Investigating faster quantum gate schemes for molecular nanomagnet qubits

Rebecca Dalphin

December 2024

Advised by Professor Charles Collett

A Thesis presented to the Faculty

of the Hamilton College Physics Department

in Partial Fulfillment of the Requirements

for the Degree of Bachelors of Arts

Abstract

Quantum computers are one of the most exciting technologies being developed currently. Molecular nanomagnets (MNMs) present unique advantages as potential qubits, but for MNMs to be a viable qubit option we must be able to perform a universal basis set of quantum gates. For the MNM Cr_7Mn, the rotation gate and phase gate, forming two of the three gates in the universal basis set, have been demonstrated. A passive scheme for the third gate, a CNOT, has previously been proposed but is relatively slow, so in this work we explore alternative active schemes to find a faster option (C. Collett *et al.*, 2020). We propose a scheme that performs a CNOT with only a single Gaussian pulse, acting significantly faster than the previous scheme. Our simulations show that it performs almost as expected, with the exception that it adds a conditional phase as well, creating an iCNOT gate with high fidelity. We demonstrate that the iCNOT gate is effectively interchangeable with the CNOT gate by simulating its use in a quantum phase estimation algorithm.

Acknowledgments

I would like to thank my advisor Charles Collett for his support and guidance throughout my thesis project. His patient explanations and enthousiasic encouragement ensured not only the success of my project, but also that is was fun.

Additionally, I would like to thank the physics department. This department has become my home at Hamilton. I would not be the scientist I am today without the experiences and support I found here.

I would also like to thank my friends and family. I couldn't have done it without you.

Contents

List of Figures

List of Tables

1 Introduction

Quantum computing is one of the technologies poised to significantly impact the twenty-first century. Recognition of this has sparked significant research recently, with many important advancements in the field. These advancements include IBM's computer with 1121 qubits (Rao, 2024) and the development of the AlphaQubit by Google to start to address quantum error correction (Google, 2024). The key advantage of quantum computers is the speed with which they can solve problems in which a large amount of data needs to be manipulated, or a large search space examined. Examples of where quantum computers could be applied are machine learning, financial modeling, or cybersecurity (Roundy, 2023).

The fundamental difference between classical computers and quantum computers that allows for this dramatic speed change is the way information is stored. In classical computers, the basic information storage is a bit, which can be expressed as either a 0 or a 1. However, quantum computers use quantum bits, or qubits. Qubits also have two states, again either $|0\rangle$ or $|1\rangle$, but due to their quantum nature, they can exist in a superposition of these two states. Unlike bits, which exploit classical systems to store information, qubits exploit the quantum properties of a given system (Schneider and Smalley, 2024). There are many different choices for quantum systems that could function as qubits. Some of the more popular qubits currently being researched are superconducting, trapped ions, and quantum dots (Schneider and Smalley, 2024).

The qubit system we have chosen to use is molecular nanomagnets (MNMs). Much like the other qubit modalities, such as quantum dots and photons (Schneider and Smalley, 2024), this system exploits the quantum nature of spin, in this case the spin ensemble of the molecule. Despite not being as well developed currently as other systems, we are investigating MNMs due to how easy they are to chemically engineer, and for their potential solutions to quantum error correction (Chiesa *et al.*, 2024). In this system we define the $|\uparrow\rangle$ spin state to be 1, and the $|\downarrow\rangle$ spin state to be 0. Due to the quantum nature of spin, the qubit can be in any superposition of these states. The values of these superpositions can be changed by 'tipping' the spins (described further in background), which amounts to a basic logic operation.

The long term aim of this lab is to show the viability of this system for quantum computing by performing multi-qubit operations. To this end we need to be able perform all the gates in a logical basis for computing, which is the scope of this thesis. The logical basis we have chosen is comprised of the R_x and R_y rotation gates, the phase gate, and the CNOT gate (QuEra, 2023). We interact with the qubits using Electron spin resonance (ESR), which works by applying radio frequency (RF) pulses to the spins, which means we must be able to find ways to perform the logic gates with RF pulses.

We already have ways to perform the rotation and phase gates; however, the CNOT gate, which requires interacting with two coupled qubits, is more complicated. As will be discussed further in the background section, a passive scheme which performs a CNOT gate has already been found (C. Collett *et al.*, 2020). However, it requires a slow free evolution step and thus takes approximately 924ns to act. For quantum computing it is ideal to have as fast a gate as possible to allow more gates to be applied in a given amount of time. Therefore the goal of this thesis is to develop an active CNOT gate scheme which can be significantly faster.

2 Background

As mentioned above, molecular nanomagnets (MNMs) are one option currently being investigated for their viability as qubits. MNMs consist of a magnetic core composed of one or more metal ions, and organic ligands surrounding this center. The organic ligands separate the core from the rest of the environment, resulting in strong coupling between the metal ions and weak dipolar coupling between neighboring molecules (Chiesa *et al.*, 2024). The strong core coupling means the spins of these molecules act as one 'giant spin'. The energy states of this 'giant spin', which are determined primarily by internal anisotropy from the molecular structure, can be used as qubit states. The multiple energy levels (spin values can vary from $\frac{1}{2}$ to more than 10) in MNMs lead to potential solutions to quantum error correction, where the same computation could be done multiple times on the same qubit, and the results averaged to reduce the possibility of an error. One such MNM, Cr_7Mn, has been studied as a qubit candidate for the ease with which its traits can be engineered leading to its long coherence time, caused in large part by the clock transition at zero field (Garlatti *et al.*, 2014).

To understand the importance of coherence time (T_2) it is worth noting that one of the requirements of any qubit is that quantum gates must be able to be implemented in a consistent and predictable way. This means the qubit must be able to maintain its quantum state long enough for many gates to be performed, since actual computations will take a long time. At low temperatures, in our system, the main factor to cause decoherence is dipolar interactions (Chiesa *et al.*, 2024). If decoherence occurred too quickly, a state could be randomized before the computation was completed, so measurements would not reflect only the effects of the gates used in a computation, but also the effects from fluctuations of the qubit's spin due to dipolar interactions. Thus, the quality of the qubit is proportional to T_2. However, the number of gates a qubit can perform (corresponding to the complexity of the computation) within T_2 depends on how long the gates take to act (t_{gate}). This means that the quality of the qubit is also inversely proportional t_{gate}. We therefore define one figure of merit η for a qubit to be the coherence time (T_2) divided by the time it takes to perform a gate operation (t_{gate}),

$$\eta = \frac{T_2}{t_{gate}}. \tag{1}$$

Clearly, one way to increase the quality of a given qubit is to increase the coherence time. Here MNMs present a significant advantage. MNMs are chemically engineered resulting in it being possible for their traits to be altered by chemical synthesis (Chiesa *et al.*, 2024). This includes being able to engineer the spin Hamiltonian of the molecule, controlling the energy levels and the coupling (Chiesa *et al.*, 2024). Crucially, this engineering allows clock transitions to be added, and coupling between spins in heterodimers to be varied. The spins in Cr_7Mn have been engineered such that the ground state is an $S = 1$ state. In a magnetic field this leads to the expected splitting into $2n + 1 = 3$ states. Figure 1 shows the energy spectrum for the ground state effective spin system for Cr_7Mn (C. Collett *et al.*, 2020), plotted against the strength of an applied magnetic field. Figure 1 also shows a diagram of the structure of the molecule, clearly showing the hetero-metallic ring that is surrounded by organic ligands. The energies for the three spin states are represented by the blue, orange, and green lines. In high field, the orange and blue lines have the expected divergence from each other due to the Zeeman effect. In high fields these correspond to the states $|+1\rangle$, $|-1\rangle$ and $|0\rangle$, these being the S_z eigenstates. However, near zero field there is an 'avoided crossing', where the expected degeneracy between the two states is broken by transverse anisotropy. The energy eigenstates in this region are,

$$|-\rangle = \frac{|+1\rangle - |-1\rangle}{2}, \tag{2}$$

and

$$|+\rangle = \frac{|+1\rangle + |-1\rangle}{2}, \tag{3}$$

along with the constant $|0\rangle$ state. This 'avoided crossing' is a clock transition. In this region, to first order, the energy of the state does not depend on the strength of the magnetic field. This reduces the energy fluctuations of the states due to fluctuations in the local field environment. Energy fluctuations change the precession frequencies of the spins, which would cause them to dephase faster.

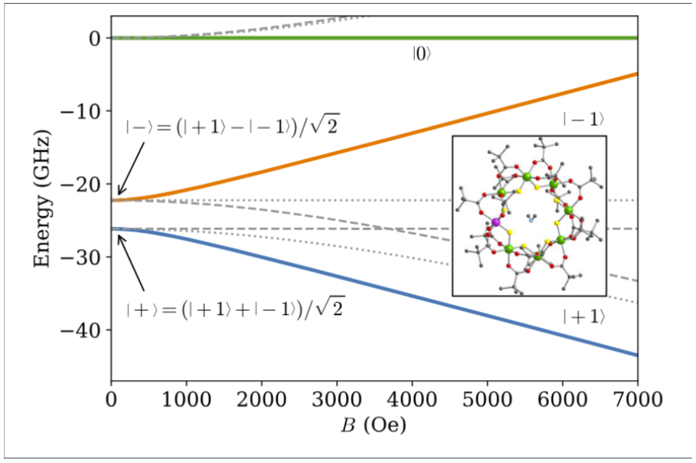


Figure 1: The lowest energy $S = 1$ spin system for Cr_7Mn, plotted as a function of applied axial field, along with the structure of the molecule. The blue and orange lines correspond to the $|+\rangle$ and $|-\rangle$ states, while the green line corresponds to the $|0\rangle$ state. Figure from C. Collett *et al.*, 2020.

