Quantum Benchmarking on the [[4,2,2]] Code

Atsushi Hu, Joey Li, Rebecca Shapiro

July 2018

Abstract

For roughly two decades, theoretical quantum algorithms have been known to provide significant speedup over classical techniques for certain classes of problems. However, building reliable quantum computers in the real world has been challenging due to problems of decoherence in computation. Thus, the field of quantum error correction is extremely important. In this paper, we describe the results of two experiments conducted on real quantum computers available through the IBM Q Experience that experimentally examine the benefits of quantum error coding. We find that carefully designed fault-tolerant operators for the [[4,2,2]] code decrease the infidelity of computation by a factor of ten, from 2.08 to 0.19. We also confirm that in the [[4,2,2]] code, centralizing operators provide no significant benefits over normalizing operators.

1 Real Randomized Benchmarking

While quantum error-correcting codes can theoretically reduce problems of decoherence, they have not been tested extensively on real quantum computers. The first goal of this project was to benchmark the [[4,2,2]] CSS code on a 5-qubit computer made available through IBM's Quantum Experience in order to show experimentally that computations in the encoded space could have lower infidelity than computations in the uncoded space. In particular, we demonstrate the advantage of fault-tolerant encoded operators across different lengths of circuits. Following the experimental procedures outlined in [1], we replicated their results, confirming a factor of ten decrease in infidelity, from 2.08 to 0.19, when performing computation in the [[4,2,2]] code.

1.1 Experimental Design

The procedure for both the uncoded and encoded implementations was as follows:

1. Choose a set of circuits lengths $\{l_i\}$. For each l_i, sample $l_i - 1$ logical gates chosen randomly from the Realizable Group $R(2)$, defined later, and compose these into a circuit.

2. Add an appropriate inversion gate at the end so that the function of the circuit is to perform a net identity operation.
3. Run the circuit through IBM's quantum computer for 1024 shots, and determine the success rate.
4. Repeat the previous steps while varying the logical input to the system by compiling in II, XI, IX, XX operators into the circuit. Run each circuit with its specific set of inputs nine times.
5. Repeat the process again in the same manner while adding a phase gate at the beginning of the circuit and an inverse phase gate at the end. This changes the effect operator E, and gives information necessary for calculation of the average gate fidelity.

Physical Gates	Logical Gates
$X \otimes I \otimes X \otimes I$	$X \otimes I$
$X \otimes X \otimes I \otimes I$	$I \otimes X$
$Z \otimes Z \otimes I \otimes I$	$Z \otimes I$
$Z \otimes I \otimes Z \otimes I$	$I \otimes Z$
$H \otimes H \otimes H \otimes H$	$SWAP_{12} \circ (H \otimes H)$
$P \otimes P \otimes P \otimes P$	$(Z \otimes Z) \circ (CZ)$
$SWAP_{12}$	$CNOT_{12}$
$SWAP_{13}$	$CNOT_{21}$

Table 1: Set of Fault-Tolerant Gates

1.2 Implementation Details

The Realizable Group $R(2)$ is the subgroup of real Clifford elements generated by the gates specified in Table 1. The randomly chosen gates in this paper were sampled from $R(2)$ for several reasons. First, $R(2)$ is an orthogonal 2-design. Second, $R(2)$ is a small subset of the Cliffords, with only 576 elements as compared to 1,152 in the Real Clifford group and 11,520 in the overall Clifford Group on 2 qubits. Moreover, each element in $R(2)$ is generated by an average of 4 generators, as compared to 7 for the Real Clifford group. Finally, there are simple fault-tolerant implementations of Realizable Group generators, as listed in Table 1 above. For more thorough discussion of the choice of $R(2)$ for this benchmarking procedure, see [1].

All experiments were run on the IBM Q 5 Tenerife, or ibmqx4. IBM quantum computer chips have a life cycle that is relevant to experiment accuracy; the ibmqx4 is calibrated twice a day (at approximately 8AM and 8PM EST), and reports the lowest readout error rate and gate error rate immediately after calibration. Therefore, we tried to start experiments right after the machine was calibrated. Moreover, since there is a limit on the job size, the runs were broken down according to circuit lengths and multiple jobs were submitted.

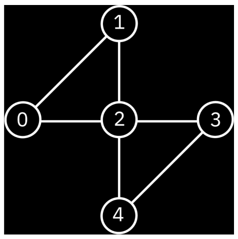


Figure 1: connectivity of qubits on ibmqx4

The lengths of circuits we ran were 6, 15, 24, 42 (corresponding to 5, 14, 23, 41 logical gates). For each length, we compiled in 4 different inputs (II, XI, IX, XX) and repeated the circuit 9 times for each compiler. We conducted phased and unphased runs, and we did all of this for uncoded and encoded qubits, for a total of $4 \times 4 \times 9 \times 2 \times 2 = 576$ experiments, each running 1024 shots.[1]

The data gathered was used to determine an average success rate $\bar{q}$. For uncoded runs, $\bar{q}$ was determined for each run by dividing the number of correct outputs by 1024. For encoded runs, $\bar{q}$ was determined for each run by dividing the number of correct outputs by the number of outputs inside the code space (results outside the code space were discarded as part of the error-detection procedure). [1] proves that the average success rate can be fit into the model

$$\bar{q}(m, E, \rho) = A + b^m B + c^m C \tag{1}$$

where m is the number of logical gates, which is equal to circuit length minus , ρ is the initial state, and E is an effect operator of a POVM. The value of A is provided by [1] to be 0.25. B and C are constants

$$B = \mathrm{Tr}[E\rho_+] \text{ and } C = \mathrm{Tr}[E\rho_-],$$

where $\rho_\pm = \frac{1}{2}(\rho \pm \rho^T)$. Per [1], for two qubits, the average fidelity is given by

$$F(\varepsilon) = (9b + 6c + 5)/20. \tag{2}$$

In order to determine the values of b and c, we chose appropriate ρ to eliminate B and C, one at a time, in the equation. In the computational basis $|00\rangle\langle 00|$, $\rho_- = 0$ and thus $C = 0$. This means that if the operation is done in the computational basis, the success rate $\bar{q}$ is modeled by the equation $\bar{q}(m, |00\rangle\langle 00|) = 0.25 + b^m B$. To find out the value of C, we need to rotate ρ. To achieve this, we apply a phase gate followed by a Hadamard gate. Since phase gate cannot be applied fault-tolerantly in the code space, we apply a non-fault-tolerant phase gate first and apply an encoded Hadamard gate. These are inverted after the all the logical gates are applied and $|00\rangle$ is measured in the normal way.

[1] The code used in the experiment is available at `https://github.com/Octophi/quantum_rb`.

