# Surface Code Threshold Calculation and Flux Qubit Coupling

by

Peter Groszkowski

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

Waterloo, Ontario, Canada, 2009

## Abstract

Building a quantum computer is a formidable challenge. In this thesis, we focus on two projects, which tackle very different aspects of quantum computation, and yet still share a common goal in hopefully getting us closer to implementing a quantum computer on a large scale. The first project involves a numerical error threshold calculation of a quantum error correcting code called a surface code. These are local check codes, which means that only nearest neighbour interaction is required to determine where errors occurred. This is an important advantage over other approaches, as in many physical systems, doing operations on arbitrarily spaced qubits is often very difficult. An error threshold is a measure of how well a given error correcting scheme performs. It gives the experimentalists an idea of which approaches to error correction hold greater promise. We simulate both toric and planar variations of a surface code, and numerically calculate a threshold value of approximately $6.0 \times 10^{-3}$, which is comparable to similar calculations done by others [44, 45, 54]. The second project deals with coupling superconducting flux qubits together. It expands the scheme presented in [43] to a three qubit, two coupler scenario. We study L-shaped and line-shaped coupler geometries, and show how the coupling strength changes in terms of the dimensions of the couplers. We explore two cases, the first where the interaction energy between two nearest neighbour qubits is high, while the coupling to the third qubit is as negligible as possible, as well as a case where all the coupling energies are as small as possible. Although only an initial step, a similar scheme can in principle be extended further to implement a lattice required for computation on a surface code.

## Acknowledgements

I would like to thank my supervisor Prof. Frank Wilhelm for taking me on as a graduate student, and providing his guidance and always useful feedback. I look forward to us working together during my PhD. I also very much appreciate the input I received from the rest of my committee, Jan Kycia, Daniel Gottesman and Ray Laflamme. Further thanks go to the members of the Quantum Device Theory group (Brendan Osberg, Felix Motzoi, Farzad Qassemi, Jay Gambetta, Bill Coish, Ioana Serban) and others from IQC, with whom I've had many illuminating discussions, and often exhausting, but always fun ball-hockey games. This work would probably not be possible without Austin Fowler, who was very much the driving force behind the initial ideas for both of the projects that make up this thesis. He has always put 110% into providing guidance, even if on occasions it meant matching my often nocturnal schedule and staying up until the early hours of the morning. His patience, excitement about our work, and all the time and effort that he has put into it, will be always greatly appreciated. Further acknowledgements go to NSERC for funding this project, and SHARCNET for providing the high-performance computing infrastructure, without which some of the results presented here would be difficult to obtain. Finally, I would like to say thanks to my parents for all their love and support over the years, and Silvia for all the good times we've had.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Quantum Mechanics and Computing

Classical computers have been around for many years. Although originally they were used only in highly computationally intensive research, in the recent couple of decades they have made their way into our everyday lives in a way no one could have ever predicted. Quantum computers promise to revolutionize the way we perform computation even further. They provide a way to factor numbers [47], search databases [24], and an ability to effectively simulate physical systems [20, 48] — all much more efficiently than their classical counterparts ever could. In classical computation, a bit is used to store information. It can either represent a 0 or a 1. Quantum computing has an analogous construct — a qubit. Qubits are quantum two level systems that are described by vectors in a two dimensional Hilbert space. They are different from classical bits in that they can store an arbitrary "superposition" of 1 and 0, meaning that a qubit can in a sense be in both states at the same time. Using Dirac's bra-ket notation [13] we can describe such a state of a given qubit as

$$|\Psi\rangle = \alpha\,|0\rangle + \beta|1\rangle \tag{1.1}$$

where $\alpha$ and $\beta$ are complex numbers and $|0\rangle$, $|1\rangle$ quantum basis states - often called the "computational basis". These basis states are assumed to be orthonormal with respect to the scalar product, and hence

$$\langle i|j\rangle = \delta_{ij} \tag{1.2}$$

must be true. Furthermore, in order to ensure that any state is normalized, we need the condition

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.3}$$

to hold. A helpful way of visualizing pure two-level quantum states is using a Bloch sphere. By rewriting $\alpha = \cos\frac{\theta}{2}$ and $\beta = e^{i\varphi}\sin\frac{\theta}{2}$ in Eq. 1.1, with $0 \leq \theta \leq \pi$ and

$0 \le \varphi \le 2\pi$, we can think of the vector $|\psi\rangle$ as being oriented on a unit sphere, as shown in Fig. 1.1.

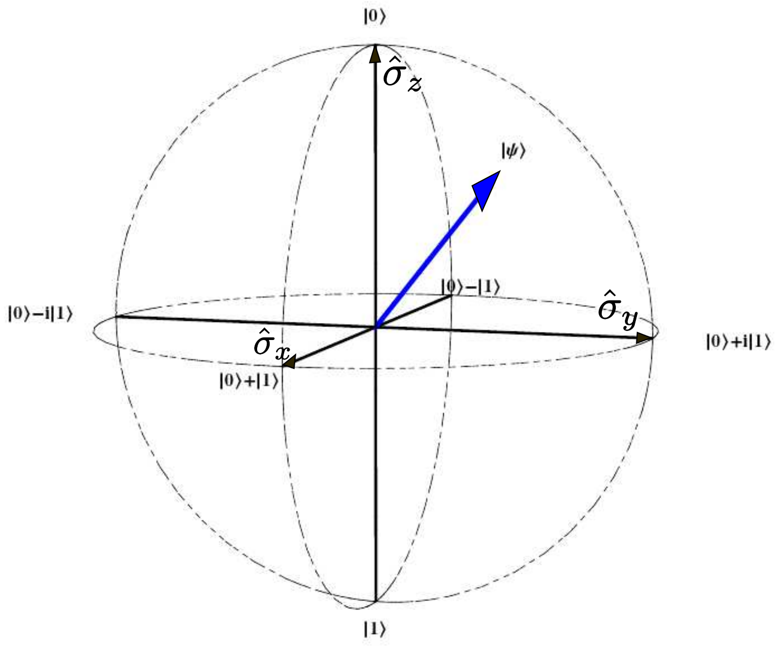


Figure 1.1: A Bloch sphere. $|\psi\rangle$ represents an arbitrary qubit state.

Over the last few years, many different proposals for qubit implementations have been suggested. They range from polarized light [30] and NMR [15], through trapped atoms [11] to superconducting circuits [12], with many others in between. Although at this stage it is impossible to predict which will turn out to be most usable in the end, substantial advances have been made, and some few-qubit systems have been demonstrated experimentally. The actual physics in each proposal can be rather different, but the two-state nature of qubits is most often described in terms of a Hamiltonian

$$H = -\frac{\epsilon}{2}\sigma_z - \frac{\Delta}{2}\sigma_x \tag{1.4}$$

where $\sigma_x$, $\sigma_y$ and $\sigma_z$ are the Pauli matrices (which are explained in Section 1.5), and both $\epsilon$ and $\Delta$ are related to physical parameters which depend on a given physical implementation. Either of them (or both) can vary in time. The eigenvalues of this Hamiltonian are $\pm\frac{1}{2}\sqrt{\epsilon^2 + \Delta^2}$. Quantum mechanics tells us that a qubit's dynamics are governed by the Schrödinger's equation

$$i\hbar\frac{d}{dt}|\psi(t)\rangle = H(t)|\psi(t)\rangle \tag{1.5}$$

which, when solved for a time independent Hamiltonian leads to a unitary evolution

$$|\psi(t)\rangle = \exp\left\{\frac{-itH}{\hbar}\right\}|\psi(0)\rangle \tag{1.6}$$

with $|\psi(0)\rangle$ representing the initial state of the system, and $|\psi(t)\rangle$ the evolved state at some arbitrary time t. Full control of the qubit involves appropriately adjusting (to the task at hand) the parameters of the Hamiltonian in Equation 1.4. This often includes sporadic changes and/or continuous "driving" of one or both $\epsilon$ and $\Delta$.

## 1.2 Multi-Qubit Systems

In order to perform useful quantum computation, many qubits are needed. When two quantum systems are involved, the combined state space is a tensor product of the state spaces of the constituents. This concept can be further generalized to an arbitrary number of qubits $n$. The dimension of the combined $n$ qubit Hilbert space is then $2^n$. If we suppose that the state of the first system is $|\psi_1\rangle$ and the second $|\psi_2\rangle$, and the combined state can be written as a tensor product of these, namely as

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle . \tag{1.7}$$

then we say that the system is separable, and not entangled. If, on the other hand, a state of a multi-qubit system cannot be written as a simple tensor product of the states of all the constituents, the combined system is entangled. The dynamics of multi-qubit systems still follow the Schrödinger's equation (see Eq. 1.5 and 1.6), except the dimension of the Hilbert space involved is greater.

## 1.3 Density Operators And Mixed States

So far we have only looked at pure quantum states - these are states which have a definite state vector. A more general case involves a description which only assumes a particular state vector with a given probability, say $p$. Such a state is called a mixed state, and one can describe it as set of pairs

$$\rho = \{(p_1, |\psi_1\rangle), (p_2, |\psi_2\rangle), ..., (p_k, |\psi_k\rangle)\}. \tag{1.8}$$

This notation means that there is a probability $p_i$ of the system being in a quantum state $|\psi_i\rangle$. It is easy to see that a pure state is simply a state with a single $p_i = 1$, and all others set to zero. A more compact notation can be introduced, and we can rewrite Equation 1.8 as

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| . \tag{1.9}$$

$\rho$ is often called a density operator. It is always a Hermitian, semi-positive matrix, which may be infinite dimensional, depending on the type of quantum system it describes. Furthermore, the trace of any density matrix is always equal to 1. The evolution of the system under some unitary operation $U$ can be described as simply applying $U$ to every term in Equation 1.8, which when written using Equation 1.9 leads to

$$\sum_i U p_i |\psi_i\rangle \langle\psi_i| U^\dagger = U \left( \sum_i p_i |\psi_i\rangle \langle\psi_i| \right) U^\dagger \tag{1.10}$$

$$= U\rho U^\dagger. \tag{1.11}$$

Finally, it is worth stressing that a density matrix describes a pure state if and only if $\rho = \rho^2$ — i.e. if $\rho$ is a projector.

## 1.4 Circuit Model

We can describe quantum computation as a sequence of unitary operations which are made to act on a given quantum system. The most well known model for quantum computation is the circuit model. It is a direct quantum generalization of its counterpart from the classical world. It consists of a register of $n$ qubits with a portion of them containing some initial state on which the computation is to be preformed. During computation, some of those qubits undergo reversible unitary evolution. At the end of the processing, the qubits are read out. An example of a

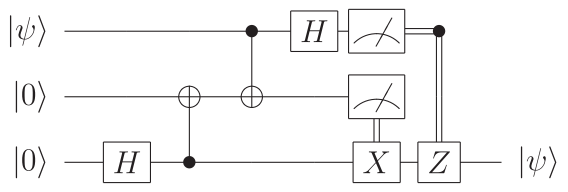


Figure 1.2: A teleportation circuit. The "computation" involves teleporting an arbitrary state $|\psi\rangle$ from the 1st qubit to the 3rd.

circuit representing quantum teleportation is shown in Fig. 1.2.

## 1.5 Single-Qubit Gates

The unitary operations which are involved in the computation are called quantum gates. Some of the most important one-qubit gates are the Pauli operators. They can be written as

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{1.12}$$

For simplicity, we will often refer to $\sigma_x$ as $X$, $\sigma_y$ as $Y$, and $\sigma_z$ as $Z$. These matrices are linearly independent, and therefore along with $I$, can be used to form any $2 \times 2$ matrix. Arbitrary qubit rotations can be made along any axis. Suppose that we wish to rotate our qubit around some arbitrary vector $\hat{n}$. We could use the Pauli operators shown in Eq. 1.12 to construct a rotation operator

$$\begin{aligned} R_{\hat{n}}(\theta) &= \exp\left\{-i\frac{\theta}{2}\left(\hat{n}\cdot\hat{\sigma}\right)\right\} \qquad (1.13)\\ &= \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right)\left(n_x X + n_y Y + n_z Z\right) \qquad (1.14)\end{aligned}$$

where we took $\hat{n} = (n_x, n_y, n_z)$ and $\hat{\sigma} = (X, Y, Z)$. We easily observe that $\theta$ determines the angle of rotation. Euler has shown that a rotation about any axis can be written explicitly in terms of rotations around only two distinct axes. In a special case where $\hat{m}$ and $\hat{n}$ are orthogonal, this can be expressed as

$$U = e^{\{i\alpha\}} R_{\hat{m}}(\beta) R_{\hat{n}}(\gamma) R_{\hat{m}}(\kappa) \qquad (1.15)$$

with $\hat{m}$ and $\hat{n}$ representing these different rotation axis, and $\alpha$, $\beta$, $\gamma$ and $\kappa$ the rotation angles. This result turns out to be useful in quantum computing, as in principle at least, it simplifies the control Hamiltonian; we only need control along at most two axes. A few important one-qubit operators are presented in Eq. 1.16 below

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{pmatrix}, \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}. \qquad (1.16)$$

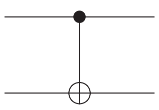
H stands for the Hadamard gate, T for a $\pi/8$ gate and S for a phase gate.

## 1.6 Two-Qubit Gates

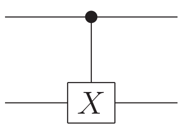
Two-qubit gates play an important role in quantum computation as they are needed in order to achieve entanglement between qubits. Furthermore, it has been shown that only arbitrary one-qubit rotations along with certain two-qubit gates are needed to perform any computation on any number of qubits. If we assume that $U$ is some one-qubit gate, then we can define a Controlled-$U$ operation on two-qubits, to act in the following way. If the first qubit (control) is in the $|1\rangle$ state, then $U$ is applied to the second one (target). Otherwise, if the first (control) qubit is in a $|0\rangle$ state, nothing happens to the second qubit. Two important examples of such operations are CNOT and CPHASE gates. Their matrix representation, as well as circuit symbols are shown in Figure 1.3.

CNOT

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

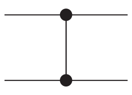


or



CPHASE

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$



or

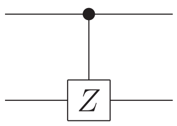


Figure 1.3: Names, matrix representations and circuit model symbols for CNOT and CPHASE two-qubit gates.

## 1.7 Organization Of This Thesis

In this last section, we outline how this thesis is organized. In Chapter 1, we gave a brief introduction to the area of quantum mechanics, talked about the fundamental building blocks of quantum computers — the qubits, and explained the mathematical formalism used to describe them. In Chapter 2, we first outline the basis of quantum error correction and describe the stabilizer formalism and how it applies to error correcting codes. We then introduce surface codes in some detail and finally outline what the threshold theorem is, and its importance. Chapter 3 includes a discussion of the numerical threshold calculation of a surface code. Here we present the approach that was used, assumptions that were made and actual results that were obtained. In Chapter 4 we give another introduction, this time to quantum computing with superconducting circuits. After a brief historical treatement, we study a Josephson junction, flux qubit, DC-SQUID, and finally describe how coupling of flux qubits can be done. Chapter 5 includes a discussion of the three qubit, two DC-SQUID coupler. Here we study its properties, and in particular concentrate on numerical calculations of the coupling energy and how it varies in different coupler geometries. Finally, Chapter 6 contains the conclusions of the work presented throughout the thesis.

# Chapter 2

# Quantum Error Correction

## 2.1 Basics

Although the theory of quantum information is far along when it comes to potential benefits quantum computers can bring, actually building a functional quantum computer is a formidable challenge. To a large extent this has to do with the contradicting nature of how such a device has to behave. On one hand it needs to be able to interact with its environment in order for us to manipulate and measure its constituents — the qubits, but on the other hand this interaction has to be kept to a minimum in order to minimize decoherence, which destroys the very advantage that a quantum computer brings to the table over its classical counterpart. A realistic assumption is therefore that a functional quantum computer will have to be able to cope with errors that will naturally develop from unwanted interactions with the environment. The study of quantum error correction deals with the very nature of this unwanted system-environment interaction, and in particular provides a clear path of how one can overcome it. The fact that arbitrary, yet reliable quantum computation can still be performed is not entirely obvious and is covered in some detail in Section 2.4 where a discussion of threshold error rate is presented. Here we concentrate on some properties of quantum error correcting codes.

Quantum correcting codes provide a means of encoding quantum states and keeping them safe from certain types of errors. These codes are described in terms of three numbers: $[[n, k, d]]$, $n$ being the total number of qubits used, $k$ the number of encoded qubits, and $d$ the code distance. The code distance is related to the amount of errors $t$ that a given code can tolerate by the relation $t = \lfloor (d-1)/2 \rfloor$. A convenient way to think about the code distance is to count the minimum number of single qubit operations one has perform in order to go from one codeword to any other.

As already described, an arbitrary 2x2 operator can be expressed as a linear superposition of Pauli matrices $\mathbb{1}, X, Y, Z$,

$$E = \alpha_1 \mathbb{1} + \alpha_2 X + \alpha_3 Y + \alpha_4 Z \tag{2.1}$$

with $\alpha_i \in \mathbb{C}$. A consequence of quantum mechanics is that by solely correcting for errors $X$ and $Z$ we can correct for completely arbitrary, unwanted rotations of the qubit. Furthermore, quantum mechanics is linear, and therefore we can be assured that if a given error correcting code can protect against some errors, it will be able to correct against a linear combination of these errors.

A condition for an existence of a recovery operation is:

$$\langle \Psi_i | E_k^\dagger E_l | \Psi_j \rangle = C_{kl} \delta_{ji} \tag{2.2}$$

where $|\Psi_i\rangle$, $|\Psi_j\rangle$ are codewords, $E_k$, $E_l$ the errors and $C_{kl}$ a positive semi-definite matrix. We can easily see that the requirement in Equation 2.2 means that a result of an error operator acting on a codeword, necessarily needs to be orthogonal to an error (the same or different) acting on any other codeword. This orthogonality implies that one can come up with a recovery operation which can get rid of the encountered error. The condition allows for different error operators acting on a given state to overlap. This happens when the matrix $C$ does not have a maximum rank, and we say that the the code is *degenerate*. A code in which the matrix $C$ does have a maximum rank is called is *non-degenerate*.

There is an important bound that can shed a little light on the properties of error correcting codes. It is called the quantum Hamming bound, which is only valid for non-degenerate codes, and is analogous to the Hamming bound known from the study of classical error correction. It helps us determine how many qubits $n$ we need to encode $k$ logical qubits while still being able to tolerate at most $t$ errors. To answer this question, and formulate the Hamming bound, let us assume that we can have $j$ errors, with $j \leq t$. This will lead to $\binom{n}{j}$ locations where errors could occur. Furthermore there are three possible errors at each location; $X$, $Y$ or $Z$, and therefore $3^j$ possible errors. We can then conclude that the total number of errors that can happen is at most $\sum_{j=0}^{t} \binom{n}{j}$. Since the code is non-degenerate, and therefore any error will take any codeword to an orthogonal state to any other codeword (possibly being acted upon by another error), we need the total size of the Hilbert space of $n$ qubits to "fit" all of the possibly erroneous states. This leads to an inequality

$$\sum_{j=0}^{t} \binom{n}{j} 3^j 2^k \leq 2^n \tag{2.3}$$

A well known example is a case with $k = 1$ and $t = 1$, which leads to $n = 5$ — a smallest known quantum error correcting code that can account for one arbitrary error on any of the qubits. The quantum Hamming bound tells us that there cannot be a non-degenerate error correcting code that will use fewer than 5 qubits.

## 2.2 Stabilizer Codes

The stabilizer formalism was invented by Gottesman in 1997 [23]. It provides an elegant and efficient method for describing a collection of states and their dynamics. Although its application is limited to a subset of all possible states, it is very useful when dealing with error correcting codes. Let us first briefly review the mathematical formalism of groups which will let us concisely define what a stabilizer is.

A group $G$ is a set of elements with binary multiplication, which satisfies the following properties:

- Closure; $g_1 \cdot g_2 \in G$ for all $g_1, g_2 \in G$
- Associativity; $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$ for all $g_1, g_2, g_3 \in G$
- Identity; $\exists e \in G$ such that $e \cdot g = g$ for all $g \in G$
- Inverse; $\forall g \in G$, $\exists g^{-1} \in G$ such that $g^{-1} \cdot g = e$ for some $e \in G$

A group $G$ is Abelian if $g_1 \cdot g_2 = g_2 \cdot g_1 \;\; \forall \;\; g_1, g_2 \in G$. A simple example of a group is the one-qubit Pauli group $G_1$, which consists of elements

$$G_1 = \{\pm \mathbb{1}, \pm \mathrm{i}\mathbb{1}, \pm X, \pm \mathrm{i}X, \pm Y, \pm \mathrm{i}Y, \pm Z, \pm \mathrm{i}Z\}. \tag{2.4}$$

The factors of $\pm 1$ and $\pm \mathrm{i}$ are needed to satisfy the closure condition defined above. When studying quantum error correction, we are usually interested in a group of Pauli operators on $n$ qubits (denoted as $G_n$). Such a group consists of the $n$-fold product of all the Pauli matrices along with the multiplication factors of $\pm 1$ and $\pm \mathrm{i}$. It is easy to see that it has $4^{n+1}$ elements. Luckily, any group can be described in a more compact way using the concept of group *generators*. We say that a set of operators $g_1, g_2, ..., g_l$ generates a given group $G$ if every element in $G$ can be written as a (possibly repeated) product of elements from this set. A shorthand notation for $G$ in terms of its generators is $G = <g_1, g_2, ..., g_l>$. It can be shown [37] that any group can be described by at most $\log_2 |G|$ generators with $|G|$ representing the number of elements in $G$. We are now ready to define a stabilizer.

**Definition** A stabilizer $S$ for a given vector space $V$, is a subgroup of $G_n$ which fixes every element in a subset of states defined by $V$.

It is useful to note that in order for the vector space $V$ to include non-null elements, the stabilizer $S$ cannot contain the $-\mathbb{1}$ element (which is in $G_n$). One can easily see this by noting that for any state $|\Psi\rangle$ we have $-\mathbb{1}\,|\Psi\rangle = |\Psi\rangle$ only if $|\Psi\rangle$ is null.

The stabilizer formalism is not only useful when describing the state of a given system, but also its dynamics. Let us suppose that stabilizer $S$ describes a quantum

| Input | Unitary Operation $U$ | Evolution | Result (output) |
|---|---|---|---|
| $X$ | $H$ | $HXH^\dagger$ | $Z$ |
| $Z$ | $H$ | $HZH^\dagger$ | $X$ |
| $X$ | $S$ | $SXS^\dagger$ | $Y$ |
| $Z$ | $S$ | $SZS^\dagger$ | $Z$ |
| $X_1I_2$ | $C_{1,2}$ | $C_{1,2}X_1I_2C_{1,2}^\dagger$ | $X_1X_2$ |
| $I_1X_2$ | $C_{1,2}$ | $C_{1,2}I_1X_2C_{1,2}^\dagger$ | $I_1X_2$ |
| $Z_1I_2$ | $C_{1,2}$ | $C_{1,2}Z_1I_2C_{1,2}^\dagger$ | $Z_1I_2$ |
| $I_1Z_2$ | $C_{1,2}$ | $C_{1,2}I_1Z_2C_{1,2}^\dagger$ | $Z_1Z_2$ |

Table 2.1: A table showing how the different members of the Pauli group ($X$, $Z$ — omitting $Y$, as it can be constructed from these) evolve under the influence of different unitary operations (shown in the 2nd column). $C_{i.j}$ implies a CNOT gate with $i$th qubit acting as control, and $j$th as target.

state $|\Psi\rangle$ and we want to apply a unitary operator $U$. Then since $|\Psi\rangle$ is stabilized by any element $g \in S$ we have

$$U|\Psi\rangle = Ug|\Psi\rangle = UgU^\dagger U|\Psi\rangle \tag{2.5}$$

which leads us to conclude that $UgU^\dagger$ now stabilizes the newly evolved state $U|\Psi\rangle$. Some important examples of this evolution in a case of unitary gates $H$, $S$ and $CNOT$ are shown in Table 2.1.

In addition to understanding how the stabilizer evolves, we need to understand what happens as we perform measurements. Let us suppose we want to measure some operator $M$, and that $g$ represents any of the generators that describe a stabilizer $S$. There are three different scenarios that we need to consider.

