, seq):
    H(target, seq)
    CZ(control, target, seq)
    H(target, seq)
```