1.3 Results

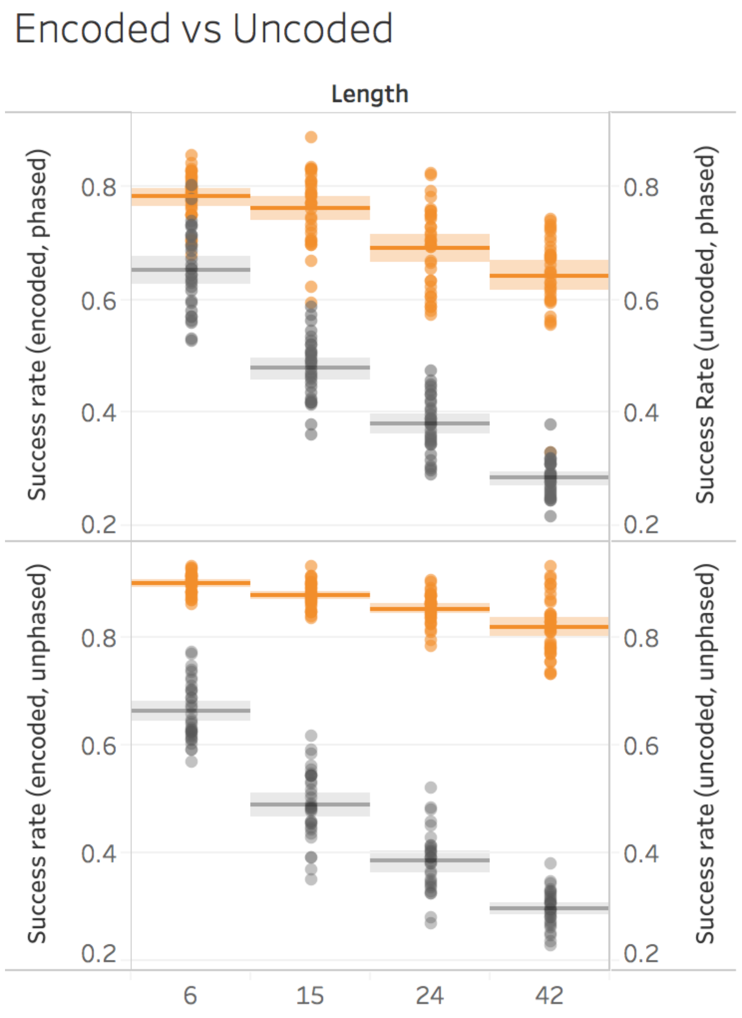


Figure 2: Encoded vs uncoded

Figure 2 compares the effect of encoding the qubits. Based on calculations using equation 1 and 2, the infidelity rate is $F(\varepsilon) = 0.19\%$ for the [[4,2,2]] code and the infidelity rate is $F(\varepsilon) = 2.08\%$ for the two physical qubits[2]. This difference was about a factor of ten and shows the significance of encoding.

However, since we conducted the post-selection of the results, it is unclear how much improvement comes from the error-detecting ability of the [[4,2,2]] code and how much comes from the $CNOT$ gate in the codespace. To solve

[2]The data can be found online at `https://github.com/Octophi/quantum_rb`.

this problem, we conduct calculations in which the results that are not in the codespace are not discarded. This creates a direct comparison between the performance of codes with error detection and codes without error detection. This leads to a calculated infidelity rate of 0.51%, which is only slightly higher than the infidelity rate after discarding the results outside the codespace. Comparing this result to the infidelity given by the uncoded circuits (2.08%), we can tell that most of the improvement comes from error-detecting and a small amount of improvement is from not using a noisy $CNOT$ gate.

2 Benchmarking Centralizing versus Normalizing Solutions

When constructing a logical operator $\bar{h}$ on a code, we want to ensure that the operator preserves the code space. Mathematically, this condition can be expressed succinctly by the statement $\bar{h} \in N_{U_N}(S)$, the normalizer of the stabilizer group for our code within the $N \times N$ unitary matrices. However, this condition does not necessarily fix the subspaces constructed by the code. In order to prevent this permutation of subspaces, we can impose the stricter condition that $\bar{h}$ belong to $C_{U_n}(S)$, the centralizer.[3] Though theoretically this provides more structure to syndrome measurements, it may come at the cost of a more complex implementation.

[2] recently presented an algorithm for generating all normalizing and centralizing solutions for a Clifford gate associated with a given encoding scheme. Using the MATLAB code presented in this paper to generate normalizing and centralizing solutions, we benchmarked the performance of each type of implementation for the [[4,2,2]] code on the ibmqx4 quantum computer. Since the [[4,2,2]] code is error-detecting and not error-correcting, we would expect that centralizing implementations would show no advantage, since both normalizing and centralizing implementations preserve the code space, and all results outside the code space were automatically discarded without regard to syndrome. This is in fact, exactly what we found.

2.1 Experimental Procedure

The procedure was carried out as follows:

1. Choose a logical operator and obtain centralizing and normalizing implementations of the operator within the [[4,2,2]] code using the MATLAB code at `https://github.com/nrenga/symplectic-arxiv18a`

2. For both the set of centralizing solutions and the set of normalizing solutions, choose a representative circuit with the smallest number of gates.

[3]For a more thorough discussion of the theory behind centralizing and normalizing solutions for logical operators, see section B.3.

Additionally, consider the implementation of the operator in uncoded space.

3. For each implementation of an operator, construct a corresponding circuit.

4. For each implementation and for each possible logical input, chosen from $\{00, 01, 10, 11\}$, run the corresponding circuit 9 times for 1024 shots each, taking the average of the success rates.

2.2 Implementation Details

Because the experiment was only meaningful when the shortest normalizing implementation of an operator was distinct from the shortest centralizing implementation of an operator, the only operator we benchmarked so far was the CZ gate. Each circuit consisted simply of state preparation, the implemented operator, and then measurement in the standard basis. Each operator had three corresponding circuits: the centralizing implementation, the normalizing implementation, and the uncoded implementation. As with the first experiment, each circuit was run 9 times for each of 4 inputs, and success rate was calculated out of 1024 or out of the number of outcomes in the code space, as appropriate. All experiments were run on the ibmqx4. Unlike with the real randomized benchmarking procedure, the data was gathered at more scattered times throughout the day, not necessarily right after IBM's system calibrations. As it turned out, the differences between the implementations were generally large enough to be significant even accounting for potential issues of noise.