1. *$M$ can be expressed as a product of stabilizers*
   No change is needed as the state is already an eigenstate of $M$.
   Example: $S =< X_1, X_2 >$, $M = X_1X_2$. We can rewrite the stabilizer as $S =<X_1X_2, X_2>$ and easily see that the state must be in a $+1$ eigenstate of $M$.

2. *$M$ cannot be expressed as a product of the generators of $S$, and commutes with every $g$*
   The operator is added to the list of generators that represent the stabilzier, with a sign which depends on whether the state was projected onto $+1$ or $-1$ eigenvalue of $M$.
   Example: $S =<X_1X_3, X_2X_3>$, $M = X_1$. We simply need to add $\pm X_1$ to the stabilizer, with the sign depending on the measurement outcome. This results in the final state stabilized by $S =<\pm X_1, X_1X_3, X_2X_3>$

3. *$M$ cannot be expressed as a product of the generators of $S$, and anti-commutes with one or more generators $g_i$ of $S$.*

One of the anti-commuting generators is selected, and the other anti-commuting generators are multiplied by it, to make them commute with $M$. The first generator is then replaced by $M$ with a sign which depends on whether the state was projected onto $+1$ or $-1$ eigenvalue of $M$.
Example: $S = < X_1X_2, X_2X_3, X_3X_4, Z_1Z_2Z_3Z_4 >$, $M = Z_2$. We can first rewrite the stabilizer as $S = < X_1X_3, X_2X_3, X_3X_4, Z_1Z_2Z_3Z_4 >$, so that only one of the generators anti-commutes with the measurement operator. The second generator is then replaced by the measurement operator with the sign depending on the measurement outcome. The final state can then be written as $S = < X_1X_3, \pm Z_2, X_3X_4, Z_1Z_2Z_3Z_4 >$.

Now that we have an understanding of how to perform stabilizer manipulation, and how one can represent measurements, we can focus on an important limitation. It turns out that only a subset of all operators on $n$ qubits can be easily expressed in a stabilizer formalism. Let us consider a set of gates that take any element in $G_n$ to elements in $G_n$ under conjugation. Such a set is referred to as a *normalizer* of $S$. It can be proven that any gate in this set can be constructed from at most $O(n^2)$ gates solely consisting of CNOT, Hadamard and Phase. This set, however, is not universal, which means that it cannot be used to construct a completely arbitrary operation (which possibly takes gates out of $G_n$) on n qubits, although it still allows us to study some very important concepts in quantum computing — quantum error correction being one of them. Furthermore, a theorem by Gottesman and Knill (known as the Gottesman-Knill Theorem) [23] states that any quantum computation which consists of only CNOT, Hadamard and Phase gates, with a possibility of classical control which may depend on measurement outcomes can be simulated on a classical computer.

| 5-Qubit Code [[5,1,3]] | | | | | 7-Qubit Code [[7,1,3]] | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | I | I | I | X | X | X | X |
| X | X | Z | I | Z | I | X | X | I | I | X | X |
| Z | X | X | Z | I | X | I | X | I | X | I | X |
| I | Z | X | X | Z | I | I | I | Z | Z | Z | Z |
| Z | I | Z | X | X | I | Z | Z | I | I | Z | Z |
| | | | | | Z | I | Z | I | Z | I | Z |

Table 2.2: Stabilizer representation of two quantum error correcting codes; the 5-qubit code and the Steane 7-qubit code. Each column corresponds to a single physical qubit, and each row represents a stabilizer generator.

A couple of well known error correcting codes that can be described in terms of the stabilizer formalism are the 5-qubit code, and the 7-qubit Steane code. Their stabilizers are shown in Table 2.2. Detecting errors using the stabilizer formalism is straightforward. When an error acts on any of the qubits, we simply measure all the stabilizer generators. Given that the number of errors that occur is within the range of what a given code can tolerate, which qubit suffered an error, and

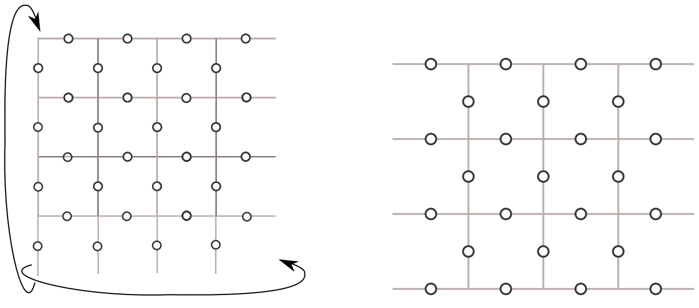
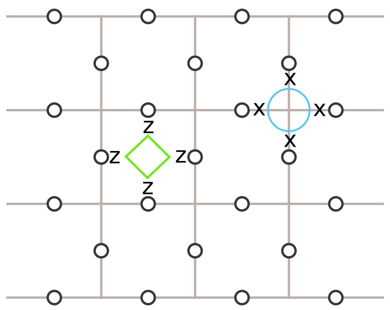

(a) Toric code (b) Planar code

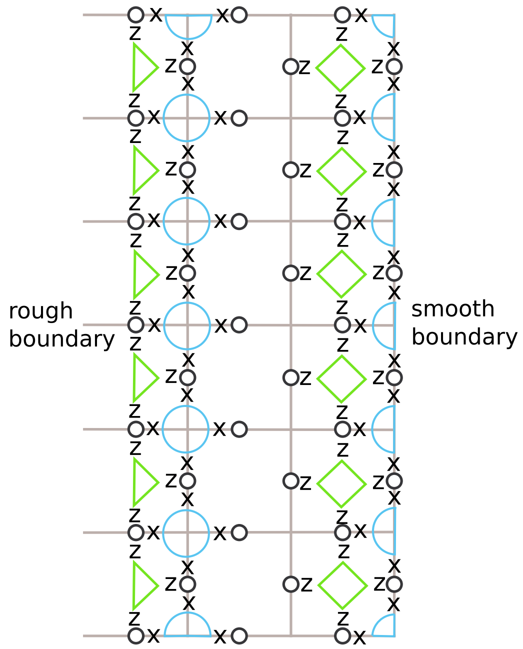
Figure 2.1: Two examples of a surface code. In both scenarios, qubits are arranged on a square lattice. The two cases differ in boundary conditions. a) In the toric case, one can think of the lattice as being the surface of a torus, with the left (top) boundaries in the diagram "touching" the right (bottom) boundaries. b) In the planar code case, the top and bottom as well as left and right pairs of boundaries are the same — either smooth or rough (see text for explanation).

what kind of error it was, can be determined from the results of the measurement. As a simple example let us consider the 7-qubit Steane code shown in Table 2.2. Furthermore, let us suppose that an $X$ error happens on the first qubit. If we were to measure all the stabilizer generators, we would obtain an outcome of $+1$ from all but the 6th one (6th row in Table 2.2), since the error operator $X$ anti-commutes with it. The 6th generator would give a $-1$. From this information, and the fact that measuring the other generators would give an eigenvalue of $+1$, we would know that applying $X$ to the first qubit would undo the error.

## 2.3 Surface Codes

Surface codes were invented by Alexei Kitaev in 1997 [28]. They are topological quantum error-correcting codes in which we can think of qubits being arranged on a lattice which in turn lives on some surface. Surface codes belong to the family of stabilizer codes, and therefore can be elegantly described using the stabilizer formalism presented above. Surface codes have some appealing properties. They are what is called "local check codes", and this means that in order to perform syndrome measurements (and understand where the errors occurred) one only ever needs to measure operators that are nearest neighbors. This in practice may prove to be a very useful feature, since for many systems interacting qubits that are close to each other is substantially less difficult than ones that are further apart. We can think of physical qubits as being arranged on the edges of a lattice as shown in Figure 2.1. In this discussion we concentrate on two different cases, a toric code and a planar code. In the toric code, the qubits are arranged on a lattice which can be thought of as spread over a surface of a torus, and in a planar code case we think of the qubits as living on a simple 2-D plane. Mathematically we can describe a general surface code as a simultaneous $+1$ eigenstate of mutually commuting check


(a)



(b)

Figure 2.2: a) Check operators consist of $ZZZZ$ around every plaquette and $XXXX$ around every vertex (where possible — there is one less operator around the edges of the lattice in a planar code case). b) Two different types of surface code boundaries; rough (on the left) with three-term $Z$ operators, and smooth (on the right) with three-term $X$ operators.

operators (which are stabilizer generators). We can define these check operators as

$$A_j = \prod_{j \in V} X_j \tag{2.6}$$

and

$$B_j = \prod_{j \in P} Z_j \tag{2.7}$$

with $V$ representing any vertex on a lattice and $P$ any plaquette, as is schematically shown in Figure 2.2a. We call the outcomes of the measurements of $A_j$ and $B_j$, syndromes. It is worth noting that not all these operators are independent, and in fact one has conditions $\sum_{j \in V} A_j = \mathbb{1}$ and $\sum_{j \in P} B_j = \mathbb{1}$. Furthermore, operators $A_j$ and $B_j$ always share either zero or two edges, hence they commute.

A planar code has four boundaries, two that are called “smooth” and two that are called “rough”. Smooth boundaries have four-term $Z$ stabilizer generators, and three-term $X$ stabilizer generators, whereas rough boundaries have four-term $X$ stabilizer generators and three-term $Z$ stabilizer generators. A planar code, with two rough and two smooth boundaries can encode a single logical qubit. One can easily see this by noting that a square lattice (such as the one in Figure 2.1b) with

$n$ qubits on each of the sides has a total of $2n^2 + 2n + 1$ qubits and $2n^2 + 2n$ independent stabilizer generators. This leads to $2n^2+2n+1-2n^2+2n = 1$ degrees of freedom, and hence one logical qubit. In a toric code case, a square lattice (such as the one shown in Figure 2.1a) with $n$ qubits across would have a total of $2n^2$ qubits, and $2n^2 - 2$ independent stabilizer generators, hence 2 degrees of freedom, and could be used to encode two qubits.

Detecting errors involves measuring check operators, and observing which ones give a value of $-1$ (due to anti-commuting with errors). This information helps us guess where errors occurred.

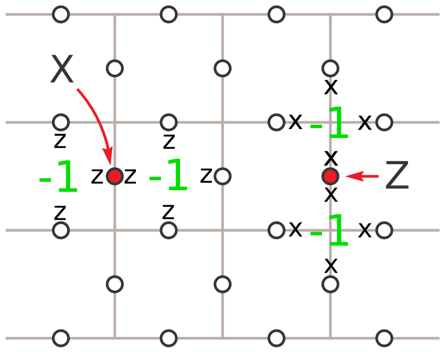


Figure 2.3: Errors can be detected by measuring stabilizer generators around each plaquette and each vertex.

Figure 2.3 shows this schematically. A single $X$ error on one of the qubits anti-commutes with the $B_j$ operators directly to the right and left of the qubit. Likewise, a single $Z$ error anti-commutes with the $A_j$ operators directly above and below the qubit suffering an error. In practice, of course, errors do not have to occur on their own, and often one can observe multiple instances next to each other. In these cases, the error operators form error chains throughout the lattice. Since only the ends of such error chains anti-commute with the check operators, determining where errors occurred often involves guessing the most likely scenario.

There are two types of chains that we need to discuss in more detail. The first involves ones that form closed loops. It turns out that these operators are within the stabilizer of the code, meaning they in fact can be written as a product of the stabilizer generators (check operators $A_j$ and $B_j$). Furthermore, they commute with each check operator, hence have no effect on the encoded qubits.

The other types of errors that we need to consider are chains that span across the whole dimension of the lattice. In the planar case this means chains that connect opposite boundaries of the same type (either left to right, or top to bottom), and in the toric case, chains that span all the way across a given dimension of the lattice. These operators, despite commuting with all the check operators, cannot be expressed as their product. They turn out to the change the encoded, logical state of the qubit and hence are called logical errors. Two examples are shown in Figure 2.4. $Z_L$ is a chain of $Z$ operators that connects two rough boundaries, and

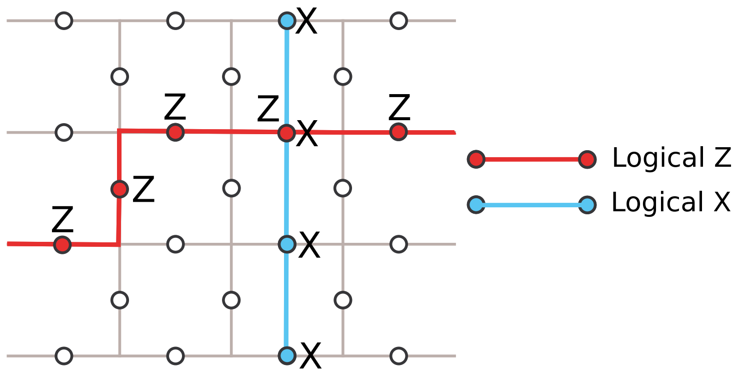


Figure 2.4: On a planar surface code, a logical $Z$ ($X$) error is a chain of $Z$ ($X$) operators that spans the whole lattice, and connects rough (smooth) boundaries.

$X_L$ a chain of $X$ operators that connects two smooth ones. Figures 2.5a and 2.5b show an analogous case in a toric code. Since a toric code has no boundaries, the chains need to go all the way around one of the dimensions of the lattice (torus). A $Z_L$ along the vertical dimension acts on one qubit, and along the horizontal dimension, on the other. Likewise in a case of $X_L$. It is worth stressing that since these operators change the logical state of the encoded qubit, when done purposely, they can be regarded as simply the logical versions of gates $X$ and $Z$, and not as errors. Applying a logical version of $Y$ would naturally include applying both $X_L$ and $Z_L$. In Section 2.3.1 we will come back to the notion of performing other gates on a surface code, and outline techniques that can be used to make the computation universal.

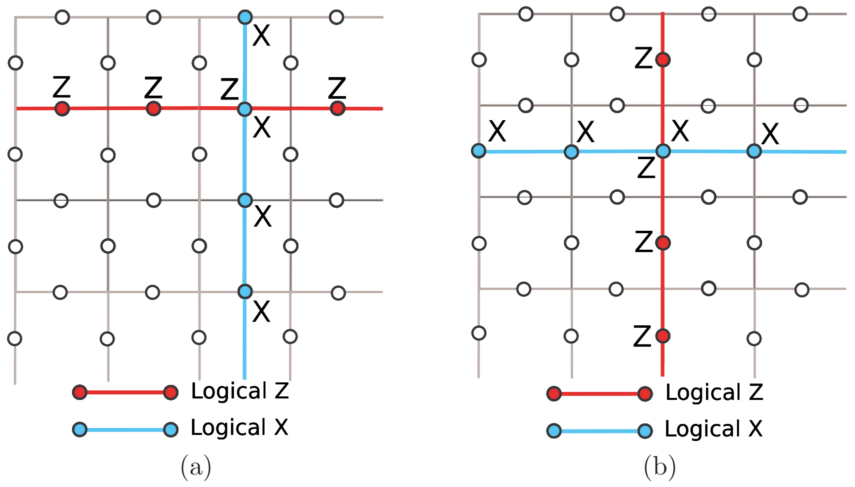


Figure 2.5: We can encode two qubits in a toric code, since there are no boundaries. Figure a) shows how logical operations are done on the first qubit, and b) on the second. In both cases, the logical operations involve applying a set of operators (either $X$ or $Z$) in a chain that goes around one of the dimensions of the torus.

### 2.3.1 Computation On A Surface Code

In our discussion so far, we have reviewed general properties of surface codes and showed the general principles of how errors can be detected and corrected. We have also described how a logical $X$ or logical $Z$ operation can be performed on a single encoded qubit in the planar code case or two qubits in a toric case. These operations, although important, do not paint the full picture. In this section, we give a brief outline of how a surface code lattice can actually be used to perform universal quantum computation. Due to the fact that this topic is rather involved, we only outline the most general ideas, and point the reader to other more detailed discussions [44, 22, 21].

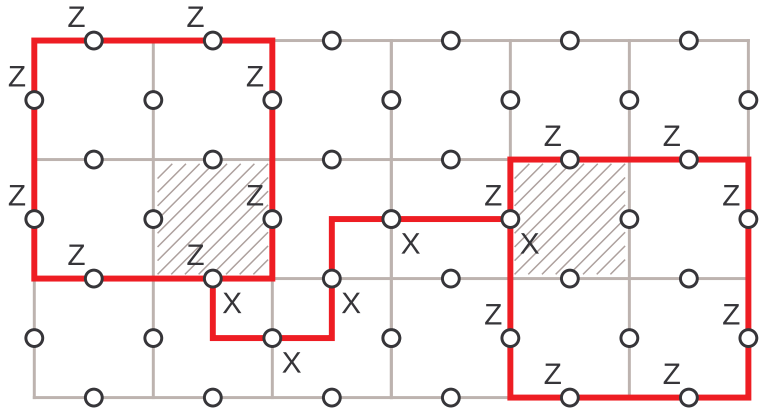


Figure 2.6: A logical qubit can be represented by two defects in the lattice. The defects are created by simply neglecting to enforce that the eigenvalues associated with their respective check operators around a given plaquette (in the case of a smooth defect) are +1. Logical operations on such a qubit are done by applying a chain of $X$ operators between the two defects in the case of $X_L$, and a chain of $Z$ operators around one of the defects in the case of $Z_L$. An analogous construction exists for defects that originate on the vertices and not plaquettes, as explained in the text. Figure created by Austin Fowler, and taken from [22] with permission.

We have already established that logical operations can be described by chains of operators that span the lattice and connect boundaries (in a planar code case) or simply surround a given dimension of the torus (in the toric code case). Another approach in creating a logical qubit on a surface code, is by introducing artificial boundaries. This is done by simply stopping to enforce that check operators at certain spots in the lattice are always in their respective +1 eigenvalues, and hence creating “holes” or “defects”. An explicit example is shown in Figure 2.6, where we observe two plaquette stabilizer generators (shaded regions), which we choose to neglect. These two newly created defects have boundaries, which can be used for logical operations. The type of boundary a defect has depends on whether we chose to neglect stabilizer generators associated with plaquettes (as shown in Figure 2.6) or vertices. The first gives us a smooth boundary, whereas the second a rough one. A chain of $X(Z)$ operators between two smooth (rough) defects represents a logical $X(Z)$ error, and a chain of $Z(X)$ operators surrounding one of those smooth (rough) defects represents a logical $Z(X)$ operator. Defects can be easily moved around the lattice, and even enlarged. A detailed procedure of how this can be done is explained in [22]. Hence we see that two lattice defects can be used to describe a single logical qubit. Assuming the lattice we are using is big, and each pair can be sufficiently separated, this approach provides us with a consistent way of preparing many logical qubits by simply creating lattice defects.

Another useful consequence of this approach is that a CNOT gate between two pairs of defects can be easily performed. Such an operation involves braiding parts of the qubits (i.e. one of the defects in each) around each other. To show this, we can use the fact that a quantum gate can be specified by how it acts on a set of basis states, or basis of stabilizers. In particular, a CNOT gate between qubits 1 (control) and 2 (target), is described by how it acts on a set of generators $X_1I_2$, $I_1X_2$, $Z_1I_2$, $I_1Z_2$, with the result expressed in the last four rows of Table 2.1. Therefore in order to show how braiding defects around each other could be equivalent to a CNOT gate, we need to show that the evolution of these stabilizer generators under braiding gives the same results. An example of one case where the input state is originally stabilized by the operator $X_1I_2$ is presented in Figure 2.7. We see that we are dealing with two qubits, the top one formed out of two smooth defects, and the bottom one out of two rough ones. We assume that the top qubit is initialized in the state stabilized by $X_L$. The process involves moving one of the defects of the top qubit between the defects of the bottom one (Figure 2.7 a, b, c) until the defects comes back to its original starting position. This operation "wraps" the $X_{1,L}$ operator around one of the defects of the rough qubit (Figure 2.7), which in turn is equivalent to performing a $X_L$ operation on it. This in turn leads to the final state stabilized by the operator $X_{1,L}X_{2,L}$, agreeing with the desired result from Table 2.1. A similar argument can be used in the three remaining cases. Finally, it is worth stressing that in the shown construction, the CNOT gate is performed between two different types of qubits — the first being a smooth one, and the other a rough one. In practice this may not be entirely ideal, and slightly modified approaches which allow to perform a CNOT gate between two qubits that are of the same type (smooth or rough) have been devised. They involve a temporary creation of a dual qubit (a rough if dealing with two smooth ones, and vice-versa), which at the end of the operation disappears.

So far we have discussed how a logical version of Pauli $X$, $Z$, $Y$ and a CNOT gates can be applied. Unfortunately as described in Section 2.2 this set of gates is not universal, but luckily can be made universal by providing an ability to perform gates $\exp(i\frac{\pi}{8}Z)$, $\exp(i\frac{\pi}{4}Z)$ and $\exp(i\frac{\pi}{4}X)$ [45]. The first step in accomplishing the task of turning this scheme into one where arbitrary quantum computation can be performed, is done by being able to prepare a couple of special logical states

$$|Y\rangle \quad = \quad |0\rangle + i\,|1\rangle \tag{2.8}$$

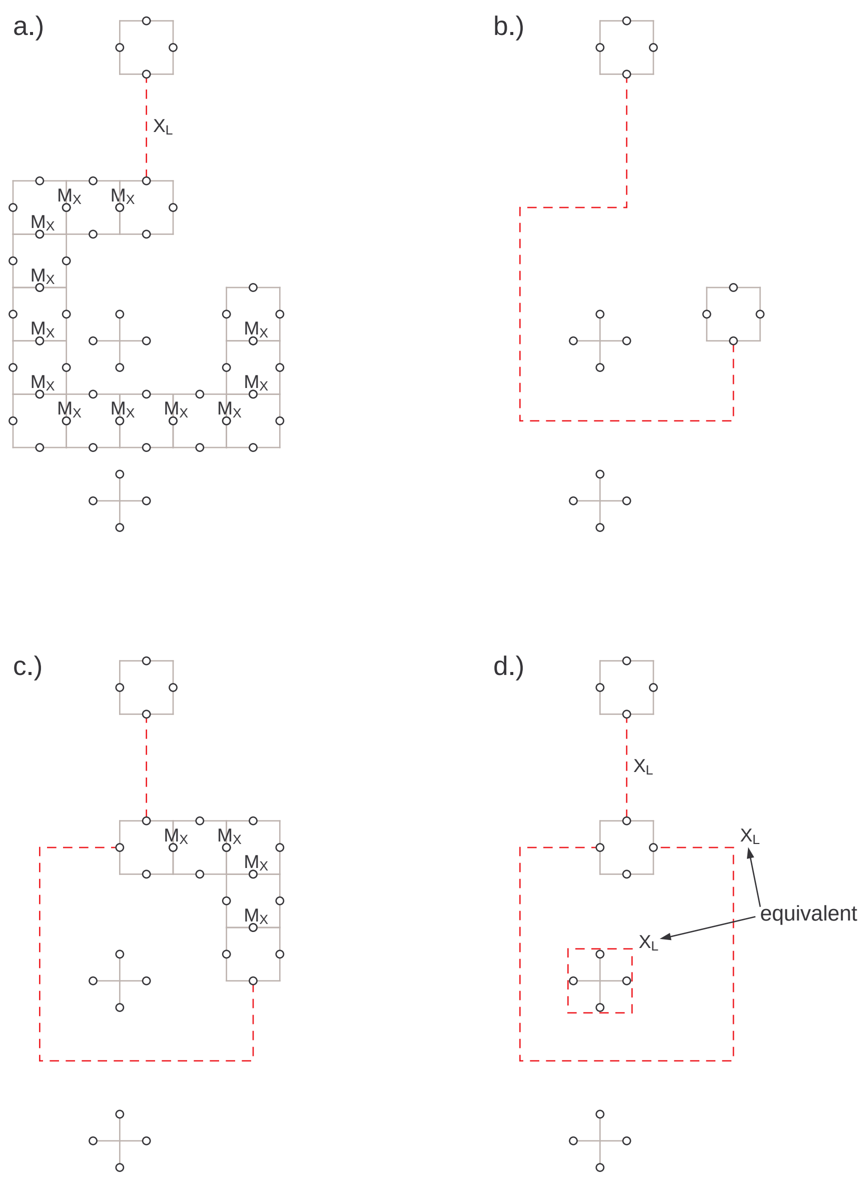


Figure 2.7: A CNOT gate executed between two logical qubits. The top one is a smooth qubit, playing the role of control, whereas the bottom one is a rough qubit, playing the role of the target. The gate involves braiding one of the defects of one of the qubits, around a defect of the other qubit. a) The two qubits start in a state $X_L I$, where the first and second operators are acting on Hilbert spaces associated with the top and bottom qubits respectively. b) One of the defects of the top qubit is moved around one of the defects of the bottom qubit. c) The motion continues until the defect ends up back in its starting position. d) This results in the operator $X_L I$ evolving to $X_L X_L$, which in turn completes a CNOT gate. Figure created by Austin Fowler, and taken from [22] with permission.

and

$$|A\rangle \;=\; |0\rangle + \exp\left(\frac{i\pi}{4}\right)|1\rangle\,. \tag{2.9}$$

Assuming we can rotate a single physical qubit to some arbitrary superposition state $\alpha |0\rangle + \beta |1\rangle$, via a rather involved set of operations [22] a smooth or rough logical qubit with that same superposition state can be created. Constructing our desired states just in this way is not fault tolerant, however, and methods of state distillation [7, 46] need to be employed with which the desired states can be prepared with fidelity arbitrarily close to 1. These naturally add extra complexity to computation, but have been shown to be efficient. For a more detailed discussion which describes these procedures in detail, we point the reader to [44, 22].