Transitioning between the different spin states in Figure 1 is done by adding or taking away energy. In a lab setting, these transitions can be induced with electron spin resonance (ESR). This can be most easily understood using Bloch sphere diagrams, pictured in figure 2. The z axis represents the alignment of the intrinsic magnetic field of the qubit. The $|+\rangle$ state is perfectly aligned with this field, while the $|-\rangle$ state occurs when the spins are perfectly anti-aligned with the field. Any point on the surface of the sphere represents a possible superposition of these two states.

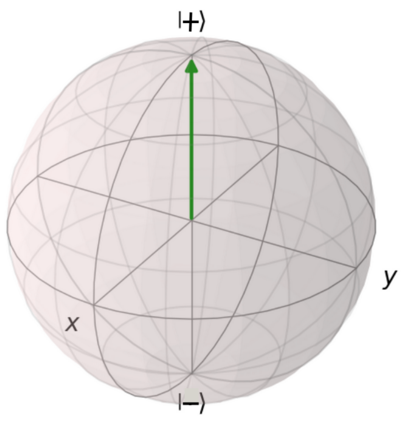


Figure 2: A Bloch sphere, showing the $|+\rangle$ and $|-\rangle$ states, with a $|+\rangle$ vector shown in green. All possible spin states represented by points on the surface of the sphere.

Using radio frequency pulses set to a frequency corresponding to the energy difference between two states, transitions between energy states can be induced. In the case of figure 2, a transition could be induced from the $|-\rangle$ state to the $|+\rangle$ state by adding the exact energy difference between these two states. For MNMs, any pair of states can form a qubit, and the spin energy state acts as bits, or in classical computing terms, a $|+\rangle$ could correspond to a 1, and a $|-\rangle$ could correspond to a 0. Thus, a pulse (or a series of pulses) which changes the superposition of the spin state is acting as a logic gate.

In quantum computing a universal basis set is a combination of logic gates that would be able to produce any desired computation. One of the common basis sets is comprised of rotation gates, a phase gate, and a controlled not (CNOT) gate (Neilsen and Chuang, 2010). Rotation gates consist of a rotation away from the z axis (the easy axis), e.g. a rotation from the $|+\rangle$ to the $|-\rangle$ state. Typically, these are referred to in Cartesian coordinates, where are rotation can be about the x or y axis, to change the energy value, or z projection. A phase gate produces a rotation about the easy axis, and does not change the z projection.

Using ESR, we can achieve single qubit rotation, and induce phase changes (C. A. Collett *et al.*, 2024). A CNOT gate functions as the entangling gate, and requires the operation to be applied to two interacting qubits. Since it is a controlled gate, there must be a control qubit, whose state influences the outcome of the target qubit. If we take the simplest two qubit case, there are four permutations of the states of these two qubits: 00, 01, 10, and 11. Here, the first digit corresponds to the control qubit, and the second digit, the target qubit (Williams, 2011). Since one requirement for a quantum gate is that they must be reversible, there must be an exact one to one mapping between the input state to the output. This one to one mapping can be visualized in the form of a matrix, with the standard matrix representation of a CNOT being:

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

Here the rows and the columns each correspond to a possible permutation of the qubit pair. The matrix therefore represents how one permutation would be affected by the CNOT gate. For example, if we take the first row and first column to be the permutation $1, 1$, where the columns are the start state, and the rows are the end state, the 1 in the [1,1] element shows that no change to the end state would be made. This is not the case when the control qubit starts as $|0\rangle$ - a fact that is inherent to this functioning as a CNOT gate. When the control qubit starts as $|0\rangle$, the target qubit's state is inverted. To further clarify this function, a full diagram of start states and their corresponding end states is shown below.

$|1,1\rangle \rightarrow |1,1\rangle$

$|1,0\rangle \rightarrow |1,0\rangle$

$|0,1\rangle \rightarrow |0,0\rangle$

$|0,0\rangle \rightarrow |0,1\rangle$

Of course, the input state could be any superposition of these permutations, and then these would be proportionally mapped to their respective output states. This matrix representation holds for a two qubit system with two $S = \frac{1}{2}$ spin qubits, however; for a 2 qubit system with $S = 1$ spin qubits, a nine by nine matrix is required, along with a more complicated scheme for implementing this gate.

In 2020, Collett et al. published a paper simulating a scheme for driving a CNOT gate for the Cr_7Mn system (C. Collett *et al.*, 2020). They did this by using the four lowest energy states. They were able to do this because when the easy axis (axial) energy term is larger than than the other terms the system can be truncated to ann effective $S = \frac{1}{2}$ system (C. Collett *et al.*, 2020). However, this system requires a free evolution of state, resulting in a t_{gate} of 924ns. T_2 for $\mathrm{Cr_7Mn}$ is 15μs, so this t_{gate} would not allow the application of many gates within T_2, making this significantly slower than ideal for a quantum gate. From equation 1, if we assume that T_2 has been maximized, the way to improve a qubit is to decrease the time it takes for a gate to operate. This thesis aims to investigate possible alternative active schemes that act as a CNOT gate. These schemes, by avoiding the free evolution state, should be faster than the previously found time.

3 Theory

The giant spin Hamiltonian of Cr$_7$Mn can be expressed as

$$\mathcal{H} = -DS_z^2 + E(S_x^2 + S_y^2) + g\mu_B B_z S_z. \tag{4}$$

The constants D and E are values determined empirically, and are inherent to a given system. The D term corresponds to the energy along the easy axis (axial anisotropy), and the E term to the energy along the hard axis (the axis perpendicular to the alignment of the intrinsic magnetic moment, also called the transverse anisotropy). The final term corresponds to the energy due to the spin interacting with an external static field B_z. Since we are interested in exploiting the clock transition which occurs in zero external field, we will be setting $B_z = 0$ (C. Collett *et al.*, 2020). This means the Hamiltonian for one monomer can be expressed as

$$\mathcal{H}_\rangle = -D_i S_{zi}^2 + E_i(S_x^2 + S_y^2). \tag{5}$$

Individual molecules of Cr$_7$Mn can be combined into two molecule pairs, dimers, which interact through magnetic spin-spin interactions (Hore, 1999). This interaction functions as a bilinear exchange interaction, which introduces the following term to the Hamiltonian,

$$\mathcal{H}_{coupling} = \vec{S}_1 \cdot \hat{J} \cdot \vec{S}_2 \tag{6}$$

where $\vec{S}_1$ is the spin of qubit 1, $\vec{S}_2$ is the spin of qubit two, and $\hat{J}$ is the tensor which describes the coupling. $\hat{J}$ depends on the distance between the monomers, the linker that forms the dimer, the magnetic moments of the qubits, and will be determined empirically in the future. The dimer Hamiltonian can thus be expressed as

$$\mathcal{H}_{dimer} = \mathcal{H}_1 + \mathcal{H}_2 + \mathcal{H}_{coupling}, \tag{7}$$

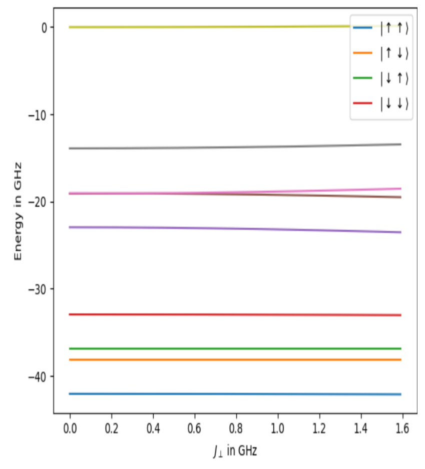
where $\mathcal{H}_1$ is the Hamiltonian for one monomer in the dimer, and $\mathcal{H}_2$ is the Hamiltonian for the second monomer. For the purposes of this thesis, we have chosen units where $\hbar = 1$, so the energy can be discussed in terms of frequency. In this form, we used the empirically determined constants $D_1 = 21\text{GHz}$, $E_1 = 1.95\text{GHz}$, $D_2 = 16.5\text{GHz}$, and $E_2 = 2.6\text{GHz}$ to define the individual Hamiltonians (C. Collett *et al.*, 2020).

As mentioned in the background section, one way of interacting with these spin states is through electron spin resonance. To achieve a transition, a radio frequency (RF) pulse is applied to the sample. Spin transitions are induced by applying RF pulses with the transition frequency set to be at or near the energy difference between quantum states. As seen in appendix A, specific transitions require certain spin operators. These spin operators define the axis of applied torque in the transition. These transitions act like rotations (see the Bloch sphere in figure 2 for a visualization), so to transition to a desired state from a specific state, the correct axis about which the torque is applied must be chosen. When the RF contribution to the spin Hamiltonian is added in, this yields the full expression for the Hamiltonian

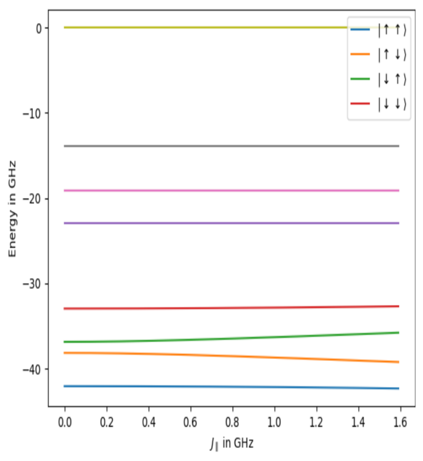
$$\mathcal{H}_{dimer} = \mathcal{H}_1 + \mathcal{H}_2 + \mathcal{H}_{coupling} + \mathcal{H}_{RF}. \tag{8}$$

How $\mathcal{H}_{RF}$ is constructed will be discussed further in the methods section.

The heterodimer is constructed in such a way as to have some degenerate energy transitions. However, as can be seen in equation 7, this degeneracy is broken by the coupling between the two molecules. The $\hat{J}$ term in equation 6 can be approximated by splitting it into an axial term and a transverse term. This value cannot exceed the value used for E without violating the assumptions leading the giant spin approximation. Therefore, we are restricted to look at $\hat{J}$ values less than E. The energy levels for our system are plotted against various axial coupling values is shown in figure 3a, and against transverse coupling values in figure 3b.



