Flag Fault-Tolerant Error Correction with Qudits

by

Elijah Durso-Sabina

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Physics (Quantum Information)

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis, I explore fault-tolerant quantum error correction using qudits. I show that a specific flag-fault-tolerant procedure for the five-qubit code may be extended to work on the five qudit code for any prime dimension. Using insights from this proof, I go on to show how to extend a procedure for doing flag-fault-tolerant quantum error correction on any stabilizer code to any qudit stabilizer code. These low overhead codes will be useful in development of quantum error correction on devices in the NISQ era.

Acknowledgements

Thank you to:

Crystal Senko, for taking in an academic refugee.

Daniel Gottesman, for the ideas.

Raymond Laflamme, for slogging through the results.

My friends and family, for dealing with me while I wrote this thing.

Dedication

This is dedicated to curiosity.

Table of Contents

List of Figures

Chapter 1

Introduction

Given all the amazing things that we've shown quantum computers can do[1][14][4], we'd like the biggest, most powerful quantum computers as quickly as possible. I imagine that this shocks no one. To this end, we're exploring diverse quantum computing architecture options.

One of the options in question is to compute with larger systems than just qubits[8]. Qubits are quantum two-state systems (or misspelled archaic units of length). Qudits are quantum d-state systems. Clearly, qudits should be able to do everything that qubits can do since they are just qubits if you never access the higher states.

In some ways, qudits are easier to build than qubits. Nature doesn't give exactly two options very often, so most quantum systems are more naturally qudits than qubits. Unfortunately, qudits are generally harder control and are more prone to errors that qubits. With only two states, control pulses are simpler and it is easier to exploit symmetries of the system to reduce noise sensitivity[11]. Some examples of potential qudits are energy levels of single ions and energy levels of superconducting quantum oscillators.

Okay, so if qudits are hard to make, we should probably look at what we stand to gain by making them. Since the size of the Hilbert space for qudits is larger than for qubits, a computer with n qudits supports larger calculations than one with n qubits (by a factor of $\log_2(d)$). Recent research indicates that for some kinds of calculations, qudits have an advantage greater than just the log factor[5]. Further, qudits support different (and potentially better) quantum error correcting codes than do qubits [10]. The different group

structure of qudits compared to qubits means that they inherently support different error correcting codes. Some of these codes will encode a larger number of logical qudits for the same number of physical qudits at the same distance compared to qubit codes.

Quantum error correction is exactly what it sounds like[12]. That's where the simplicity ends, though. Quantum computers rely on superposition and entanglement. Thus, if we're correcting errors on a quantum computer, we can't perform operations which ruin superposition and entanglement. Specifically, we can't directly measure any of the qudits being used in the calculation since that may destroy the relationships between the qudits which encode the information.

Instead of measuring the qudits directly, we use extra qudits to encode a smaller amount of information, then measure aggregate properties of the larger system of qudits. In this way, we can detect errors on the whole system without destroying the information stored in the system. These larger collections of qudits, together with the correction rules, are called quantum error correcting codes. The precise definitions of many of these terms will appear in the following chapters.

1.1 Outline

In Chapter 1, we introduce quantum computing and give some reasons why you might care about it. We also include an outline of the structure of the paper (you are here) and a description of the new results I have proven.

In Chapter 2, we introduce some of the concepts necessary to have a discussion about fault-tolerant error correction.

In Chapter 3, we discuss a procedure for doing fault-tolerant error correction with the five qudit code. This chapter both shows a useful code and serves as a testing ground for some of techniques used in the next chapter.

In Chapter 4, we show an algorithm for doing fault-tolerant syndrome measurement for any qudit stabilizer code. This algorithm uses very few ancilla qudits to achieve fault-tolerance.

1.2 Statement of Contribution

There are two novel results proven in this thesis.

Chapter 3 introduces a procedure for doing fault-tolerant error correction for the five qudit code which works for qudits of any prime dimension. This is a generalization of the work in [2] which shows the procedure for qubits. We cover the measurement procedure explicitly, gate by gate, to make hardware implementation easy for anyone who wants to run this on a real quantum computer. This code also happens to be the smallest quantum error correcting code which can correct any single qudit Pauli error, and so is well suited as a test case for contemporary, small quantum processors.

Chapter 4 shows how to do fault-tolerant syndrome measurement for any stabilizer code for qudits of any prime dimension. The number of ancilla qudits needed for this procedure scales with the distance of the code unlike most procedures which scale with the stabilizer weight. This scaling is important for near term devices where minimizing the number of qudits necessary to run a program is a high priority. It may also have applications in efficient use of larger devices in the more distant future. This is a generalization of the work in [3].

Chapter 2

Background: Fault-Tolerant Error Correction

This chapter defines the concepts necessary to understand what is going on in the next two chapters. We cover, as concisely as possible, qudit operations, how error correcting codes work, and what fault-tolerance is.

This thesis is chiefly about fault-tolerance, and not about error correction itself. For that reason, we'll leave out a lot of the general theory of quantum error correction and just discuss the features which are directly relevant to showing fault-tolerance.

2.1 Generalized Pauli Operators

Since we're going to be talking about errors and groups and the like, now seems to be a good time to introduce the generalized Pauli operators[8]. Just as the standard Pauli operators (I, X, Y, Z) span the vector space of 2×2 Hermitian matrices (up to a phase), the generalized Pauli operators for dimension d span the space of $d \times d$ Hermitian matrices. Thus, when considering all possible errors on a qudit, it suffices to consider all generalized Pauli operators. These operators are defined for dimension d as:

$$X : X|a\rangle = |a \oplus 1\rangle \tag{2.1}$$
$$Z : Z|a\rangle = \omega^a|a\rangle \tag{2.2}$$

Where ω is the primitive dth root of unity, and $\oplus$ indicates addition modulo d. These definitions are identical to the qubit operators when $d = 2$, though in qudit land, we don't talk about Y operators.

The generalized Pauli operators come with the cumbersome but extremely useful commutation relation:

$$(X^r Z^s)(X^t Z^u) = \omega^{st-ru}(X^t Z^u)(X^r Z^s) \tag{2.3}$$

A general element of the generalized Pauli group (GPG) is of the form $\omega X^i Z^j$ for $i, j \in \mathbb{Z}$. When discussing operators on multiple qudits, the elements of the GPG are joined by tensor products. For example, operators on two qudits take the form $(X^i Z^j) \otimes (X^k Z^l)$. These tensored operators span the space of multiple qudits in the same way that, for spaces $\mathcal{A}$ and $\mathcal{B}$, the set of tensor products of the basis vectors $\{|a\rangle \otimes |b\rangle\}$ spans the space $\mathcal{A} \otimes \mathcal{B}$.

To avoid lots of annoying typesetting, I will omit the $\otimes$ between single-qudit operators when discussing multi-qudit operators. You should assume that each operator in a string acts on a new qudit. For example, when I write XZX, I mean the Pauli operator acting on three qudits with an X applied to the first, a Z applied to the second, and another X applied to the third. When we need to talk about both X and Z operators on the same qudit, we'll indicate that they act on the same qudit with parentheses. For example, $X(XZ)Z$ is a three qudit operator with both X and Z acting on the middle qudit.

For multi-qudit Pauli operators, we will use the notion of "weight" to describe the number of qudits on which the action of the operator is non-trivial. For example, the operator $XIXIX$ has weight three while the operator $ZIIII$ has weight one, despite both being five qudit operators.

2.2 Stabilizers and Stabilizer Codes

Most of the well studied quantum error correcting codes belong to a family of codes called stabilizer codes[12]. The term "stabilizer" comes from the same concept in group theory. For some state $|s\rangle$ and group G, the stabilizer subgroup is the set of operators with trivial action on $|s\rangle$, $\{g \in G : g|s\rangle = |s\rangle\}$. We may also discuss the stabilizer of a *set* of elements, which is defined exactly like you might imagine. For a set of states S and the same group G the stabilizer is $\{g \in G : \forall |s\rangle \in S, g|s\rangle = |s\rangle\}$.

One can write quite a lot about stabilizers[7], but we'll just discuss a few salient characteristics here. The elements of a stabilizer form an Abelian group. Also, the elements of the stabilizer are almost invariably described by elements of the GPG.

The stabilizer group, being a group, must be closed under the group operation. Given that the elements of the stabilizer group are from the GPG, the group operation is operator composition. The stabilizer group is typically described not by listing out all the elements of the group, but by listing a minimal set of generators for the group. These generators, together with all their powers and compositions can be used to construct the stabilizer group in a discrete analog of the way basis vectors can be used to construct a vector space.

Usually, stabilizer codes are constructed "backwards." That is, you begin by picking the number of qudits you want. Then you pick a set of commuting Pauli operators on that number of qudits. This set is the generating set for your stabilizer. The codewords are all the states which are stabilized by chosen operators. Of course, if you just pick a random set of commuting operators, it probably won't define a **useful** set of codewords since there may be no efficient decoding algorithm. With that, let's take a look at the properties of stabilizer codes that make them good (or bad).