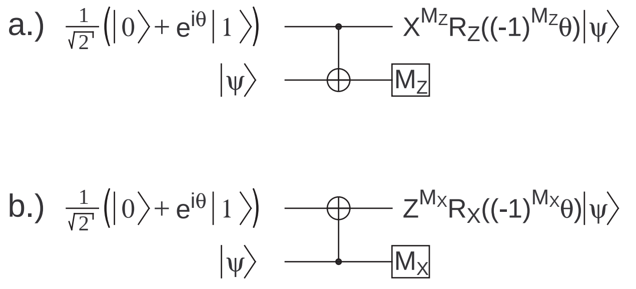

a.)
$\frac{1}{\sqrt{2}}(|0\rangle + e^{i\theta}|1\rangle)$
$|\psi\rangle$
$X^{M_Z}R_Z((-1)^{M_Z}\theta)|\psi\rangle$
$M_Z$
b.)
$\frac{1}{\sqrt{2}}(|0\rangle + e^{i\theta}|1\rangle)$
$|\psi\rangle$
$Z^{M_X}R_X((-1)^{M_X}\theta)|\psi\rangle$
$M_X$

Figure 2.8: Two circuits that can be used to obtain arbitrary rotations around a) $Z$ and b) $X$ axis. In both cases it is assumed a specially prepared ancilla qubits can be created. Figure created by Austin Fowler, and taken from [22] with permission.

Given that our ancilla qubits can be prepared in these special states $|Y\rangle$ $|A\rangle$, the final step in achieving universal computation is using them to perform rotations of $\pi/4$ and $\pi/8$ around the $X$ and $Z$ axis. This can be done by non-deterministic circuits shown in Figure 2.8. The general idea involves entangling the specially prepared ancilla qubits with a qubit which we wish to rotate, and measuring it. The ancilla qubit will then have a chance of containing the desired, rotated state. However, after one application of such a circuit, the resulting gate may be a rotation in the opposite direction, for example a $R_X(-\theta)$ as opposed to $R_X(\theta)$. Luckily, these cases can be determined from a measurement of a negative eigenvalue of the qubit which originally contained the input state, and therefore easily corrected.

This discussion, although brief, lets us conclude that full universal quantum computation on a surface code is possible.

## 2.4 Error Threshold

Although it has been known for some time that quantum computation may be possible, initially it was not clear how much physical error a quantum computer can tolerate. The threshold theorem, which attempts to provide an answer to this question, was presented by Knill, Laflamme and Zurek [31] and independently by Aharonov and Ben-Or [2]. It states that there exists an error rate $\epsilon$, such that as long as all physical error rates are smaller than $\epsilon$, arbitrary quantum computation is possible. An alternative way to think about it is to note that precisely at the threshold, increasing the resources devoted to error correction does not increase or decrease the reliability of the computer. This is a remarkable result! It gives hope that large scale quantum computation is feasible and provided an initial estimate of its experimental requirements. Threshold error rates give a benchmark for different error correcting codes and architectures, and shed light on which ones may be more promising candidates for implementation.

There are two main approaches to threshold calculations. The first is analytical, which usually results in lower bounds. These lower bounds are often pessimistic as this simplifies calculations. Alternative approaches utilize Monte Carlo simulations, which usually result in numbers that are often as much as an order of magnitude higher than their analytical counterparts. It is believed that actual thresholds are somewhere between the two.

Known numerical threshold results vary between $10^{-5}$ [49] for a restrictive 1-D architecture and $10^{-2}$ [29] for a case with long-range qubit-qubit interactions. Much work over the last few years has been concentrated on making this number as large as possible, as that implies that larger physical error rates can be tolerated. The actual value of the threshold is largely dependent on the architecture used as well as many of the assumptions that are made during calculations and modeling (for a list of assumptions made during our surface code threshold calculation, see Section 3.4).

For completeness, we reproduce a simple, yet very elegant proof of a threshold for concatenated codes. Concatenated codes are ones in which logical encoding is used in successive levels. For example, we might imagine using the Steane 7-qubit quantum error correcting code to encode a single qubit — this would require 7 physical qubits. We can imagine going further and encoding every one of those 7 qubits with 7 new physical qubits — and so on. Now, let us suppose that $p$ is the probability that any of the physical qubits in our computer can suffer an error. After one level of encoding, the probability of unrecoverable failure to the encoded state is reduced to $cp^2$ for some constant $c$. If we include another level of encoding, the probability of failure reduces further to $c(cp^2)^2 = c^3p^4$. We can generalize this, and note that after $k$ levels of encoding, the probability of failure of a logical qubit can be written as $\frac{1}{c}(cp)^{2^k}$. It is easy to see then, that the probability of failure will decrease with every new concatenation level, as long as $p < \frac{1}{c}$. This number $\frac{1}{c}$ is precisely the error threshold.

In Chapter 3 will explain the details of how the numerical threshold can be obtained for a surface code.

# Chapter 3

# Surface Code Threshold Calculation

## 3.1 Introduction

As discussed in Chapter 2, an error threshold is a useful measure of how well a particular error correcting code can perform. Our strategy when calculating such a threshold for a surface code, is to simulate random generation of errors on a lattice of qubits, while observing how long it takes for the encoded quantum state to be compromised. By repeating this procedure for different lattice sizes, and different physical error rates (denoted as $p$), we can generate a plot, which can let us estimate the actual threshold value. Before we move on, it is useful to point out other outcomes of similar calculations. Numerical results were presented in [45, 44, 54], where values of $\approx 7.5 \times 10^{-3}$ and $\approx 7.8 \times 10^{-3}$ were obtained. While the method of calculation differed from the one presented here, we nevertheless expect our results to be similar, since the underlying nature of the error correcting code used is the same. Another impressive result was given in [17], where an analytical lower bound of $\approx 1.7 \times 10^{-4}$ was derived.

## 3.2 Syndrome Readout Cycle

Although the explanation of a surface code (both toric and planar) in Section 2.3 is complete in a mathematical sense, we need to discuss in more detail the procedure, according to which the check operators $A_j$ and $B_j$ (defined in 2.6 and 2.7) are read out. Since in many systems directly measuring the four-term Z and X operators has not been shown to be possible, we use another approach where only one-qubit measurements are needed. We assume that an ancilla qubit is placed in the center of every plaquette, as well as at every vertex. This is schematically shown in Figure 3.1. These ancilla qubits are used for determining the measurement outcomes of the four-term (and three-term on boundaries) check operators without actually

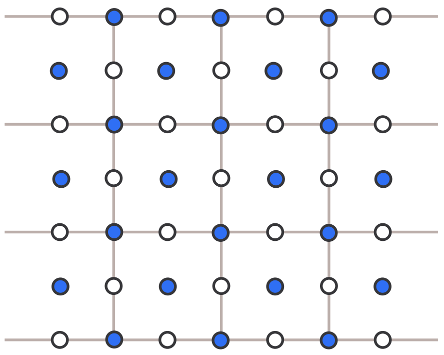


Figure 3.1: A planar surface code lattice with ancilla qubits shown in blue.

needing to measure them directly. We call these outcomes syndromes, and use them to determine where errors have occurred. The readout circuits, shown in Figure 3.2, explain how this is done. The approach consists of initializing the ancilla qubits, performing a collection of CNOT gates with neighboring data qubits, and finally reading out (measuring) the ancillas. Therefore the full readout cycle needs

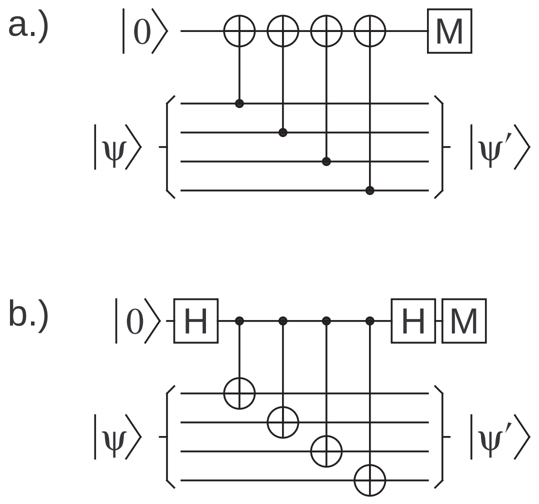


Figure 3.2: Circuit showing how an additional ancilla qubit (top line of each figure) is used to measure a) $ZZZZ$ around each plaquette and b) $XXXX$ around each vertex.

to include six simulation steps. It is worth noting that in a case of the readout of $XXXX$ around each vertex, the circuit needs to include Hadamard gates right after initialization and right before the measurement of the ancilla qubits[1], but in our simulations we simply combine these Hadamard gates with the neighbouring CNOTs and hence do not need to introduce any extra steps into the readout cycle. The orientation of the CNOT gates is different when dealing with ancilla qubits

[1]An equivalent way to think about it would be to say that we initialize the ancillas that live on vertices to the $|+\rangle$ state (and not $|0\rangle$), and read it out in the eigenstates of the $X$ operator, namely $\{|+\rangle, |-\rangle\}$.

that are located on the vertices, from the ones that sit on the plaquettes. In the former case, the ancilla qubits play the role of control, while the data qubits are targets. The situation is reversed in the latter scenario. An example of the plaquette readout is shown in Figure 3.3. It turns out that the actual order in which

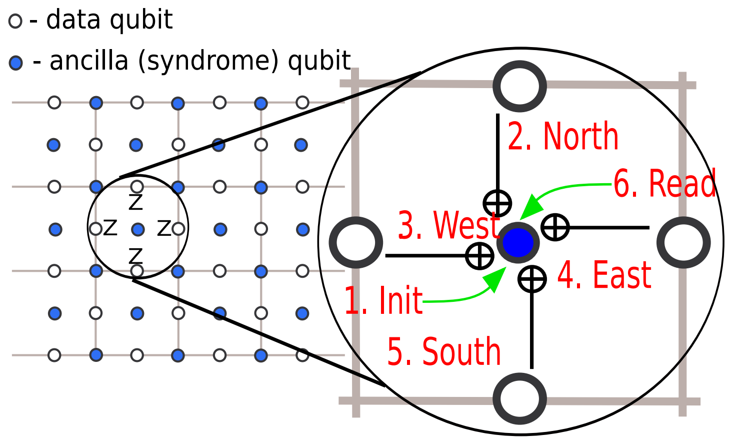


Figure 3.3: A syndrome measurement typically involves six steps; ancilla qubit initialization, CNOTs with the four surrounding data qubits (fewer on boundaries in a planar code case) and finally ancilla qubit readout. This example shows a temporal order of the CNOT gates of NWES.

the CNOT gates are applied needs to be chosen carefully. This is due to the fact that certain patterns may lead to entanglement between some of the ancilla qubits. Entanglement is not desired, because it leads to non-deterministic measurement results. An easy way to see this is to consider a scenario where both the lattice (not including ancilla qubits) starts off in a state that corresponds to +1 eigenvalues of all the four-term check operators, and all the ancilla qubits are also initialized to states $|0\rangle$, the +1 eigenstate of $Z$. If we further assume that no errors can occur, and perform the readout cycle, then at the end we expect to find all the ancilla qubits still in the +1 eigenstate, each agreeing with its respective check operator. However, if say two of the ancillas are entangled in the state $(|00\rangle + |11\rangle)/\sqrt{2}$ , which can be the case in certain patterns (occurs more than once), then measuring will not necessarily give +1 eigenvalue, and hence with only a probability of 1/2 will the result reflect the value of its corresponding four-term check operator. Figure 3.1 shows a few different patterns, and describes whether each of them is capable of ancilla qubit entanglement by the end of the readout cycle. The letters N, E, S, W, stand for North, East, South and West respectively.

To figure out whether a given pattern may lead to entanglement by the end of the readout cycle, we calculate the evolution of the full stabilizer of the lattice through the six-step readout cycle with different orders of CNOT gates, and simply observe whether at the end, the ancilla qubits are in product states or not. Manual calculations of this sort are very tedious and a software tool was written in order to simplify and automate this procedure. A more detailed discussion and

| CNOT readout pattern | Lattice type | Syndrome Entanglement |
|---|---|---|
| NWES | Toric | No |
| NWSE | Toric | Yes |
| NSEW | Toric | Yes |
| NWES | Planar | No |
| NWSE | Planar | Yes |
| NSEW | Planar | Yes |

Table 3.1: A table showing a few readout patterns and whether they lead to ancilla qubit entanglement at the end of the readout cycle. For a more detailed discussion and an explanation how these results were obtained see Appendix A.

actual calculations are presented in Appendix A. In our threshold simulations, we concentrate on the non-entangling readout order of NWES.

## 3.3 Error Model

The threshold error rate is derived from four different error rates in our simulations — initialization error $p_i$, readout error $p_r$, memory error $p_m$ and the error associated with a two-qubit gate $p_2$. As outlined before, the single qubit Hadamard gates associated with the readout cycle of vertex syndromes are combined with neighbouring CNOT gates. In order to be able to compare our result with other threshold calculations, we set all these error rates to the same value of $p$. We call this $p$ a physical error rate, and it is one of the inputs into our simulator. Below, we describe the four error types in some detail, keeping in mind that $\rho$ represents the density matrix of the lattice and indices $i$ and $j$ represent the qubits on which the error operator acts.

1. Memory errors
   These happen to qubits which sit idle during a particular step. For example, data qubits may suffer a memory error during ancilla qubit initialization and readout. Furthermore, in a planar code case, memory errors may be applied to boundary qubits which are not involved in a two-qubit gate. We can describe the transformation that the lattice undergoes from a memory error on the $i$th qubit as
   $$\rho \to \frac{p}{3}\left(X_i\rho X_i^\dagger + Y_i\rho Y_i^\dagger + Z_i\rho Z_i^\dagger\right) + (1-p)\rho \qquad (3.1)$$
2. Two-qubit gate errors
   These errors occur during two-qubit gates between data and ancilla qubits. The error is an application of one of the 15 tensor products of $I$, $X$, $Y$ and $Z$ (excluding $I \otimes I$). Assuming that the gate takes place between qubits $i$ and

$j$, we can describe it as:

$$\rho \rightarrow \frac{p}{15}\left(I_i X_j \rho X_j^\dagger I_i^\dagger + \cdots + Z_i Z_j \rho Z_j^\dagger Z_i^\dagger\right) + (1-p)\rho \tag{3.2}$$

3. Initialization errors
   By initialization, we mean initialization of the ancilla qubits to the state $|0\rangle$. An initialization error is therefore accidental preparation of state $|1\rangle$ with probability $p$. We can describe this mathematically with:

$$\rho \rightarrow pX_i \rho X_i^\dagger + (1-p)\rho \tag{3.3}$$

4. Readout errors
   By readout, we mean readout in the $Z$ basis. A readout error is a classical error — the qubit is projected into the $\pm 1$ eigenstate of $Z$, but with probability $p$ the eigenstate reported by the measurement device is incorrect.

Finally, it is worth stressing that in our simulations we apply the error operators (for initialization, readout, two-qubit gates) after first executing all operations perfectly — one could think of this approach as the errors being applied between simulation steps. Figure 3.4 shows an example of a possible error pattern that gets applied after a) the initialization step, and b) CNOTs North step. We see that in both cases, appropriate errors are applied to random qubits. An obvious difference between a toric code and a planar code is that in the toric code, during steps 2, 3, 4 and 5 (when two-qubit gates are being applied), no qubits ever sit idle, whereas in the planar code case both ancilla and data qubits can possibly suffer a memory error if they are located on a boundary (see Figure 3.4b for an example).

## 3.4 Assumptions

Our calculation makes certain assumptions that need to be clearly stated. We outline them below.

- Fast (instantaneous!) classical processing.
  This means that we are assuming that in a quantum computer controlled by classical electronics, classical processing is not taking any of the simulation steps.

- Uncorrelated errors
  We assume that errors which are generated at random, are not correlated in space or in time (the actual error model used is described in Section 3.3). It is important to stress, that here we are only talking about the process of random error generation, meaning that the probability of an error being generated at random at any given position on the lattice during any time step

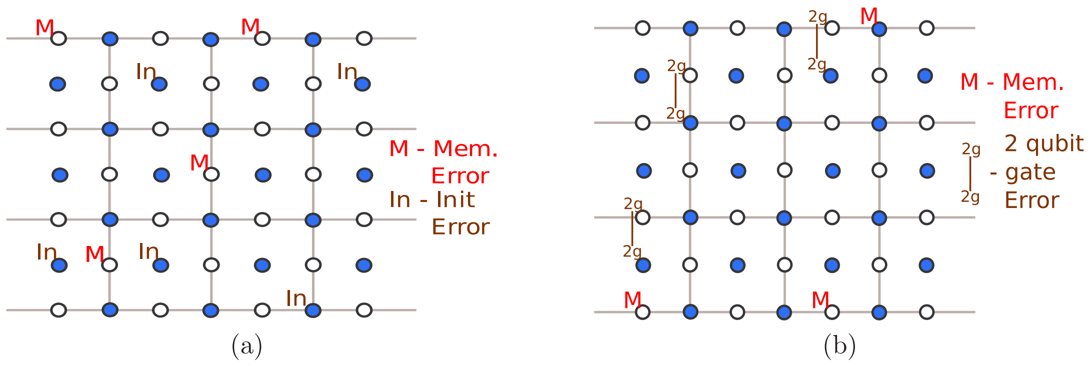


Figure 3.4: In our simulations, errors are applied to qubits at random. The type of errors depends on the current step of the readout cycle — in particular whether a given qubit has been sitting idle, has been involved in a two-qubit gate, or has undergone the process of initialization or readout. Above, we see error application a) after initialization and b) after CNOTs to the north of every ancilla qubit. In this particular step, ancilla (data) qubits in the top (bottom) row can suffer a memory error, as they are not involved in two-qubit gate operations.

is independent from an error being randomly generated on a different part of a lattice, during any other time step. Once an error has occurred on some part of the lattice however, depending on the time step and its location, it may get "copied" around to other qubits. This "copying" is properly accounted for by the simulation.

- Constant error rate
  This assumption implies that the physical probability of failure is independent of the number of the qubits in our computer.

- Quick measurement and initialization
  We assume that initialization and measurement of the ancilla qubits can be done as fast as any of the gates.

- Concurrent computation
  We assume that all the gates in different parts of the lattice can be done concurrently. Mathematically, this is not a problem, as all the operations done during any particular readout cycle step always commute. Here, we assume something different however — namely that the hardware required to do perform all the operations simultaneously would have to exist. This assumption is the basis of our approach where we are using six steps to perform a full syndrome readout cycle (explained in Section 3.2).

## 3.5 Method

We are now ready to describe the calculation procedure in more detail. We assume that the lattice starts off with some encoded state that was prepared beforehand, and in a +1 eigenstate of all the check operators. As was explained in Section 3.1, the simulation involves checking how long it takes for this encoded state to be compromised. In practice this means repeating the readout cycle discussed in Section 3.2 over and over, until a logical error can be observed. Figure 3.5 shows a schematic diagram of this procedure. The simulation software takes a physical gate

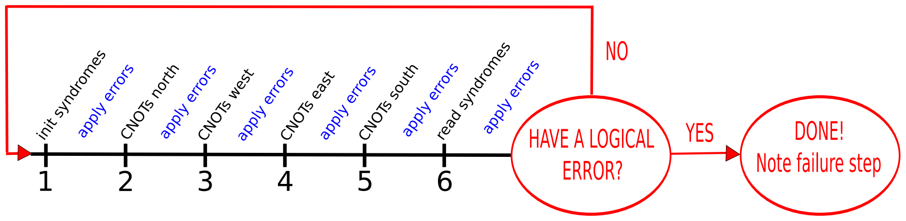


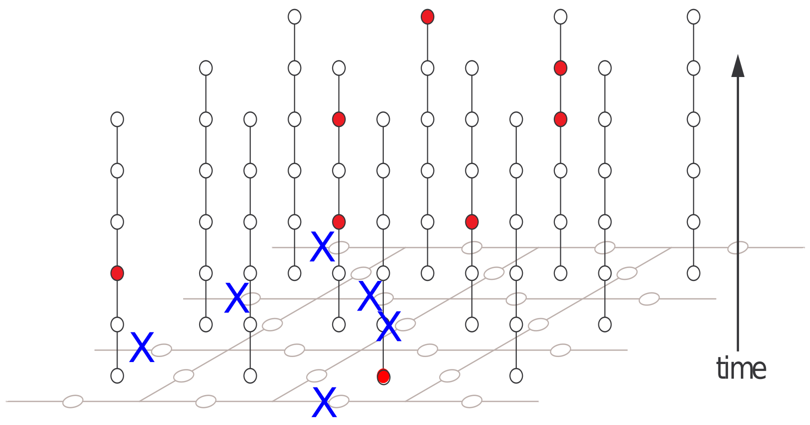
Figure 3.5: A schematic representation of the algorithm used in the threshold calculation. This particular case represents a scenario where the order of CNOT gates is NWES.

failure rate $p$ as well as a lattice size as input. We can also adjust whether a planar or a toric lattice should be simulated. Since $X$ and $Z$ errors can be treated independently, in order to decrease simulation times, we only concentrate on one type of error at a time (although in initial runs both error types were being accounted for).

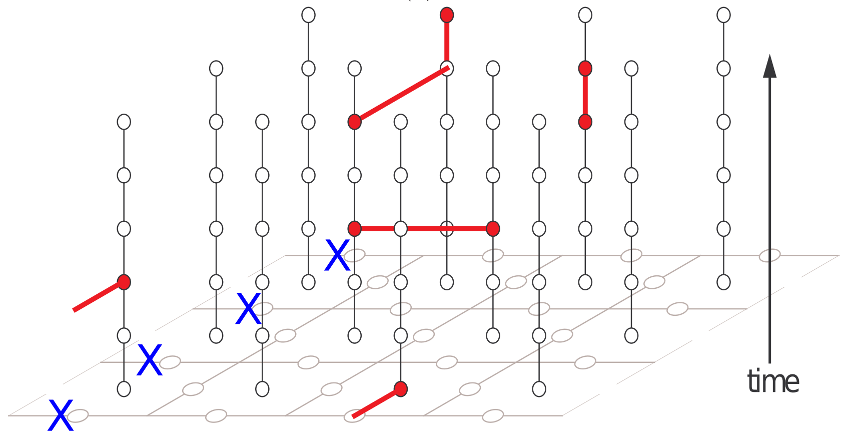
## 3.6 Measurement and Matching

In order to extract all the syndromes, the last step of the readout cycle — step 6, involves ancilla qubit readouts. After each ancilla is read, its value is checked against a result from the previous iteration, and if the values differ, the syndrome change location (in time and space) is recorded. Next, a matching of all the syndrome changes collected up to this point is used to guess where errors occurred. An example of this is shown in Figure 3.6a, where we can see that in this particular scenario, a collection of $X$ errors (shown as $X$s in blue) after six readout cycles, have lead to the given space-time locations of syndrome changes (red dots). We stress that one could get the same readout pattern from a different set of errors, hence the best we can do when guessing where the errors occurred is find a guess that is the most likely scenario.