(a) Energy against transverse coupling



(b) Energy against axial coupling

Figure 3: The energy eigenstates for Cr_7Mn against the coupling values

This figure shows how the degeneracy could be broken, with greater differences in between degenerate states as the coupling strength is increased. Hinted at in the image is the fact that without coupling, there are several degenerate transition. Any transition in one qubit is unaffected by the state of the other, so for example the transitions $|--\rangle$ to $|+-\rangle$ is degenerate to $|-0\rangle$ to $|+0\rangle$. The first two of these transitions are explored in this thesis. With coupling, the degeneracy of these transtions is broken. For $|--\rangle$ to $|+-\rangle$ and $|-0\rangle$ to $|+0\rangle$ with an axial coupling value of 0.16GHz, there is a difference of 0.016GHz. This difference can be exploited to conditionally drive transitions in one molecule based on the state of the other. Since the only difference between the first transition and the second is the state of the second qubit, the difference in energy comes from the different coupling values when the qubit is in either the $|-\rangle$ state or the $|0\rangle$ state. Thus, the overall outcome is governed by the state of the second qubit. The first qubit is in both cases changing from a $|-\rangle$ state to a $|+\rangle$ state, performing an inversion. By definition, a CNOT acting on a qubit pair drives an inversion on one qubit, conditioned by the state of the other qubit. Thus, driving the $|-0\rangle$ to $|+0\rangle$ and not $|--\rangle$ to $|+-\rangle$ would act as a CNOT gate. By exploiting coupling, the gate can be achieved using active transitions. This has has the potential to be faster than the passive scheme used by C. Collett *et al.*, 2020, when similar coupling values are used.

Potential RF pulses to create this active gate were simulated using Quantum Tools in Python (QuTiP) (QuTiP, 2024). In order to evaluate the efficacy of these pulses, a figure of merit for the gate is needed. The one chosen here is fidelity,

$$\mathcal{F}(\rho_1, \rho_2) = tr\sqrt{\rho_1^{\frac{1}{2}}\rho_2\rho_1^{\frac{1}{2}}}, \tag{9}$$

where ρ_1 corresponds to a reference state, and ρ_2 to the end state of the simulation (Chiesa *et al.*, 2024). $\mathcal{F}$ functions by comparing the end state of the simulation to a known outcome. It is worth noting that these fidelities are calculated in the interaction picture. The interaction picture has both time evolving operators and states, and therefore removes the constant z precession.

The simulations solve the time dependent Schrodinger equation numerically. This can be done either by directly simulating the evolution of states given a time dependent Hamiltonian, or by simulating the transformation matrix that corresponds to that evolution, called the propagator (QuTiP, 2017). The propagator can be used to calculate the state evolution for any given input state with significantly less computation, so that is the method used in this thesis

3.1 Quantum Algorithms

To ensure that a gate will function as intended in a quantum computer, we can simulate quantum algorithms with this gate. If the algorithm runs as intended, then mathematically the gate must be achieving the same function as the ideal gate in the algorithm.

One commonly used quantum algorithm is phase estimation (Classiq, 2024). This uses the inverse quantum Fourier transform which, both in function and implementation, is the exact reverse of the quantum Fourier transform. Therefore, we begin with a discussion of the quantum Fourier transform.

The quantum Fourier transform performs the same function as the classical Fourier transform, but due to its implementation on a quantum computer it would be significantly faster. For an arbitrary state this can be expressed as,

$$\sum_{j=0}^{N-1} x_j \left|j\right\rangle \rightarrow \sum_{j=0}^{N-1} y_k \left|k\right\rangle . \tag{10}$$

Here the arrow represents the application of the quantum Fourier transform, where the quantum state $\left|j\right\rangle$ is transformed into the quantum state $\left|k\right\rangle$. x_j, the amplitude is transformed into the corresponding amplitude y_k. This can be understood with the same applications as the classical fourier transform. For example, $\left|j\right\rangle$ could be a state in time space, which could then be transformed into $\left|k\right\rangle$, some state in frequency space. The formal definition of the Fourier transform is given by

$$\left|j\right\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{2\pi ijk/2^n} \left|k\right\rangle , \tag{11}$$

where n is the number of qubits, such that $\left|0\right\rangle .... \left|2^n - 1\right\rangle$ represents the computational basis. From equation 11, one can arrive at the result for n qubits,

$$|j_1, j_2, \ldots, j_n\rangle = \frac{(|0\rangle + e^{2\pi i 0 j_n} |1\rangle)(|0\rangle + e^{2\pi i 0 j_{n-1} j_n} |1\rangle) \cdots (|0\rangle + e^{2\pi i 0 j_1 j_2 \ldots j_n} |1\rangle)}{2^{n/2}}, \tag{12}$$

Here we are transforming n arbitrary quantum states, much like we saw in equation 10, except that our state $|k\rangle$ is now represented in the various superpositions of $|0\rangle + |1\rangle$. For a full walk through of the algebra, please see Neilsen and Chuang, 2010.

Equation 12 will be useful when looking at quantum phase estimation (Neilsen and Chuang, 2010). By examining how rotation gates act upon each state, it can be shown that one can build up this representation by applying R_n rotation gates to each qubit. Following these rotations, swap operations (swapping the states of the two qubits in the operation) then need to be performed to reorder the qubits to the output to match equation 12. To achieve the inverse quantum Fourier transform these steps are just performed exactly in reverse. The inverse fourier transform is the heart of the quantum phase estimation algorithm. A brief explanation of this is below, and an in depth discussion can be found in Neilsen and Chuang, 2010.

The goal of the phase estimation algorithm is to determine the phase of a given controlled unitary operator, $\hat{U}$. As will be shown below, an operator of this description must have eigenvalues of the form $e^{2\pi i\varphi}$. The phase estimation algorithm produces an estimate for the value of φ (Neilsen and Chuang, 2010).

To begin a discussion of the quantum phase estimation algorithm, a controlled unitary operator, $\hat{U}$, is constructed. By definition,

$$U|\lambda\rangle = \lambda|\lambda\rangle, \tag{13}$$

where λ is the eigenvalue corresponding to the eigenvector $|\lambda\rangle$. Since $\hat{U}$ is unitary, if equation 13 is multiplied by its Hermitian conjugate we find,

$$\langle\lambda| U^\dagger U |\lambda\rangle = \lambda\lambda^* \langle\lambda|\lambda\rangle , \tag{14}$$

$$\langle\lambda|\lambda\rangle = \lambda\lambda^*, \tag{15}$$

$$1 = |\lambda|^2. \tag{16}$$

This necessitates that

$$\lambda = e^{i\varphi} \tag{17}$$

where φ is the phase (H. Gupta, 2021). So we now know that we are searching for φ, as it appears in

$$U |\lambda\rangle = e^{i\varphi} |\lambda\rangle . \tag{18}$$

The key to extracting φ is phase kickback. First a control qubit is prepared in an equal superposition of the $|0\rangle$ and $|1\rangle$ states using a Hadamard gate. Then the controlled U gate is applied to the target qubit in state $|\lambda\rangle$, with the resulting state of the combined control qubit and target qubit being

$$\frac{|0\rangle + e^{i\varphi} |1\rangle}{\sqrt{2}} |\lambda\rangle . \tag{19}$$

This is because it is a controlled operator being applied and the $|0\rangle$ state will not allow a transition, while the $|1\rangle$ state does. Since it is an eigenvector, the state, $|\lambda\rangle$, of the target qubit remains unchanged. Therefore the only change is the addition of a phase to the control qubit (H. Gupta, 2021).

In practice, because the phase value φ can always be represented by

$$\varphi = 2\pi\varphi', \tag{20}$$

where φ' has values only between 0 and 1, the phase φ can be represented as a binary fraction. A binary fraction is a way of representing a base ten fraction in binary. For example, the fraction $\frac{3}{4}$ is 0.11 as a binary fraction. There is a 1 in the first decimal place, corresponding to $\frac{1}{2}$ and a 1 in the second decimal place, corresponding to $\frac{1}{4}$. Adding these results in the fraction $\frac{3}{4}$. With a small leap in intuition, given this representation, it would be ideal for one qubit to contain one 'bit' of information in the representation of φ. If

$$\varphi_n = 0.\varphi_1\varphi_2\varphi_3..., \tag{21}$$

then one qubit would be required for φ_1, one qubit for φ_2 etc.

One way to see how this is achieved is to look at how a system with the control qubit in an equal superposition of $|0\rangle$ and $|1\rangle$ states behaves when the controlled operator is applied 2^j times. Here, j is a number between 0 and n (the number of qubits) -1. From equation 19, one can see that for 2^j applications the result would be

$$(\frac{|0\rangle + e^{i\varphi 2^j}|1\rangle}{\sqrt{2}})|\lambda\rangle . \tag{22}$$

Again, here $|\lambda\rangle$ is the state from the target qubit, and the rest of the expression is the state of the control qubit. From the concepts discussed above, this can be re expressed as

$$\frac{|0\rangle + e^{2\pi i\varphi' 2^j}|1\rangle}{\sqrt{2}}|\lambda\rangle . \tag{23}$$

Due to this computation being in base 2, multiplying by 2^j just moves the information in the j^{th} place to the left by j places in the the binary fraction expression. If we take for example equation 21, and multiplied by 2^2, the result would be

$$2^2\varphi_n = \varphi_1\varphi_2.\varphi_3\varphi_4... \tag{24}$$

It is worth noting the shift of the decimal point in the result. With this understanding, and following some simplification (H. Gupta, 2021), this results in

$$= \frac{|0\rangle + e^{2\pi i 0.\varphi_{j+1}\varphi_{j+2}\varphi_{j+3}\cdots\varphi_n}|1\rangle}{\sqrt{2}} |\lambda\rangle . \tag{25}$$

In order to set up for the last step of the algorithm, the controlled operation is applied to n qubits. This determines the resolution of the estimation, because each of these qubits contains the information for the $j+1^{th}$ place on the phase, in the first decimal place for the operator applied 2^j times. As mentioned earlier, $|\lambda\rangle$ can be factored out, since it is unchanged throughout the operation. All together, this results in the following state of the control qubit:

$$\frac{1}{2^{n/2}}(|0\rangle + e^{2\pi i 0.\varphi_n}|1\rangle)(|0\rangle + e^{2\pi i 0.\varphi_{n-1}\varphi_n}|1\rangle)...(|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2\varphi_2\cdots\varphi_n}|1\rangle). \tag{26}$$

Comparing this to equation 12, we can see they have the same form. Therefore, by applying the inverse Fourier transform we should be able to extract the information in phase bit by bit (Neilsen and Chuang, 2010). In this case the nth qubit will correspond to the nth decimal place of φ_n.