2.3 Results

		Input States				
	Implementation	00	01	10	11	**Avg**
Uncoded	CZ_{12}	0.95	0.86	0.90	0.82	**0.88**
Normalized	$P_4 \circ P_3 \circ P_2 \circ P_1$	0.89	0.87	0.90	0.87	**0.88**
Centralized	$Z_4 \circ CZ_{23} \circ CZ_{13} \circ CZ_{12}$	0.72	0.75	0.74	0.75	**0.74**

Table 2: Success Rates for Implementations of CZ Gate on Two Qubits

The data corroborated our expectation that centralizing gates would show no inherent advantage over normalizing gates in the [[4,2,2]] code, as seen in Table 2. Rather, in every experiment, the simplest gate performed the best, and this is to be expected, because generally, shorter depth circuits are less likely to incur errors.

The data in these tables may be interpreted as showing that uncoded operations do in fact show higher fidelity than encoded operations. This is likely generally true for individual gates; however, a key point of quantum error correction which was demonstrated in the first experiment is that encoding is useful precisely for long computations. Not only can operations be made fault-tolerant and sequences of gates simplified when computations are conducted in encoded space, but errors can be detected and/or corrected, which, as our other experiment showed, has a significant impact.

2.4 Future Work

This experiment should be extended to error-correcting codes in order to test the potential benefits of centralizing implementations. We made considerable progress on building a framework for error correction coding in the [[5,1,3]] code, following the flagged syndrome extraction procedure outlined in [3]. However, as of the time of writing, such experiments cannot be run because IBM's publicly available quantum computers currently do not support measurement in the middle of a circuit, a crucial component to error correction. Further investigation of this problem will be carried out when such functionality is added. We propose that any such experiment compare fault-tolerant error correction methods with non fault-tolerant ones, as well as compare normalizing operators against centralizing operators, particularly over varying lengths of circuits.

References

[1] R. Harper and S. Flammia. Fault tolerance in the IBM Q Experience. *ArXiv e-prints*, June 2018.

[2] N. Rengaswamy, R. Calderbank, S. Kadhe, and H. D. Pfister. Synthesis of Logical Clifford Operators via Symplectic Geometry. *ArXiv e-prints*, March 2018.

[3] R. Chao and B. W. Reichardt. Quantum error correction with only two extra qubits. *ArXiv e-prints*, May 2017.

[4] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

A Background

A.1 Introduction to Heisenberg Weyl

A fundamental algebraic group is the dihedral group D_4, the group of the symmetries of the square, which we can describe as the group with presentation

$$\langle X, Z \mid X^2 = Z^2 = 1, XZ = Z^{-1}X \rangle.$$

We may check that if we take

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

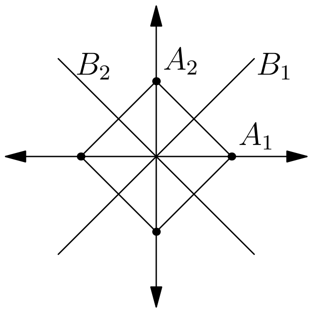


Figure 3: Symmetries of the Square

we may describe this group as a matrix group. Geometrically, X corresponds to reflection in axis B_1 and Z corresponds to reflection in axis A_1 in figure 3.

An important mapping related to the group D_4 is the Walsh Hadamard Transform, given by

$$H_2 = H_2^T = H_2^{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

which geometrically corresponds to rotation anticlockwise by $\pi/4$ and then reflection in B_1. This transform is important because it takes X to Z and vice versa under the action of conjugation:

$$H_2^{-1} X H_2 = Z \qquad H_2^{-1} Z H_2 = X.$$

Naturally, it also interchanges the eigenvectors A_1, A_2 for Z with the eigenvectors B_1, B_2 for X. As an exercise, prove that the map $g \mapsto H_2^{-1} g H_2$ describes an automorphism of D_4.

A.2 Kronecker Products

Given a $p \times p$ matrix $X = [x_{ij}]$ and a $q \times q$ matrix $Y = [y_{ij}]$, the Kronecker product $X \otimes Y$ is defined by

$$X \otimes Y = \begin{bmatrix} x_{11}Y & \dots & x_{1p}Y \\ \vdots & \ddots & \vdots \\ x_{p1}Y & \dots & x_{pp}Y \end{bmatrix}.$$

The important idea of Kronecker products is that they allow us to "multiply" together objects in a way that preserves the individual information of the objects. A fundamental fact of Kronecker products that will repeatedly come up is the following equality:

$$(x \otimes y)(x' \otimes y') = (xx') \otimes (yy') \tag{3}$$

Now given a real or complex vector of length 2^m, we may index its coordinates from left to right as 0 to $2^m - 1$, and label the i^{th} coordinate by the binary expansion of i. Then let e_v be the standard unit vector with 1 in coordinate $v \in \mathbb{F}_2^m$ and 0 everywhere else.

For example, we have $e_0 = (1,0), e_1 = (0,1)$, and

$$e_{110} = (00000010) = (0,1) \otimes (0,0,1,0) = e_1 \otimes e_1 \otimes e_0.$$

We may prove that, in general, for binary m-tuple $v = (v_{m-1} \dots v_0)$,

$$e_v = e_{v_{m-1}} \otimes \cdots \times e_{v_0}.$$

A.3 Time Shifts and Frequency Shifts

Given a binary m-tuple $a = (a_{m_1} \dots a_0)$, define a $2^m \times 2^m$ matrix $D(a,0)$ by

$$D(a,0) = X^{a_{m_1}} \otimes \cdots \otimes X^{a_0}$$

Then we may define the time shift group $X_N == \{D(a,0) \mid a \in \mathbb{F}_2^m\}$. The time shift group is useful because we have defined it so that $e_v D(a,0) = e_{v+a}$, which you should prove as an exercise.

We may analogously define the frequency shift group. Let $b = (b_{m_1} \dots b_0)$ and then define $D(0,b)$ by

$$D(0,b) = Z^{b_{m_1}} \otimes \cdots \otimes Z^{b_0}.$$

The frequency shift group is $Z_N = \{D(0,b) \mid b \in \mathbb{F}_2^m\}$, and we have the nice property that $e_v D(0,b) = (-1)^{bv^T} e_v$.

Finally, given binary m-tuples a and b, define $D(a,b) = D(a,0)D(0,b)$. Now since $D_4 = \{\pm I_2, \pm X, \pm Z, \pm XZ\}$, we see that every matrix $D(a,b)$ can be written as a Kronecker product

$$\pm g_{m-1} \otimes g_{m-2} \otimes \cdots \otimes g_0$$

where $g_i \in \{I_2, X, Z, XZ\}$. The Heisenberg Weyl group HW_N, where $N = 2^m$, is the set of all matrices $\pm D(a,b), \pm iD(a,b)$ where $i = \sqrt{-1}$.

We can prove a number of useful statements about the Heisenberg Weyl group. Here are a few:

1. HW_N is a group with respect to matrix multiplication and $D(a,b)D(a',b') = (-1)^{a'b^T + b'a^T} D(a',b')D(a,b)$.

2. $D(a,b)^T = (-1)^{ab^T} D(a,b)$.