To do this, we observe that shorter error chains are more likely than longer ones and therefore use a minimum-weight matching algorithm [14] to match the syndrome change locations and obtain a likely error pattern. Before the matching algorithm can find a minimum-weight solution, however, we need to convert our



(a)



(b)

Figure 3.6: a) An example of syndrome change locations (red dots) after six readout cycles. The $X$ operators represent the actual errors that the lattice suffered, which lead to the given syndrome change location pattern. These now have to be matched to obtain a guess as to where the errors happened. b) The matching of syndrome changes gives us information on which errors should be corrected.

matching results into something that the matching algorithm can understand. This is done by converting all the syndrome change results into a graph, with the locations of the syndrome changes representing the graph's nodes, and edges between these nodes having a weight which depends on the distance between them. The edge weight is measured in faces along the spatial dimensions and ancilla qubit readout cycles along the time dimension.

In the case of a planar code, some error chains may begin at the boundary and end somewhere inside the lattice (see Figure 3.7). In such cases, we can only observe the syndrome changes on the inner lattice. To account for this (meaning enable the matching algorithm to guess that the error chain started on a boundary), in a planar code case, for every inner node, we always create a closest boundary node. The edges between different boundary nodes are set to be of weight zero in order to minimize the cost of their removal by the minimum-weight perfect matching algorithm. This is not needed when we are dealing with a toric code, since toric codes have no boundaries (i.e. the lattice "wraps around").

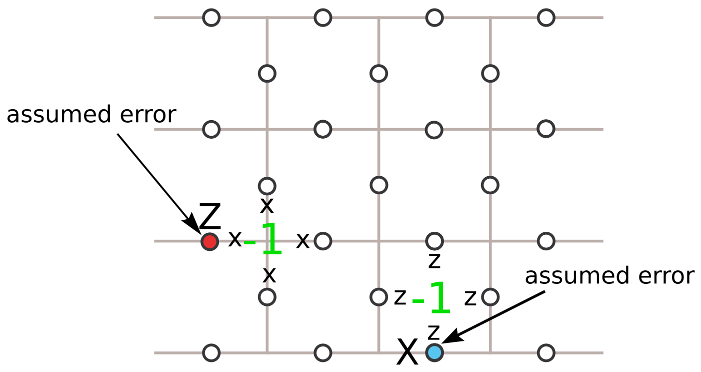


Figure 3.7: A planar surface code where we only observe a single syndrome change location (for a given error type) tells us that the error chain possibly originated at the boundary. The algorithm assumes the least damaging scenario — meaning the shortest path to the closest boundary.

One way to prepare our graph would be to include an edge between every pair of nodes (since in principle we don't know where actual errors occurred), but in practice we only do this for the toric code case, as it is not necessary for the planar surface code in which some edges will never be a part of a minimum-weight perfect match. The preparation of the graph takes this into account and only includes edges that connect nodes which are not further from each other than the sum of the weights between each node and its closest boundary node (since matching with the boundary gives a smaller total edge weight). An example that illustrates this is shown in Figure 3.8. We further optimize graph creation in the planar code case by noting that matches which are temporally far behind the current time step will not be changed by recent syndrome changes and therefore can be "remembered" from previous iterations. These techniques let us minimize the size of the graph that is passed to the matching algorithm which, despite scaling polynomially in the

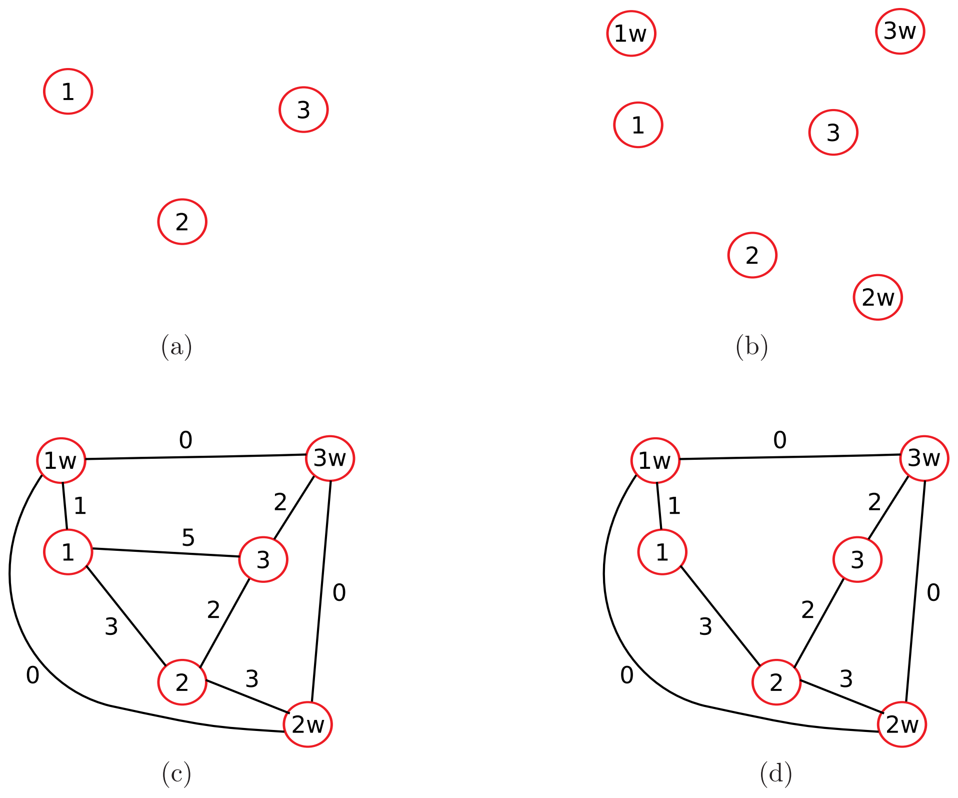


Figure 3.8: a) Let us suppose that after some number of simulations steps of a planar surface code, we have a collection of syndrome change locations labeled 1, 2 and 3 (which we can call “inner nodes”). These directly correspond to red dots in Figures 3.6a and 3.6b. b) The first thing we do when creating a graph which will be passed to the matching algorithm is to include a closest wall (boundary) node for every inner node. These are labeled 1w, 2w and 3w. c) The next step involves connecting the nodes using edges with weight that depends on the spatial and temporal distance between the nodes. One way to do this would be to put an edge between every pair of inner nodes, as well as between every node and its closest wall node, and finally to connect all the wall nodes together with edges of weight zero. d) A more efficient way of handling a planar code scenario is to note that only edges that connect inner nodes which are not further from each other than the sum of weights of edges between themselves and their respective boundary nodes can contribute to the minimum-weight perfect matching. In this example we can then remove the direct edge between nodes 1 and 3 as it is not needed.

number of edges and nodes, can often still take substantial computing time. An example of a successful minimum-weight perfect match is shown in Figure 3.6b. In it we see that the correction did eliminate most of the errors — but not all. The matching algorithm guessed that the vertical line of 3 $X$s was less likely than the $X$ in the bottom left corner, thus in fact introducing a logical error, which leads us to the next important aspect of the simulation — logical error detection.

## 3.7 Logical Error Detection

As outlined in Section 3.5, in order to know if the simulation should continue or not, we need to determine whether the lattice suffered a logical error (and hence the encoded state has changed). Logical errors are chains of operators ($X$ or $Z$) that span across the whole lattice. In the case of a planar code, this means chains which connect boundaries of the same type — either smooth or rough. In a case of a toric code, the chain needs to "wrap around" the surface. Detecting these chains is not entirely trivial. One approach would be to start at a boundary (in a planar case, or simply chose an artificial boundary in a toric case) and try to find paths of error operators that would lead to the other boundary (or cross the artificially chosen one in the toric case). However, an even number of paths of operators that cross the lattice are equivalent to no errors at all (i.e. $XX = ZZ = \mathbb{1}$), and therefore should not be considered as logical errors. Furthermore, one has to consider that some paths may be a part of closed loops and as we outlined in Section 2.3, closed loops are within the stabilizer of the code, and do not contribute to logical errors. Hence any direct path detection algorithm would have to take these things into account. Figure 3.9 shows a specific example of a case where multiple paths of $Z$ operators exist between two boundaries, and where the logical state has not been compromised. Furthermore, all the errors can be "decomposed" into loops which are within the stabilizer of the code. Therefore, in order to simplify logical error

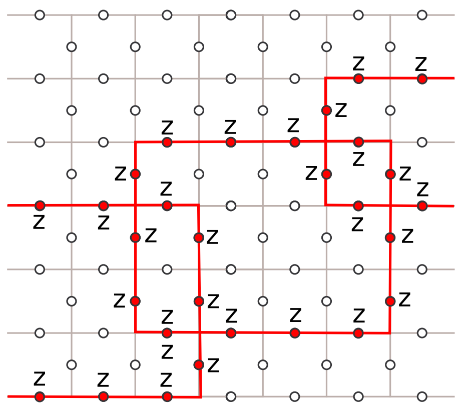


Figure 3.9: Z errors that form an even number of chains across the lattice do not form logical operators. Furthermore, the loops of errors presented in the diagram are within the stabilizer of the code.

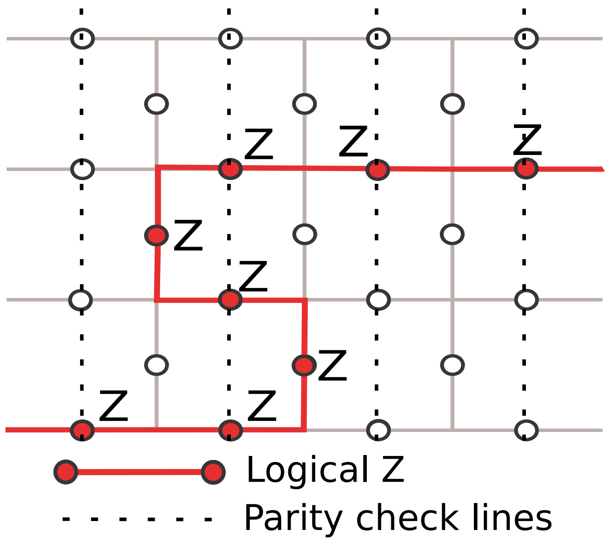
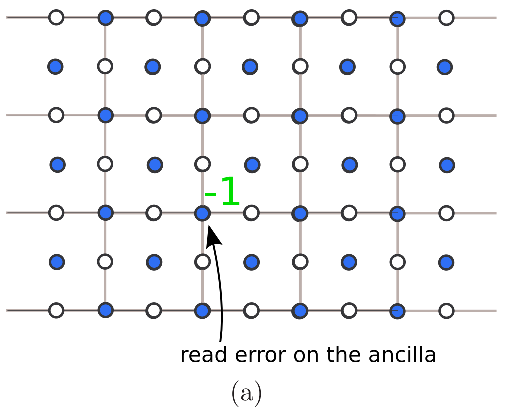


Figure 3.10: A logical Z(X) error detection involves checking if the parity of Z(X) operators along any of the vertical(horizontal) lines of qubits is odd. Above we show an example of a logical Z error.

detection procedure, we approach the problem form a different angle. To detect if a logical error has occurred, we repeat the readout cycle with all the error sources set to zero (i.e. set $p_i = p_r = p_m = p_2 = 0$). This allows us to be certain that any logical error can be recognized by solely checking the parity of operators crossing a line of qubits which is perpendicular to the direction of the chain needed to produce a logical error. Figure 3.10 shows this in more detail in a case of a logical $Z$ error on a planar lattice. A chain of $Z$ operators spans from the left boundary to the right. The number of times it passes through any of the dashed lines is always odd and whether this is the case or not can be easily detected by the simulation software. It is worth noting that without such a "perfect readout" simply checking the operator parity crossing a line of qubits in the way outlined above would not work. To see this we can consider a simple example of having a single readout error on one of the syndrome qubits. This error would lead to a syndrome change, and if it was sufficiently isolated from others, it would be matched to the boundary (in a planar code case), which, when corrected for, would lead to a single chain of errors on qubits between the ancilla where the syndrome was obtained and the closest boundary. Hence measuring the parity along a path that would cut this chain, would yield an odd operator number leading to a wrong conclusion that a logical error has occurred. This example is shown graphically in Figures 3.11a and 3.11b. In a case of a toric code a readout error presented above could lead to an odd number of syndrome readout nodes, which the matching algorithm would not be able to handle.

Now that we've explained the logical error detection in some detail, we can summarize just how it fits into the whole picture. At the end of every syndrome extraction cycle the current state of the simulation is recorded (all relevant data



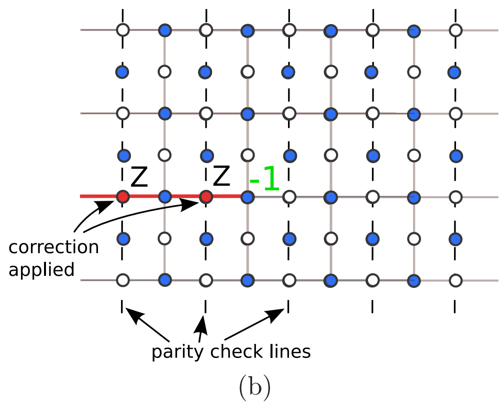


Figure 3.11: A simple example which shows that the detection of logical errors by simply checking the parity of operators along the parity check lines does not work without a final error-free syndrome extraction cycle. a) We suppose that there have not been any errors on the lattice, except for a readout error on one of the ancilla qubits, resulting in the syndrome value of $-1$. b) The matching algorithm guesses that the most likely errors that could cause this situation occurred on the path between the ancilla qubit and its closest boundary. By "correcting" we introduce an unwanted chain of $Z$ operators. Finally, measuring the parity of these $Z$ operators crossing the two parity check lines closest to the boundary, gives an odd result, which in turn is wrongly interpreted as a logical error. If the final error-free readout cycle was used, the original readout error would be detected precisely as a readout error, and no matching to the boundary would occur.

structures, variables, etc. copied). Another full readout cycle is then performed with all failure rates set to zero. The last step of this readout cycle involves ancilla qubit measurements, and hence if any syndrome changes are detected they are recorded. Next, the full set of syndrome changes, those from before as well as from this latest readout cycle are used for minimum-weight perfect matching. The resulting guess is then used for applying a correction. Finally, the corrected lattice is then passed onto error detection routines which can determine whether a logical error has occurred. If it has, then the simulation is stopped and the previous cycle step (at which the simulation was "frozen") recorded. If no logical error has been detected, the simulation is reverted to the state just before the "perfect readout" cycle began and continues on.

## 3.8 Results

As was described above, in order to obtain a value for the threshold, one needs to calculate time-until-failure for different size lattices as well as different physical error rates $p$. The time-until-failure may differ from run to run, even when the same lattice size and physical error rate is used, and therefore an average over many (we use at least 5000) repeated simulation runs is needed. The crossing point of all the curves gives the threshold. We can easily see why that is the case by remembering that by definition, a threshold is an error rate at which increasing the resources devoted to error correction does not increase or decrease the reliability of the computer. The resources are the qubits (expressed here in terms of the lattice size), and the reliability can be described in terms of the number of readout cycles before an encoded state suffers a logical error. In the case of the toric code, we look at lattice sizes between 3 and 13 faces across (or codes with distances between 3 and 13). Although it would be beneficial to look at larger sizes, we are limited by the matching algorithm and the fact that optimization techniques used when dealing with the planar code (more about it in Section 3.6) cannot be applied to the toric code case. The resulting plot is presented in Figure 3.12. In it we see that except for the two curves associated with the smallest lattice sizes, the crossing is observed at $\approx 6.0 \times 10^{-3}$. The situation is similar in the planar code case shown in Figure 3.13. Here however, using the optimization techniques described in Section 3.6, we are able to look at larger lattice sizes — between 4 and 20 faces across (corresponding to the code distance between 5 and 21). The crossing is now substantially more noisy, and only visible for much larger larger lattice sizes (than in the toric code). This is likely due to the fact that the boundary check operators have three terms as opposed to four, which is the case for the ones on the inside of the lattice (for more on this, see Section 2.3, and in particular Figure 2.2). The ratio of the three-term to total check operators (of a particular kind — either composed of $X$s or $Z$s) scales as $2/(n+1)$, where $n$ is the number of faces across the square lattice (or alternatively as $2/d$ with d being the code distance). For smaller lattices, where this number is large, changing the lattice size has an effect on the average probability of an

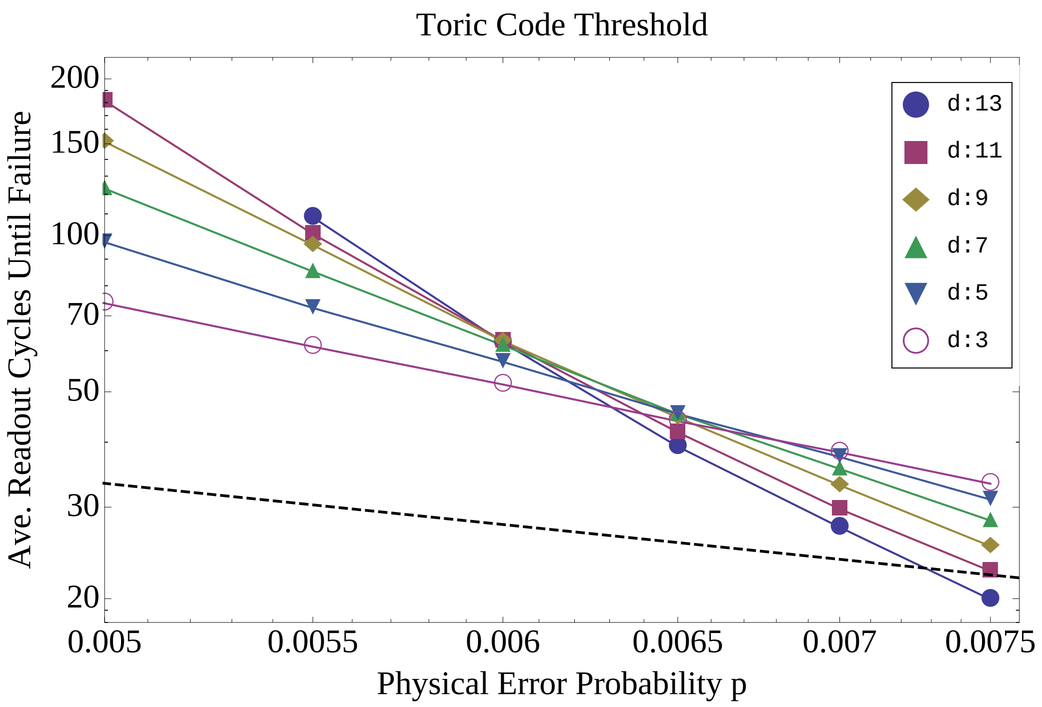


Figure 3.12: A log-log plot of average time-until-failure versus the physical error rate $p$ on a toric code. We study lattice sizes ranging from 3 to 13 faces across (code distance from 3 to 13). Except for cases of the two smallest lattices, a crossing is observed at $\approx 6.0 \times 10^{-3}$. A case where no error correction is used (single qubit) is represented with a dashed line.

ancilla qubit reporting errors in a single readout cycle (whereas in a toric code for example, this stays constant as the lattice size changes). From Figure 3.13 we can observe that the crossing is slowly converging to a value slightly above $6.0 \times 10^{-3}$ in particular for lattice sizes larger than 12 faces across (distance 13). We can suspect that if we were able to look at even larger lattice sizes, the crossing would likely agree with that of the toric code case. We therefore estimate our threshold value to be $6.0 \times 10^{-3}$.

These results let us conclude that below the threshold, the average lifetime of the encoded qubit can be increased arbitrarily by increasing the size of the lattice. Furthermore, the numbers we obtain are in general agreement with those presented by others [45, 44, 54], which are only slightly higher at $7.5 \times 10^{-3}$ and $7.8 \times 10^{-3}$. The reason for this small discrepancy likely has to do with the differences in the details of the simulations. To further solidify the values presented above, it would be beneficial to observe the behaviour of the system with larger lattice sizes. In order to make this possible one approach might involve using an approximation to the minimum-weight perfect matching algorithm as this is currently the biggest

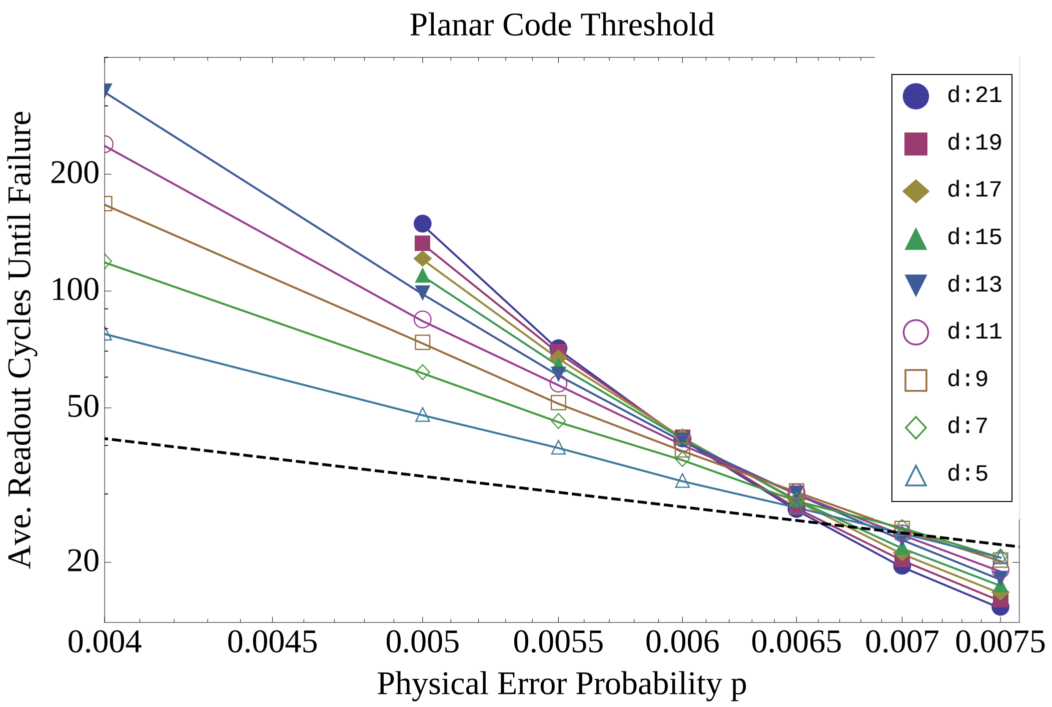


Figure 3.13: A log-log plot of average time-until-failure versus the physical error rate $p$ on a planar code. We study lattice sizes ranging from 4 to 20 faces across (code distance between 5 and 21). For lattice sizes larger than 12 faces across (corresponding to distance 13), a crossing is converging to value just above $\approx 6.0 \times 10^{-3}$. A case where no error correction is used (single qubit) is represented with a dashed line.

bottleneck in our calculations. The drawback could be that the threshold estimate calculated in such a way, might be lower than what is presented here, but depending on the approximations used, it might not be by much. In summary, we state that it is encouraging that both our results and those presented elsewhere for a surface code threshold, are higher than error thresholds of many other approaches to error correction (in particular, ones that assume only nearest neighbour interaction).

## 3.9 Other Possible Approaches To The Threshold Calculation

The approach to the surface code threshold calculation presented above requires large computational resources. The matching algorithm is invoked after every readout cycle. As is observed in Section 3.8, in the cases of larger lattices and lower physical rates, logical errors are not detected until often as many as couple of hundred of iterations. Furthermore, in the cases of larger lattice sizes, even with optimizations associated with preparation of the graphs which represent syndrome changes (more about this in Section 3.6), the matching takes substantial time. Expecting this, our original approach to the threshold calculation differed. In order to minimize the computing time, it was envisioned that instead of attempting to determine whether a logical error occurred after every readout cycle, we could run the simulation for some time (say a few hundred cycles), record all the errors (in space and time) along with syndrome change locations, and from this information try to determine how many logical errors the lattice suffered over the given period of time. The leap of faith that one had to make to justify this approach was that the density of logical errors would be constant as the simulation progressed, which is in tune with the general assumptions shown in Section 3.4. At the same time, we were also experimenting with the alternative logical error detection method where we explicitly searched for boundary-to-boundary paths of error operators (more on this method in Section 3.7). An artistic rendering of this is shown in Figure 3.14.