4 Methods

The primary methods for this thesis can be divided into two parts. The first of these is direct pulse level simulation, done in QuTiP, as described in the theory section (QuTiP, 2024). This method is used to find specific pulses tailored to our system. The second of these is a circuit level simulation in Qiskit (IBM, 2024). This allowed for easy implementation of quantum algorithms, but being higher level, did not use specific system-determined pulses.

For pulse level simulations, a Hamiltonian of the form of equation 7 is used to define the spin system. An RF pulse can be applied, the contribution of this term to the Hamiltonian can be expressed as

$$\mathcal{H}_{RF} = A(t)cos(\omega t)(S_{1i} + S_{2i}), \tag{27}$$

where $A(t)$ is the time dependent amplitude of the pulse, ω is the frequency used to drive a given transition, and $(S_{1i} + S_{2i}$ are the spin operators used to drive the transition. As discussed in Theory, a given transition needs a specific spin operator. Appropriate spin operators for transitions are listed in Appendix A. For their ease of simulation (and creation later in the lab), we simulated RF pulses in the form of square waves, with amplitudes

$$A(t) = 0.1\text{GHz}. \tag{28}$$

This amplitude is significantly higher than that used in C. Collett *et al.*, 2020. However, pulses of this strength are feasible with current amplifiers and higher amplitudes make it significantly easier to implement an active gate scheme on our system. Of interest were transitions whose degeneracy was broken by coupling. We investigated two schemes, alpha (α) using the following transitions: $|-0\rangle \rightarrow |+0\rangle$ and $|--\rangle \rightarrow |+-\rangle$ and the beta (β) scheme using the following transitions: $|+-\rangle \rightarrow |+0\rangle$ and $|--\rangle \rightarrow |-0\rangle$. The frequency used was the exact energy difference between the desired end state and desired start state. This corresponded to

$$\omega_\alpha = E_{+0} - E_{-0}, \tag{29}$$

and

$$\omega_\beta = E_{+0} - E_{+-}, \tag{30}$$

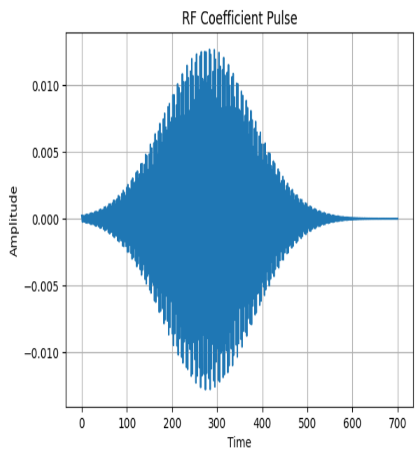
where E_n corresponds to the energy of n state, ω_α is the frequency used for transition pair α and ω_β is the frequency used for transition pair β.

We expect square pulses to be wide in frequency space, so to avoid driving nearby transitions we switched to a pulse which is narrower in frequency space. The pulses we chose to use were Gaussian pulses. These pulses were defined with the form,

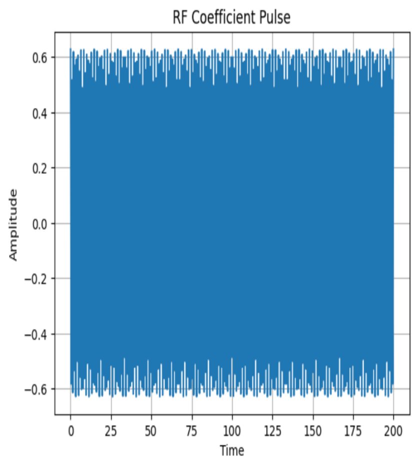
$$A(t) = w_1 e^{\frac{-(t-2d)^2}{d^2}}, \tag{31}$$

where w_1 is the amplitude, and d is the width. Appendix B contains an implementation of this pulse level simulation.

Both w_1 and d were varied when searching for the pulses with the highest fidelities. This search was conducted with an evolutionary algorithm which can be found in Appendix C. This algorithm is how the pulse presented in Results was found.



(a) Energy against time for a gaussian pulse, amplitude of 2.04MHz, and width set to 140ns.



(b) Energy against time for a square pulse, amplitude of 0.1, and width set to 40ns.

Figure 4: The shapes of both types of pulses used in this thesis

Figure 4 shows both of these shapes of pulses in time space, the Gaussian pulse in figure 4a and the square pulse in figure 4b.

The second type of method in this thesis is a higher level quantum circuit implementation, done in Qiskit (IBM, 2024). A quantum circuit is a set of qubits which are subsequently acted upon by a series of gates, and typically some part of the qubits' state is measured at the end of the circuit to see the effect of the gates. Rather than work at a pulse level, and simulate the pulses which act as gates, entire pre-made gates were applied. These gates are assumed to be perfect, and achieve exact transitions every time they were applied. This is a sharp difference from the gates made from pulses, which do not necessarily behave like a perfect gate. This method is how the quantum phase estimator was implemented to test the viability of the iCNOT gate. Figure 5 shows the circuit used in this phase estimation. It shows 3 qubits (q_0, q_1, q_2), which act as the control qubits and along with the eigenvector (c) have the controlled U gate applied to them 2^j times. This gate is given by `circuit-647` in the diagram. Shown also are the Hadamard gates at the start (`H`) to set up the equal superposition, and the inverse quantum Fourier transform (`IQFT_dg`), which are built into Qiskit.

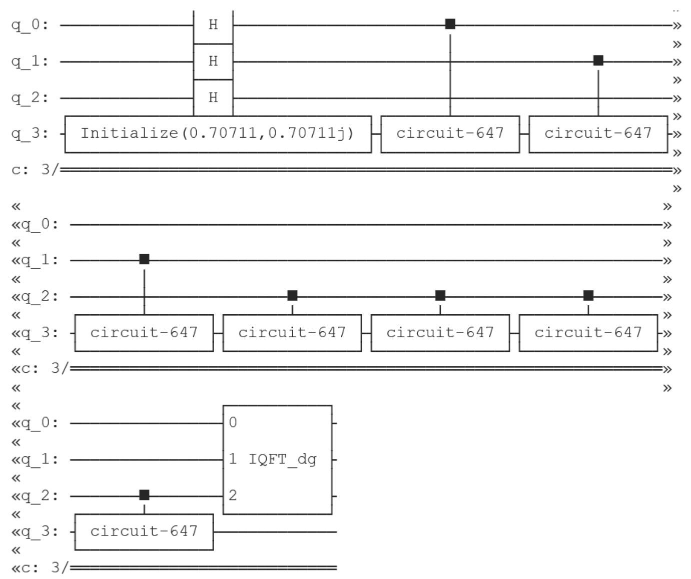


Figure 5: A diagram of the quantum circuit used for phase estimation

Crucially, here ideal gates were used, either Quskit's predefined gates, or the matrix defined iCNOT. The pulse level activities that were the subject of method one were not looked at. This means it was assumed we had the perfect way to implement every gate, and every gate was precisely and perfectly defined. This implementation will be discussed further in results, and the code can be found in Appendix D.

5 Results

We will begin by presenting the results from the QuTiP pulse simulations, attempting to find a CNOT gate with high fidelity. The first pulses tested were square waves set to drive the transition $|-0\rangle \rightarrow |+0\rangle$ (see figures 3a and 3b), while not driving the $|--\rangle \rightarrow |+-\rangle$ transition. In Figure 6 the probability with respect to time of the qubit being found in the desired $|+0\rangle$ state starting from the $|-0\rangle$ state is shown. In the plots below, the fidelity is plotted, which is defined in the Theory section. In a pure state, an intuitive mathematical definition is

$$Fidelity = |\langle\psi_a|\psi_b\rangle|^2 \tag{32}$$

where ψ_a is the simulated state, and ψ_b is the desired ideal state. The orange line is the fidelity that the qubit starting in $|-0\rangle$ will be found in the target state (in this case $|+0\rangle$). Here we show that the desired transition is taking place, with the final fidelity being 99.99%.

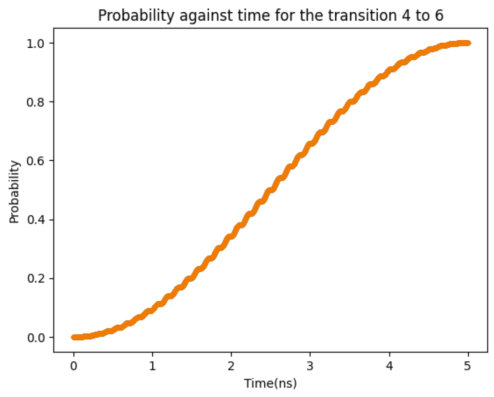


Figure 6: Simulated transition using a square wave shape, with the transition frequency set to be correct for a $|-0\rangle$ to $|+0\rangle$ transition, acting on a pure $|-0\rangle$ state

However, Figure 7 shows that a square wave is untenable. The undesired $|--\rangle$ to $|+-\rangle$ transition still has a 92.17% chance of occurring when driven by the frequency appropriate for a $|-0\rangle$ to $|+0\rangle$ transition in a square wave shape.