An error is some element of the GPG which is applied to the data by some imperfection in the operation of the quantum computer. This error may arise from a lack of precision in control pulses or from noise in the environment.

We define the distance (d) of a stabilizer as the weight of the lowest weight Pauli operator which commutes with every element of the stabilizer but is not itself an element of the stabilizer. This is the lowest weight operator needed to transform one codeword into another codeword. This type of error cannot normally be detected since it takes valid

states to other valid states. This means that the error is equivalent to a logical operator on the encoded data. An error correcting code which detected logical operators would not be useful since it would always trigger whenever you manipulated the data during a real calculation.

If an error occurs and we want to be able to tell what the original codeword was, the error weight must be strictly less than half the distance. Thus, the largest correctable error has weight $t = \lfloor \frac{d-1}{2} \rfloor$. Intuitively, if we detect an error and assume that low weight errors are the norm, then the original state of the data is probably the one with the fewest differences from our current state.

Our stabilizer *code* is defined by the stabilizer generators and the rules for returning states which are not codewords back to states which are. Those rules map a set of syndrome measurements to a set of correction operators.

When we discuss "measuring a stabilizer," what we mean is determining the phase accumulated by commuting whatever error is on the data past each of the generators stabilizer. The vector consisting of the measurements of each of the stabilizer generators for a particular error is called the syndrome of that error. Two errors are distinguishable if they have different syndrome vectors. If the correction operator corresponding to the syndrome of an error is the inverse of the error (up to multiplication by a stabilizer), that error is correctable. We will see an explicit example of the process of syndrome measurement and error distinguishing in the next chapter.

2.3 Fault-Tolerance

Error correction is all well and good, but if faults occur during the correction phase we might still be in trouble[9]. For this reason, we want faults during error correction not to mess things up too badly. This condition is called "fault-tolerance."

There are many equivalent ways to define fault-tolerance, but we'll start with providing some intuition about the subject.

First, fault-tolerant error correction should probably still correct errors. I imagine this is not terribly controversial, so we'll leave it at that.

Second, after the fault-tolerant error correction happens, there should be no *more* errors left on the data than the next error correction step is capable of correcting. Error correcting codes alone make no promises about their output if the input has more errors than the code can correct. You can think of this condition a bit like leaving the kitchen after a cooking disaster. You can leave dirty dishes and burnt food around until tomorrow, but there may not be any kitchen left to come back to if you leave a pot of burning oil. Likewise, if there are too many errors coming out of an error correcting step, it might not *just* ruin this data, but everything this data touches in the future.

We will now define these conditions a bit more formally. A fault-tolerant error correcting gadget (circuit acting on a code block whose purpose is to correct errors) must satisfy:

First, a fault-tolerant error correction gadget for a code that corrects t errors must satisfy the error correction correctness property. That is, if there are r errors coming into the gadget and s faults during the gadget, and $r + s \leq t$ then the output is correctable with an ideal decoder. That is, there must be no logical errors.

Second, the gadget must satisfy the error correction recovery property. That is, for a fault-tolerant error correction gadget with $s \leq t$ faults, the output should have at most s errors regardless of how many errors there were on the input. The output might not be the right state, but it should at least be correctable to *a* valid state.

The second condition can be guaranteed by repeating syndrome measurements some finite (and reasonably small) number of times[15], so we will focus mostly on the first condition.

Chapter 3

Flag Fault-Tolerant Error Correction with the 5-Qudit Code

In this chapter, we will show that with slight modification, the fault-tolerant protocol proposed by [2] works for qudits of any prime dimension [6]. This serves two purposes. First, it is an introduction to the notion of flag-fault-tolerance using a simple code. Second, it gives an explicit set of gates for a simple family of codes which can be used in real experiments.

The five qubit and qudit codes are the smallest (least number of physical qubits) error correcting codes which can correct general Pauli errors of weight one. The code consists of, predictably, five qubits (qudits) encoding one logical qubit (qudit). The five qudit code exists for qudits of any prime dimension. The stabilizer group representatives for the five qubit code are[6]:

$$\begin{array}{ccccc} X & Z & Z & X & I \\ I & X & Z & Z & X \\ X & I & X & Z & Z \\ Z & X & I & X & Z \end{array}, \tag{3.1}$$

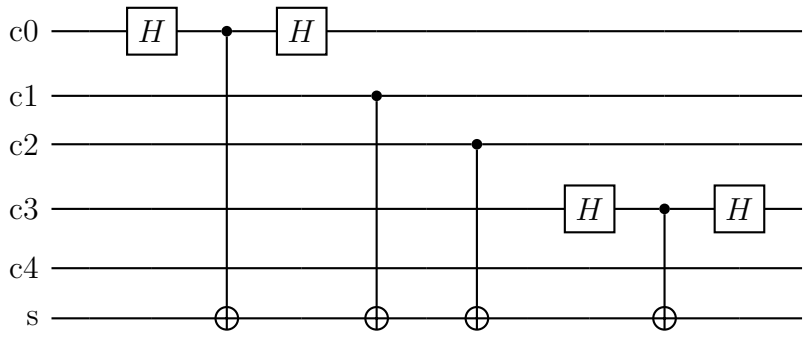
and those for the five qudit code are:

$$\begin{array}{ccccc} X & Z & Z^{-1} & X^{-1} & I \\ I & X & Z & Z^{-1} & X^{-1} \\ X^{-1} & I & X & Z & Z^{-1} \\ Z^{-1} & X^{-1} & I & X & Z \end{array}, \tag{3.2}$$

The five qudit code has distance three, and therefore corrects errors of up to weight one. You can check this explicitly for yourself, but plenty of trustworthy people including myself have done so already.

3.1 Syndrome Measurement in Practice

This is the qubit circuit for measuring the $XZZXI$ stabilizer for the qubit version of the code [2]:



The code qubits are $c0$ through $c4$ and the stabilizer qubit is s. The stabilizer qubit is initialized into state $|0\rangle$.

You might ask "how does this circuit actually measure the stabilizer?" Since, in real quantum computers it is much easier to measure in the amplitude(energy) basis rather than the phase basis, the goal of this circuit is to produce a 0 or a 1 on the s qubit according to whether the error commutes or anti-commutes with the stabilizer. To do this, the circuit goes qubit by qubit and flips the s qubit from a 0 to a 1 (or back) for each single qubit Pauli operator which anti-commutes with the stabilizer $XZZXI$.

To verify that the circuit does what we just discussed, we'll need the following commutation relations for qubit gates[12]:

$$XZ = -ZX \tag{3.3}$$
$$HX = ZH \tag{3.4}$$
$$CNOT(X \otimes I) = (X \otimes X)CNOT \tag{3.5}$$
$$CNOT(I \otimes X) = (I \otimes X)CNOT \tag{3.6}$$
$$CNOT(Z \otimes I) = (Z \otimes I)CNOT \tag{3.7}$$
$$CNOT(I \otimes Z) = (Z^{-1} \otimes Z)CNOT \tag{3.8}$$

You can verify these relations by doing some multiplication of the matrix representations of these operators if you really want to. For our purposes, it is enough to just take these as definitions. From these commutation relations, we can see that $CNOT$ gates copy X errors from the control to the target qubit and copy Z errors from the target qubit to the control qubit. H gates turn X errors into Z errors and vice versa. Remember that in circuit representations, time increases from left to right, while in the operator representation, time goes from right to left.

So, for the first qubit, we can check what happens when the error has either an X or a Z or both at that location. If an X comes in on the left, the first thing that happens is that the H turns it into a Z. Then, since Z errors only move up $CNOT$ gates, it passes the $CNOT$ without doing anything. It then gets turned from a Z back into an X and goes on its merry way, having made no change to the syndrome qubit. Alternatively, if a Z comes in on the left, the H converts it into an X. The X is then copied by the $CNOT$ onto the syndrome qubit, flipping it. If the error has both an X and a Z at that location, it still flips the syndrome bit because the X does nothing and the Z flips the bit. This is exactly the behavior we require, since the stabilizer we are measuring has an X at the first location, so an error with a Z at that location anti-commutes and an error with an X commutes.

For the second qubit, the roles are reversed, since there are no H gates on this one. An incoming X will flip the syndrome qubit and an incoming Z will do nothing. Again, this is what we want since the stabilizer has a Z at this location so Zs commute and Xs anti-commute.

Doing this for each of the qubits in the code, we can see that an incoming error will produce a 0 on the syndrome qubit if an even number $(0, 2, 4)$ of the single qubit operators anti-commute, and a 1 on the syndrome qubit otherwise. This effectively copies the power

of the phase accumulated by commuting the error past the stabilizer to the amplitude of the stabilizer qubit.