In the end, the results were not what was expected — in particular no crossing of curves representing different lattice sizes was observed. Since we were provided with access to Sharcnet (a high-performance computing cluster), in the end we switched our approach to the "check for error after every step" method described in the other parts of this chapter.

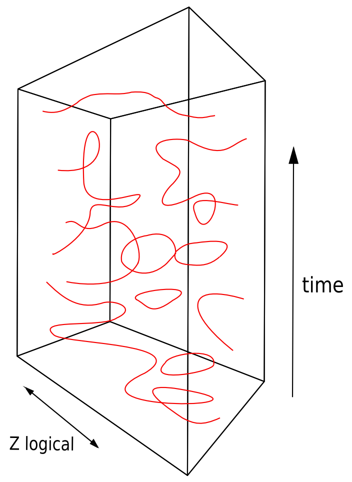


Figure 3.14: Error chains crossing a planar lattice structure. The vertical axis is time, and the horizontal axis represent the spatial dimensions of the lattice. Logical errors are chains that span the lattice, starting at one boundary and ending at the other. In order to detect logical errors in the lattice, such paths had to be found, keeping in mind that closed loops are within the code stabilizer, hence don't actually contribute to logical errors.

## 3.10 A Few Notes On Implementation

Our simulation software was written in either C (the original version, see Section 3.9 for more details) or C++ (the final version which was ultimately used to produce results presented here). Naturally there are multiple ways to implement a simulation such as this. One way is to simply simulate the stabilizer evolution of the lattice, accounting for ancilla qubit initialization, CNOTs between ancilla and data qubits, ancilla measurements, and of course appropriate error operators that would be randomly generated. A well established method for this might be to use a check matrix [37] where a $g \times 2n$ matrix can be used to represent a stabilizer with $g$ generators, and $n$ qubits. 1s in a position $(i, j)$ with $1 \leq j \leq n$ denote a $Z_j$ operator in the $i$th generator, and 1s in a position $(i, j)$ with $n + 1 \leq j \leq 2n$ represent a $X_{j-n}$ operator in the generator with index $i$. Evolution of the stabilizer involves updating such a check matrix according to the rules shown in Table 2.1. An efficient approach to do this, which is capable of dealing with a rather arbitrary gate order, and even with available software was presented in [1]. In our case, the situation can be simplified, because we are not dealing with a completely unpredictable stabilizer evolution. When one does not consider any errors, the order of the CNOT gates and hence the evolution of the stabilizer is the same in every cycle. Furthermore, by noting that errors only change the signs of certain generators (which ones, depends on where and when they occurred) and not the structure, we can calculate what each generator will look like after each step of the readout cycle (up to a possible sign difference, which depends on the errors). By then figuring out what generators are affected by an error happening on a given qubit during any of the six cycle steps and storing this information in some static data structure, we have a very efficient way to run the simulation.

Our first version of the code worked in precisely the way described above, however in the later incarnations we modified the way we kept track of errors. Instead of dealing with the stabilizer, we imagined a lattice with qubits which was implemented as a 2D array. Each entry of the array represented a count of the number of errors (of a particular type) which acted on a qubit with the same lattice index. Since two $X$s (or $Z$s or $Y$s) are equivalent to $I$, this number could be either a 1 or a 0. As the simulation progressed, the CNOT gates "copied" the errors around. The readout involved checking whether each syndrome is a 0 or a 1 corresponding to the $+1$ and $-1$ eigenvalues respectively.

The two approaches presented above are equivalent in the sense that both can be used to describe the error propagation throughout the lattice, and in turn provide enough information to ultimately determine a numerical error threshold when a non-entangling readout cycle is used (see Section 3.2 for more information on this). We ended up sticking with the latter, simply because it was more "geometrical" and hence easier to deal with.

In order to perform minimum-weight perfect matching, our software was interfaced to a matching code written by Cook and Rohe [14], which is freely available on the internet. The code was originally written in C, but a C++ wrapper was provided to us by K. Goyal, which we slightly modified to simplify its integration into our project. It turned out that the matching process was computationally by far most demanding. The optimizations in syndrome change graph preparation in the planar code case, explained in Section 3.6 gradually let us go to lattices with over 40 qubits across (counting ancilla qubits), but exploring large lattice sizes without choosing to approximate the process of perfect matching would not be computationally practical.

# Chapter 4

# Quantum Computing With Superconducting Circuits

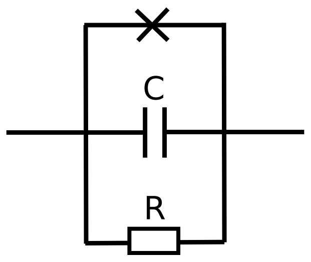
## 4.1 Preliminaries

Superconductivity was first discovered in 1911 by a Dutch physicist named Heike Kamerlingh Onnes [39]. He noticed that when when mercury is cooled below 4.2K, its resistivity to current disappears. Over the last century, many scientists have tried to explain this fascinating phenomenon. An important breakthrough was made in 1957, when a microscopic (BCS) theory was proposed by Bardeen, Cooper, and Schrieffer [4]. In this theory it is established that atomic level vibrations force electrons to "team up" into pairs, which in turn enables them to flow freely through the superconducting material. The BCS theory has been very successful in explaining low temperature superconductivity and led to many advances in the field.

In the last decade, physicists realized how to build qubits using superconducting circuits. Many different approaches have been introduced [10, 12, 18, 34, 36, 41, 53, 56, 57]. Their differences come from the circuit layouts and components used. In this chapter we explain the workings of a flux qubit, as well as a DC-SQUID. Both of these devices play an important role in proposals that envision large scale quantum computing with superconducting circuits.

## 4.2 Josephson Junctions

Before we move on to a more detailed discussion of quantum computing applications, let us review the Josephson effect, as it will play a central role in what follows. The Josephson effect was named after British physicist Brian David Josephson, who in 1962 predicted the phenomenon of current flow through a junction created by separating two superconductors with a thin insulating layer [26]. Such a junction



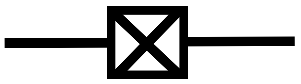


Figure 4.1: a) A Josephson junction as described in the RCSJ model, by a resistor R, capacitor C and the Josephson element (represented by a cross). b) A simpler representation that does not explicitly show the resistor and capacitor.

can be schematically represented as a resistor, capacitor and what's called a Josephson element in parallel (this is often called the RCSJ model [50]), an example of which is shown in Figure 4.1. Josephson discovered two equations that govern the behavior of these junctions. The first is

$$I_s = I_0 \sin\phi \tag{4.1}$$

with $\phi$ being the difference in the phase of wavefunctions in the superconductor on either side of the junction and $I_0 = I_0(T, \Delta)$ the critical current — the maximum supercurrent that can flow without dissipation. In general it depends on the temperature of the sample as well as the properties of the superconductor (many of which govern the size of the energy gap $\Delta$) [3]. The other equation is

$$\frac{d\phi}{dt} = \frac{2eV}{\hbar} \tag{4.2}$$

and it tells us that a varying phase difference across the junction implies non zero voltage. The two equations lead to an interesting conclusion that the supercurrent can flow even if the potential across the junction is zero. We can go a little further, and use Equations 4.1 and 4.2 to derive the free energy stored in a junction by integrating the work done by the current source that is required to change the phase. This leads to

$$E = \int I_S V dt = \int I_0 \sin(\phi')\frac{\hbar}{2e}d\phi' = constant - E_J\cos(\phi) \tag{4.3}$$

with $E_J = \frac{\hbar I_0}{2e}$ called the Josephson energy. When describing Josephson tunnel junctions, one also needs to consider the capacitance associated with their geometry, and therefore capacitative single-electron charging energy, defined as $E_c = \frac{e^2}{2C}$.

One can also define a Josephson inductance as

$$L_J = \frac{\Phi_0}{2\pi I_s \cos\phi}. \tag{4.4}$$

with $\Phi_0$ being the "flux quantum" and expressed in terms of other fundamental constants as $\Phi_0 = \frac{h}{2e}$. The fact that it is not linear, plays a critical role in quantum computing as it leads to non-uniform differences in energy levels, which in turn makes addressing specific transitions possible. This addressability, along with the fact that under some critical temperature and voltage these junctions are non-dissipative, makes Josephson junctions extremely important in building quantum computing devices with superconducting circuits.

## 4.3 Flux Qubits

Flux qubits can be constructed from a loop of superconducting wire, and three Josephson junctions [35], shown schematically in Figure 4.2. We assume that the qubit is operated in the non-dissipative regime, hence the resistance branch of each junction is not shown in the picture. We define $\phi_i$ to represent the Cooper pair wavefunction phase difference across junction $i$, and $E_{J,i}$ the $i$th junction's Josephson's energy. In this design, two of those junctions are of the same size,

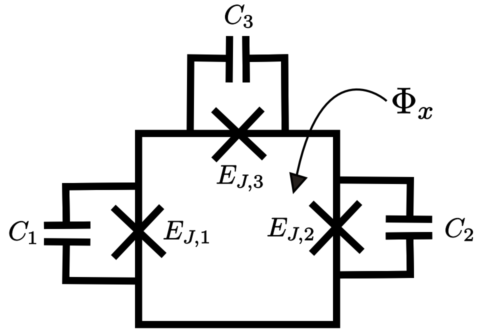


Figure 4.2: A three junction flux qubit, often called a "persistent current qubit".

and the third is fabricated to be a factor $\frac{1}{\alpha}$ smaller. This leads to $E_{J,3}/E_{J,1} = E_{J,3}/E_{J,2} = \alpha$, with $\alpha$ usually being between 0.6 and 0.8. We assume that an applied flux $\Phi_x$ can be threaded through the loop, and easily controlled by the experimental apparatus. Furthermore, by taking the loop to be small, we can assume that its geometric inductance $L$ is small, and hence observe that the total flux $\Phi$ passing through the loop is roughly the same as the applied flux $\Phi_x$, namely $\Phi \approx \Phi_x$. Using fluxoid quantization [50, 16], we can then relate the applied flux with the phases $\phi_i$ by

$$\phi_1 + \phi_2 + \phi_3 + \frac{2\pi\Phi_x}{\Phi_0} = 2\pi m \tag{4.5}$$

for $m$ integer. This lets us get rid of $\phi_3$ in the Hamiltonian by writing it in terms of the flux and other junction phases. All this leads to

$$H = \sum_{i=1}^{3} \frac{Q_i^2}{2C_{J,i}} - E_J \left( \cos\phi_1 + \cos\phi_2 + \alpha \cos\left( \frac{2\pi\Phi_x}{\Phi_0} - \phi_1 - \phi_2 \right) \right). \tag{4.6}$$

The second term in Equation 4.6 describes the potential energy landscape, which when plotted along $\phi_1 = \phi_2 = \phi$ leads to a double well potential. The minima of the wells can be changed relative to each other by changing the applied flux through the qubit. This is shown in Figure 4.3. The two wells correspond to

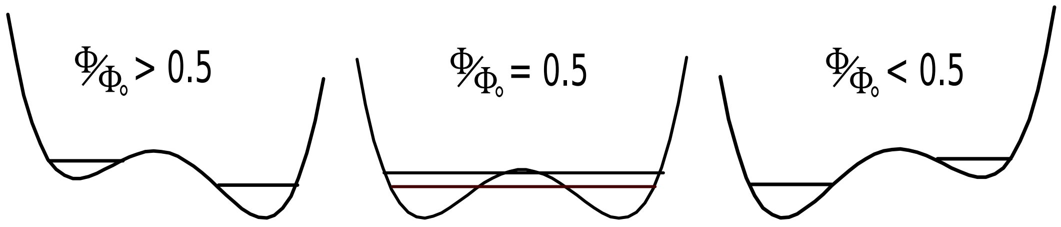


Figure 4.3: Potential energy of a flux qubit. We observe that the double-well shape of the potential energy can be shifted by the applied flux threaded through the qubit.

a circulating, persistent currents in the clockwise or counter-clockwise directions. Mathematically, they can be described in terms of the parameter $\alpha$ as

$$I_p = I_0\sqrt{1 - \frac{1}{4\alpha^2}}. \tag{4.7}$$

where $I_0$ is the critical current of junctions 1 and 2. They represent two basis states of the qubit, usually denoted as $|0\rangle$ and $|1\rangle$. When the applied flux $\Phi_x$ is equal to $\frac{\Phi_0}{2}$, the lowest energy states are symmetric and anti-symmetric superpositions of $|0\rangle$ and $|1\rangle$, and as shown in Figure 4.4, an energy anti-crossing is observed. In the

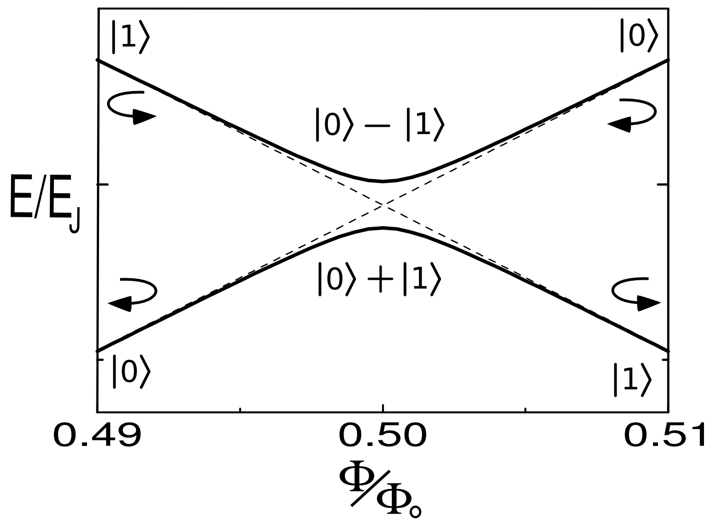


Figure 4.4: An energy diagram of a flux qubit. We observe an anti crossing where the flux is equal to $\Phi_x = \frac{\Phi_0}{2}$. States $|0\rangle$ and $|1\rangle$ are represented by clockwise and counterclockwise directions of the circulating currents.

two state approximation, the Hamiltonian shown in Equation 4.6 can be further rewritten in the form presented in Chapter 1.4 as

$$H_i = -\frac{1}{2}(\epsilon\sigma_z + \Delta\sigma_x) \tag{4.8}$$

with $\epsilon = 2I_p\left(\Phi_x - \frac{\Phi_0}{2}\right)$ and $\Delta$ often called the tunneling energy. The parameter $\Delta$ is dependent on the physical properties of the circuit (in particular on the Josephson junctions used) and in most implementations it is fixed at fabrication time. However, approaches where $\Delta$ is adjustable have been demonstrated both theoretically [35] and experimentally [42] by simply replacing the third junction by a DC-SQUID (reviewed in detail in the next section).

Finally, for completeness, it is worth stressing that the original proposal for a flux qubit only included one Josephson junction [8]. It had similar properties to the approach presented above, but in order to obtain a double-well potential the geometrical inductance needed to be large. This meant that the physical size of the qubit was large, which lead to substantial interaction with the environment, rendering the qubit very sensitive to decoherence and experimentally difficult to use.

## 4.4 DC-SQUIDs

A DC-SQUID consists of a loop of superconducting material, which is interrupted by two Josephson junctions, as is shown in Figure 4.5. For simplicity, we'll assume

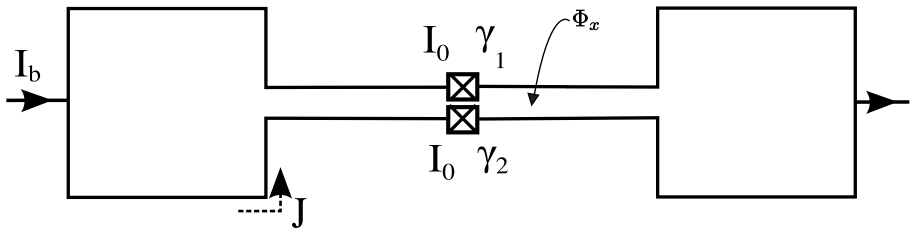


Figure 4.5: A DC-SQUID is a superconducting loop, interrupted by two Josephson junctions. $\gamma_1$ and $\gamma_2$ are the differences in phase of the wavefunction of Cooper pairs on either side of junctions 1 and 2 respectively.

that both junctions have the same parameters, and their critical current is described by $I_0$ and capacitance by $C$. $I_b$ represents the bias current sent through the loop. Using Kirchhoff's current laws, we can write

$$\frac{I_b}{2} - J = I_0 \sin\gamma_1 + \frac{C\Phi_0}{2\pi}\ddot{\gamma}_1 \tag{4.9}$$

and

$$\frac{I_b}{2} + J = I_0 \sin\gamma_2 + \frac{C\Phi_0}{2\pi}\ddot{\gamma}_2 \tag{4.10}$$

where $J_n$ is the circulating current of the DC-SQUID. By adding and subtracting these, using $\gamma_- = \frac{\gamma_1 - \gamma_2}{2}$, $\gamma_+ = \frac{\gamma_1 + \gamma_2}{2}$ and a couple of trigonometric identities, we can rewrite Equations 4.9 and 4.10 as

$$I_b = 2I_0 \sin\gamma_+ \cos\gamma_- + \frac{2C\Phi_0}{2\pi}\ddot{\gamma}_+ \tag{4.11}$$

$$J = I_0 \sin\gamma_- \cos\gamma_+ + \frac{C\Phi_0}{2\pi}\ddot{\gamma}_-. \tag{4.12}$$

Next, using fluxoid quantization around the loop we have a relation between the $\gamma$s, the applied flux $\Phi_x$ and the circulating current $J$

$$\gamma_1 - \gamma_2 + \frac{2\pi\Phi_x}{\Phi_0} - LJ = 0 \tag{4.13}$$

where L is the geometric inductance of the DC-SQUID. At this stage we can simplify things a little and assume that L is small, and $LI_0 \ll \frac{\Phi_0}{2}$. In this case, the Josephson inductance dominates the geometrical inductance, and the last term in Equation 4.13 can be neglected (Note that Chapter 5 does not neglect the geometric inductance, and full numerical treatment is presented). This lets us write the Hamiltonian [9] as simply

$$H = \frac{\hbar^2}{4E_C}\dot{\gamma}_+^2 - 2E_J \cos\left(\frac{\pi\Phi_x}{\Phi_0}\right)\cos\gamma_+ - \frac{\hbar}{2e}I_b\gamma_+. \tag{4.14}$$

We see that a DC-SQUID acts very much like a single Josephson junction with the important difference that its critical current can be controlled by the applied flux $\Phi_x$. In the limit of small $L$, we can express the critical current $I_c$ of the DC-SQUID as

$$I_0(\Phi_x) = 2I_0 \cos\left(\frac{\pi\Phi_x}{\Phi_0}\right). \tag{4.15}$$

DC-SQUIDs are turning out to play important roles in applications of superconducting circuits to quantum computing. For example, they can be used as readout devices for flux qubits [40, 52, 32]. One readout method involves injecting a dc-current and observing at what point the device switches into a non-zero voltage state. Since the critical current is dependent on the flux, which in turn can be affected by the nearby flux qubit, the applied current that is needed for the DC-SQUID to switch will differ depending on the actual state of the qubit. Hence, by first calibrating the DC-SQUID appropriately, one can determine the state of the qubit. Another very useful application of DC-SQUIDs in quantum computing applications is coupling. Since this topic is very important to what follows, we'll review it in more detail in the next section.

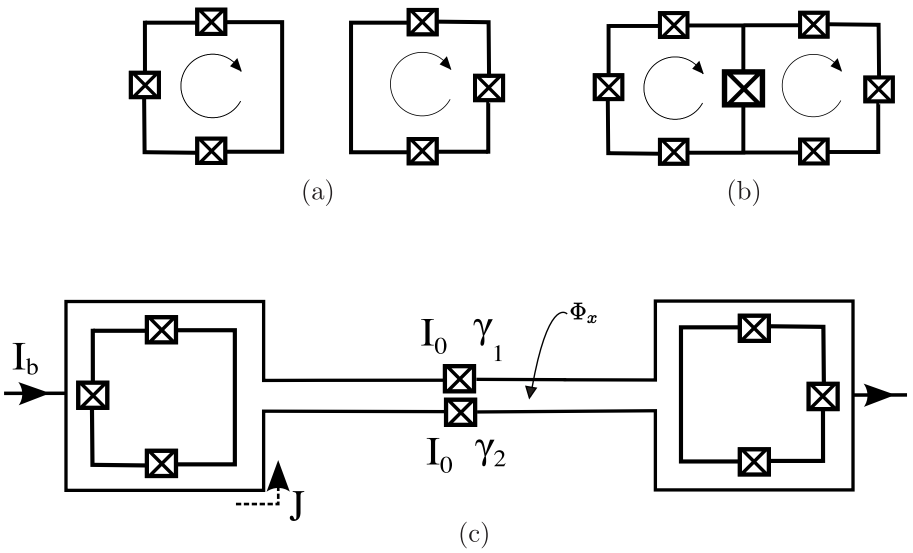


Figure 4.6: Different approaches to coupling flux qubits. a) Direct coupling via mutual inductance. b) Coupling via a shared Josephson junction, which is typically larger than junctions used in the qubits. c) A DC-SQUID used as a coupler. By applying an appropriate bias current $I_b$ one can completely turn off any qubit-qubit interactions.

## 4.5 Coupling Flux Qubits

Many different approaches have been proposed to couple multiple qubits together. In the simplest case, one can consider only direct qubit-qubit coupling via mutual inductance, as shown in Figure 4.6a. In more complicated cases, qubits can share an edge with or without extra junctions (Figure 4.6b), or even couple via another high-excitation energy object such as a DC-SQUID (Figure 4.6c) [43] or even another qubit [38]. Recently proposals of coupling qubits to a cavity also have been demonstrated [33, 6], but we will not further discuss those here. An important aspect of building a useful quantum computer is making multi-qubit coupling controllable, hence proposals that provide some way of switching off, or at least being able to, at some level, control the coupling strength are preferred.

In this section we'll concentrate on briefly reviewing an approach presented in [43] as it forms the starting point of the three qubit, two coupler work from Chapter 5. In this scenario, two flux qubits are coupled via both direct inductive coupling, as well as via a third object — a DC-SQUID. The circuit representation is shown in Figure 4.6c. The energy between the ground and first excited state of the coupler is chosen to be large when compared to the energies of the qubits, and therefore one can assume that the coupler always stays in its ground state. We will study the details of an extended version of this circuit in Chapter 5, but it is worth stressing a few important results. The total coupling energy $K$ between the two qubits is due to both direct coupling $K_0$ as well as coupling through the DC-SQUID, denoted $K_s$. The direct coupling is due to the mutual inductance $M$

between both of the qubits and takes the form of $MI_1I_2$, where $I_i$ is the circulating current in the qubit $i$. In the next chapter we will show that $K_s \propto \frac{\partial J}{\partial \Phi_x}$, with $J$ representing the circulating current in the coupler, and $\Phi_x$ the applied flux. The authors of [43], have shown that for certain values of the bias current $I_b$ and applied flux $\Phi_x$, this derivative is negative, and therefore the direct coupling between qubits can be exactly negated by the interaction through the DC-SQUID. A plot of the coupling strength for experimentally achievable parameters is presented in Figure 4.7. It shows that at $\Phi_x = 0.45\Phi_0$ and the bias current $I_b = 0.57I_c$, with $I_0$ being the critical current of the coupler, the total coupling energy $K$ is zero. Being able to tune $K$ is a useful feature of this design. It simplifies control of the

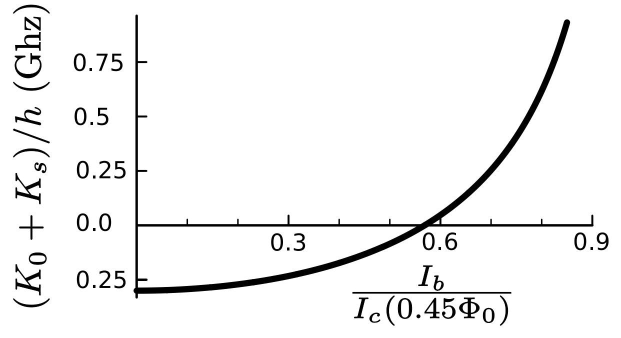


Figure 4.7: Coupling strength due to both direct qubit-qubit coupling $K_0$ as well as qubit-coupler-qubit $K_s$. By tuning the bias current $I_b$, the total coupling energy can be set to zero. Plot taken from [43].

qubits, and in particular makes performing two-qubit gates a less daunting task. Many other suggestions for controllable coupling architectures have been suggested [55, 33, 38, 51], but at this stage it is not yet clear which will become most useful on a large scale.