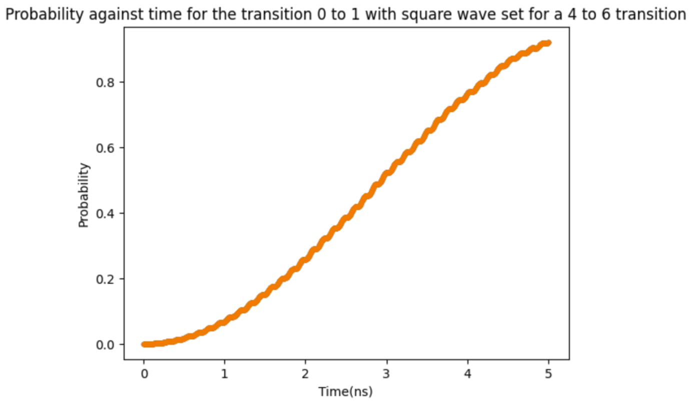


Figure 7: Simulated transition using a square pulse, with the transition frequency set to be correct for a $|-0\rangle$ to $|+0\rangle$ transition, acting on a pure $|--\rangle$ state

To avoid driving the non-target transition, Gaussian pulses were used. Gaussian pulses have a smaller spread in frequency space (and this can be tightened further by extending the pulse in time space). This allows a more precise targeting of a given transition. In all of the following Gaussian pulse trials, the pulse used had an amplitude of 2.04MHz, and a width of 140.5ns, which was pulse selected for its high fidelity using the evolutionary algorithm in Appendix C. In figure 8, the transition from a $|-0\rangle$ to $|+0\rangle$ is shown, comparing to a pure $|+0\rangle$ state, and driven by a Gaussian pulse with an energy tailored for this $|-0\rangle \rightarrow |+0\rangle$ transition. The fidelity of being in the $|+0\rangle$ state is 99.9%, showing a successful transition.

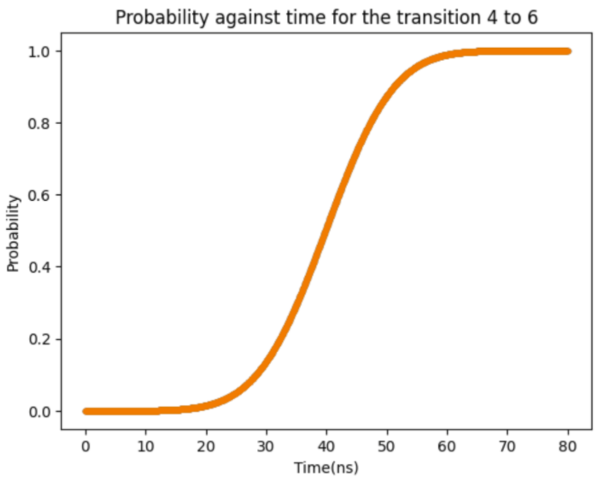


Figure 8: Pure state fidelity of driving a $|-0\rangle \rightarrow |+0\rangle$ transition when using a $|-0\rangle \rightarrow |+0\rangle$ frequency. The pulse width is 140.5ns, and amplitude 2.04MHz

With this set up, and in contrast to the square wave result, when this pulse is applied to a pure $|--\rangle$ state, the probability of transition to a $|+1\rangle$ state is only approximately 10%. This transition is shown in figure 9.

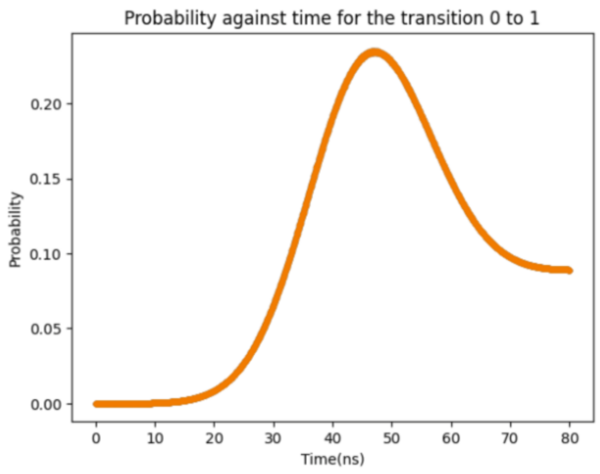


Figure 9: Pure state fidelity of driving a $|--\rangle \rightarrow |+-\rangle$ transition when using a $|-0\rangle \rightarrow |+0\rangle$ frequency. The pulse width is 140.5ns, and amplitude 2.04MHz

To check how well this pulse worked on a more general state, the same Gaussian pulse was applied to an equal superposition of the $|--\rangle$ and $|-0\rangle$ states. This is shown in figure 10. The fidelity of this transition is only 54.0%, represented as previously by the orange line. However, the blue represents the fidelity not in the interaction picture. Noticing that the fidelity does reach 100% in the amplitude suggests that the result from the pulse is differing by some phase to the predicted output.

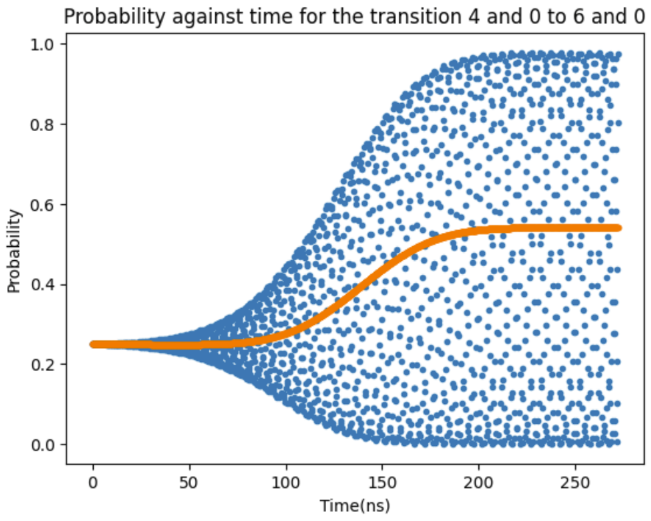


Figure 10: Transition from the super-positioned state of $|--\rangle$ and $|-0\rangle$ compared to a final state of $|--\rangle$ and $|+0\rangle$. The pulse width is 140.5ns, and amplitude 2.04MHz

This supposition is reinforced by examining the propagator corresponding to a Gaussian pulse with width 140.5ns, and amplitude 2.04MHz, which is given by equation 33.

$$P = \begin{bmatrix} -0.52+0.83j & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.96+0.24j & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.94-0.32j & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.47-0.88j & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.01+0.01j & 0 & 0.59+0.81j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.31+0.95j & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.59+0.81j & 0 & -0.01-0.01j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.31-0.95j & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{33}$$

This equation shows the propagator corresponding to the simulated Gaussian pulse with a width of 140.5ns, and an amplitude of 2.04MHz. The elements are presented to two significant figures. Considering in particular the off diagonals at [4,6] and [6,4] (where the first row and column are indexed as 0), which are the indices driving the energy transition, we see that they contain a non-negligible imaginary that differs from any global phase change. This indicates that there is a phase change being driven here as well as an energy transition.

Due to the realizations above, the simulation was rerun, but comparing the fidelity to the outcome that would be from a CNOT with $1j$ terms in the indices driving the transition (referred to as an iCNOT). The matrix used is shown below

$$iCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{34}$$

The fidelity of the pulse output compared to the ideal iCNOT ouptut acting on $\frac{1}{\sqrt{2}}(|--\rangle + |-0\rangle)$ is shown in figure 11. This figure shows that compared to the ideal output from a CNOT with a phase change, the pulse has a very high probability of driving the state $\frac{1}{\sqrt{2}}(|--\rangle + |-0\rangle)$ to the state $\frac{1}{\sqrt{2}}(|--\rangle + |+0\rangle)$, which is exactly the desired transition for a CNOT. The measured fidelity was 99.92%.

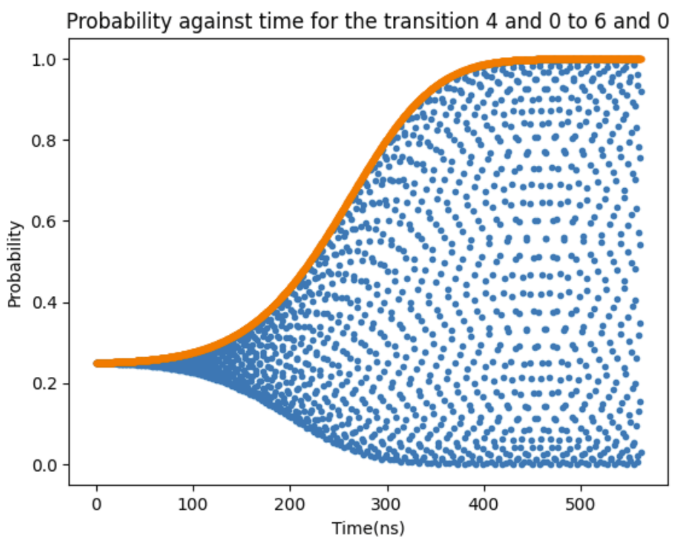


Figure 11: Fidelity of being in the state given by multiplying the iCNOT matrix above by the start state compared to the state due to the transition from the Gaussian pulse. The pulse width is 140.5ns, and amplitude 2.04MHz

In the case of these simulations, the quantum system was initialized in states that formed a mutually orthogonal basis set, so the entire vector space could be tested, for the possible superpositions of the states: $|--\rangle$, $|+-\rangle$, $|-0\rangle$, and $|+0\rangle$ (Fields and Wootters, 1989). The average of these fidelities was 99.88%.

An alternative scheme of driving the $|+-\rangle$ to $|+0\rangle$ transition, while not driving the $|--\rangle$ to $|-0\rangle$ transition was also investigated. These states have the same relationship as the previously described scheme, and the results of the application of the Gaussian pulse on an equal superposition of the $|--\rangle$ and $|+-\rangle$ states is shown below in figure 12.