Importantly, the error itself is also not changed by the circuit. If it were, the next stabilizer measurements would give information about the wrong error (or would have to be tediously adjusted for the sloppiness in this measurement).

Each of the other stabilizers for the 5-qubit code is just a cyclic permutation of the first stabilizer, so the circuit to measure it is the same, just with the registers permuted.

3.2 Commutation Relations and Moving Errors Through Qudit Circuits

In this section, we will discuss qudit gate commutation relations and and how to use them to trace errors through quantum circuits. The main commutation relations useful in analyzing qudit circuits are [8]:

$$XZ = \omega^{-1} ZX \tag{3.9}$$
$$FX = ZF \tag{3.10}$$
$$FZ = X^{-1} F \tag{3.11}$$
$$SUM(X \otimes I) = (X \otimes X) SUM \tag{3.12}$$
$$SUM(I \otimes X) = (I \otimes X) SUM \tag{3.13}$$
$$SUM(Z \otimes I) = (Z \otimes I) SUM \tag{3.14}$$
$$SUM(I \otimes Z) = (Z^{-1} \otimes Z) SUM \tag{3.15}$$
$$\tag{3.16}$$

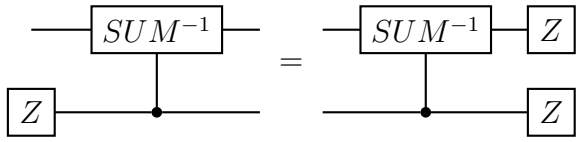
where F is the quantum Fourier transform and SUM adds the amplitude of the first argument to the amplitude of the second argument.

This set is not, by itself, enough to analyze the qudit circuit 3.4. For that, we must derive some further relations using the ones we have. A sample of such a derivation follows:

$$
\begin{aligned}
SUM(I \otimes Z) &= (Z^{-1} \otimes Z)SUM \\
SUM(I \otimes Z)(Z \otimes I) &= (Z^{-1} \otimes Z)SUM(Z \otimes I) \\
SUM(Z \otimes Z) &= (Z^{-1} \otimes Z)(Z \otimes I)SUM \\
SUM(Z \otimes Z) &= (I \otimes Z)SUM \\
(SUM^{-1})SUM(Z \otimes Z)SUM^{-1} &= (SUM^{-1})(I \otimes Z)SUM(SUM^{-1}) \\
(Z \otimes Z)SUM^{-1} &= (SUM^{-1})(I \otimes Z)
\end{aligned}
$$

By this means, we retrieve a commutation relation showing that Z errors on the target of SUM^{-1} gates are copied (without inversion) onto the control. This type of commutation relation allows you to visually trace errors through a circuit:

$SUM^{-1}(I \otimes Z) = (Z \otimes Z)SUM^{-1} \implies$



(3.17)

Analogous derivations show the relevant commutation relations for X errors. The list of relevant new commutation relations is:

$$SUM^{-1}(I \otimes Z) = (Z \otimes Z)SUM^{-1} \tag{3.18}$$
$$SUM^{-1}(Z \otimes I) = (Z \otimes I)SUM^{-1} \tag{3.19}$$
$$SUM^{-1}(X \otimes I) = (X \otimes X^{-1})SUM^{-1} \tag{3.20}$$
$$SUM^{-1}(I \otimes X) = (I \otimes X)SUM^{-1} \tag{3.21}$$

Using these commutation relations, we can trace faults during the circuit to the end. By this means, we can determine which types of faults produce correlated errors and whether these errors may be detected and corrected.

3.3 Fault-Tolerance and Flags

So far, we have only been looking at the circuit in terms of what it does for incoming errors. However, there is nothing magical about error correcting circuits that prevents

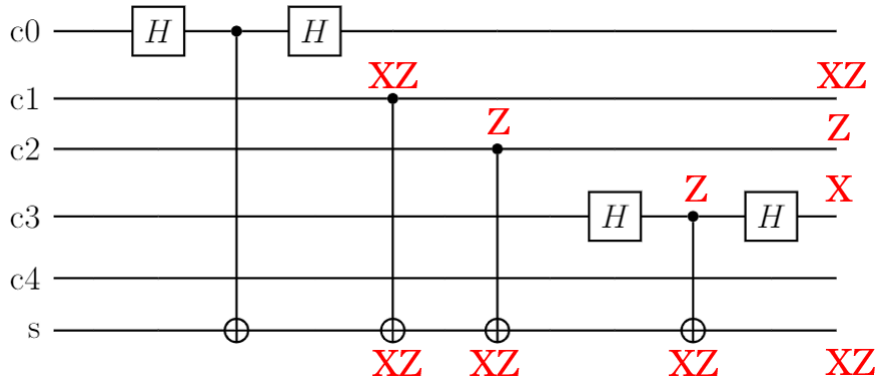


Figure 3.1: This shows a fault on a $CNOT$ gate and how the fault propagates through the circuit to produce a multi-qubit error at the end of the circuit. The red letters first appear at the location of the fault. The letters are repeated at every gate with which the errors interact, and show the error on the wire at that location and time. The letters are repeated once more at the end of the circuit to indicate the final error after the circuit has completed.

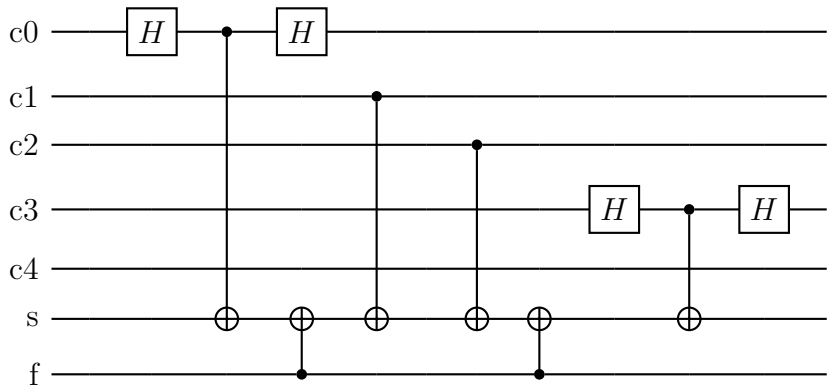
errors from occurring **during** the circuit. For clarity, we will call physical errors during error correction "faults," and henceforth reserve the term "error" for what comes into and goes out of the error correction circuit.

Since the five qudit code only corrects weight one errors, it would be a big problem if a fault during the correction produced more than one outgoing error. We will define such an error as a "correlated error." Unfortunately, the circuit 3.1 can produce such errors if faults occur on the right gates. In particular, two qubit gates like the $CNOT$ gate sort of automatically produce higher weight errors, but the problem is actually worse than that. Not only can two qubit gates produce multiple errors with a single fault, but they can also copy faults on one qubit onto other qubits. Figure 3.1 shows a sample of such a fault and the associated error.

That doesn't look good, does it? With a fault on a single gate, we are left with a multi-qubit data error and the wrong syndrome bit. Worse yet, a fault on the next of the $CNOT$ gates will produce a *different* multi-qubit data error. We'd better figure out a way to deal with this.

Actually, we don't have to. Someone else already did [2]. They added an extra qubit

to the circuit and connected it to the syndrome bit with $CNOT$ gates on either side of the center two gates. This new qubit will flag whether or not an fault a fault of the type we've been discussing has occurred. Incidentally, the new qubit is called the flag ancilla. It is initialized in the $|+\rangle$ state.

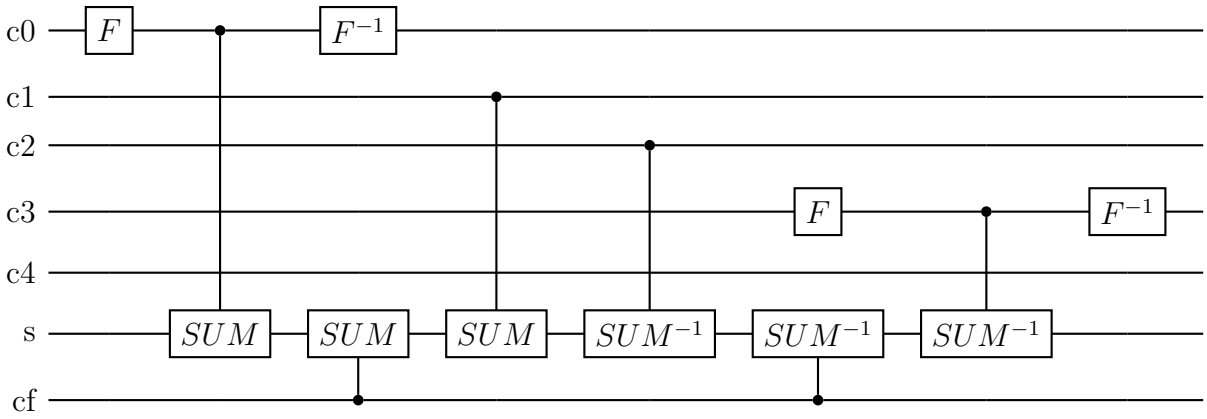


You can see that, if a Z fault occurs on the syndrome ancilla for either of the two middle gates, not only will it be copied to the data qubits, but it will be copied to the flag ancilla as well. By measuring the flag ancilla in the $+/-$ basis, we can tell if such a fault has occurred.