# Chapter 5

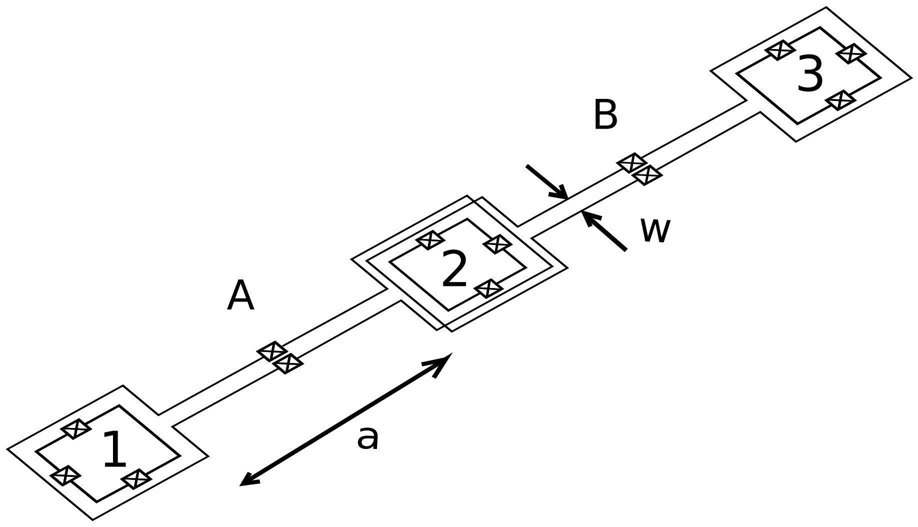
# Three Qubit Coupler

## 5.1 Motivation

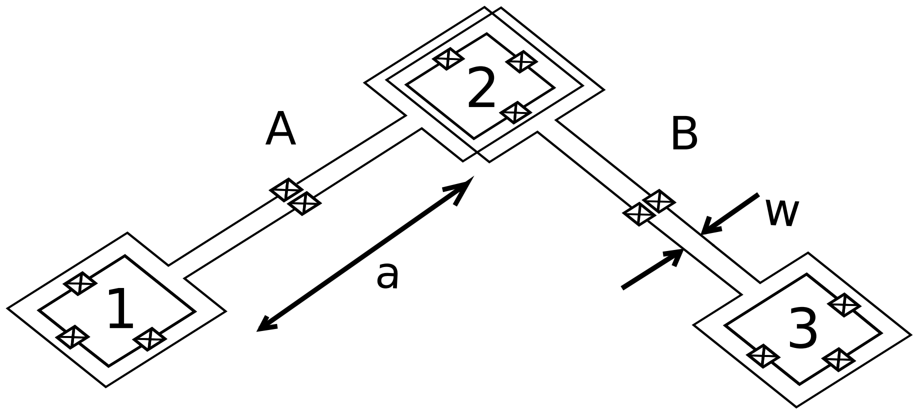
Coupling qubits is a critical aspect of building a large scale quantum computer. From the previous chapters of this thesis we see that even a simple two-dimensional lattice in theory provides an adequate "resource" and furthermore seems to yield a high error threshold, which naturally leads to its usefulness in experimental realizations. So far however, scientists have only begun to construct multi-qubit systems out of superconducting circuits. In this chapter we study a specific scenario where three flux qubits are coupled using two DC-SQUIDs. We explore how the coupling energy changes as we vary geometrical properties of the couplers for a set of experimentally achievable parameters, and concentrate on two cases of interest; the first where coupling between all the qubits is small, and another where it is large between one pair, and small between the other two. Although only a starting point, these results could be extended further to more complicated designs with both more qubits as well as couplers.

## 5.2 Geometry

The performance of any inductively coupled qubit architecture will strongly depend on the geometry of the qubit-coupler layout. In this chapter, we study two geometries that partially describe a square lattice, which could be used (for example) to implement a surface code. The geometries we study are both shown in Figure 5.1. In the layout (a), the coupling DC-SQUIDs form a straight line and in (b), a capital letter "L". In both cases we have two qubits at the end points (labeled as 1 and 3) and the third (labeled 2) in the middle. The middle qubit is therefore surrounded by both of the couplers. We consider the qubits at the edge to be nearest-neighbours to the middle qubit. The qubits sit inside the couplers, hence maximizing the qubit-coupler inductance. We take the edge length of our square qubits to be 44$\mu$m, roughly the same as those used by the Clarke group [25]. The



(a)



(b)

Figure 5.1: Two different arrangements of qubits 1, 2, 3 and couplers A, B. “a” represents the arm length of the couplers, and “w” its width. We assume that the wires of couplers A and B overlap at most in two places around qubit 2. a) Straight-line shape. b) Capital “L” shape.

distance between the qubit and the surrounding wire of the coupler is between 2 and 4$\mu$m (depending on whether a qubit is surrounded by one or two coupling DC-SQUIDs). Since qubit 2 is always surrounded by two DC-SQUIDs, an experimental realization of this scenario would require crossing the wires of the couplers in at least two places. For simplicity we assume a fully symmetric situation meaning the qubits that are situated at the edges look the same to the qubit in the middle.

To understand what impact qubit separation has on the coupling energy, we vary both the DC-SQUIDs' arm length and arm width (shown as "a" and "w" respectively in Figure 5.1). The arm length "a" varies between 50$\mu$m and 300$\mu$m and we study cases of the arm width $w = 48\mu$m and $w = 2\mu$m (although in the latter case we keep a $10 \times 10 \mu m^2$ loop in the center of the arm to simplify delivering a well defined flux through the couplers). In order to numerically calculate the inductance of any given configuration, we use a program called "FastHenry" [27]. To simplify its use, a "wrapper code" has been written, which is easily able to prepare rather involved configuration files for various geometries. We have assumed that both the qubit loops and the coupling DC-SQUIDs are made out of aluminum, and use a penetration depth of $51nm$ [5] (which in practice can vary depending on purity). The result is a $5 \times 5$ inductance matrix. As an example, we can look at the L-shaped geometry with the thin arm width of $w = 2\mu$m. With the arm length $a = 50\mu$m, the inductance matrix was obtained to be

$$\begin{array}{c} \\ Q1 \\ Q2 \\ Q3 \\ A \\ B \end{array}\begin{array}{c} \begin{array}{ccccc} Q1 & Q2 & Q3 & A & B \end{array} \\ \begin{pmatrix} 171.5483 & -0.4689 & -0.1659 & 73.9081 & -0.8751 \\ -0.4689 & 171.5483 & -0.4600 & 75.6132 & 75.5106 \\ -0.1659 & -0.4600 & 171.5483 & -0.8751 & 73.9081 \\ 73.9081 & 75.6132 & -0.8751 & 457.8219 & 91.8483 \\ -0.8751 & 75.5106 & 73.9081 & 91.8483 & 457.8219 \end{pmatrix} \end{array} \times 10^{-12}\text{H}, \quad (5.1)$$

and in a case of $a = 300\mu$m,

$$\begin{array}{c} \\ Q1 \\ Q2 \\ Q3 \\ A \\ B \end{array}\begin{array}{c} \begin{array}{ccccc} Q1 & Q2 & Q3 & A & B \end{array} \\ \begin{pmatrix} 171.5483 & -0.0101 & -0.0041 & 74.6179 & -0.0200 \\ -0.0101 & 171.5483 & -0.0089 & 76.3539 & 76.2424 \\ -0.0041 & -0.0089 & 171.5483 & -0.0200 & 74.6179 \\ 74.6179 & 76.3539 & -0.0200 & 716.4041 & 94.2563 \\ -0.0200 & 76.2424 & 74.6179 & 94.2563 & 716.4041 \end{pmatrix} \end{array} \times 10^{-12}\text{H}. \quad (5.2)$$

where in both cases the columns (and rows) correspond to the first, second, third qubits and couplers A and B respectively. The inductance matrices for other geometries are similar in order (hundreds of pH for DC-SQUIDs' self inductance). We point out that the matrix elements that correspond to the mutual inductance between each of the qubits and their nearest coupling DC-SQUIDs (there are two in a case of qubit 2) are much greater than the ones that corresponds to the mutual inductance between qubits themselves. The significance of this is that smaller currents (in the couplers) can still provide substantial coupling energy.

## 5.3 Hamiltonian

In this section we look at the mathematics of the coupling in more detail. We do this by expanding on the results presented in [43]. First we define the parameters of our system of which a schematic representation is shown in Figure 5.2. The three flux qubits are labeled 1, 2 and 3, and the couplers A and B. The variables $\gamma_{n,i}$ are phase differences across $i$th junction in a coupler $n$. $\Phi_n$ is the applied flux threaded through a coupler $n$ and $J_n$ its circulating current. $I_0$ is the critical current of the Josephson junctions used in the coupling DC-SQUIDs — for simplicity we assume that all the junctions, and hence their critical currents are the same. We further assume that we are able to control the flux independently through each of the DC-SQUIDs and qubits.

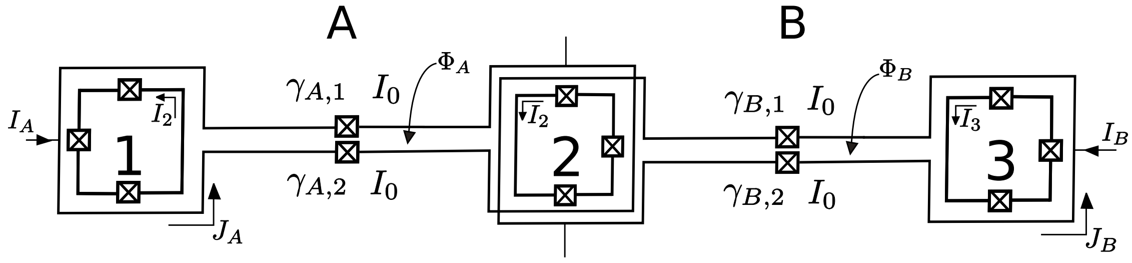


Figure 5.2: The three flux qubits labeled 1, 2 and 3, and two DC-SQUIDs labeled A and B, form a three qubit, two coupler system. $\gamma_{n,i}$ is the phase difference across $i$th junction in a coupler $n$. $\Phi_n$ and $J_n$ are the applied flux and circulating current of coupler $n$ respectively. $I_0$ is the critical current of the Josephson junctions used in the coupling DC-SQUIDs — for simplicity we assume that all the junctions, and hence their critical currents are the same.

Using the discussion presented in Chapters 1 and 4, we can write an effective Hamiltonian of the three flux qubits and interaction between them as

$$H = \sum_{i=1,2,3} H_i - \sum_{i \neq j} K_{ij} \sigma_z^i \otimes \sigma_z^i \tag{5.3}$$

where $H_i$ are the Hamiltonians of the individual qubits as described in Equation 4.8, and $K_{ij}$ correspond to the coupling energy between qubits $i$ and $j$. In writing this Hamiltonian, we make an important assumption that the energy difference between the ground and excited states of the couplers is much greater than that of the qubits. This lets us assume that the coupling DC-SQUIDs will stay in their ground state, and hence we can neglect the dynamics of the couplers. Furthermore we stick to the sign convention used in [43] and note that for $K < 0$ the minimum energy configuration corresponds to anti-parallel fluxes. The coupling terms $K_{ij}$ can be split into two parts; the first, $K_{ij}^0$ consisting of direct coupling between qubits $i$ and $j$ and the second $K_{ij}^c$ relating to the indirect coupling that is mediated by the two DC-SQUIDs themselves. The second term comes from the fact that as the persistent current of qubit $j$, $I_j$ changes direction, it modifies the DC-SQUIDs' circulating currents $J_A$ and $J_B$ which in turn alter the flux in qubit $i$. Mathematically we can describe the two coupling contributions as

$$K_{ij} = K^0_{ij} + K^c_{ij}. \tag{5.4}$$

To calculate these terms, we start by writing the change in the energy of qubit $i$ due to qubit $j$. This is simply

$$K_{ij} = I_i \Delta\Phi^j_i \tag{5.5}$$

where $\Delta\Phi^{(j)}_i$ refers to the change in flux of qubit $i$ due to qubit $j$. We can rewrite it using three terms, which are related to changes in the current of qubit $j$ and the circulating currents of both of the couplers

$$K_{ij} = I_i M_{ij} \Delta I_j - (M_{iA}\Delta J^{(j)}_A + M_{iB}\Delta J^{(j)}_B). \tag{5.6}$$

$\Delta I_j$ is the change in the persistent current of qubit $j$, and $\Delta J^{(i)}_A$ represents the change in the circulating current of the coupler $A$ only due to changes in qubit $j$ — likewise for coupler $B$. The negative sign in front of the second and third terms comes from the fact that the coupling that is mediated through the DC-SQUIDs is anti-ferromagnetic in nature — for a detailed justification of this, we point the reader to [19]. Each of the circulating currents depends on the applied fluxes $\Phi_A$ and $\Phi_B$, hence for fixed bias currents $I_A$ and $I_B$, we can write

$$\Delta J^{(j)}_A = \frac{\partial J_A}{\partial \Phi_A}\Delta\Phi^{(j)}_A + \frac{\partial J_A}{\partial \Phi_B}\Delta\Phi^{(j)}_B \tag{5.7}$$

where similarly as before, $\Delta\Phi^{(j)}_A$ and $\Delta\Phi^{(j)}_B$ correspond to the changes in applied fluxes of the coupling DC-SQUIDs $A$ and $B$ due to qubit $j$. We can write them as $M_{jA}\Delta I_j$ and $M_{jB}\Delta I_j$. Following the same arguments for $\Delta J^{(j)}_B$ and rewriting $\Delta I_j$ as simply $I_j$ (since the computational basis correspond to persistent currents in either one or the other direction, and as we're dealing with a two level system, only the signs of $I_j$ can change), we can write the total interaction energy between qubits $i$ and $j$ as

$$\begin{aligned} K_{ij} &= K^0_{ij} + K^c_{ij} && (5.8) \\ &= M_{ij} I_i I_j - \sum_{n=A,B}\sum_{k=A,B} M_{n,i} M_{k,j} \frac{\partial J_n}{\partial \Phi_k} I_i I_j. && (5.9) \end{aligned}$$

From the above we see that the coupling strength depends on the transfer functions of the form $\frac{\partial J_n}{\partial \Phi_k}$. And these ultimately depend on the DC-SQUIDs' control parameters; the bias current and the applied flux. In the following section, we will show how.

## 5.4 Transfer Function $\frac{\partial J}{\partial \Phi}$

Although in this work we are are limiting ourselves to treating a situation where only two coupling DC-SQUIDs are present, it is not much harder to generalize this particular section to an arbitrary number of coupling DC-SQUID devices. We will therefore initially treat a more general situation, and only at the end explain what the equations reduce to in our particular case.

We start with Kirchhoff's current laws for coupler $n$. In the limit of $\omega = 0$, (meaning the coupler stays in its ground state) and using results from Section 4.4 we know they can be written as

$$\begin{aligned} I_n &= I_0 \sin \gamma_{n1} + I_0 \sin \gamma_{n2} \\ 2J_n &= I_0 \sin \gamma_{n2} - I_0 \sin \gamma_{n1} \end{aligned} \tag{5.10}$$

Using $\gamma_{n-} = \frac{\gamma_{n1}-\gamma_{n2}}{2}$ and $\gamma_{n+} = \frac{\gamma_{n1}+\gamma_{n2}}{2}$, we rewrite equations 5.10 as

$$I_n = 2I_0 \sin \gamma_{n+} \cos \gamma_{n-} \tag{5.11}$$

and

$$J_n = I_0 \sin \gamma_{n-} \cos \gamma_{n+} \tag{5.12}$$

From fluxoid quantization [16], we have another condition, which relates the flux passing through the $n$th DC-SQUID to a change in phases across the junctions. This lets us write

$$\gamma_{n-} = \frac{\pi}{\Phi_0}(\Phi_n - M_n J_n - \sum_{l,l\neq n} M_{nl} J_l) \tag{5.13}$$

where $\Phi_n$ is the applied flux, $M_n$ the geometric inductance of the $n$th DC-SQUID, $M_n J_n$ flux due to its circulating current, and the last term involving a sum over $l$ is related to the flux passing through the $n$th coupler due to the circulating currents in all the others. By fixing the applied flux $\Phi_n$ and bias current $I_n$ for all $n$ (i.e. in all the coupling DC-SQUIDs) we can solve equations 5.11, 5.12 and 5.13 to obtain circulating currents $J_n$ as well as the phases $\gamma_{n-}$ and $\gamma_{n+}$. Next we fix the bias currents $I_n$ and implicitly differentiate Equations 5.11, 5.12 and 5.13 in order to obtain the transfer functions $\frac{\partial J_n}{\partial \Phi_k}$ (at this stage $k$ may equal $n$), which we'll ultimately need to calculate the interaction energy $K_{ij}$. Therefore differentiating 5.11 with respect to $\Phi_k$ gives

$$\frac{\partial I_n}{\partial \Phi_k} = 0 = 2I_0 \frac{\partial}{\partial \Phi_k} (\sin \gamma_{n+} \cos \gamma_{n-}) \tag{5.14}$$

which further leads to

$$\cos \gamma_{n+} \frac{\partial \gamma_{n+}}{\partial \Phi_k} \cos \gamma_{n-} - \sin \gamma_{n-} \frac{\partial \gamma_{n-}}{\partial \Phi_k} \sin \gamma_{n+} = 0. \tag{5.15}$$

Now solving for $\frac{\partial \gamma_{n+}}{\partial \Phi_k}$ gives

$$\frac{\partial \gamma_{n+}}{\partial \Phi_k} = \tan^2 \gamma_{n+} \tan^2 \gamma_{n-} \frac{\partial \gamma_{n-}}{\partial \Phi_k}. \tag{5.16}$$

Next, by differentiating 5.12 and substituting in Equation 5.16 we arrive at

$$\begin{aligned} \frac{\partial J_n}{\partial \Phi_k} &= I_0 \left(\cos \gamma_{n-} \cos \gamma_{n+} - \sin \gamma_{n-} \sin \gamma_{n+} \tan \gamma_{n-} \tan \gamma_{n+}\right) \frac{\partial \gamma_{n-}}{\partial \Phi_k} && (5.17) \\ &= I_0 \cos \gamma_{n-} \cos \gamma_{n+} \left(1 - \tan^2 \gamma_{n-} \tan^2 \gamma_{n+}\right) \frac{\partial \gamma_{n-}}{\partial \Phi_k} && (5.18) \end{aligned}$$

Finally, we're left with obtaining $\frac{\partial \gamma_{n-}}{\partial \Phi_k}$, which will now depend on whether $n$ and $k$ are the same, or different. Differentiating Equation 5.13 with respect to $\Phi_k$ and treating a case where $k = n$, we get

$$\frac{\partial \gamma_{n-}}{\partial \Phi_n} = \frac{\pi}{\Phi_0} \left( 1 - M_n \frac{\partial J_n}{\partial \Phi_n} - \sum_{l, l \neq n} M_{nl} \frac{\partial J_l}{\partial \Phi_n} \right). \tag{5.19}$$

Now assuming that $k \neq n$ yields

$$\frac{\partial \gamma_{n-}}{\partial \Phi_k} = \frac{\pi}{\Phi_0} \left( -M_n \frac{\partial J_n}{\partial \Phi_k} - \sum_{l, l \neq n} M_{nl} \frac{\partial J_l}{\partial \Phi_k} \right). \tag{5.20}$$

Substituting the results of equations 5.20 and 5.19 into 5.18 and solving for $\frac{\partial J_n}{\partial \Phi_n}$ and $\frac{\partial J_n}{\partial \Phi_k}$ leads to

$$\frac{\partial J_n}{\partial \Phi_n} = \frac{1}{2L_n^J} \frac{1 - \tan^2 \gamma_{n+} \tan^2 \gamma_{n-}}{1 + \frac{M_n}{2L_n^J} \left[1 - \tan^2 \gamma_{n+} \tan^2 \gamma_{n-}\right]} \left( 1 - \sum_{l, l \neq n} M_{nl} \frac{\partial J_l}{\partial \Phi_n} \right) \tag{5.21}$$

and for a case where $n \neq k$

$$\frac{\partial J_n}{\partial \Phi_k} = \frac{-1}{2L_n^J} \frac{1 - \tan^2 \gamma_{n+} \tan^2 \gamma_{n-}}{1 + \frac{M_n}{2L_n^J} \left[1 - \tan^2 \gamma_{n+} \tan^2 \gamma_{n-}\right]} \sum_{l, l \neq n} M_{nl} \frac{\partial J_l}{\partial \Phi_k}, \tag{5.22}$$

where we have taken

$$L_n^J = \frac{\Phi_0}{2\pi I_0 \cos \gamma_{n-} \cos \gamma_{n+}} \tag{5.23}$$

to be the Josephson inductance associated with the $n$th coupling DC-SQUID. For a system with $N$ couplers, one would have to simultaneously solve $N^2$ equations in order to calculate all the combinations of $\frac{\partial J_n}{\partial \Phi_k}$. Naturally, things could be simplified by accounting for geometrical factors. For example DC-SQUIDs that would be "far away" from each other would have small mutual inductance terms $M_{nl}$ and hence the correction to the flux threaded through a given coupler due to another might be neglected (and therefore their equations decoupled).

As a check of our calculations, we can observe that in a case of one coupler, $M_{nl} = 0$ in Equation 5.21 and Equation 5.22 does not come into play since there is only one applied flux in the problem. This lets us reproduce the results derived in [43]. We are also now ready to adapt the results from Equations 5.21 and 5.22 and explicitly write them for two coupler scenario that we are dealing with. As before, labeling the coupling DC-SQUIDs $A$ and $B$ gives

$$\frac{\partial J_A}{\partial \Phi_A} = \frac{1}{2L_A^J}\frac{1-\tan^2\gamma_{A+}\tan^2\gamma_{A-}}{1+\frac{M_A}{2L_A^J}\left[1-\tan^2\gamma_{A+}\tan^2\gamma_{A-}\right]}\left(1-M_{AB}\frac{\partial J_B}{\partial \Phi_A}\right) \tag{5.24}$$

$$\frac{\partial J_A}{\partial \Phi_B} = \frac{-1}{2L_A^J}\frac{1-\tan^2\gamma_{A+}\tan^2\gamma_{A-}}{1+\frac{M_A}{2L_A^J}\left[1-\tan^2\gamma_{A+}\tan^2\gamma_{A-}\right]}M_{AB}\frac{\partial J_B}{\partial \Phi_B} \tag{5.25}$$

$$\frac{\partial J_B}{\partial \Phi_B} = \frac{1}{2L_B^J}\frac{1-\tan^2\gamma_{B+}\tan^2\gamma_{B-}}{1+\frac{M_B}{2L_B^J}\left[1-\tan^2\gamma_{B+}\tan^2\gamma_{B-}\right]}\left(1-M_{AB}\frac{\partial J_A}{\partial \Phi_B}\right) \tag{5.26}$$

$$\frac{\partial J_B}{\partial \Phi_A} = \frac{-1}{2L_B^J}\frac{1-\tan^2\gamma_{B+}\tan^2\gamma_{B-}}{1+\frac{M_B}{2L_B^J}\left[1-\tan^2\gamma_{B+}\tan^2\gamma_{B-}\right]}M_{AB}\frac{\partial J_A}{\partial \Phi_A} \tag{5.27}$$

Hence for a given geometry (which defines the inductance matrix M), the applied fluxes $\Phi_A$, $\Phi_B$ as well as the bias currents $I_A$, $I_B$, we can now have a means of calculating the interaction energy from Equation 5.9, as well as the Hamiltonian from Equation 5.3.