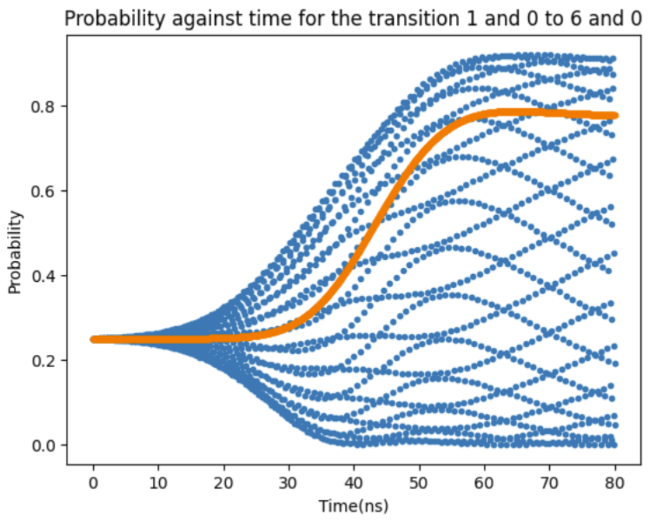


Figure 12: An equal superposition of the $|--\rangle$ and $|+-\rangle$ states being driven by the Gaussian pulse, compared to outcome of the iCNOT gate. The pulse width is 140.5ns, and amplitude 2.04MHz

Even when compared to the iCNOT, there was still only mediocre probability of being in the expected state ($|+0\rangle + |--\rangle$), and the amplitude not in the interaction picture never reached one. This hinted that there were more than differences in phase at play, and therefore this scheme was not pursued further.

6 Discussion

The results above indicate that a pulse has been found that has a very high fidelity for driving the $|-0\rangle \rightarrow |+0\rangle$ transition while not driving the $|--\rangle \rightarrow |+-\rangle$ transition. However, we have also shown that doing this transition with the Gaussian pulse shown in methods, with a width of 140.5ns and an amplitude of 2.04MHz, accumulates an extra phase from that of the traditional CNOT gate, creating what we are calling an iCNOT gate. To ensure that the iCNOT gate could function equivalently to the CNOT gate a quantum phase estimation algorithm (as described in the Quantum Algorithms portion of Theory) was implemented using the second type of simulation involving Qiskit. Described in full above, this method involved defining ideal gates in matrices, and then applying these to a quantum system. These were tested for the mutually orthogonal basis set of quantum states (Fields and Wootters, 1989). The simulation produced binary fraction values for the phase of the gate of interest, which were then converted into traditional fractions. These fractions correspond to the value of φ, the phase of the controlled operator that is being tested. Due to the probabilistic nature of the calculation, small differences in the proportion of the eigenvalues can be ignored. The results of phase estimation for the iCNOT and the CNOT are in table 1.

Input	Phase CNOT	Phase iCNOT	Phase Difference
[0, 1]	$(512)0 + (512)\frac{1}{2}$	$(504)\frac{1}{4} + (520)\frac{3}{4}$	$\frac{1}{4}$
[1, 0]	$(522)0 + (502)\frac{1}{2}$	$(511)\frac{1}{4} + (513)\frac{3}{4}$	$\frac{1}{4}$
$\frac{1}{\sqrt{2}}$[1, 1]	$(1024)0$	$(1024)\frac{1}{4}$	$\frac{1}{4}$
$\frac{1}{\sqrt{2}}$[1, -1]	$(1024)\frac{1}{2}$	$(1024)\frac{3}{4}$	$\frac{1}{4}$
$\frac{1}{\sqrt{2}}$[1, i]	$(523)0 + (501)\frac{1}{2}$	$(523)\frac{1}{4} + (501)\frac{3}{4}$	$\frac{1}{4}$
$\frac{1}{\sqrt{2}}$[1, -i]	$(510)0 + (514)\frac{1}{2}$	$(518)\frac{1}{4} + (506)\frac{3}{4}$	$\frac{1}{4}$

Table 1: Results of the phase estimation algorithm for both the CNOT and iCNOT gates. The fractions represent the phase and the coefficients in front of each fraction represent the number of individuals in the simulated population (total number = 1024) that had this phase.

Table 1 shows that no matter the input, there is a consistent $\frac{1}{4}$ phase difference between the two gates. This means that not only does the iCNOT gate function in a quantum algorithm as intended, and produces the expected eigenvalues for its eigenvectors, but it differs from the CNOT gate by a constant and therefore predictable phase. The iCNOT gate drives the $|-0\rangle$ to $|+0\rangle$ transition, and not the $|--\rangle$ to $|+-\rangle$ transition, as desired, and if doing a computation requiring phase, one could correct just by this constant factor of $\frac{1}{4}$. Thus, the iCNOT gate presented above in the form of a Gaussian pulse would be able to function as a CNOT gate.

As expected, this active scheme was significantly faster than the passive scheme initially proposed. The pulse found with highest fidelity had a gate duration of 140.5ns, while the passive scheme took 924ns (C. Collett *et al.*, 2020). This speed up would allow more many more gates to be operated within the same time period, which would significantly increase the value of η. There is also evidence that further speed increases are possible by optimizing pulses with shorter widths. There were numerous pulses in the 60ns range with fidelities of 97% before being fully optimized, leaving open the possibility of fidelities close to the 99% reported above.

The work presented here made several assumptions and idealizations of the qubits involved. One of these was that there was no decoherence. This could be introduced into the QuTiP simulation, and would provide a much more realistic picture of how this gate would work in practice. Including this, states would not stay perfectly the same value throughout the simulation, which would account for one of the greatest contributions to experimental error. The impact of this idealization is hopefully small because the effects of decoherence are already limited. This is due to the choice of an MNM with a clock transition, which maximizes the coherence time.

We also chose an arbitrary value for the coupling between spins. While this was in the expected range of greater than 0 and less than the E value (see equation 7), it has not yet been determined experimentally. Since this value will affect how quickly the phase changes, the specific pulse width and amplitude will have to be adjusted based on the actual coupling. This should be straightforward with an evolutionary search algorithm such as that presented in Appendix C. It is likely similar fidelities can be achieved across the possible range of coupling values. However, there is evidence that increasing the coupling value may decrease fidelity (C. Collett *et al.*, 2020), or possibly since our system uses the energy difference induced by coupling we may see a reduction in fidelity with lower coupling values. We do not expect there to be a significant reduction in fidelity but this requires more study.

These pulse level simulations are also limited in quantum computational scope. Only two qubits were used, far fewer than needed for a functional quantum computer. Additionally, the higher energy spin states (not the ground state) which could be used for quantum error correction were not explored. However, this project exists as a proof of concept for the viability of specific MNMs as qubits (using Cr_7Mn as our example MNM). As such, the work here presents a successful active scheme that would perform the role of a CNOT gate for a system using Cr_7Mn molecules as qubits. Along with the existing gates for the Cr_7Mn system we already have, the iCNOT gate presented here, with its significantly faster time to act, improves the potential viability of Cr_7Mn as a qubit.

The next steps of this project will include testing quantum algorithms from a pulse level, verifying that our exact pulse will function as an iCNOT gate, not only the idealized iCNOT as shown here. Additionally, adding in decoherence and testing a variety of coupling strengths to the simulations will provide more convincing evidence as to the viability of this system.

References

Chiesa, A, P Santini, E Garlatti, F Luis, and S Carretta (2024). “Molecular nanomagnets: a viable path toward quantum information processing?” In: *Reports on Progress in Physics* 87.

Classiq (2024). *Quantum Phase Estimation (QPE).* Available at: `https://www.classiq.io/insights/quantum-phase-estimation-qpe`.

Collett, Charles, Paolo Santini, Stefano Carretta, and Jonatha Friedman (2020). "Constructing clock-transition-based two-qubit gates from dimers of molecular nanomagnets". In: *Physical Review Research*.

Collett, Charles A., Sofia M. Davvetas, Abdulelah Alsuhaymi, and Grigore A. Timco (2024). *An Inexpensive, Configurable Two-Tone Electron Spin Resonance Spectrometer*. arXiv: `2407.21782 [physics.ins-det]`. Available at: `https://arxiv.org/abs/2407.21782`.

Fields, Brian and William Wootters (1989). "Optimal state-determination by mutually unbiased measurements". In: *Annals of Physics*.

Garlatti, Elena, Morten A. Albring, Michael L. Baker, Rebecca J. Docherty, Hannu Mutka, Tatiana Guidi, Victoria Garcia Sakai, George F. S. Whitehead, Robin G. Pritchard, Grigore A. Timco, Floriana Tuna, Giuseppe Amoretti, Stefano Carretta, Paolo Santini, Giulia Lorusso, Marco Affronte, Eric J. L. McInnes, David Collison, and Richard E. P. Winpenny (2014). "A Detailed Study of the Magnetism of Chiral Cr7M Rings: An Investigation into Parametrization and Transferability of Parameters". In: *Journal of the American Chemical Society*.

Google (2024). *AlphaQubit tackles one of quantum computing's biggest challenges*. Available at: `https://blog.google/technology/google-deepmind/alphaqubit-quantum-error-correction/`.

Gupta, Aman and Devesh (2022). *Implementing Quantum Phase Estimation Algorithm Using Qiskit*. Available at: `https://darveshiyat.medium.com/implementing-quantum-phase-estimation-algorithm-using-qiskit-e808e8167d32`.

Gupta, Harshit (2021). *Intro to QPE and Phase Encoding — QPE algorithms*. Available at: `https://medium.com/quantum-untangled/intro-to-qpe-and-phase-encoding-qpe-algorithms-15638a8554b9`.

Hore, P.J. (1999). *Spin-Spin Coupling.* Available at: `https://www.sciencedirect.com/topics/physics-and-astronomy/spin-spin-coupling`.

IBM (2024). *Qiskit.* Available at: `https://www.ibm.com/quantum/qiskit`.

Neilsen, Micheal and Isaac Chuang (2010). *Quantum computation and quantum information.* Cambridge University Press.

QuEra (2023). *Universal Gate Set.* Available at: `https://www.quera.com/glossary/universal-gate-set`.