Note that if a Z fault occurs on the first of the data $CNOT$ gates, it is copied onto the flag ancilla twice. Since $Z = Z^{-1}$ for qubits, this causes the flag not to be raised. This is perfect, since the fault will only produce a weight one error (up to multiplication by a stabilizer) on the data, which we are prepared to accept. For a full analysis of this circuit, take a look at [2].

3.4 General Correlated Errors in the Qudit Circuit

Up to this point, we have just been discussing existing results and giving some context. This is where the new stuff starts. The next circuit does the same thing as the the one in the previous section, but for qudits.



If you compare this circuit to the one in fig. 3.3, you can see that they are essentially the same, but with some gates replaced with their qudit equivalents. H gates have been replaced with F or F^{-1} gates and $CNOT$ gates have been replaced with SUM or SUM^{-1} gates. The main challenge to this extension is figuring out which gates need to be inverted and which not. Intuitively, at locations in the stabilizer where X and Z are inverted, so must be the SUM gate. Additionally, the flag must be "turned off" by a SUM^{-1} gate since, for qudits of $d > 2$, $SUM \neq SUM^{-1}$.

Now that we know how to trace errors through the circuits, we can figure out the effect of single faults during the circuit on the circuit output.

We will ignore errors which will never be measured. For instance, since the syndrome ancilla is measured in the amplitude basis, we can ignore phase errors on it except insofar as they travel from the syndrome ancilla to other qudits. For clarity, we will show an example of the fully general case (fig. 3.2), then the same example in this simplified form (fig. 3.3).

Note, in the simplified version, the X error on the flag ancilla is removed since it is never measured and i replaces $i - l$ on the syndrome ancilla since i and l are arbitrary anyway. On the syndrome ancilla, the Z^j error is never measured, but it is transferred to the data by commuting with the SUM and SUM^{-1} gates. The stabilizer being measured in this case is $XZZ^{-1}X^{-1}I$. The final data error is equivalent to $(X^jIIII)(XZZ^{-1}X^{-1}I)^{-j}$ which is a weight one error multiplied by a stabilizer. Clearly, a fault on the SUM^{-1} gate on the flag ancilla can only affect one data qudit and cannot produce a correlated error. For the same reason, faults at locations A and D cannot produce correlated errors.

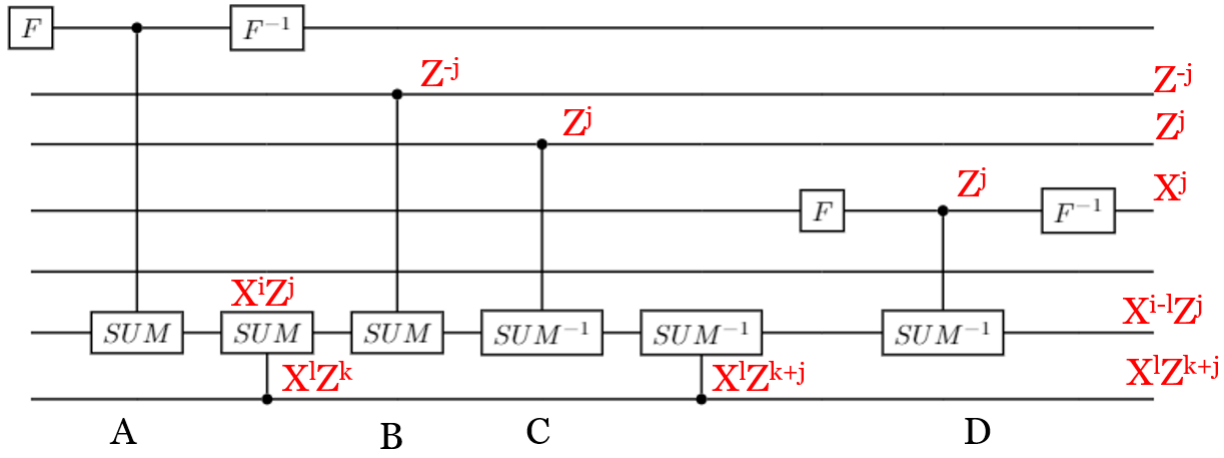


Figure 3.2: Sample cumbersome error tracing for the fault-tolerant syndrome measurement circuit of the five qudit code.

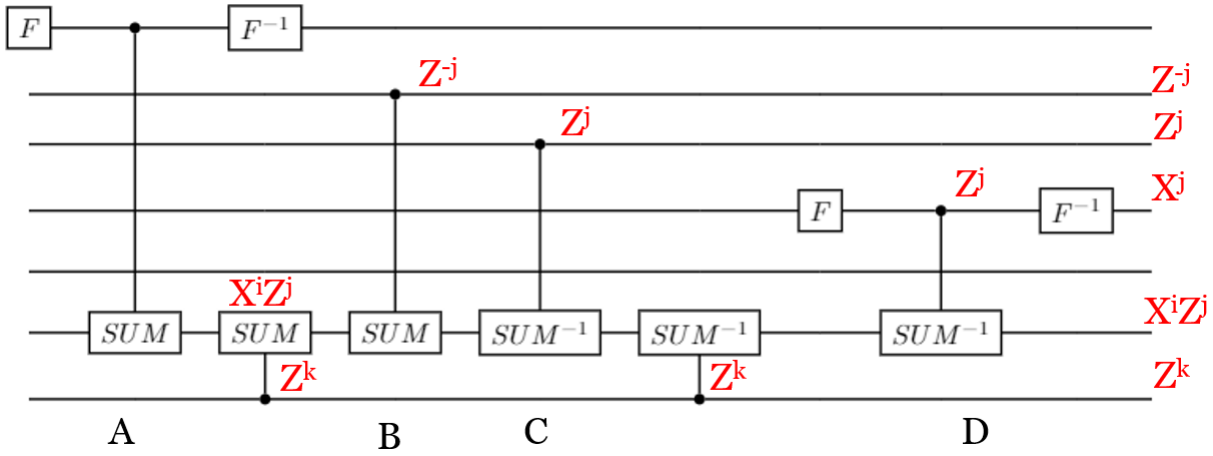


Figure 3.3: The same error tracing exercise as fig. 3.2, but without redundancies.

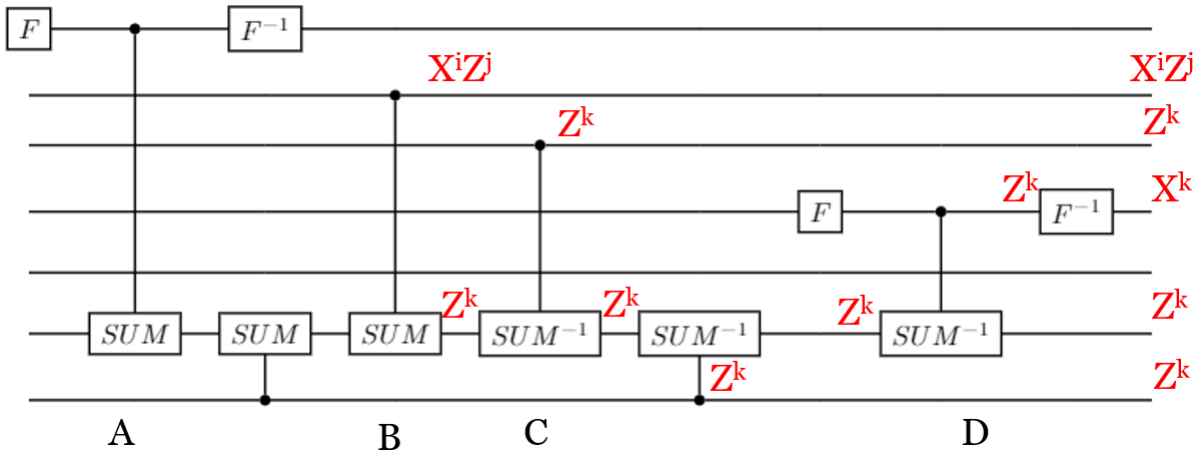


Figure 3.4: Error tracing for a general fault on the SUM gate at location B

Thus, the only locations about which we have to worry are B and C. Figures 3.4 and 3.5 show the error tracing for cases B and C respectively.