## 5.5 The Critical Current Of The Coupling DC-SQUIDs

In Section 4.4 we outlined that the critical current of a DC-SQUID can be written as shown in Equation 4.15. This result however, was for a case where the inductance of the DC-SQUID was ignored, and in this section we want to see what impact including it may have on the value of the critical current for the coupler geometries we study. The strength of the interaction energy $K_{ij}$, which will be calculated in the next section, clearly does not depend on the critical current itself, however it does depend on the bias current of both of the couplers, hence we need to have an idea about the maximum value that a bias current can have before the couplers switch out of the zero voltage state. It is worth stressing that we only consider the self inductance of a given DC-SQUID, and neglect further effects from the mutual inductance between the coupler in question and its neighbour.

Rewriting Equations 5.11 and 5.12 using 5.13 and neglecting the mutual inductance between different couplers (the term with $M_{nl}$ in Equation 5.13) we obtain

$$I_n = 2I_0 \sin\gamma_{n+}\cos\left(\frac{\pi}{\Phi_0}\left(\Phi_n - M_n J_n\right)\right) \tag{5.28}$$

and

$$J_n = I_0 \sin\left(\frac{\pi}{\Phi_0}\left(\Phi_n - M_n J_n\right)\right)\cos\gamma_{n+}. \tag{5.29}$$

Here, as before, $n$ represents either coupler $A$ or $B$. In order to find the critical current for a given value of applied flux $\Phi_n$, we simply fix $\Phi_n$ and maximize Equation 5.28 with respect to $\gamma_{n+}$, subject to 5.29. Plotting this results in the dashed, red line in Figure 5.3. For comparison we also include a plot of the critical cur-

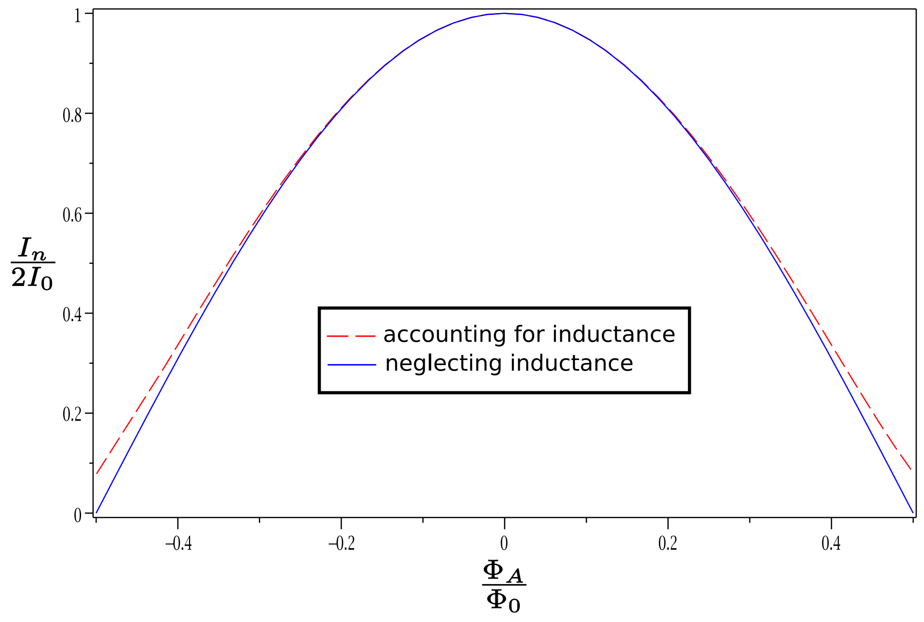


Figure 5.3: Critical current of a DC-SQUID. The solid (blue) line represents a case where all effects due to the inductance are neglected, and the dashed (red) line a situation where the self-inductance of the DC-SQUID is taken into account. In the latter case, we take the scenario of the largest inductance of all the geometries we study — where the arm width $w = 48\mu$m and arm length $a = 300\mu$m, which leads to $M_A = M_B = 1.011$nH.

rent, obtained using Equation 4.15, where the inductance is not taken into account — solid, blue line. We look at a geometry where the inductance of the coupling DC-SQUID is the largest; namely the case where the arm width is $w = 48\mu$m and arm length $a = 300\mu$m, which leads to $M_A = M_B = 1.011$nH. We can see that that in our case the critical current is not affected by much, and the main differences come in only for high (absolute) values of the applied flux $\Phi_n$. An ideal situation would be to also include effects of mutual inductance between the two coupling DC-SQUIDs, but since we're only using the resulting critical current values as a rough guide to what bias currents can be applied to the couplers, and not for calculations of the actual coupling energy (where the extra terms are included — see previous section), we do not consider these mutual inductance effects in our critical current calculations.

## 5.6 Coupling Strength

In this section we use the equations obtained above to numerically calculate all the interaction energies $K_{ij}$ of the Hamiltonian shown in Equation 5.3. In our calculations, we take the qubits' persistent currents values to be $0.46\mu$A and the critical currents of the Josephson junctions that are used in DC-SQUIDs as $0.11\mu$A — both in the experimentally achievable ranges. In order to answer whether all coupling can be turned off, or whether we can selectively make certain interaction strong, while keeping others weak, we need to understand what happens to all the coupling terms for all possible input parameters. To do this, we scan both of the applied fluxes $\Phi_A$ and $\Phi_B$ between $-\pi\Phi_0/2$ and $-\pi\Phi_0/2$ and the bias currents between 0 and their critical value for a given flux (as discussed in Section 5.5). The inductance matrix is obtained for each geometry and incorporated into the calculations to obtain values for all possible $K_{ij}$. The results are explicitly shown for both line-shape and L-shape geometries, in a case where the arm width "w" is $2\mu$m, in Figures 5.4 and 5.5 respectively. The vertical axis corresponds to the values of coupling $K_{12}$, the horizontal to coupling $K_{23}$ and finally the "out-of-page" direction to the crosstalk term $K_{13}$. Obtaining these results included calculating thousands of data points, which were then used to obtain a surface that spans all possible combinations of the three coupling energies. We further repeated the process for all the different geometries, while varying the arm length and arm width of the coupling DC-SQUIDs.

The first question we wish to address is whether we can keep the coupling between two nearest neighbour qubits (say 1 and 2) high, the other nearest neighbour coupling (say 2 and 3) turned off, while eliminating the crosstalk (interaction between qubits 1 and 3). We can get the answer to this by looking at plots in Figures 5.4 and 5.5. We first fix $K_{23}$ (horizontal axis) at zero, and then traverse in the vertical direction along the dashed line (note that in order to simplify this procedure, each plot has two dashed lines which represent cases of $K_{12} = 0$ and $K_{23} = 0$). We find that in the straight-line geometry (see Figure 5.4), in the case of the shortest arm length of $a = 50\mu$m, when the interaction energy $K_{12} \approx 1.0$GHz, while $K_{23} \approx 0$, the crosstalk is at $\approx 16$MHz, which is roughly 1.6% of $K_{12}$. Furthermore, as we increase the couplers' arm width it becomes clear that one can easily find a value of $K_{12}$ beyond which $K_{13}$ is always zero (for example when $a = 300$, $K_{13} \approx 0$ for $K_{12} \geq 0.7$GHz).

In the case of the L-shape geometry (Figure 5.5), the situation is less favorable — which is expected since qubits 1 and 3 are closer together and their direct mutual inductance is larger. Furthermore, they interact with the coupler that is further away from them more than is the case in the line-shaped geometry. From Figure 5.5 we see that the best we can do for a case of short couplers ($a = 50\mu$m) and the coupling strength $K_{12} \approx 1.0$GHz, is $K_{13}$ with a magnitude of $\approx 62$MHz — which is a few times larger than in the straight-line geometry scenario. However, as expected, we can reduce the crosstalk term $K_{13}$ by simply increasing the arm length of the coupling DC-SQUIDs — for example in a case of $a = 300\mu$m, for $K_{12} > 1.0$GHz, the magnitude of $K_{13}$ is smaller than 2MHz. We have further found that increasing the arm width of the couplers made no qualitative difference in the results above, however for the same set of parameters the magnitudes of the interaction energies ($K_{ij}$) were smaller. This is due to the fact that the mutual inductance elements between these wider coupling DC-SQUIDs and qubits are smaller than in cases of the narrow couplers. The discussion above leads us to conclude that minimizing the crosstalk coupling (term $K_{13}$) can be done to a great degree, even when fairly small DC-SQUIDs are used and more importantly in both of the studied geometries.

The next question we wish to answer is whether the interaction between all pairs of qubits can be turned off completely. This may be useful when one wants to perform single gate operations on any (or all) of the qubits without affecting the others. To determine this, we once again turn to Figures 5.4 and 5.5 (and their analogues for cases of $w = 48\mu$m — not explicitly shown). We can now look at the crossing of the two dashed lines in each plot in the figures, which correspond to $K_{12} = K_{23} = 0$, and try to figure out the corresponding value of $K_{13}$. We plot the results in Figure 5.6 for both geometries and two different arm widths. From the data it is clear that once again the straight-line geometry is much less (over 3 times in a case of short arm length) susceptible to the unwanted interaction $K_{13}$. However the crosstalk energy dies off quickly, and in a case of arm length $a > 200\mu$m, its magnitude stays below 5MHz. Given that the typical energies of the qubits are of the order of a few GHz, the crosstalk strength of roughly three orders of magnitude smaller may not be very damaging. We also find that the width of the couplers have little effect on how well all coupling energies can be turned off.

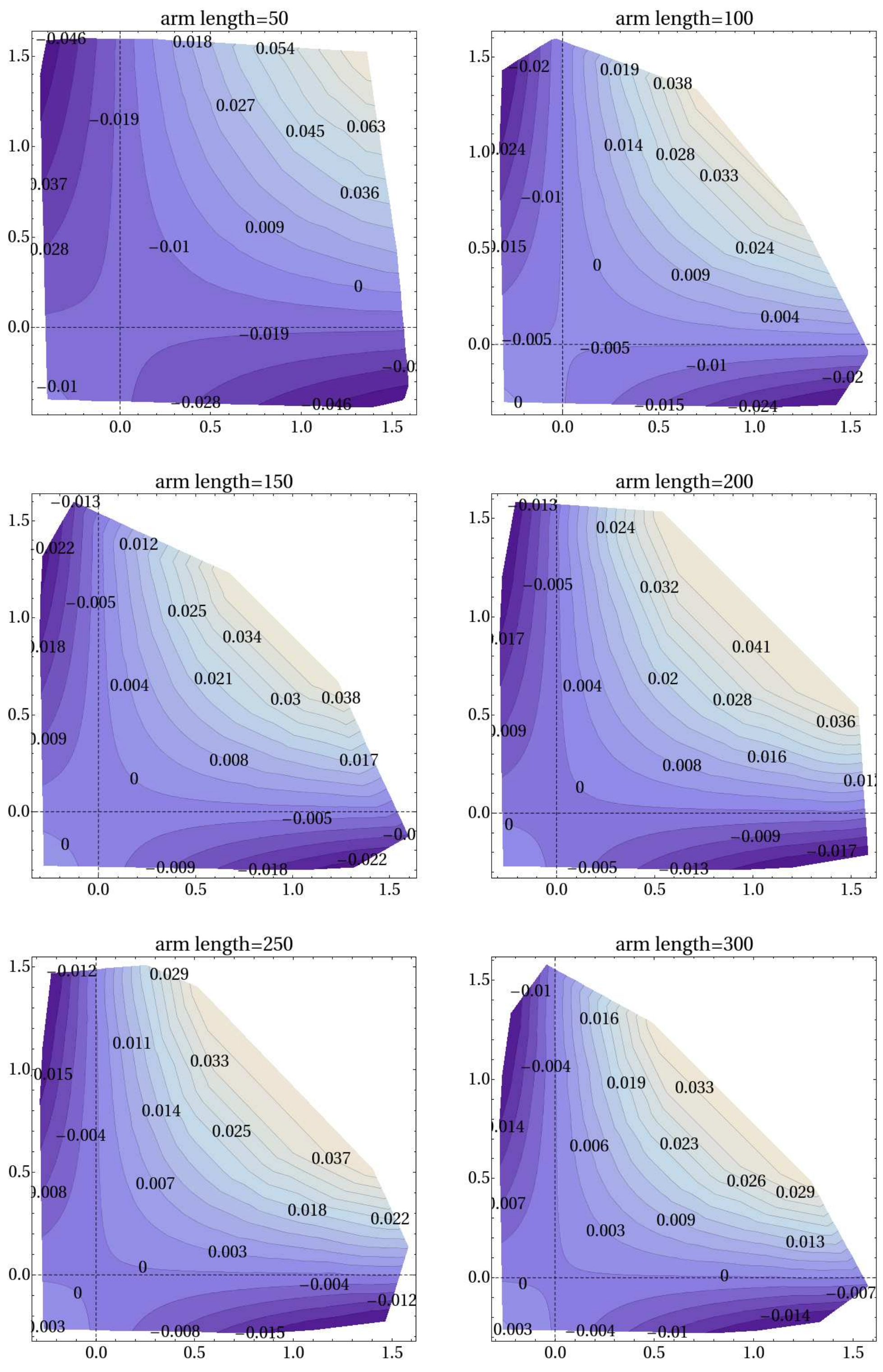


Figure 5.4: Interaction energies $K_{12}$ (vertical axis), $K_{23}$ (horizontal axis) and $K_{13}$ (out-of-page direction) for a straight-line geometry with couplers' arm width of $w = 2\mu$m. Even in a case of a short arm lengths the crosstalk is only a few percent of the interaction between qubits 1 and 3 or 2 and 3. All values are shown in GHz.

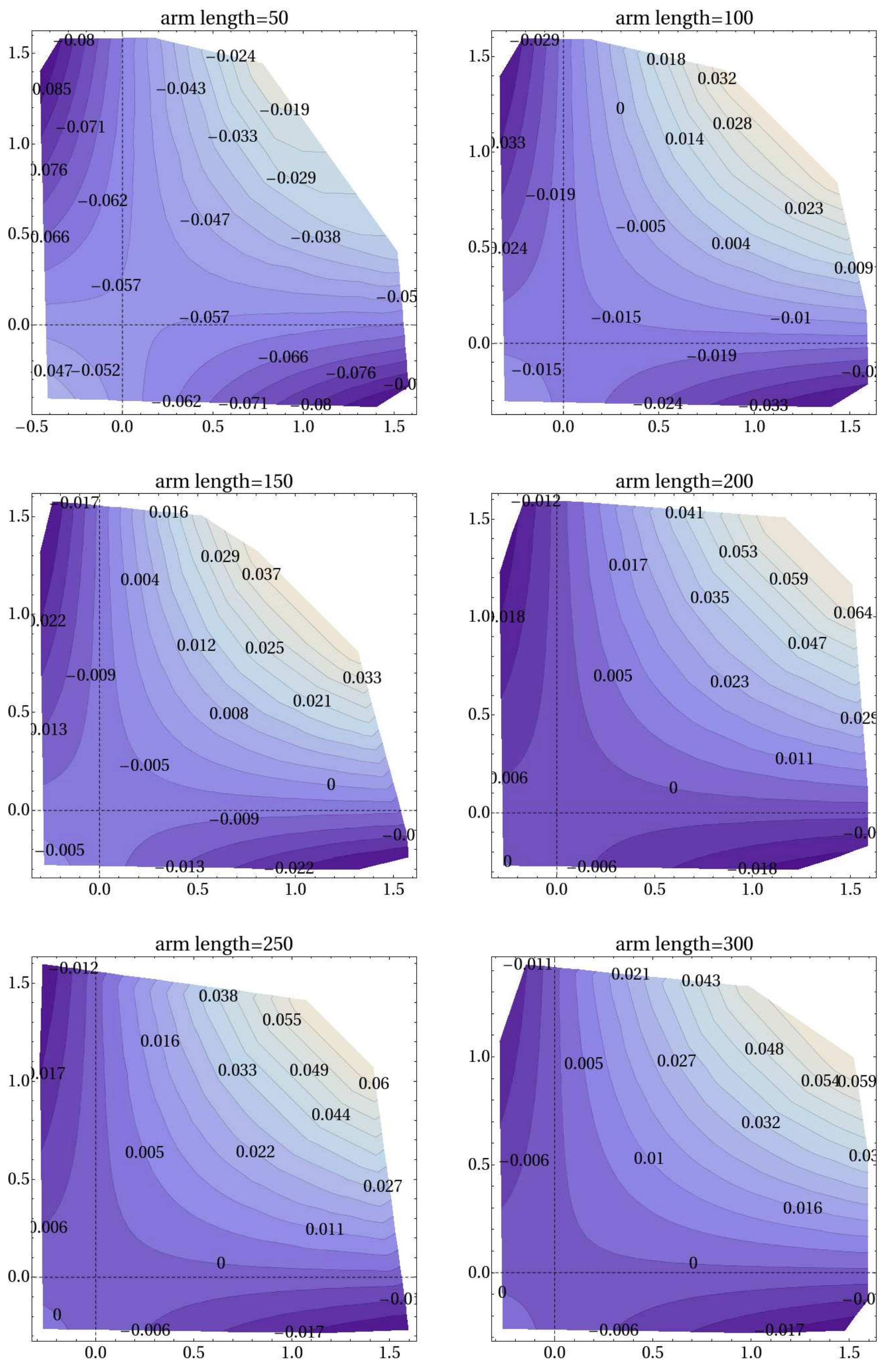


Figure 5.5: Interaction energies $K_{12}$ (vertical axis), $K_{23}$ (horizontal axis) and $K_{13}$ (out-of-page direction) for an L-shape geometry with couplers' arm width of $w = 2\mu$m. All values are shown in GHz.

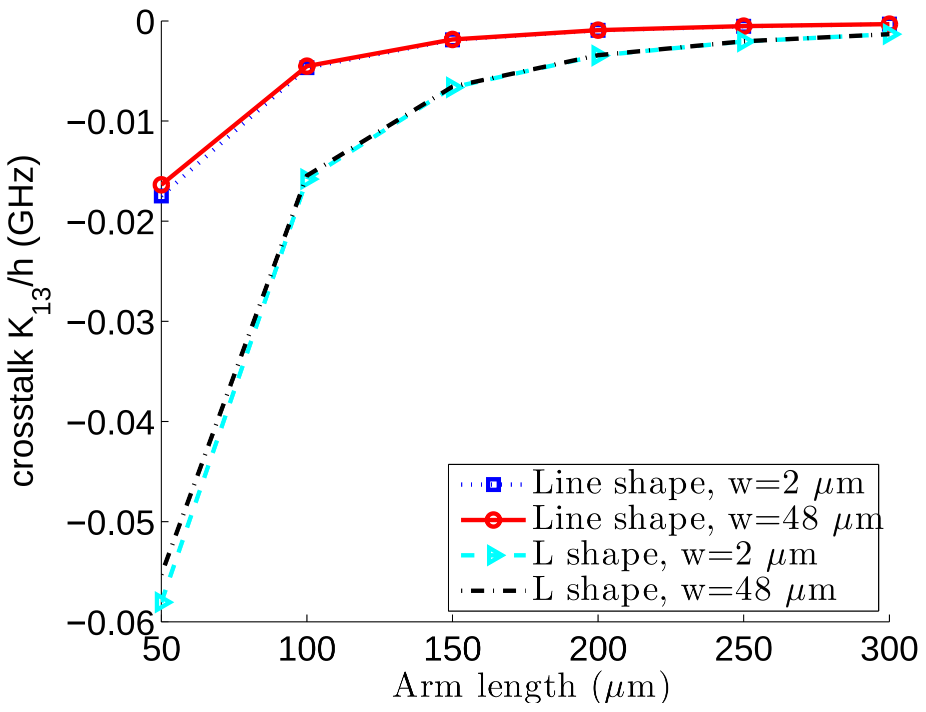


Figure 5.6: A plot consisting of values for the crosstalk $K_{13}$, while keeping $K_{12} = K_{23} \approx 0$. The horizontal axis represents different arm lengths of the coupling DC-SQUIDs. We find that the magnitude of $K_{13}$ is substantially smaller in the case of the straight-line geometry, but even in the L-shaped scenario dies off quickly.

# Chapter 6

# Conclusions

It is still unclear what form large scale quantum computing devices will take. It is likely that whatever the physical hardware will be, some form active error correction will be required[1]. An error threshold can tell us what physical error rate can be tolerated by a particular architecture and error correcting code, while still allowing for arbitrary quantum computation. In this thesis, we concentrated on numerically calculating such an error threshold for two variations of surface codes; toric and planar. We obtained an estimate of $6.0 \times 10^{-3}$, which is similar to other results presented in literature [45, 44, 54]. We gave a detailed description of the calculations and methods that were involved in obtaining these numbers. We also stressed the importance of proper syndrome readout order and explicitly showed how some orders can lead to syndrome qubit entanglement, which could potentially be very decremental to the threshold value (although no actual threshold calculations with entangling readout orders were performed).

In the second part of this thesis, we included a discussion of flux qubit coupling which provides a very first step in the direction of multi-qubit devices built out of superconducting circuits. In particular we extended the coupling mechanism first presented in [43], to a three-qubit, two couplers scenario. We studied multiple geometries of the coupling DC-SQUIDs; a straight-line shape, and an L-shape, with couplers' arm widths of $50\mu$m and $300\mu$m, and arm widths between 2 and $48\mu$m. We hoped to answer two important questions; the first, whether by choosing appropriate DC-SQUIDs' parameters, we can completely turn off the coupling between all three qubits, and the second, whether we can keep a coupling energy between two nearest neighbours hight while at the same time eliminating their interaction with the other qubit. Both of these questions will play a central role in larger systems, where many more qubits are involved. In particular, turning off all interaction energy is desirable when one needs to perform single qubit gates. On the other hand, keeping the interaction high between any two qubits, but not others is important when a two-qubit gate between those two qubits is being implemented.

[1]This may not be true in a case of topological quantum computers which require special states of matter. However, at this time it is not clear that their implementation will even be possible.

In the end we found that, as expected, for shorter arm lengths, in the straight-line geometries it was easier to keep the interaction between all qubits small, than it was for the L-shaped scenario; in particular we concentrated on turning the coupling $K_{12}$ and $K_{23}$ off completely, while observing how the crosstalk term $K_{13}$ varies with different geometries. In both cases, for arm lengths larger than $200\mu$m, the magnitude of the interaction between the two furthest qubits ($K_{13}$) stayed below 5MHz. Furthermore, we found that the arm width of the couplers made virtually no difference (although it did affect the maximum strength that a given coupling could have). When it came to turning on the coupling between two nearest neighbour qubits (for example $K_{12}$), to the strength of approximately 1.0GHz, and keeping the interaction $K_{23} \approx 0$, we observed that in the case of straight-line geometry, the crosstalk was roughly 1.6% of $K_{12}$ — even when the smallest arm lengths were considered. In the L-shape geometry, with the shortest arm length of $50\mu$m, the crosstalk increased to as much as 6% of $K_{12}$. However in both cases, by increasing the couplers' lengths to $300\mu$m the ratio of $K_{13}/K_{12}$ could be reduced substantially to well below a few MHz. The situation was completely analogous in the case where wide arm couplers were used, once again with the exception that maximum achievable coupling energies were smaller than in their narrow arm counterparts.

In summary, our results show that in a two coupler, three qubit case, even with short arm lengths one can substantially reduce unwanted crosstalk (term $K_{13}$), while still keeping wanted interaction ($K_{12}$ or $K_{23}$) high. It is clear that our design is only the very first step when it comes to implementing a full lattice which could be used for surface code computation. Future work may involve modeling of a system with additional qubits and couplers, and also studying methods of single and multi-qubit control using optimization techniques such as the GRAPE algorithm, while taking into account the constraints on the coupling energies due to the system's geometrical properties.

# APPENDICES

# Appendix A

# Readout Cycle Stabilizer Evolution

In this chapter we show the stabilizer evolution during the six step readout cycle for different CNOT gate readout orders. For brevity, we study a planar 2 by 2 faces (5 by 5 qubits, including ancillas) lattice. This has turned out to be important, as certain orders of CNOT gates during the six step readout cycle may entangle some of the ancilla qubits together. We explicitly show the full calculations for two different CNOT orders in a planar code case. Although not conceptually difficult, these calculations if done by hand are very tedious, and therefore special software was written to simplify and automate the procedure.