QuTiP (2017). *quitip.propagator.* Available at: `https://qutip.org/docs/4.0.2/modules/qutip/propagator.html`.

— (2024). *QuTiP.* Available at: `https://qutip.org/`.

Rao, Ravi (2024). *Breakthroughs in Quantum Computing.* Available at: `https://www.wevolver.com/article/breakthroughs-in-quantum-computing`.

Roundy, Jacob (2023). *Explore 7 future potential quantum computing uses.* Available at: `https://www.techtarget.com/searchdatacenter/tip/Explore-future-potential-quantum-computing-uses`.

Schneider, Josh and Ian Smalley (2024). *What is a Qubit.* Available at: `https://www.ibm.com/topics/qubit`.

Williams, Colin (2011). *Explorations in Quantum Computing.* Springer London.

A List of possible transitions

Provided below is a list of all of the possible transitions, with the transitions' energies and spin operators. Given below are the states (corresponding to $-, 0, or +$), and also the numerical value which is commonly used in the python libraries and code in Appendices B and C.

Table 2: Table of all available transitions with transition frequencies (with no coupling) and spin operators

Transition (states)	Transition (numerical value)	Spin Operator	Frequency of transition
$0+ \rightarrow 00$	$7 \rightarrow 8$	S_y (or S_x)	$E_1 + D_1$ (S_x with $D_2 - E_2$)
$++ \rightarrow +0$	$3 \rightarrow 6$	S_x	$D_2 - E_2$
$-+$ (or $+-$) $\rightarrow$-0	$2 \rightarrow 4$	S_x	$D_2 - E_2$
$0-$ (or $+0$) $\rightarrow 00$	$5 \rightarrow 8$	S_y	$E_2 + D_2$
$+-$ (or $-+$) $\rightarrow +0 (or 0$-)	$1 \rightarrow 6$	S_y	$E_2 + D_2$
$-- \rightarrow$-0	$0 \rightarrow 4$	S_y	$E_2 + D_2$
$+0$ (or $0-$) $\rightarrow 00$	$6 \rightarrow 8$	S_x	$D_1 - E_1$
$++ \rightarrow 0+$	$3 \rightarrow 7$	S_x	$D_1 - E_1$
$+-$ (or $-+$) $\rightarrow$0-$(or+0)$	$1 \rightarrow 5$	S_x	$D_1 - E_1$
$0-$ (or $+0$) $\rightarrow 0+$	$5 \rightarrow 7$	S_z	$2E_2$
$+-$ (or $-+$) $\rightarrow ++$	$1 \rightarrow 3$	S_z	$2E_2$
$-- \rightarrow$-+$(or+$-)	$0 \rightarrow 2$	S_z	$2E_2$
$-+$ (or $+-$) $\rightarrow ++$	$2 \rightarrow 3$	S_z	$2E_1$
$-- \rightarrow$+-$(or$-+)	$0 \rightarrow 1$	S_z	$2E_1$
$-0 \rightarrow +0(or0$-)	$4 \rightarrow 6$	S_z	$2E_1$
$-0 \rightarrow 00$	$4 \rightarrow 8$	S_y	$E_1 + D_1$
$-- \rightarrow$0-$(or+0)$	$0 \rightarrow 5$	S_y	$E_1 + D_1$
$-+$ (or $+-$) $\rightarrow 0+$	$2 \rightarrow 7$	S_y	$E_1 + D_1$

B QuTiP implementation of pulse comparison

Presented below is the python function which was used to test a given transition against the ideal iCNOT matrix.

```
def GaussianPropagatorSuperI(width, amp, nstart1, nstart2, nend1, nend2,
    a, base):
   """Function to compare a given evolution of a super positioned state
```

```
    to the output an iCNOT would give
This is specifically for testing with a 4-6 transition. Initial
    structure from Daniel Rodriguez.
Arguments are:
   width: width of pulse applied
   amp: Amplitude of pulse applied
   nstart1: Start state
   nstart2: Start state - combination of above creates a 50-50
       superposition
   nend1: End state 1 (The transition frequency is determined by
       nend1-nstart1)
   nend2: End state 2
   a: modifies the width of the pulse applied (larger a gives a
       smaller pulse in frequency space)
   base: defines the coeffcients of the superpositioned state

"""
#Define w1, to keep notation consistent
w1 = amp

#Define Hamiltonian with no coupling and zero field
Ham_dimer = df.cr_h_s1(D1, E1, D2, E2, J_a, J_t)



#rf operator. Dependent on transition.
Ham_dimer_rfop2 = Sz1+Sz2

#Define rf coefficient term for our Hamiltonian
def Ham_dimer_rf_coeff(t, args):
   w1 = args['w1']
```

```
    w = args['w']

    return (w1)*np.cos(w*t)*np.exp((-(t-2*width)**2)/(a*width)**2)


#Define full Hamiltonian

Ham_dimer_full = [Ham_dimer, [Ham_dimer_rfop2, Ham_dimer_rf_coeff]]


#Current states and energies

dimer_energies, dimer_estates = Ham_dimer.eigenstates()


#Frequency to verify

dimer_w = dimer_energies[6]-dimer_energies[4]


dimer_args = {

'w1': w1,

'w': dimer_w

}


#Number of time points

tlen = 2001


#Define the list of times over which the simulation will be run. 2nd
    parameter = end point

tlist = np.linspace(0, 4*width , tlen)


#Define the propagator

prop = qt.propagator(Ham_dimer_full, tlist, args=dimer_args)


a = base[0]

b = base[1]

c = base[2]
```

```
d = base[3]

# Initializing real numbers
x = 0
y = -1.0

# converting x and y into complex numbers
i = complex(x, y)

#Define the iCNOT matrix
CNOT_z = qt.Qobj([[1,0,0,0,0,0,0,0,0],
                [0,1,0,0,0,0,0,0,0],
                [0,0,1,0,0,0,0,0,0],
                [0,0,0,1,0,0,0,0,0],
                [0,0,0,0,0,0,i,0,0],
                [0,0,0,0,0,1,0,0,0],
                [0,0,0,0,i,0,0,0,0],
                [0,0,0,0,0,0,0,1,0],
                [0,0,0,0,0,0,0,0,1]],
               dims = [[3, 3], [3, 3]])
#Transform the matrix into the energy eigenbasis
CNOT = CNOT_z.transform(dimer_estates, True)

#Initial state
dimer_psi0 = a*dimer_estates[nstart1] + b*dimer_estates[nstart2] +
    c*dimer_estates[nend1] + d*dimer_estates[nend2]
#Final state to be compared to
EndState = CNOT*(a*dimer_estates[nstart1] + b*dimer_estates[nstart2]
    + c*dimer_estates[nend1] + d*dimer_estates[nend2])
```

```
#Array of evolved states, calculated with the propagator
dimer_ev_states = [prop[n]*dimer_psi0 for n in range(len(prop))]

#Z-axis noise reduction operator
def U(t):
    return (-1j*Ham_dimer*t).expm()

#Noice reduced states
dimer_ev_states_inter = [U(t).dag()*state for t, state in zip(tlist,
    dimer_ev_states)]
#Probability calculator function using expression for fidelity.
dimer_flip_probsz = [[np.abs(EndState.overlap(st))**2 for st in
    state]
                    for state in zip(dimer_ev_states,
                        dimer_ev_states_inter)]

#Plot
fig, ax = plt.subplots()
ax.set_title('Probability against time for the transition
    '+str(nstart1)+ ' and '+ str(nstart2)+' to '+str(nend1)+' and '+
    str(nend2))
ax.set_xlabel('Time(ns)')
ax.set_ylabel('Probability')
ax.plot(tlist, dimer_flip_probsz, '.');

#Print the last point in the array, so the overall fidelity of the
    transition can be reported
mpoint = np.array(dimer_flip_probsz)[:,1].argmax()
print("probability of overlap = ",
    str(100*dimer_flip_probsz[2000][1]) + "%")
    return prop, dimer_estates
```