3. Elements of HW_N have order 1,2, or 4, and we may characterize elements of each order.

4. The conjugacy classes of HW_N are all size 1 or 2.

5. A subgroup H of HW_N is normal if and only if $-I_N \in H$.
6. $HW_N/\langle \pm iI_n \rangle \cong \mathbb{F}_2^{2m}$

Let $E(a,b) = i^{ab^T} D(a,b)$ for all $a, b \in \mathbb{F}_2^m$. Then we may prove several more facts:

1. $E(a,b)$ is Hermitian and $E(a,b)^2 = I_N$ for all $a, b \in \mathbb{F}_2^{2m}$
2. If $E(a,b)$ and $E(c,d)$ commute, then
$$E(a,b)E(c,d) = \pm E(a+c, b+d)$$
and if they do not commute, then
$$E(a,b)E(c,d) = \pm iE(a+c, b+d).$$
3. The matrices $\frac{1}{\sqrt{N}}E(a,b)$ form an orthonrmal basis for the real vector space of $N \times N$ Hermitian matrices
4. The only eigenvalues of $E(a,b)$ are ± 1.
5. Let V_+ and V_- be the eigenspaces of $E(a,b)$ corresponding to the eigenvalues 1 and -1 respectively. Then
$$\dim(V_+) = \dim(V_-) = 2^{m-1} = N/2.$$

Finally, revisiting the Walsh Hadamard Transform, we see that we may generalize the Walsh Hadamard Matrix by letting $H_N = \underbrace{H_2 \otimes \cdots \otimes H_2}_{m \text{ times}}$. Then we define Walsh functions as rows of H_N and their negatives, ie,
$$\hat{e}_b = e_b H_N.$$
Then we may prove that Walsh functions are eigenvectors of binary time shifts, and thus are the analogue of spikes under the Walsh Hadamard Transform.

B Quantum Error Correction

Quantum computation is built on qubits, the quantum analogue of classical bits. Physically, there are many different possible implementations for these qubits, such as th espin of an electron or an ion. However, mathematically, a *qubit* is a 2 dimensional Hilbert space over $\mathbb{C}$. A *pure state* is a ray in Hilbert space, usuall written as a unit norm complex vector
$$v = \alpha e_0 + \beta e_1 \text{ with } \alpha, \beta \in \mathbb{C} \text{ and } |\alpha|^2 + |\beta|^2 = 1.$$
Intuitively, we may think of the qubit as existing in some sperposition of the 0 state (e_0) and the 1 state (e_1).

A state of m qubits may be written as a vector in $\mathbb{C}^{2^m} = \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2$. Then a pure state of m qubits would be written

$$x = \sum_{v \in \mathbb{F}_2^m} \alpha_v e_v \text{ with } \alpha_v \in \mathbb{C} \text{ and } \sum_{v \in \mathbb{F}_2^m} |\alpha_v|^2 = 1.$$

A von Neumann measurement of a quantum state begins with a resolution of the identity, namely, t Hermitian operators $P_1, \ldots, P_t$ such that

$$P_i P_j = \begin{cases} 0, & \text{if } i \neq j \\ P_i, & \text{if } i = j \end{cases}$$

and

$$P_1 + \cdots + P_t = I_{2^m}.$$

When we measure $v \in \mathbb{C}^{2^m}$ with respect to $P_1, \ldots, P_t$ we obtain $z = \frac{1}{\sqrt{v^\dagger P_i v}} v^\dagger P_i$ with probability $v^\dagger P_i v$ and we learn the index i of the projection that occurs. Notice that von Neumann measurement cannot distinguish two states that are scalar multiples of each other.

Entanglement refers to the phenomenon where states exist which cannot be written as pure Kronecker products. The prototypical example is the Bell state or EPR pair $v = \frac{1}{\sqrt{2}}(e_{00} + e_{11})$. Entanglement is considered a resource in quantum computation because it can provide some extra information. For example, if we knew we had an EPR pair, and we measured the first qubit, we would be able to determine the state of the second qubit without measuring it.

Gates in quantum computation correspond mathematically to unitary operators, and quantum circuits are simply compositions of quantum gates. The advantage of quantum computation lies in the idea of quantum parallelism, which allows a quantum computer to explore exponentially many potential solutions to a problem simultaneously. Roughly, the idea is that quantum algorithms are designed to take in a uniform superposition as input, perform some some sequence of unitary transformations to increase the probability of measuring a desired solution from this superposition, and then measure the result. Quantum algorithms such as Grover's Algorithm and Shor's Algorithm have been known to provide significant speedup over classical techniques for over two decades.

The problem with quantum computing, however, is the prevalence of decoherence errors. Because errors inevitably build up over the course of a computation, they can render a theoretically correct algorithm useless in practice because of noise. Correspondingly, the field of quantum error correction has received increasing attention over time.

In the quantum error model, we genreally assume that errors come from the group HW_N. While it may seem odd to focus on a discrete group of errors when quantum errors are theoretically arbitrary, this assumption is justified because the elements of HW_N forma basis for all errors that will occur in practice.

Theorem 1. *Any code that corrects errors from HW_N will be able to correct errors in arbitrary models, assuming that the errors are not correlated among large numbers of qubits, adn that the error is small.*

To see how an error-correcting code is designed, it is most instructive to start with an example. Suppose we are asked to protect the information in a single qubit, and only against single bit-flip errors. We may do this with the bit flip code, as explained below.

We begin with the following map:

$$|0\rangle \rightarrow |000\rangle$$
$$|1\rangle \rightarrow |111\rangle$$

Thus, we have mapped a two-dimensional vector into eight-dimensional space. For the bit-flip code, we then specify what we call our *stabilizer group*, which we will take to be $S = \langle Z_1Z_2, Z_2Z_3\rangle$. Notice each of our stabilizer generators is an element from the general error group, and furthermore, each splits eight-dimensional space into equally-sized positive and negative eigenspaces. Therefore, the using the two generators, we can split the eight-dimensional space once into two four-dimensional spaces, and then again into four two-dimensional spaces. Then based our encoding and choice of stabilizer elements, our two-dimensional vector of information lies in one of the four two-dimensional spaces. In particular it lies in the space that is the intersection of the positive eigenspaces of the stabilizer generators, which we call the code space. Furthermore, we have chosen our stabilizer generators carefully so that any single bit-flip error will take the information into a unique subspace other than the code space. Then by measuring our information to the stabilizer generators, we can determine which subspace we lie in, and then apply correction terms as appropriate to fix the error and return to the code space.