So, we have that the general correlated errors for the B and C locations are $I(X^iZ^j)Z^kX^kI$ and $II(X^iZ^j)X^kI$ respectively. For brevity, we will term these B-type and C-type errors respectively. With that, our job is now to use flag and syndrome information to distinguish and correct these two different types of correlated errors.

3.5 Distinguishability of Errors

3.5.1 Correlated Errors

For this section, we will make extensive use of the commutation relation in equation 2.3. With this relation, we can calculate the phase accumulated by commuting the correlated errors past each of the stabilizers and therefore the syndromes resulting from them. These can be distinguished from single qudit errors by the flipped flag qudit. Further, the value of the flag qudit measurement tells us the value of k on in the correlated errors from the previous section.

If it turns out that the combination of flag and stabilizer information is sufficient to distinguish the correlated errors from one another, then we will be able to do fault-tolerant

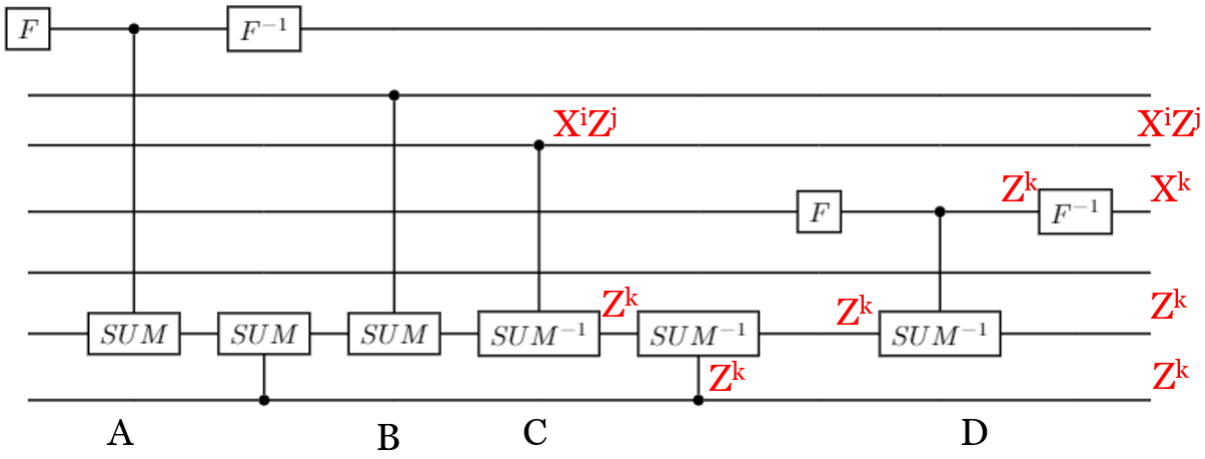


Figure 3.5: Error tracing for a general fault on the SUM gate at location C

error correction with this circuit.

Overall, the procedure for detecting measuring, and correcting the error goes as follows:

1: Measure nontrivial flag
2: Remeasure all stabilizers non-fault-tolerantly
3: Use flag and stabilizer information to determine error
4: Correct Error

Note that, for correctness, we are not worried about faults during re-measurement or about incorrect flag . This is a distance-three code, so we only promise to tolerate one fault. If the flag measurement is non-trivial, there *must* already have been at least one fault. If there is a fault during re-measurement, we have exceeded our allotment. If there was an error during flag measurement, the stabilizer re-measurement will come back trivial and no correction will be issued.

With that, we will now actually do the calculation we've been talking about. We calculate the accumulated phase qudit by qudit, then report the total accumulated phase for each at the end. We'll start with the $XZZ^{-1}X^{-1}I$ stabilizer and the B-type correlated error $I(X^iZ^j)Z^kX^kI$.

$$
\begin{aligned}
(X)(I) &= (I)(X) : Phase = 0 \\
(Z)(X^i Z^j) &= \omega^i (X^i Z^j)(Z) : Phase = i \\
(Z^{-1})(Z^k) &= (Z^k)(Z^{-1}) : Phase = 0 \\
(X^{-1})(X^k) &= (X^k)(X^{-1}) : Phase = 0 \\
(I)(I) &= (I)(I) : Phase = 0
\end{aligned}
\tag{3.22}
$$

For a total contribution of i.

For the stabilizer $IXZZ^{-1}X^{-1}$:

$$
\begin{aligned}
(I)(I) &= (I)(I) : Phase = 0 \\
(X)(X^i Z^j) &= \omega^{-j} (I)(X) : Phase = -j \\
(Z)(Z^k) &= \omega^i (X^i Z^j)(Z) : Phase = 0 \\
(Z^{-1})(X^k) &= \omega^{-k} (Z^k)(Z^{-1}) : Phase = -k \\
(X^{-1})(I) &= (X^k)(X^{-1}) : Phase = 0
\end{aligned}
\tag{3.23}
$$

For a total contribution of $-j - k$.

For the stabilizer $X^{-1}IXZZ^{-1}$:

$$
\begin{aligned}
(X^{-1})(I) &= (I)(X^{-1}) : Phase = 0 \\
(I)(X^i Z^j) &= (X^i Z^j)(I) : Phase = 0 \\
(X)(Z^k) &= \omega^{-k} (Z^k)(X) : Phase = -k \\
(Z)(X^k) &= \omega^k (X^k)(Z) : Phase = k \\
(Z^{-1})(I) &= (I)(Z^{-1}) : Phase = 0
\end{aligned}
\tag{3.24}
$$

For a total contribution of 0.

For the stabilizer $Z^{-1}X^{-1}IXZ$

$$
\begin{aligned}
(Z^{-1})(I) &= (I)(Z^{-1}) : Phase = 0 \\
(X^{-1})(X^i Z^j) &= \omega^j (X^i Z^j)(X^{-1}) : Phase = j \\
(I)(Z^k) &= (Z^k)(I) : Phase = 0 \\
(X)(X^k) &= (X^k)(X) : Phase = 0 \\
(Z)(I) &= (I)(Z) : Phase = 0
\end{aligned}
\tag{3.25}
$$

For a total contribution of j.

Thus, for a B type correlated error, we measure a syndrome of $(i, -j-k, 0, j)$ when we redo the syndrome measurement after measuring a flipped flag.

Following an equally tedious calculation, we can show that the syndrome for a C type correlated error is $(-i^{'}, i^{'}-k, -j^{'}-k, 0)$.

Or job now is to determine if these two syndromes of these forms are always distinguishable from one another.

Importantly, the flag measurement gives us the value of k in both cases, so we may regard it as known.

If we can perform operations on the pair of syndrome vectors in such a way as to isolate $i, j, i^{'}$ and $j^{'}$ and to make the vectors perpendicular to one another, then we will have succeeded. We have to do the same line-by-line operation to both vectors at once, though, since for a real measurement, we can't know which case (B or C) we are in. The entries of the vectors are labelled in order by α, β, γ and δ. This process is analogous to Gaussian elimination. We start with the two different possible syndrome measurements, B-type on the left and C-type on the right. Each successive line applies the same transformation to both syndrome vectors. If the end result is two linearly independent vectors, the two types of errors are distinguishable.

$$\begin{matrix} i \\ -j-k \\ 0 \\ j \end{matrix} \qquad \begin{matrix} -i^{'} \\ i^{'}-k \\ j^{'}+k \\ 0 \end{matrix} \qquad \text{add } k \text{ to row } \beta$$

$$\begin{matrix} i \\ -j \\ 0 \\ j \end{matrix} \qquad \begin{matrix} -i^{'} \\ i^{'} \\ j^{'}+k \\ 0 \end{matrix} \qquad \text{let } j^{'}+k=j^{*}$$

$$\begin{matrix} i \\ -j \\ 0 \\ j \end{matrix} \qquad \begin{matrix} -i^{'} \\ i^{'} \\ j^{*} \\ 0 \end{matrix} \qquad \text{let } \alpha \longrightarrow \alpha+\beta+\delta$$

$$\begin{matrix} i \\ -j \\ 0 \\ j \end{matrix} \qquad \begin{matrix} 0 \\ i^{'} \\ j^{*} \\ 0 \end{matrix} \qquad \text{let } \beta \longrightarrow \beta+\delta$$

$$\begin{matrix} i \\ 0 \\ 0 \\ j \end{matrix} \qquad \begin{matrix} 0 \\ i^{'} \\ j^{*} \\ 0 \end{matrix}$$

Thus, we have shown that the two syndromes are distinguishable in all cases. Furthermore, since we have isolated $i, j, i^{'}$, and $j^{'}$ onto different bits of the modified syndrome vector, we can determine the appropriate correction for any of these errors.

3.5.2 Distinguishing Flag Errors from Correlated Errors

While, in the last section, we established that an error on the first flag SUM gate does not produce a weight two error, there is still potentially a problem. What if such an error occurs and we mistake it for an error at location B or C? Then we might issue a *correction* which produces a weight two error.