C QuTiP implementation of pulse evolutionary algorithm

Below is the code used as the evolutionary algorithm. This is how the pulse of width of 140.5ns, amplitude of 2.04MHz and fidelity of 99.9% was found.

```
import sys
sys.path.append('../../')
import numpy as np
import qutip as qt
import matplotlib.pyplot as plt
import dimerfuncs as df
from qutip.qip.operations import rotation
import cmath
import random
import copy

#Initialize all constants
#Parameters of our sample, some variation to be expected.
D1 = 21*2*np.pi
E1 = 1.95*2*np.pi
D2 = 16.5*2*np.pi
E2 = 2.6*2*np.pi
```

```
#Define operators corresponding to this configuration

Sx1, Sy1, Sz1, Sx2, Sy2, Sz2 = df.two_spin_system(1)


#Coupling Values

J_a = 1

J_t = 0


#Define Hamiltonian with no coupling and zero field

Ham_dimer = df.cr_h_s1(D1, E1, D2, E2, J_a, J_t)


#Current states and energies

dimer_energies, dimer_estates = Ham_dimer.eigenstates()

#Coefficients which define the superposition

a = 1/np.sqrt(2)

b = 1/np.sqrt(2)


#Initial state

dimer_psi0 = a*dimer_estates[0] + b*dimer_estates[4]


# Initializing real numbers

x = 0

y = -1.0


# converting x and y into complex number

i = complex(x, y)


#Define the iCNOT matrix

CNOT_z = qt.Qobj([[1,0,0,0,0,0,0,0,0],

                [0,1,0,0,0,0,0,0,0],

                [0,0,1,0,0,0,0,0,0],
                [0,0,0,1,0,0,0,0,0],
                [0,0,0,0,0,0,i,0,0],
                [0,0,0,0,0,1,0,0,0],
                [0,0,0,0,i,0,0,0,0],
                [0,0,0,0,0,0,0,1,0],
                [0,0,0,0,0,0,0,0,1]],
               dims = [[3, 3], [3, 3]])
#Transform the iCNOT matrix intro the energy eigen basis
CNOT = CNOT_z.transform(dimer_estates, True)

#Final state
EndState = CNOT*(a*dimer_estates[0] + b*dimer_estates[4])

#Number of time points
tlen = 2001

#Define the pulse class
class Pulse:

    def __init__(self, width, amp, freq):
        self.width = width
        self.amp = amp
        self.freq = freq
        self.tlist = np.linspace(0, 4*self.width , tlen) #Second
            parameter = end point
        self.prop = self.getProp()
        self.fidelity = self.fidelityCalc()

    def __repr__(self):
        return f"Width: {self.width:.2f} | Amplitude: {self.amp:.5f} |
            Fidelity: {self.fidelity:.2f}\n"

    def pulsePartOfHamDimer(self, t):
        """Define the pulse part of the dimer Hamiltonian
        """
        pulse =
            (self.amp)*np.cos(self.freq*t)*np.exp((-(t-2*self.width)**2)/(self.width)**2)
        return pulse

    def getProp(self):
        """Find the propagator for a given Hamiltonian
        """
        #rf operator. Dependent on transition.
        spinOp = Sz1+Sz2

        dimer_args = {
            'w1': self.amp,
            'w': self.freq
        }

        #Current states and energies
        dimer_energies, dimer_estates = Ham_dimer.eigenstates()

        #Define full Ham
        Ham_dimer_full = [Ham_dimer, [spinOp, self.pulsePartOfHamDimer]]

        #Define the propagator
        prop = qt.propagator(Ham_dimer_full, self.tlist, args=dimer_args)

        return prop
    def fidelityCalc(self):
        """Calculate the fideltity for a given state
        """
        dimer_ev_states = [self.prop[n]*dimer_psi0 for n in
            range(len(self.prop))]

        #Z-axis noise reduction operator
        def U(t):
            return (-1j*Ham_dimer*t).expm()

        #Noice reduced states
        dimer_ev_states_inter = [U(t).dag()*state for t, state in
            zip(self.tlist, dimer_ev_states)]
        #Expected final state (Used to measure overlap (success of
            transition).
        dimer_end_statez = EndState
        #Probability calculator function. Takes in the evolved states
            and measures how much they overlap with end state
        dimer_flip_probsz = [[np.abs(dimer_end_statez.overlap(st))**2
            for st in state]
                       for state in zip(dimer_ev_states,
                           dimer_ev_states_inter)]

        return 100*dimer_flip_probsz[2000][1]

#Set up the actual evolutionary algorithm using the pulse class

n = 20 #number of individuals in a population
ratio = 0.8 #ratio of winners to randos in a population
```

```
generations = 5 # number of generations of evolution
tournamentSize = 4 # number of tournaments to be held
stepSizeWidth = 1 # how much to change the width by when mutating
stepSizeAmp = 0.0001 # how much to change the amplitude by when mutating
allowedWidths = np.arange(60, 80, 0.1).tolist() #range of allowed widths
allowedAmps = np.arange(0, 1, 0.0001).tolist() #range of allowed
    amplitudes

#Current states and energies
dimer_energies, dimer_estates = Ham_dimer.eigenstates()

#Frequency to verify
dimer_w = dimer_energies[6]-dimer_energies[4]

def pulseRandos():
    """Function to create pulses with random widths and amplitudes
    Creates pulses only with the allowed width and amplitudes
    """
    width = random.choice(allowedWidths)
    amp = random.choice(allowedAmps)
    pulse = Pulse(width, amp, dimer_w)
    return pulse

def tournament(pulses, size):
    """Function which holds a tournament between a certain number of
        pulses
    The pulse with the highest fidelity is declared the winner
    pulses: List of current pulses
    size: size of the tournament to be held
    """
    competitors = [None] * size
    for i in range(size):
        competitors[i] = random.choice(pulses)
    winner = max(pulses, key=lambda pulse: pulse.fidelity)
    return winner

def ampMutate(amp):
    """Function which mutates the amplitude
    Randomly chooses whether to increase or decrease the amplitude
    amp: Amplitude of the pulse
    """
    coinFlip = random.randint(0, 1)
    if coinFlip == 0:
        amp += (random.randint(1, 50) * stepSizeAmp)
    else:
        amp -= (random.randint(1, 50) * stepSizeAmp)
    return amp

def widthMutate(width):
    """Function which mutates the width
    Randomly chooses whether to increase or decrease the width
    width: Width of the pulse
    """
    coinFlip = random.randint(0, 1)
    if coinFlip == 0:
        width += (random.randint(1, 50) * stepSizeWidth)
    else:
        width -= (random.randint(1, 50) * stepSizeWidth)
    return width
```

```
def mutate(pulse):
    """Function which decides how to mutate a pulse
    Based off a 3 way coin flip either mutates amplitude, width or both
    pulse: the pulse the be mutated
    """
    coinFlip = random.randint(0, 2)
    amp = pulse.amp
    width = pulse.width
    freq = pulse.freq
    if coinFlip == 0:
        newAmp = ampMutate(amp)
        newWidth = width
    elif coinFlip == 1:
        newWidth = widthMutate(width)
        newAmp = amp
    else:
        newAmp = ampMutate(amp)
        newWidth = widthMutate(width)
    mutatedPulse = Pulse(newWidth, newAmp, freq)
    return mutatedPulse

#Initialize a list of random pulses
pulses = [pulseRandos() for _ in range(n)]
#Initialize arrays
probability = [None] * n
bestPulses = [None] * generations

#For the number of generations (gen) run the evolutionary algorithm
for gen in range(generations):
    print(f" This is generation {gen}")
    currentBest = max(pulses, key=lambda winner: winner.fidelity)
    bestPulses[gen] = currentBest
    print(f"This is the current best pulse {bestPulses[gen]}")

    #sending pulses from generation to tournament, with competitions of
        size x
    #Decide who gets to evolve
    evolvers = [tournament(pulses, 5) for _ in range(int(ratio * n))]
    #Evolve the winning pulses
    mutated = [mutate(pulse) for pulse in evolvers]
    print(mutated)
    #Replenish the gene pool with random pulses
    randoms = [pulseRandos() for _ in range(n - int(ratio * n))]
    #Send back to start again the evolved pulses and some new random ones
    pulses = mutated + randoms

print(bestPulses)
overallBest = max(bestPulses, key=lambda pulse: pulse.fidelity)
print(f" This is the overall best {overallBest}")
```

D Qiskit implementation of the Quantum Phase Estimation algorithm

Below is the code used to simulate the quantum phase estimate algorithm for the CNOT and iCNOT gates. The starting point for this code can be found at A. Gupta and Devesh, 2022.

```
from qiskit_aer.primitives import SamplerV2
from qiskit_aer import AerSimulator
from qiskit import *
#from qiskit.jupyter import *
from qiskit.visualization import *
import numpy as np
from qiskit.circuit.library import QFT
from qiskit.quantum_info import Operator

def my_qpe(w_qubits,s_qubits, gate, initial_state = None, trotter_number
    = 1):
    """Function which returns the result of the quantum phase estimation
        algorithm
    w_qubits: Number of ancillae qubits
    s_qubits: Number of qubits needed to initialize the state of the
        eigenvector
    gate: The gate which the QPE is evaluating
    initial_state: Starting state of the simulation
    trotter_number: Time scale for the trotterization
    """
    repetitions=1
    #Initializes the Quantum cicuit with the ancillae and simulation
        qubits
    qpe_0 = QuantumCircuit(w_qubits+s_qubits,w_qubits)
    if (initial_state != None):
        #Initializes the state
        qpe_0.initialize(initial_state,list(range(w_qubits,w_qubits+s_qubits)))
    for i in range(w_qubits):
        qpe_0.h(i)
    #Performs the trotterization
    for j in range(trotter_number):
        for counting_qubit in range(w_qubits):
            #Performs the U^2j repititions
            for i in range(repetitions):
                qubit_list =
                     [counting_qubit]+list(range(w_qubits,w_qubits+s_qubits))
                qpe_0.append(gate,qubit_list)
            repetitions *= 2
        repetitions = 1
    #Uses the inbuilt QFT gate in Qiskit
    qpe_1 = QFT(w_qubits, 0, True , True)
    l = [*range(w_qubits)]
    #Final result, composed IQFT result
    qpe = qpe_0.compose(qpe_1, l)
    return qpe

#Code which runs the QPE on a CNOT (easily switched to an iCNOT)
simulator = AerSimulator()



trotter_number = 1
t = 1

# Initialize the quantum circuit with the appropriate number of qubits
#w = ancillae qubits
w_qubits = 3
#s = number of qubits needed to initialize the eigenvector
s_qubits = 1

initial_state = [1/np.sqrt(2), 1.0j/np.sqrt(2)]
```

```
#Defining the Not matrix, and the iNot (which will become the CNOT and
    iCNOT respectively)
inot_matrix = np.array([[0.0, 1.0j],
                      [1.0j, 0.0]])
not_matrix = np.array([[0.0, 1.0],
                      [1.0, 0.0]])
#Initialize the circuit
cir = QuantumCircuit(1)
#Make sure the gate is unitary
cir.unitary(not_matrix, [0])
#Create a controlled gate out of either the iNot or Not gate
CNOT_gate = cir.to_gate().control(1)

# Build the QPE circuit
qpe = my_qpe(w_qubits, s_qubits, CNOT_gate, initial_state=initial_state)

# Draw the QPE circuit
display(qpe.draw())

# Perform the measurement on the QPE circuit (using the correct qubits)
qpe.measure([0, 1, 2], [0, 1, 2])

# Transpile the QPE circuit before running on the simulator
qpe_transpiled = transpile(qpe, simulator)

# Run the simulator with the QPE circuit
result = simulator.run(qpe_transpiled).result()

# Get counts from the QPE circuit
counts = result.get_counts(qpe_transpiled)
```

```
# Plot the histogram of the counts

plot_histogram(counts, title='Counts from the simulation')
```