	Measure		
State	$D(000, 110)$	$D(000, 011)$	Diagnosis
$\alpha\,|000\rangle + \beta\,|111\rangle$	+	+	No error
$\alpha\,|100\rangle + \beta\,|011\rangle$	-	+	Flip first qubit
$\alpha\,|010\rangle + \beta\,|101\rangle$	-	-	Flip second qubit
$\alpha\,|001\rangle + \beta\,|110\rangle$	+	-	Flip third qubit

Table 3: Summary of bit flip code error correction procedure

We can use an analogous procedure to protect against phase flip errors. The main difference is that we will choose our stabilizer to be $S = \langle x_1x_2, x_2x_3\rangle$ instead. Continuing, we might ask whether we can protect qubits from all errors rather than just bit flip errors or just phase flip errors.

The answer of course, is yes, and there is a very natural procedure for doing so based on what we have seen already. We simply begin by encoding our single qubit with three qubits in the bit flip code. Following, we encode each of those three qubits using another three qubits in the phase flip code. This encoding procedure can be summarized in the following circuit:

This encoding scheme is called the Shor code, and was historically the first indication that quantum error correcting codes analogous to those in classical coding theory could be designed.

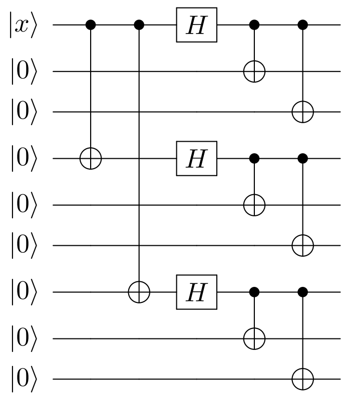


Figure 4: Shor Code Encoding Circuit

B.1 Stabilizer Codes

Stabilizer codes generalize the idea of the bit flip code and Shor code. However, construction of stabilizer codes is somewhat different. To construct a stabilizer code, we begin by choosing some elements $g_1, \ldots, g_i \in HW_N$, which we call our stabilizer generators. Then the stabilizer group is given by $S = \langle g_1, \ldots, g_i \rangle$, and correspondingly, the code space is given by

$$V_s \triangleq \{|\psi\rangle \mid g\,|\psi\rangle = |\psi\rangle \;\forall g \in S\}.$$

To check that an element does belong in V_s, we clearly only have to check if it is stabilized by each of the generators. This construction of S is useful because it allows us to detect errors using the same method as the bit flip code: measure a quantum state against each of the stabilizer generators, determine the subspace we lie in, and then perform a quantum operation to correct as necessary.

This process of determining the subspace our information lies in is analogous to syndrome decoding procedures in classical linear codes. In fact, stabilizer codes are best thought of as the quantum equivalent of linear codes. For example, there is also a valid interpretation of the generator matrix for stabilizer codes. Using the map γ given in Theorem 1, we may describe the stabilizer generators as rows of a generator matrix. For example, for the bit flip code, our generator matrix would be

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

since we may write $z_1 z_2 = D(000, 110)$ and $z_2 z_3 = D(000, 011)$.

Then it is clear that the stabilizer group corresponds to the rowspace of the generator matrix. This structure makes it easier to analyze the behavior of stabilizer codes. As with linear codes, carefully choosing the elements of our stabilizer generator is crucial to designing effective codes.

To discuss different quantum error correcting codes, it is useful to employ a naming convention that is similar to the one for classical coding. We generally describe a quantum code as a $[[m,\ m-k,\ d]]$ code, where m refers to the number of physical qubits the code is implemented on, k is the dimension of the stabilizer group, and d is the distance of the code. While the significance of $m-k$ is not immediately obvious, note that the dimension of our information qubits is equal to the total dimension of the physical space minus the dimension of the stabilizer group. We must also define distance in quantum codes. We choose the natural definition: the *distance* of a quantum code is the minimum *weight* of an error taking one element of the code to another element of the code, where weight is measured by the number of non-identity elements in an error. Thus, for example, the reader should check that the Shor code described above is a [[9,1,3]] code.

One other connection between quantum codes and classical codes worth exploring is the quantum Hamming bound, the analogue of the sphere-packing bound in classical coding. Recall that the sphere-packing bound states that for an $[[n,k,d]]$ code, we need

$$2^n \geq 2^k \cdot \sum_{i=0}^{\frac{d-1}{2}} \binom{n}{i}.$$

In the case of quantum stabilizer codes, we see that the total dimension of our physical qubits is 2^m and the dimension of our information qubits is 2^{m-k}. Then the number of an elements in a t-sphere around a quantum state is $\sum_{i=0}^{e} 3^i \binom{2^m}{i}$, and so we get the bound

$$2^m \geq 2^{m-k} \cdot \sum_{i=0}^{t} 3^i \binom{2^m}{i}$$

for a quantum [[m,m-k]] code which can correct t errors.

In particular, we may consider the smallest number of physical qubits necessary to protect a single qubit from single errors. Then we need $t=1$ and $m-k=1$, and we obtain the inequality

$$2(3m+1) \leq 2^m,$$

which implies $m \geq 5$, with equality precisely when $m=5$. In fact, we can indeed find a code which uses 5 physical qubits to protect 1 logical qubit from single errors, which we call the [[5,1,3]] perfect code and which has stabilizer $\langle XZZXI, IXZZX, XIXZZ, ZXIXZ \rangle$. Analogous to classical codes, quantum codes are called perfect when they satisfy equality in the quantum Hamming bound.

B.1.1 CSS Codes

CSS codes, named after Calderbank, Shor, and Steane, are a special class of stabilizer codes that are of particular interest in quantum computing for a few reasons. First, CSS codes with specific error-correcting capabilities may be constructed from known classical codes. Second, this construction provides a systematic procedure for determining the the encoding schemes, which are not always easy to determine for stabilizer codes. Finally, the stabilizer elements in CSS codes are all pure products of X's and Z's.

To construct a CSS code, one begins with two classical codes C_1 and C_2 on m bits such that $C_2 \subset C_1$ and C_1 and $C_2^\perp$ both correct t errors. Then we take our stabilizer to be given by

$$G = \begin{bmatrix} & G_1^\perp \\ G_2 & \end{bmatrix}$$

where G_2 is a generator matrix for C_2 and $G_1^\perp$ is a generator matrix for $C_1^\perp$.

Theorem 2. *This code corrects t errors.*

Proof. To see this is true, consider the distance of the code. We want the minimum weight vector (e, f) orthogonal, with respect to the symplectic inner product, to everything in G. This gives the equation

$$\begin{bmatrix} & G_1^\perp \\ G_2 & \end{bmatrix} \begin{bmatrix} f \\ e \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Then we must have that $e \in C_1$ and $f \in C_2^\perp$, and since both C_1 and $C_2^\perp$ can correct t errors, at least one of e and f must contain at least $2t+1$ nonzero entries, as desired. $\square$

It should be clear from this construction why the stabilizer is made of elements with only X's or only Z's. To see how we may explicitly write down an encoding scheme, let us consider a specific example.