To make sure this doesn't happen, let's take another look at the analysis for the flag error circuit, fig. 3.3. We establish that there was a fault by measuring the flag qudit and thereby determine the value of k. If k is non-trivial, we know something has gone wrong, but we're not yet sure if it's a flag fault or one of the two correlated errors. To distinguish the two cases, we can measure the syndrome qudit in the same basis to determine the value of j. Once we know j, then the problem reduces to case B, but with j taking the role of k. Conveniently, if there is actually a B type fault, and we measure the syndrome qudit, we just get k back again, so we can use the same procedure for both cases. Similar analysis shows that a fault on the SUM^{-1} gate which turns off the flag reduces to case C with $i = j = 0$.

For the new procedure, we still start by measuring the flag qudit to determine whether any correction is necessary. The only change to our procedure due to the possibility of flag faults is that we use the power of Z on the syndrome qudit instead of the flag qudit to determine our correction behavior.

With that, we have shown that any singe fault which triggers the flag can be corrected perfectly including faults which would propagate to higher weight errors. Thus, we have a circuit which fault-tolerantly measures the syndrome for our code.

3.6 Fault-Tolerant Error Correction

Now that we have a method for fault-tolerantly measuring the syndrome, we need to turn that into an error correction procedure. This procedure closely mirrors that laid out in [2], but differs in one key respect. The five qu**b**it code is a perfect code, so every error is a weight one error away from a code-word. This is not the case for the general five qu**d**it code. Thus, part of our non-flagged correction procedure must include what to do if we measure a syndrome corresponding to a weight two error. If this happens, we just reinitialize the data to some predetermined codeword. This, of course, destroys whatever

information was initially encoded, but it prevents the problems from spreading uncontrollably to the rest of the calculation.

Our overall fault tolerant error correction procedure is then:

1: **for** $a = 0$ to 2 **do**
2: Measure the syndrome using the fault-tolerant circuit.
3: **if** A flag is raised **then**
4: Remeasure the whole syndrome using the non-fault-tolerant version of the circuit and issue the correction indicated for the flagged circuit. Break.
5: **end if**
6: **if** Syndrome a agrees with syndrome $a - 1$ **then**
7: Issue the non-flagged correction based on the syndrome measurement. Break.
8: **end if**
9: **end for**
10: Issue the non-flagged correction based on the the last syndrome measurement.
11: Repeat 1 through 10 again.

This procedure is fault-tolerant because:

- If there are no faults during the correction procedure, it corrects one error on the data.
- If there are no incoming errors and there is at most one fault during correction, then:
  - If all syndromes and flags are trivial, there can be at most a weight one error on the data.
  - If a flag is raised or if a non-trivial syndrome is measured, the code corrects the error, correlated or otherwise.
- If there are multiple incoming errors and there is a fault during the circuit the repeated syndrome measurement ensures that the syndrome is measured correctly. Repeating the block twice ensures that interactions of a fault and incoming errors do not cause the wrong correction to be issued.

This procedure satisfies the Error Correction Correctness property since it corrects a single fault or a single error, so long as there is only one and this is a distance 3 code.

Demonstrating that this satisfies the Error Correction Recovery Property is a little trickier. The repeated syndrome measurement is key for this since a large number of incoming errors may be disguised by a single fault and leave a high weight outgoing error. If two measurements in a row agree, there cannot have been a fault on both, so the measurement can be trusted. If the first and third agree, there must have been a fault on the second, so the third can be trusted. If none agree, there must have been more than one fault, and we are allowed to give up.

However, the fault may have triggered a flag and therefore the correction of a correlated error. If the sum of the syndromes of the fault and the incoming error are equal to the syndrome of the wrong type of correlated error, the procedure could leave a weight-two or greater error. At present, it is not known whether any such error exists.

If such an error exists, running procedure in lines $1-10$ twice would solve the problem. If there is a fault in the first iteration, the data may wind up with a high weight error. Then, the second run will correct that back to a code word since we do not permit a second fault. If there is no fault on the first run, then the data are brought back to a code word before the second run. Then, even if there is a fault on the second run, it is corrected and the data are left with at most a weight one error. With that, we guarantee that we end up with no more than a weight one error on the data after the procedure regardless of the incoming error, so we satisfy the ECRP.

Thus, this is a fault tolerant error correction procedure.

Chapter 4

Flag Fault-Tolerant Error Correction for any Qudit Stabilizer Code

In this chapter, we will analyze an algorithm which allows for the fault-tolerant measurement of stabilizers for any qudit code. This algorithm is adapted from [3] which shows how to fault-tolerantly measure stabilizers for any qubit stabilizer code. The proof method here will show some interesting features of the original algorithm which the original proof method did not bring to light, though at a slight cost.

4.1 The Circuit

This section will give an explanation of the concepts used in the circuit and algorithm. We will make reference to a lot of the concepts from the previous chapter which, to a great extent, serves as an introduction for this one.

Flags detect whether a fault has occurred on a specific qudit between the time the flag is turned on and when it is turned off. That qudit is said to be protected by the flag. In particular, if the flag is turned on before a gate and turned off after it, the gate is said to be protected by the flag. In the qudit case, a flag is turned on by a SUM gate and turned off by a SUM^{-1} gate. In the previous section, due to the symmetry of the circuit, the two gates at locations B and C were protected by a single flag. In this section, gates on the

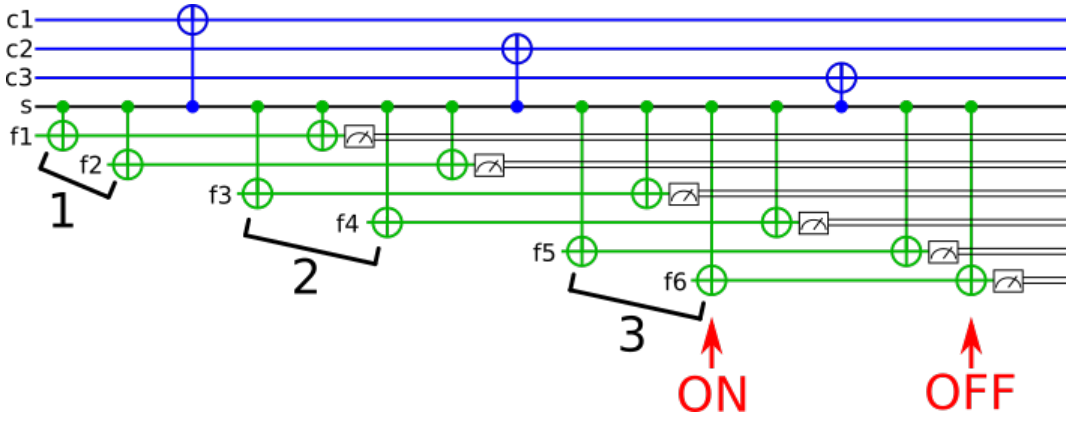


Figure 4.1: This figure shows a sample syndrome measurement circuit. There are three data qubits (c1,c2,c3), each joined to the syndrome bit (s) by $CNOT$ gates. Each of these $CNOT$ gates is protected by a pair of flags. The three pairs of flags protecting each of the $CNOT$ gates are the so called "flag rounds." The last flag has marked at the points where it is turned on and where it is turned off.

syndrome qudit will generally be protected by one or more flags.

The reason we want to protect gates during the syndrome measurement process is that the syndrome circuit connects multiple data qudits to a single syndrome qudit. If faults occur on this syndrome qudit, they can spread to the data. These faults can be detected and located by protecting the syndrome qudit with flags in places where it is connected to the data. If this is done right, the faults can be corrected or at least prevented from spreading. In general, for a higher distance code, more flags will be necessary to achieve fault tolerance.

Since some of the circuit elements in this section are difficult to picture in the abstract, I have included a sample circuit (Figure 4.1) protected by flags to be used as a visual reference. For readability, the circuit depicts qubits, but the circuit is essentially the same for qudits. The only real difference is that the flags are turned on with SUM gates and turned off with SUM^{-1} gates.

For a code with distance $d = 2t+1$, the algorithm requires that each syndrome gate be protected by at least d flags. In fig 4.1 depicts a circuit with syndrome gates protected by two flags each. Since there is no $n \in \mathcal{Z}$ such that $2n+1 = 2$, this does not correspond to any real correction scheme, but it illustrates all the salient features of a circuit protected

by flags while remaining small enough to read.

In the general case, the flags are turned on one by one before the first syndrome gate. After the first syndrome gate, the first flag for the second gate is turned on. Then the first flag for the first gate is turned off and the second flag for the second gate is turned on, etc. As the circuit continues, flags are turned on and off in alternating fashion. This way, there are at least d and at most $d+1$ flags on at any given moment. As a result, faults on the syndrome qudit will trigger either d or $d+1$ flags. Since these types of faults are the ones which can spread to larger data errors, we will call them **data** faults going forward to distinguish them from faults in the measurement of the syndrome qudit.