The tables below in Sections A.1 and A.2 show what the stabilizer looks like after the first five steps of the readout cycle (the last step which consists of the ancilla qubit measurements is not explicitly shown). The columns represent qubits (numbered as shown in Figure A.1), and the rows, the stabilizer generators. The

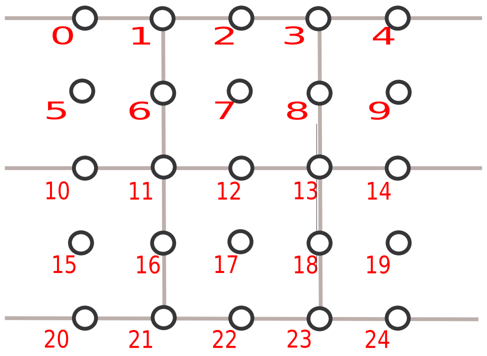


Figure A.1: A 5 by 5 qubit planar code lattice. Each of the qubits is numbered — with the top left having an index of 0 and the bottom right of 24.

order of those, is simply top left to bottom right — the first twelve are generators associated with the vertices, and the following twelve with the plaquettes. We assume that there aren't any errors during the cycle, and therefore the evolution is only due to the ancilla qubit initialization and the CNOT gates. It is worth noting that if errors were introduced, the only difference would be a potential sign change on some of the terms in the stabilizer. Before each table, a list of gates that were performed between the last step and the current step is shown. The notation of cnot$(i, j)$ implies that the $i$th qubit was taken as control, and $j$th as target. A table showing these and other outcomes from both planar and toric codes and different readout orders is presented in Table 3.1.

In Section A.1 where we study a NWES readout pattern, we observe that after the last step, the generators that describe the states of the ancilla qubits are written in a simple product form. Namely $X_1$, $X_3$, $X_{11}$, $\ldots Z_5$, $Z_7$, $Z_9$, etc., which tells us that they are not entangled. On the other hand, Section A.2 where a NWSE readout pattern is studied, shows that certain ancilla states cannot be expressed in a product form. We see for example that the states of qubits 1 and 5 are described by the operators $X_1X_5$ and $Z_1Z_5$, which correspond to $(|00\rangle + |11\rangle)/\sqrt{2}$ — an entangled Bell state. This implies that even though no errors were present, and we would hope that measuring the ancilla qubits 1 and 5 would deterministically yield eigenvalues of +1, this will be the case only only with a probability 1/2.

In summary, we stress that the order of CNOT gates during the ancilla qubit readout cycle may produce entanglement, leading to non-deterministic outcomes of the measurements results.

## A.1 Planar Code, NWES Readout

```
Step 0 (Init):
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X           X
  1     X
  2        X     X           X
  3           X
  4                    X           X     X           X
  5                                   X
  6                          X           X     X           X
  7                                         X
  8                                                  X           X     X
  9                                                                 X
 10                                                        X           X     X
 11                                                                       X
 12  Z                 Z           Z
 13                 Z
 14        Z           Z     Z           Z
 15                       Z
 16              Z           Z                 Z
 17                             Z
 18                                Z                 Z           Z
 19                                               Z
 20                                      Z           Z     Z           Z
 21                                                     Z
 22                                            Z           Z                 Z
 23                                                           Z

Step 1 (CNOT North):
cnot(11,6)
cnot(21,16)
cnot(13,8)
cnot(23,18)
cnot(0,5)
cnot(10,15)
cnot(2,7)
cnot(12,17)
cnot(4,9)
cnot(14,19)
After step 1:
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X        X  X  X
  1     X
  2        X     X        X  X  X
  3           X
  4                    X           X     X        X  X  X
  5                    X              X
  6                          X           X     X        X  X  X
  7                          X              X
  8                                                  X           X     X
  9                                                  X              X
 10                                                        X           X     X
 11                                                        X              X
 12  Z                 Z           Z  Z
 13  Z              Z
 14        Z           Z     Z        Z  Z  Z
 15        Z              Z
 16              Z           Z              Z  Z
 17              Z              Z
 18                                Z                 Z           Z  Z
 19                                Z              Z
 20                                      Z           Z     Z        Z  Z  Z
 21                                      Z              Z
 22                                            Z           Z              Z  Z
 23                                            Z              Z

Step 2 (CNOT West):
cnot(1,0)
cnot(11,10)
cnot(21,20)
cnot(3,2)
cnot(13,12)
cnot(23,22)
cnot(6,7)
cnot(16,17)
cnot(8,9)
cnot(18,19)
After step 2:
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X        X  X
  1  X  X
  2        X     X        X  X
  3        X  X
  4                    X  X        X     X        X  X
  5                    X  X        X  X
  6                          X  X        X     X        X  X
  7                          X  X        X  X
  8                                                  X  X        X     X
  9                                                  X  X        X  X
 10                                                        X  X        X     X
 11                                                        X  X        X  X
 12  Z  Z              Z           Z
 13  Z  Z           Z
 14        Z  Z        Z     Z        Z  Z
 15        Z  Z        Z  Z
 16              Z           Z              Z  Z
 17              Z           Z  Z
 18                                Z  Z              Z           Z
 19                                Z  Z           Z
 20                                      Z  Z        Z     Z        Z  Z
 21                                      Z  Z        Z  Z
 22                                            Z           Z              Z  Z
 23                                            Z           Z  Z

Step 3 (CNOT East):
cnot(1,2)
cnot(11,12)
cnot(21,22)
cnot(3,4)
cnot(13,14)
cnot(23,24)
cnot(6,5)
cnot(16,15)
cnot(8,7)
cnot(18,17)
After step 3:
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X           X
  1  X  X  X
  2        X     X           X
  3        X  X  X
  4                 X  X  X        X     X           X
  5                 X  X  X        X  X  X
  6                       X  X  X        X     X           X
  7                       X  X  X        X  X  X
  8                                               X  X  X        X     X
  9                                               X  X  X        X  X  X
 10                                                     X  X  X        X     X
 11                                                     X  X  X        X  X  X
 12  Z  Z              Z           Z
 13  Z  Z           Z  Z
 14     Z  Z  Z        Z     Z           Z
 15     Z  Z  Z        Z  Z  Z
 16           Z  Z           Z                 Z
 17           Z  Z           Z  Z
 18                                Z  Z              Z           Z
 19                                Z  Z           Z  Z
 20                                   Z  Z  Z        Z     Z           Z
 21                                   Z  Z  Z        Z  Z  Z
 22                                         Z  Z           Z                 Z
 23                                         Z  Z           Z  Z

Step 4 (CNOT South):
cnot(1,6)
cnot(11,16)
cnot(3,8)
cnot(13,18)
cnot(10,5)
cnot(20,15)
cnot(12,7)
cnot(22,17)
cnot(14,9)
cnot(24,19)
After step 4:
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X           X
  1  X  X  X           X
  2        X     X           X
  3        X  X  X           X
  4                    X           X     X           X
  5                    X           X  X  X           X
  6                          X           X     X           X
  7                          X           X  X  X           X
  8                                                  X           X     X
  9                                                  X           X  X  X
 10                                                        X           X     X
 11                                                        X           X  X  X
 12  Z                 Z           Z
 13  Z              Z  Z           Z
 14        Z           Z     Z           Z
 15        Z           Z  Z  Z           Z
 16              Z           Z                 Z
 17              Z           Z  Z              Z
 18                                Z                 Z           Z
 19                                Z              Z  Z           Z
 20                                      Z           Z     Z           Z
 21                                      Z           Z  Z  Z           Z
 22                                            Z           Z                 Z
 23                                            Z           Z  Z              Z

After step 4; rewrite
stab(1) = stab(0) x stab(1)
stab(3) = stab(2) x stab(3)
stab(5) = stab(4) x stab(5)
stab(7) = stab(6) x stab(7)
stab(9) = stab(8) x stab(9)
stab(11) = stab(10) x stab(11)
stab(13) = stab(12) x stab(13)
stab(15) = stab(14) x stab(15)
stab(17) = stab(16) x stab(17)
stab(19) = stab(18) x stab(19)
stab(21) = stab(20) x stab(21)
stab(23) = stab(22) x stab(23)
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X           X
  1     X
  2        X     X           X
  3           X
  4                    X           X     X           X
  5                                   X
  6                          X           X     X           X
  7                                         X
  8                                                  X           X     X
  9                                                                 X
 10                                                        X           X     X
 11                                                                       X
 12  Z                 Z           Z
 13                 Z
 14        Z           Z     Z           Z
 15                       Z
 16              Z           Z                 Z
 17                             Z
 18                                Z                 Z           Z
 19                                               Z
 20                                      Z           Z     Z           Z
 21                                                     Z
 22                                            Z           Z                 Z
 23                                                           Z
```

## A.2 Planar Code, NWSE Readout

```
Step 0 (Init):
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X           X
  1     X
  2        X     X           X
  3           X
  4                    X           X     X           X
  5                                   X
  6                          X           X     X           X
  7                                         X
  8                                                  X           X     X
  9                                                                 X
 10                                                        X           X     X
 11                                                                       X
 12  X                 X           X
 13                 Z
 14        Z           Z     Z           Z
 15                       Z
 16              Z           Z                 Z
 17                             Z
 18                                Z                 Z           Z
 19                                               Z
 20                                      Z           Z     Z           Z
 21                                                     Z
 22                                            Z           Z                 Z
 23                                                           Z

Step 1 (CNOT North):
cnot(11,6)
cnot(21,16)
cnot(13,8)
cnot(23,18)
cnot(0,5)
cnot(10,15)
cnot(2,7)
cnot(12,17)
cnot(4,9)
cnot(14,19)
After step 1:
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X        X  X  X
  1     X
  2        X     X        X  X  X
  3           X
  4                    X           X     X        X  X  X
  5                    X              X
  6                          X           X     X        X  X  X
  7                          X              X
  8                                                  X           X     X
  9                                                  X              X
 10                                                        X           X     X
 11                                                        X              X
 12  Z                 Z           Z  Z
 13  Z              Z
 14        Z           Z     Z        Z  Z  Z
 15        Z              Z
 16              Z           Z              Z  Z
 17              Z              Z
 18                                Z                 Z           Z  Z
 19                                Z              Z
 20                                      Z           Z     Z        Z  Z  Z
 21                                      Z              Z
 22                                            Z           Z              Z  Z
 23                                            Z              Z

Step 2 (CNOT West):
cnot(1,0)
cnot(11,10)
cnot(21,20)
cnot(3,2)
cnot(13,12)
cnot(23,22)
cnot(6,7)
cnot(16,17)
cnot(8,9)
cnot(18,19)
After step 2:
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X        X  X
  1  X  X
  2        X     X        X  X
  3        X  X
  4                    X  X        X     X        X  X
  5                    X  X        X  X
  6                          X  X        X     X        X  X
  7                          X  X        X  X
  8                                                  X  X        X     X
  9                                                  X  X        X  X
 10                                                        X  X        X     X
 11                                                        X  X        X  X
 12  Z  Z              Z           Z
 13  Z  Z           Z
 14        Z  Z        Z     Z        Z  Z
 15        Z  Z        Z  Z
 16              Z           Z              Z  Z
 17              Z           Z  Z
 18                                Z  Z              Z           Z
 19                                Z  Z           Z
 20                                      Z  Z        Z     Z        Z  Z
 21                                      Z  Z        Z  Z
 22                                            Z           Z              Z  Z
 23                                            Z           Z  Z

Step 3 (CNOT South):
cnot(1,6)
cnot(11,16)
cnot(3,8)
cnot(13,18)
cnot(10,5)
cnot(20,15)
cnot(12,7)
cnot(22,17)
cnot(14,9)
cnot(24,19)
After step 3:
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X        X  X
  1  X  X              X
  2        X     X        X  X
  3        X  X              X
  4                 X  X           X     X        X  X
  5                 X  X  X        X  X              X
  6                       X  X           X     X        X  X
  7                       X  X  X        X  X              X
  8                                               X  X           X     X
  9                                               X  X  X        X  X
 10                                                     X  X           X     X
 11                                                     X  X  X        X  X
 12  Z                 Z           Z
 13  Z  Z           Z              Z
 14     Z  Z           Z     Z        Z  Z
 15     Z  Z  Z        Z  Z              Z
 16           Z  Z           Z              Z  Z
 17           Z  Z           Z  Z              Z
 18                                Z                 Z           Z
 19                                Z  Z           Z              Z
 20                                   Z  Z           Z     Z        Z  Z
 21                                   Z  Z  Z        Z  Z              Z
 22                                         Z  Z           Z              Z  Z
 23                                         Z  Z           Z  Z              Z

Step 4 (CNOT East):
```

```
cnot(1,2)
cnot(11,12)
cnot(21,22)
cnot(3,4)
cnot(13,14)
cnot(23,24)
cnot(6,5)
cnot(16,15)
cnot(8,7)
cnot(18,17)
After step 4:
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X           X
  1  X  X  X        X  X
  2        X     X           X
  3        X  X  X        X  X
  4                    X           X     X           X
  5                    X  X        X  X  X        X  X
  6                          X           X     X           X
  7                          X  X        X  X  X        X  X
  8                                                  X           X     X
  9                                                  X  X        X  X  X
 10                                                        X           X     X
 11                                                        X  X        X  X  X
 12  Z                 Z           Z
 13  Z  Z           Z  Z           Z
 14        Z           Z     Z           Z
 15        Z  Z        Z  Z  Z        Z  Z
 16              Z           Z                 Z
 17              Z           Z  Z           Z  Z
 18                                Z                 Z           Z
 19                                Z  Z           Z  Z           Z
 20                                      Z           Z     Z           Z
 21                                      Z  Z        Z  Z  Z        Z  Z
 22                                            Z           Z                 Z
 23                                            Z           Z  Z           Z  Z

After step 4; rewrite
stab(1) = stab(0) x stab(1)
stab(3) = stab(2) x stab(3)
stab(5) = stab(4) x stab(5)
stab(7) = stab(6) x stab(7)
stab(9) = stab(8) x stab(9)
stab(11) = stab(10) x stab(11)
stab(13) = stab(12) x stab(13)
stab(15) = stab(14) x stab(15)
stab(17) = stab(16) x stab(17)
stab(19) = stab(18) x stab(19)
stab(21) = stab(20) x stab(21)
stab(23) = stab(22) x stab(23)
 ID  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  0  X     X           X
  1     X           X
  2        X     X           X
  3           X           X
  4                    X           X     X           X
  5                       X           X           X
  6                          X           X     X           X
  7                             X           X           X
  8                                                  X           X     X
  9                                                     X           X
 10                                                        X           X     X
 11                                                           X           X
 12  Z                 Z           Z
 13     Z           Z
 14        Z           Z     Z           Z
 15           Z           Z           Z
 16              Z           Z                 Z
```

```
17                   Z       Z
18                     Z           Z       Z
19                       Z       Z
20                         Z       Z   Z       Z
21                           Z       Z       Z
22                             Z       Z           Z
23                                       Z       Z
```

# References

[1] Aaronson, S., and Gottesman, D. Improved simulation of stabilizer circuits. *Physical Review A 70*, 5 (2004), 52328.

[2] Aharonov, D., and Ben-Or, M. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (1997), ACM New York, NY, USA, pp. 176–188.

[3] Ambegaokar, V., and Baratoff, A. Tunneling between superconductors. *Physical Review Letters 10*, 11 (1963), 486–489.

[4] Bardeen, J., Cooper, L. N., and Schrieffer, J. R. Theory of superconductivity. *Phys. Rev. 108*, 5 (1957), 1175–1204.

[5] Biondi, M., and Garfunkel, M. Millimeter Wave Absorption in Superconducting Aluminum. II. Calculation of the Skin Depth. *Physical Review 116*, 4 (1959), 862–867.

[6] Bourassa, J., and Blais, A. Ultra-strong coupling regime of cavity QED with flux qubits. In *American Physical Society, 2009 APS March Meeting, March 16-20, 2009, abstract# L17. 004* (2009).

[7] Bravyi, S., and Kitaev, A. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A 71*, 2 (2005), 22316.

[8] Caldeira, A., and Leggett, A. Quantum tunneling in a dissipative system. *Ann. Phys. (NY) 149* (1983), 374.

[9] Chen, G. *Quantum computing devices: principles, designs, and analysis.* Chapman & Hall/CRC, 2006.

[10] Chiorescu, I., Nakamura, Y., Harmans, C., and Mooij, J. Coherent quantum dynamics of a superconducting flux qubit. *Science 299* (2003), 1869.

[11] Cirac, J. I., and Zoller, P. Quantum computations with cold trapped ions. *Phys. Rev. Lett. 74*, 20 (1995), 4091–4094.

[12] Clarke, J., and Wilhelm, F. Superconducting qubits. *Nature 453* (2008), 1031.

[13] Cohen-Tannoudji, C., Diu, B., and Laloë, F. *Quantum Mechanics.* Wiley Interscience, Weinheim, 1992.

[14] Cook, W., and Rohe, A. *Computing minimum-weight perfect matchings.* Forschungsinst fur Diskrete Mathematik, 1997.

[15] Cory, D., Fahmy, A., and Havel, T. Ensemble quantum computing by NMR spectroscopy, 1997.

[16] de Gennes, P. *Superconductivity of metals and alloys.* Benjamin, N.Y., 1966.

[17] Dennis, E., Kitaev, A., Landahl, A., and Preskill, J. Topological quantum memory. *Journal of Mathematical Physics 43* (2002), 4452.

[18] Devoret, M., Wallraff, A., and Martinis, J. Superconducting qubits: A short review. *Arxiv preprint cond-mat/0411174* (2004).

[19] Ferber, J. Efficient creation of multipartite entanglement for superconducting quantum computers. *PhD Thesis - Munchen* (2005).

[20] Feynman, R. Simulating physics with computers. *International Journal of Theoretical Physics 21* (1981), 467.

[21] Fowler, A., and Goyal, K. Topological cluster state quantum computing. *Arxiv preprint arXiv:0805.3202* (2008).

[22] Fowler, A., Stephens, A., and Groszkowski, P. High threshold universal quantum computation on the surface code. *Arxiv preprint arXiv:0803.0272 803* (2008).

[23] Gottesman, D. Stabilizer codes and quantum error correction. *PhD Thesis - Caltech* (1997).

[24] Grover, L. Quantum mechanics helps in searching for a needle in a haystack. *Proceedings of the 28'th Annual ACM Symposium on the Theory of Computing 79* (1997), 325.

[25] Hime, T., Reichardt, P., Plourde, B., Robertson, T., Wu, C., Ustinov, A., and Clarke, J. Solid-state qubits with current-controlled coupling. *Science 314*, 5804 (2006), 1427–1429.

[26] Josephson, B. Possible new effects in superconductive tunneling. *Phys. Lett. 1*, 7 (1962), 251.

[27] Kamon, M., Tsuk, M. J., and White, J. Fasthenry: A multipole-accelerated 3-d inductance extraction program. In *Design Automation Conference* (1993), pp. 678–683.

[28] Kitaev, A. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys 52*, 6 (1997), 1191–1249.

[29] Knill, E. Quantum computing with realistically noisy devices. *Nature 434*, 7029 (2005), 39–44.

[30] Knill, E., Laflamme, R., and Milburn, G. J. A scheme for efficient quantum computation with linear optics. *Nature 409*, 6816 (2001), 46–52.

[31] Knill, E., Laflamme, R., and Zurek, W. Accuracy threshold for quantum computation. *Arxiv preprint quant-ph/9610011* (1996).

[32] Lupascu, A., Verwijs, C., Schouten, R. N., Harmans, C., and Mooij, J. E. Nondestructive readout for a superconducting flux qubit. *Phys. Rev. Lett. 93* (2004), 177006.

[33] Majer, J., Chow, J., Gambetta, J., Koch, J., Johnson, B., Schreier, J., Frunzio, L., Schuster, D., Houck, A., Wallraff, A., Blais, A., Devoret, M., Girvin, S., and Schoelkopf, R. Coupling superconducting qubits via a cavity bus. *Nature 449* (2007), 443.

[34] Martinis, J., Nam, S., Aumentado, J., and Urbina, C. Rabi oscillations in a large Josephson-junction qubit. *Phys. Rev. Lett. 89*, 11 (2002), 117901.

[35] Mooij, J., Orlando, T., Levitov, L., Tian, L., van der Wal, C., and Lloyd, S. Josephson persistent current qubit. *Science 285* (1999), 1036.

[36] Nakamura, Y., Pashkin, Y., and Tsai, J. Coherent control of macroscopic quantum states in a single-Cooper-pair box. *Nature 398* (1999), 786.

[37] Nielsen, M., and Chuang, I. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge, UK, 2000.

[38] Niskanen, A., Nakamura, Y., and Tsai, J. Tunable coupling scheme for flux qubits at the optimal point. *Phys. Rev. B 73* (2006), 094506.

[39] Onnes, H. K. The superconductivity of mercury. *Leiden Comm. 122b* (1911), 124.

[40] Orlando, T., tian, L., Crankshaw, D., Lloyd, S., van der Wal, C., Mooij, J., and Wilhelm, F. Engineering the quantum-measurement process for the persistent current qubit. *Phyisca C 368* (2003), 294.

[41] Orlando, T. P., Mooij, J. E., Tian, L., van der Wal, C., Levitov, L. S., Lloyd, S., and Mazo, J. J. Superconducting persistent-current qubit. *Phys. Rev. B 60* (1999), 15398.

[42] Paauw, F., Fedorov, A., Harmans, C., and Mooij, J. Tuning the Gap of a Superconducting Flux Qubit. *Physical Review Letters 102*, 9 (2009), 90501.

[43] Plourde, B., Zhang, J., Whaley, K., Wilhelm, F., Robertson, T., Hime, T., Linzen, S., Reichardt, P., Wu, C., and Clarke, J. Entangling flux qubits with a bipolar dynamic inductance. *Physical Review B 70*, 14 (2004), 140501.

[44] Raussendorf, R., and Harrington, J. Fault-tolerant quantum computation with high threshold in two dimensions. *Appl. Phys. Lett Phys Rev Lett 98* (2006), 190504.

[45] Raussendorf, R., Harrington, J., and Goyal, K. Topological fault-tolerance in cluster state quantum computation. *New Journal of Physics 9*, 6 (2007), 199.

[46] Reichardt, B. W. Improved magic states distillation for quantum universality. *Quantum Information Processing 4* (2005), 251.

[47] Shor, P. Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), 124.

[48] Somaroo, S., Tseng, C. H., Havel, T. F., Laflamme, R., and Cory, D. G. Quantum simulations on a quantum computer. *Phys. Rev. Lett. 82*, 26 (Jun 1999), 5381–5384.

[49] Stephens, A., and Evans, Z. Accuracy threshold for concatenated error detection in one dimension. *Arxiv preprint arXiv:0902.2658* (2009).

[50] Tinkham, M. *Introduction to Superconductivity.* McGraw-Hill, New York, 1996.

[51] van der Ploeg, S. H. W., Izmalkov, A., van den Brink, A. M., Huebner, U., Grajcar, M., Il'ichev, E., Meyer, H. G., and Zagoskin, A. M. Controllable coupling of superconducting flux qubits, 2007.

[52] van der Wal, C., ter Haar, A., Wilhelm, F., Schouten, R., C.J.P.M.Harmans, Orlando, T., Lloyd, S., and J.E.Mooij. Quantum superposition of macroscopic persistent-current states. *Science 290* (2000), 773.

[53] Vion, D., Aassime, A., Cottet, A., Joyez, P., Pothier, H., Urbina, C., Esteve, D., and Devoret, M. Manipulating the quantum state of an electrical circuit. *Science 296* (2002), 866.

[54] Wang, D. S., Fowler, A. G., Stephens, A. M., and Hollenberg, L. C. L. Threshold error rates for the toric and surface codes.

[55] Wei, L., x. Liu, Y., and Nori, F. Coupling Josephson qubits via a current-biased information bus. *Europhys. Lett. 67* (2004), 1004.

[56] Wendin, G., and Shumeiko, V. Superconducting quantum circuits, qubits and computing. *Arxiv preprint cond-mat/0508729* (2005).

[57] You, J., and Nori, F. Superconducting circuits and quantum information. *Arxiv preprint quant-ph/0601121* (2006).