Example B.1. [[4,2,2]] CSS Code

Take C_1 to be the classical [4,3,2] single parity check code and C_2 to be the [4,1,4] repetition code. We can take corresponding generators

$$G_1 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$G_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

Notice that here we actually have $C_2 = C_1^\perp$, which is not generally necessary for CSS codes. Then we have the stabilizer generator matrix

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

Because $C_2 \subset C_1$, we may form another generator matrix for C_1/C_2, given by

$$G_{C_1/C_2} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Now we can provide an explicit encoding given by the map

$$|\mathbf{x}\rangle \to \frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |\mathbf{x} \cdot G_{C_1/C_2} + c\rangle \, .$$

For example, the system $|00\rangle$ would be encoded as $\frac{1}{\sqrt{2}}(|0000\rangle + |1111\rangle)$ and the system $|10\rangle$ would be encoded as $\frac{1}{\sqrt{2}}(|0101\rangle + |1010\rangle)$. With some effort, one can check that this specified map works in general by checking that the stabilizers fix these states.

The [[4,2,2]] code specifically will reappear throughout the rest of this paper and thus a full table of encodings is given for reference:

Information	Encoded State
$\lvert 00\rangle$	$\frac{1}{\sqrt{2}}(\lvert 0000\rangle + \lvert 1111\rangle)$
$\lvert 10\rangle$	$\frac{1}{\sqrt{2}}(\lvert 0101\rangle + \lvert 1010\rangle)$
$\lvert 01\rangle$	$\frac{1}{\sqrt{2}}(\lvert 1100\rangle + \lvert 0011\rangle)$
$\lvert 11\rangle$	$\frac{1}{\sqrt{2}}(\lvert 0110\rangle + \lvert 1001\rangle)$

Table 4: Encoding Scheme for [[4,2,2]] Code

B.2 Classifying Errors

At this point, it is fruitful to stop consider what kind of errors can occur in a stabilizer code. In the following discussion, it will be helpful to refer to the tree of groups depicted in [2]. We classify our errors into three types:

- *Invisible Errors*: Suppose an error occurs and it is drawn from the group S. Then this error is in fact one of our stabilizer elements, and does not affect our code at all.

- *Indistinguishable Errors*: Suppose an error occurs an it is drawn from the group $S^{\perp}$. Then because this error normalizes our stabilizer, a syndrome measurement will not indicate any error, though our data will have changed. Indistinguishable errors cannot be fixed by syndrome measurement.

- *Correctable Errors*: Suppose an error lies in HW_N but not in $S^{\perp}$. Then by syndrome measurement, we may detect the error, and have hope to correct it.

In sum, there are a class of errors that cannot be corrected by stabilizer codes. The goal is for our code to be effective enough in correcting the other two types of errors to provide significant advantage over uncoded computations.

B.3 Implementing General Logical Operators

Before discussing how specifically to implement logical Cliffords, we can establish some conditions on general logical operators. When constructing a logical operator on a code, we want to ensure that the operator preserves the code space. In other words, given some logical operator $\overline{h}$, we always want

$$g\overline{h}\,|\overline{\psi}\rangle = \overline{h}\,|\overline{\psi}\rangle$$

for all $g \in S$. Noting that $g'\,|\overline{\psi}\rangle = |\overline{\psi}\rangle$ for all $g' \in S$, this condition may be rewritten as

$$g\overline{h}\,|\overline{\psi}\rangle = \overline{h}g'\,|\overline{\psi}\rangle$$

and so we need that for all $g \in S$, there exists $g' \in S$ with

$$g\overline{h} = \overline{h}g'.$$

Then $\overline{h} \in N_{U_n}(S)$.

Notice that this condition does not necessarily fix the subspaces constructed by the code. For example, suppose we had stabilizer $S = \langle g_1, g_2, g_3, g_4\rangle$ and state $|\overline{\psi}\rangle$ such that $g_1\,|\overline{\psi}\rangle = -\,|\overline{\psi}\rangle$ and $g_i\,|\overline{\psi}\rangle = |\overline{\psi}\rangle$ for $i = 2, 3, 4$. Then given an $\overline{h}$ that only normalizes S, this state $|\overline{\psi}\rangle$ lying in the $(-, +, +, +)$ subspace could be sent to $\overline{h}\,|\overline{\psi}\rangle$ lying in the $(+, -, +, +)$ subspace.

In order to prevent this permutation of subspaces, we can impose the stricter condition that $\overline{h}$ belong to the centralizer $C_{U_n}(S)$. In other words, we can require $\overline{h}g = g\overline{h}$ for every $g \in S$. Though theoretically this provides more structure to syndrome measurements, It is not yet clear what tangible benefits this restriction could provide to quantum error correction coding.

B.4 Finding Logical Cliffords

In order to utilize a stabilizer code with k stabilizer generators, it is necessary to find for each logical operator on $m - k$ logical qubits an analogous physical operator acting on m physical qubits. In addition to realizing the action of its logical counterpart, a physical operator must also preserve the code space. We used results from [2] as a means to synthesize logical Cliffords for our research.

The first step in synthesizing logical Cliffords is to define the logical Pauli operators. The logical Paulis are defined by their action on a logical state:

$$\begin{aligned} X_j^L\,|\underline{x}\rangle_L &= |\underline{x}'\rangle\,, x_i' = \begin{cases} x_j \oplus 1, & \text{if } i = j \\ x_i, & \text{if } i \neq j \end{cases} \\ Z_j^L\,|\underline{x}\rangle_L &= (-1)^{x_j}\,|\underline{x}'\rangle\,. \end{aligned}$$

For CSS codes, finding the physical Pauli operators $\bar{X}_j, \bar{Z}_j, j = 1, \ldots, m$ is a matter of choosing generator matrices for the classical generating code C of the form

$$G_C = \begin{bmatrix} H_C \\ G_{C/C^\perp} \end{bmatrix}.$$

We can choose two variations $G^X_{C/C^\perp}$ and $G^Z_{C/C^\perp}$, where

$$G^X_{C/C^\perp}\left(G^Z_{C/C^\perp}\right)^T = I_{m-k}.$$

For example, in the $[[4,2,2]]$ code we have:

$$G^X_{C/C^\perp} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}, \text{ and } G^Z_{C/C^\perp} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