Of course, the flags themselves can also have faults. If the fault is just on the flag, the effect only triggers that one flag. These, we will creatively call **flag** faults.

For convenience of notation, we will group all the flags protecting a single syndrome gate into a "flag round." The measurement result of the flag round will be a string of dits of length d. The number of these rounds in the algorithm is determined by the weight of the stabilizer being measured. However, if the qudits can be reset between rounds, the qudit overhead is just determined by the code distance.

A flag or flags in round l may be triggered by many different types of faults. A fault on the gate turning off flags from round $l-1$ may trigger flags in round l if it results in a fault on the syndrome qudit. A fault on the gates turning on or off flags for round l will trigger flags in round l. A fault on the gates turning on flags for round $l+1$ may trigger flags in round l if it results in a fault on the syndrome qudit. Last, a fault on the lth data SUM gate may trigger flags in round l.

The goal of the algorithm will be to catch and correct data faults before they spread. The faults spread by travelling from the syndrome qudit, up the SUM gates, back to the data. If we can figure out the location and value of the fault before it affects more than one data qudit, then we have a fault tolerant syndrome measurement circuit. To do this, the algorithm will perform a check at the end of each flag round to determine if a fault which needs correcting has occurred during that round. Because the flags are turned on and off in alternating fashion, the flag round is only finished being measured right before the next syndrome SUM gate. If that flag round indicates that a correction should be issued, the correction is issued to the *next* syndrome dit and all future ones. Thus, when we say that

a correction is issued "in round l", what we mean is that flag round l indicates that we should correct qudits $l+1$ and onward.

Without loss of generality, assume the stabilizer we are measuring for this process is $X^{\bigotimes w}$. Any other stabilizer can be made to match this form by application of single qudit unitaries on the data qudits. This condition does, however, require that the qudits be of prime dimension. If the qudits are not of prime dimension, the SUM^k is not necessarily expressible as a product of a SUM gate and single qudit unitaries.

4.2 Definitions

For the purpose of this discussion, a **fault** is a hardware fault during the correction procedure while an **error** is a logical error remaining after correction.

The measurement will take place over a number of rounds determined by the weight of the stabilizer to be measured. Let Ω be the last one of these rounds. In other words, let the circuit measure a stabilizer of weight Ω.

Let i be an index over the correction rounds from the first to the Ωth.

Let n be the dimension of the qudits used for the code.

Let k be an index over the qudit dimension from 1 to $n-1$.

Let t be the number of errors the code will correct.

Let r_i be the measurement values of the flags for round i. r_i is a string of length d with entries from the set $\{0, 1, \ldots, n-1\}$. We will call r_i the **flag pattern** for round i.

For most of the examples, we will use $t = 3$ and therefore $d = 7$ since it is large enough to illustrate most of the features in the proof.

Let m_i be another string of the same form as r_i, but which is updated each round with information from flag measurements and corrections. m_i has no intrinsic physical meaning. It is just a construct used for the algorithm to keep track of what has happened so far.

Let $|string|^{(k)}$ denote the number of entries with value k in $string$. For example, if $string = \{0, 0, 0, 1, 1, 2, 3\}$, then $|string|^{(0)} = 3, |string|^{(1)} = 2, |string|^{(2)} = 1, |string|^{(3)} = 1$ etc. We will call $|string|^{(k)}$ the k-weight of $string$.

Let $\oplus$ be the entry-wise sum of strings modulo n.

During any single round, there are three types of possible errors: a fault directly on one of the flag qudits, a fault on the data qudit while d flags are active, and a fault on the data qudit while $d + 1$ flags are active. For the purpose of this proof, it is easier to consider the $d + 1$ case as a combination of a d data fault and a single flag fault occurring immediately before or after. The reason for this will become clear as we move forward.

We denote the set of single flag faults F. Particular flag faults are described with a double index $F_{i,j}$, which is the jth fault in the ith round. For example, if the first fault in round one adds three to the amplitude of the third qudit, then $F_{1,1} = \{0, 0, 3, 0, 0, 0, 0\}$ Note that, for a round with no flag faults, the j index may be empty.

Let S be the set of faults which flip d flag dits, i.e. data faults. $S_{i,j}$ is indexed in the same way as flag faults. Data faults apply the same shift to all the flags which they affect. We will denote that shift $k_{i,j}$. For example, if $S_{2,3} = \{2, 2, 2, 2, 2, 2, 2\}$, then $k_{2,3} = 2$. Since data faults flip d flags, they may affect the flag patterns of two rounds. To avoid ambiguity, we will use the i index to refer to the round in which the fault started. So, for instance, if $S_{3,4}$ begins on the fourth dit and $k_{3,4} = 1$, then $r_3 = \{0, 0, 0, 1, 1, 1, 1\}$ and $r_4 = \{1, 1, 1, 0, 0, 0, 0\}$.

For our purposes, a fault and the flag pattern it produces are synonymous, since the only information we have about faults is the flag pattern. For this reason, when I refer to a fault, I mean the flag pattern that it produces.

4.3 The Algorithm

With our definitions and some intuition in hand, we can now state the whole fault-tolerant measurement algorithm. Before the algorithm starts, assume that we have measured a sequence of $t + 1$ flag rounds during which no flags are triggered. This ensures that the algorithm starts with no faults on the flag or syndrome ancillas. Finding such a sequence takes at most $t(t + 2)$ flag rounds.

1: $m_0 \leftarrow \{0\}^d$
2: **for** $(i = 1, \ldots, \Omega)$ **do**
3: $\quad m_i \leftarrow m_{i-1} \oplus r_i$
4: $\quad$ **for** $(k = 1, \ldots, n)$ **do**
5: $\quad\quad$ **if** $(\lvert m_i \rvert^{(k)} > t)$ **then**
6: $\quad\quad\quad m_i \leftarrow m_i \oplus \{k^{-1}\}^d$
7: $\quad\quad\quad$ Issue k^{-1} correction to data qudits $i + 1$ and onward

In human language, the algorithm initializes the m string with all 0s. Then, every round, we start by adding the new flag pattern to the m string. Then, we check if more than half of the entries of the string are the same number k for each of the possible ks. If they are, we subtract k from all the entries of the m string and issue a k^{-1} correction to all the data qudits we measure from now on. In the next section, we'll explain why each of these steps is necessary and in doing so, show that this does in fact fault-tolerantly measure the stabilizer.

At any given time, there are either d or $d + 1$ flags active. With the syndrome qudit, then, the overhead of this procedure is $d + 2$ qudits.

This overhead is one qudit higher than in the original [3] because qudits for $d \geq 3$ lack the symmetry of qubits. In the qudit case, it matters whether the flag fault comes before or after the data fault. As a result, the trick to locate and counteract the flag fault in the original paper is not applicable to the qudit case. However, the addition of a single extra qudit of overhead is sufficient to solve the problem.

An advantage of the added overhead is that the algorithm can be run in real time, which is not possible with the reduced overhead.

4.4 Proof

In this section, we will prove that this algorithm allows fault-tolerant stabilizer measurement for up to t faults.

In particular, we need not concern ourselves with the cases where there are more than t faults of any kind since the original error correcting code doesn't even promise to fix that many.

The proof relies on splitting the sum up into the data faults and flag faults, then analyzing them separately. By looking only at the data faults first, we will see that no matter how we arrange the flag faults after, we can't mess up the correction behavior.

The sum of all faults up to round l is

$$\bigoplus_{i=1}^{l} \bigoplus_{j} (F_{i,j} \oplus S_{i,j}). \tag{4.1}$$

To clarify the notation here, the sum is over the strings of flag patterns produced by the faults $S_{i,j}$ and $F_{i,j}$. It is worth noting that this sum is not usually what the algorithm "sees" since $S_{i,j}$ may not be over by the end of round i. However, this will not stop us, as you will see in a moment.

By the commutativity and associativity of $\oplus$,

$$\bigoplus_{i=1}^{l} \bigoplus_{j} (F_{i,j} \oplus S_{i,j}) = (\bigoplus_{i=1}^{l} \bigoplus_{j} S_{i,j}) \oplus (\bigoplus_{i=1}^{l} \bigoplus_{j} F_{i,j}) \tag{4.2}$$

Thus, we may consider just the data faults at first and add in the flag faults later in the analysis.

Note that, since $S_{i,j}$ flips exactly d flags and each flag round consists of d flags, a single data fault in round l can affect the flag pattern in at most rounds l and $l+1$.