We can read off the physical implementations of the Pauli operators as the rows of these two generator matrices. We then check to ensure that the physical Paulis commute with every element of the stabilizer and that they satisfy the condition

$$\bar{X}_i\bar{Z}_j = \begin{cases} -\bar{Z}_j\bar{X}_i, & \text{if } i = j \\ \bar{Z}_j\bar{X}_i, & \text{if } i \neq j \end{cases}.$$

Each Clifford operator must satisfy similar conditions with regards to its action on physical Pauli operators. Action in this case refers to conjugation by the Clifford operator. The process of finding the remaining physical Cliffords thus begins with listing these conditions and then finding a suitable physical circuit. As an example, consider the $\overline{SWAP}_{12} \circ \bar{H}_1\bar{H}_2$ physical operator for the $[[4,2,2]]$ code, which has a relatively simple implementation in the code space. This must satisfy the conditions on $\bar{X}$:

$$\bar{X}_1 = X_1X_3 \xmapsto{\overline{SWAP}_{12}\circ\bar{H}_1\bar{H}_2} \bar{Z}_2 = Z_1Z_3$$
$$\bar{X}_2 = X_1X_2 \xmapsto{\overline{SWAP}_{12}\circ\bar{H}_1\bar{H}_2} \bar{Z}_1 = Z_1Z_2$$

and likewise on $\bar{Z}$:

$$\bar{Z}_1 = Z_1Z_2 \xmapsto{\overline{SWAP}_{12}\circ\bar{H}_1\bar{H}_2} \bar{X}_2 = X_1X_2$$
$$\bar{Z}_2 = Z_1Z_3 \xmapsto{\overline{SWAP}_{12}\circ\bar{H}_1\bar{H}_2} \bar{X}_1 = X_1X_3$$

We can see that a viable circuit, which also fixes the stabilizer, is $H_1H_2H_3H_4$, as depicted in Figure B.4 below.

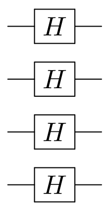


Figure 5: Circuit implementing $\overline{SWAP}_{12} \circ \bar{H}_1\bar{H}_2$ gate

Another approach, which is useful particularly for more complex physical implementations, uses symplectic geometry to find a symplectic matrix $F =$

$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ representing the operator, where F must also satisfy $F\Omega F^T = \Omega$. What follows is justification for the use of symplectic matrices to represent elements of the Clifford group.

Theorem 3. *The automorphism induced by $g \in Cliff_N$ satisfies*

$$gE(a,b)g^\dagger = \pm E([a,b]F_g), \text{ where } F_g = \begin{bmatrix} A_g & B_g \\ C_g & D_g \end{bmatrix} \in Sp(2m, \mathbb{F}_2)$$

We provide a condensed version of the proof in [2] below:

Proof. We begin by showing that for all $[a,b] \in \mathbb{F}_2^{2m}$ there exists some some matrix $F \in \mathbb{F}_2^{2m\times 2m}$ such that $gE[a,b]g^\dagger = \pm E([a,b]F)$. Consider the action of $f \in \text{Cliff}_N$ on some $E(e_i, 0)$, where $e_i \in \mathbb{F}_2^m$ with entry 1 in the i^th position, and 0 otherwise. Note that the set of all e_i forms a basis for $\mathbb{F}_2^m$, and thus the set of $E(e_i, 0)$ forms a basis for X_N. Since g is an automorphism of HW_N, it maps $E(e_i, 0) \mapsto \pm E(a_i', b_i')$, and likewise for the set of elements which form a basis for Z_N, denoted $E(0, e_j)$, g defines a map $E(0, e_j) \mapsto \pm E(c_j', d_j')$, where $[a_i', b_i'], [c_i', d_i'] \in \mathbb{F}_2^{2m}$. We can therefore define a matrix F with i^th row $[a_i', b_i']$ and $m + j^th$ row $[c_j', d_j']$ satisfying

$$gE(e_i, 0)g^\dagger = \pm E([e_i, 0]F), \text{ and } gE(0, e_j)g^\dagger = \pm E([0, e_j]F).$$

Since $[e_i, 0], [0, e_j]$ form a basis for $\mathbb{F}_2^{2m}$ we can conclude that $gE[a,b]g^\dagger = \pm E([a,b]F)$. For a particular g, we denote F as F_g. Next we show F_g is symplectic by proving $F_g \Omega F_g^T = \Omega$.
We know already that

$$E(a,b)E(a',b') = (-1)^{a'b^T} \imath^{\langle [a,b],[a',b'] \rangle_S} E(a+a', b+b'). \tag{4}$$

where $\langle [a,b],[a',b'] \rangle_S$ is the symplectic inner product. If we conjugate 4 by g we have

$$\begin{aligned} (gE(a,b)g^\dagger)(gE(a',b')g^\dagger) &= (-1)^{a'b^T} \imath^{\langle [a,b],[a',b'] \rangle_S} (gE(a+a', b+b')g^\dagger) \\ \Rightarrow E([a,b]F_g)E([a',b']F_g) &= \pm(-1)^{a'b^T} \imath^{\langle [a,b],[a',b'] \rangle_S} E([a+a', b+b']F_g) \end{aligned}$$

Defining $[c,d] \triangleq [a,b]F_g$, $[c',d'] \triangleq [a',b']F_g$ and applying 4 again, we have

$$E([a,b]F_g)E([a',b']F_g) = (-1)^{c'd^T} \imath^{\langle [a,b]F_g,[a',b']F_g \rangle_S} E([a+a', b+b']F_g)$$

By equating the two righthand sides for all $[a,b], [a',b']$ we've shown that

$$\langle [a,b]F_g, [a',b']F_g \rangle_S = \langle [a,b],[a',b'] \rangle_S$$

and thus proven that F_g is a symplectic matrix.

□

We proceed by writing our conditions as binary linear transformations:

$$
\begin{aligned}
[1010, 0000]F = [0000, 1010] &\Rightarrow [1010]A = [0000], [1010]B = [1010] \\
[1100, 0000]F = [0000, 1100] &\Rightarrow [1100]A = [0000], [1100]B = [1100] \\
[0000, 1100]F = [1100, 0000] &\Rightarrow [1100]C = [1100], [1100]D = [0000] \\
[0000, 1010]F = [1010, 0000] &\Rightarrow [1010]C = [1010], [1010]D = [0000]
\end{aligned}
$$

from which we can conclude that satisfactory choices for A, B, C and D are:

$$A = 0_4, \quad B = I_4, \quad C = I_4 \quad D = 0_4$$

which gives

$$F = \begin{bmatrix} 0 & I_4 \\ I_4 & 0 \end{bmatrix},$$

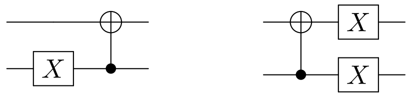
which is the symplectic matrix for the transversal Hadamard.

Note that there will typically be multiple symplectic solutions for any particular physical Clifford. The algorithm in [2] indicates the matrix generated by the simplest circuit, which is likely to have the lowest rate of error. We may also choose a matrix F that centralizes the stabilizer. The effect of selecting a centralizing versus normalizing solution is explored later in this paper.

B.5 Fault Tolerance

Not only can quantum error-correcting codes protect stored or transmitted quantum information, they can also protect quantum information that is dynamically undergoing computation. With fault tolerant quantum computation, it is theoretically possible, given certain conditions, to attain arbitrarily good quantum computation even with faulty logical gates.

After a qubit carrying quantum information is encoded as a block of qubits, encoded gates can cause errors to amplify. For instance, if the encoded block of qubits contains a $CNOT$ gate, an X error on the control qubit will lead to an error on the target qubit.



Consider the circuits above. If an X error occurs on the second qubit before a $CNOT$ gate is applied, it propagates to X errors on both qubits. We may check this mathematically by noting that

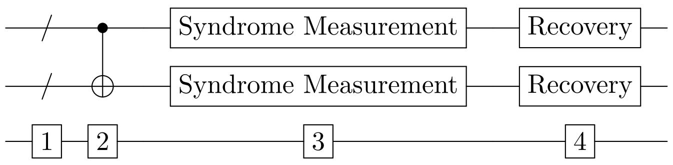
$$
\begin{aligned}
CNOT \circ X_1 &= CNOT \circ X_1 \circ CNOT^\dagger \circ CNOT \\
&= X_1 X_2 \circ CNOT
\end{aligned}
$$

Thus, an X error on the second qubit followed by a $CNOT$ is equivalent to an X error occurring on both qubits after the $CNOT$ gate being applied.

Therefore, it is important to design circuits with encoded gates that will limit the propagation of errors, and this is the idea of fault-tolerance. To be exact, a fault-tolerance procedure has the property that a failure in one elementary operation in a component will result in at most one failure in the qubits output from this component. Such elementary operations may include noisy state preparations, noisy gates, noisy quantum wires, or noisy measurements.

To better understand fault-tolerance, it is useful to think back to the error model. One of the key assumptions made was that errors are drawn from the Heisenberg-Weyl Group and are represented as tensor products of Pauli matrices.

After establishing the notion of fault tolerance, it is helpful to see how a fault tolerant gate should be implemented. The following diagram, taken from [4] will help illustrate how two or more errors might occur on an encoded block of qubits.



1. There is a pre-existing error at Step 1 on each of the two qubits, and the rest of the circuit is performed perfectly. Since an error takes place with probability c_0p, the probability of two errors happening at stage 1 is $c_0^2p^2$.

2. There is an error on qubit 1 before entering the circuit (stage 1) and there is an error on the $CNOT$ gate (stage 2), resulting in two errors in the output. Since each error occurs with probability p, the probability of having two errors in the output is c_1p^2.

3. Two failures happen at the $CNOT$ gate (stage 2), with a probability of c_2p^2.

4. One error occur at the $CNOT$ gate and one at the syndrome measurement. This has a probability of c_3p^2.

5. Two or more failures occur during the syndrome measurement step. The probability of this happening is c_4p^2, with c_2 representing the number of pairs of syndrome measurements.

6. An error happens at syndrome measurement and an error happens at recovery. The probability can be represented as c_5p^2. The value of c_5 is determined by the product of the number of syndrome measurement points and the number of recovery places.

7. Two or more failures take place at the stage of recovery. The probability, $c_6 p^2$, is thus dependent on the number of pairs of recovery steps.

Thus, the total probability of at least two errors happening is cp^2 where $c = c_0^2 + c_1 + c_2 + c_3 + c_4 + c_5 + c_6$. In the case of Steane code, the value of c is about 10^4. Since a perfect decoding only fails when there are two or more errors in the block of encoded circuits, the probability of a decoded state containing an error is at most $O(p^2)$ times larger after the action of circuits on the encoded states. Therefore, as long as p is small enough, the probability of error in decoding, cp^2, will be smaller than 1. For instance, if $p < 10^{-4}$, then cp^2 is smaller than 1, and there is a net gain by encoding the circuits.

B.5.1 Fault-tolerant initial state preparation for [[4,2,2]] codes

While Gottesman proposed that the logical $|00\rangle$ state can be prepared fault-tolerantly with the following circuit

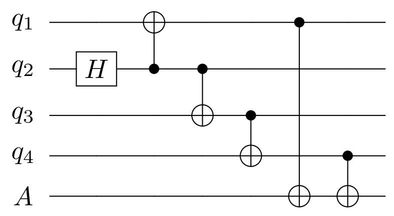


Figure 6: Fault-Tolerant Initial state preparation for the [[4,2,2]]

However, this fault-tolerant circuit could only be achieved by 5 qubits with circular connection (4 qubits for computation and 1 ancilla bit). The ancilla bit is used to determine if any error occurs during the implementation of this circuit. Condiser the connectivity of ibmqx4, it is impossible to prepare the states with this circuit. Consequently, we only use the first 4 qubits in the circuit to prepare the initial state. Since this part is not fault-tolerant, it does impact the measured results and the success rate.

B.5.2 Fault-tolerant gates for [[4,2,2]] codes

For [[4,2,2]], there is a number of logical gates that can be performed fault-tolerantly. The logical Pauli gates can be simply written as the tensor product of physical Pauli gates:

$$\bar{X}_1 = X \otimes I \otimes X \otimes I$$
$$\bar{X}_2 = X \otimes X \otimes I \otimes I$$
$$\bar{Z}_1 = Z \otimes Z \otimes I \otimes I$$
$$\bar{Z}_2 = Z \otimes I \otimes Z \otimes I$$

Performing Hadamard gates on all the physical qubits is equivalent to performing Hadamard gates on both of the logical qubits but swapping the two qubits as well. There are a number of logical gates that cannot be written as a combination of physical gates that are performed on the physical qubits in a bitwise fashion, but are fault tolerant as well. The table below summarizes the set of gates:

Physical Gates	Logical Gates
$X \otimes I \otimes X \otimes I$	$X \otimes I$
$X \otimes X \otimes I \otimes I$	$I \otimes X$
$Z \otimes Z \otimes I \otimes I$	$Z \otimes I$
$Z \otimes I \otimes Z \otimes I$	$I \otimes Z$
$H \otimes H \otimes H \otimes H$	$SWAP_{12} \circ (H \otimes H)$
$P \otimes P \otimes P \otimes P$	$(Z \otimes Z) \circ (CZ)$
$SWAP_{12}$	$CNOT_{12}$
$SWAP_{13}$	$CNOT_{21}$

Table 5: Set of Fault-Tolerant Gates