Thus, if there is no data fault in round $l+1$,

$$\bigoplus_{i=1}^{l+1} r_i = (\bigoplus_{i=1}^{l} \bigoplus_{j} S_{i,j}) \oplus (\bigoplus_{i=1}^{l+1} \bigoplus_{j} F_{i,j}). \tag{4.3}$$

Also,

$$\bigoplus_{i=1}^{l} \bigoplus_{j} S_{i,j} = \bigoplus_{i=1}^{l} \bigoplus_{j} \{k_{i,j}\}^d. \tag{4.4}$$

Let

$$\bigoplus_{i=1}^{l} \bigoplus_{j} k_{i,j} = \mathcal{K} \tag{4.5}$$

Then, if there are no corrections issued up to round l, then

$$m_{l+1} = \bigoplus_{i=1}^{l+1} r_i = \{\mathcal{K}\}^d \oplus (\bigoplus_{i=1}^{l+1} \bigoplus_{j} F_{i,j}). \tag{4.6}$$

By our assumption,

$$t \geq |F| \geq \left| \bigoplus_{i} \bigoplus_{j} F_{i,j} \right| \geq \left| \bigoplus_{i=1}^{l+1} \bigoplus_{j} F_{i,j} \right|. \tag{4.7}$$

Then,

$$t < \left| \{\mathcal{K}\}^d \oplus (\bigoplus_{i=1}^{l+1} \bigoplus_{j} F_{i,j}) \right|^{(\mathcal{K})} = |m_{l+1}|^{(\mathcal{K})} \tag{4.8}$$

and we issue a $\mathcal{K}$ correction in round $l+1$.

After this correction,

$$m_{l+1} = (\bigoplus_{i=1}^{l+1} \bigoplus_{j} F_{i,j}), \tag{4.9}$$

and we issue no more corrections until another data fault occurs, since again,

$$t \geq \left| \bigoplus_{i} \bigoplus_{j} F_{i,j} \right|. \tag{4.10}$$

Corrections after the first use this same analysis, but begin counting data faults in round $l+2$ instead of round 1.

4.5 Fault-Tolerance

4.5.1 Fault-Tolerant Syndrome Measurement

After a data fault $\{k\}^d$ in round l, the SUM gates connecting the syndrome ancilla to the data will communicate a k error to all following data qudits. When we issue a correction, we adjust m_i by $\{k^{-1}\}^d$ and apply a k^{-1} to all successive data qudits.

Our procedure corrects any data faults occurring in round l by round $l+1$ if there is no data fault in round $l+1$ and may or may not issue a correction otherwise. For fault-tolerance, we must not have one data fault spread to more than one data error. If we have one fault in round l, and correct it by round $l+1$, we end up with at most one data error by construction. If we have one fault in round l and another in round $l+1$, we may end up with the following situation:

Say the fault in round l is $\{k\}^d$ and the fault in round $l+1$ is $\{k^{'}\}^d$. Further, we issue a correction $\{c\}^d$ in round l and correction $\{c^{'}\}^d$ in round $l+1$. These corrections may be trivial and may or may not sum to $\{(k+k^{'})^{-1}\}^d$. Then, by round $l+2$,

$$m_{l+2} = (\bigoplus_{i=1}^{l+2} \bigoplus_{j} F_{i,j}) \oplus \{k + k^{'} + c + c^{'}\}^d, \tag{4.11}$$

and all future data qudits have a $k+k^{'}+c+c^{'}$ error. Data qudit l has error $k+c$ and data qudit $l+1$ has error $k+k^{'}+c+c^{'}$, and these can no longer be corrected.

If there is no data fault in round $l+2$, then we are left with two errors for two faults, which is acceptable. We then correct $k+k^{'}+c+c^{'}$ in round $l+2$ and for all successive rounds and have no future errors arising from these faults.

If there is a data fault in round $l+2$, we may have a fault $k^{''}$ and issue correction $c^{''}$. We are then left with three irreversible errors, $k+c$, $k+k^{'}+c+c^{'}$, and $k+k^{'}+k^{''}+c+c^{'}+c^{''}$ for three faults. We continue in this fashion until there is a round which does not contain a data fault or until there are more than t faults. The key feature here is that, as soon as we have a round with no data faults, we know the value of the error which will spread to the rest of the data qudits and can correct it.

Thus, we show that for $t \geq \left| \bigoplus_i \bigoplus_j F_{i,j} \right|$, x faults result in $\leq x$ errors, and our procedure is t-fault-tolerant.

This analysis also shows that, during any round where there is a data error, there is no difference between a single data fault and an arbitrary number of data faults, so long as they flip precisely d flags. All these extra faults do is change the value of k. They don't produce any more data errors.

4.5.2 Fault-Tolerant Error Correction

Now that we have a circuit for doing fault tolerant syndrome measurement, we need to use that circuit to make an error correction gadget.

Since faults may disrupt the measurement of the syndrome dit, we need to repeat the syndrome measurement circuit. If we only permit t faults, then repeating the syndrome measurement circuit until $t + 1$ of the measurements agree is sufficient to guarantee the correct measurement of the syndrome. We cannot do this dit by dit, since the total data error may change during this procedure. We need to measure the whole syndrome in one go, then do it again, etc.

Say that, for $r + s \leq t$, there are s faults during the error correction gadget. We have proven that the gadget produces no more than a weight s error on the data. For a weight r error coming into the gadget, the measured syndrome at the end must correspond to an error of weight $\leq r + s$. Since $r + s \leq t$, this error is correctable. Even if all s faults occur right after the correction is issued, we are still left with an error of weight $s < t$, so we satisfy the ECCP.

If a high weight error comes into the circuit, it may not be correctable. However, since our circuit doesn't rely on any syndrome information for fault tolerance, the errors and faults do not interact. So, we are still able to make an accurate syndrome measurement. That syndrome measurement can either tell us that we are within t of a codeword or that we are not. In the first case, we issue a correction taking us to that codeword, whether or not it is the correct one. In the second case, we can replace the data with a fresh codeword. In either case, $s < t$ faults in the circuit can produce no more than a weight s error, as

we proved in the last section. Thus, for regardless of the weight of the incoming error, our gadget leaves us with a codeword with no more than s errors on it, and we also satisfy the ECRP.

This completes the proof.

Chapter 5

Conclusion

In this thesis, we have been subjected to proofs of two fault-tolerant syndrome measurement schemes for prime-dimensional qudit codes. The first shows a gate-by-gate description of how to do fault-tolerant syndrome measurement for the five qudit code and includes a procedure for fault-tolerant error correction using the syndrome measurement scheme. The second proof describes an algorithm for doing fault-tolerant syndrome measurement for any qudit stabilizer code.

This work may be useful for people with real quantum computers who wish to correct errors on those computers. This may soon become relevant, since universal quantum processors using qudits actually exist[13], as of the writing of this particular sentence. Hopefully we have discussed the procedures in enough detail that someone can implement them after reading this document.

In the future, it may be worth exploring what happens when we try to implement these sorts of procedures on qudits of non-prime dimensions, or if anything interesting happens in the limit of large dimension.

References

[1] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560(P1):7–11, mar 2020.

[2] Rui Chao and Ben W. Reichardt. Quantum error correction with only two extra qubits. *Physical Review Letters*, 121(5), may 2017.

[3] Rui Chao and Ben W. Reichardt. Flag fault-tolerant error correction for any stabilizer code. *PRX Quantum*, 1(1), dec 2019.

[4] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics 1982 21:6*, 21(6):467–488, jun 1982.

[5] Pranav Gokhale, Jonathan M. Baker, Casey Duckering, Natalie C. Brown, Kenneth R. Brown, and Frederic T. Chong. Asymptotic Improvements to Quantum Circuits via Qutrits. *Proceedings - International Symposium on Computer Architecture*, pages 554–566, may 2019.

[6] Daniel Gottesman. Stabilizer Codes for Prime Power Qudits.

[7] Daniel Gottesman. Stabilizer Codes and Quantum Error Correction. 1997.

[8] Daniel Gottesman. Fault-Tolerant Quantum Computation with Higher-Dimensional Systems. *Chaos, solitons and fractals*, 10(10):1749–1758, feb 1998.

[9] Daniel Gottesman. An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation. 2009.

[10] Lane G Gunderman. Some Results on Qudit Quantum Error-Correction.

[11] Pei Jiang Low, Brendan M. White, Andrew A. Cox, Matthew L. Day, and Crystal Senko. Practical trapped-ion protocols for universal qudit-based quantum computing. *Physical Review Research*, 2(3), jul 2019.

[12] Michael A Nielsen and Isaac L Chuang. Quantum Computation and Quantum Information.

[13] Martin Ringbauer, Michael Meth, Lukas Postler, Roman Stricker, Rainer Blatt, Philipp Schindler, and Thomas Monz. A universal qudit quantum processor with trapped ions. sep 2021.

[14] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, aug 1995.

[15] Peter W. Shor. Fault-tolerant quantum computation. may 1996.