The Coherent Parity Check Framework for Quantum Error Correction

JOSHUA ROFFE

The Coherent Parity Check Framework for Quantum Error Correction

Joschka Roffe

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Department of Physics
Durham University

April 29, 2019

The Coherent Parity Check Framework for Quantum Error Correction

Joschka Roffe

Abstract

Quantum error correction protocols are an essential element in the design of any circuit-model quantum computer. In this thesis, I introduce the coherent parity check (CPC) framework for quantum error correction. CPC codes have a fundamental structure in which quantum parity check measurements are stored coherently and compared over time. The specific advantage of the CPC code structure is that it provides a way of creating new stabilizer codes from the starting point of any sequence of parity checks. I show that this freedom in the choice of parity checks can be used to derive methods for the construction of distance-three quantum codes based on almost any distance-three classical code. The CPC framework has further applications in machine search routines for code discovery, as well as in the design of bespoke codes tailored for the demands of a given device. Another feature of CPC codes is that they can be represented as factor graphs of the type commonly seen in classical error correction and machine learning. I outline a procedure for this mapping, and demonstrate how a quantum code can be derived by manipulating its factor graph representation. The aim of the factor graph mapping for CPC codes is to make it easier to adapt well-developed techniques from classical information theory for use with quantum codes. This will make the CPC framework a useful tool for the theoretical and practical study of quantum error correction codes as large-scale quantum computers move closer to becoming a reality.

Contents

Acknowledgements

This thesis has been three years in the making. And what a three years it has been! I have thoroughly enjoyed my time studying at Durham amongst so many brilliant colleagues and friends. Without them, this work would certainly not have been possible.

First and foremost, I would like to thank my supervisor Viv Kendon for securing the funding for me to study at Durham. You have been an excellent supervisor, providing invaluable guidance at every stage of my PhD. Thank you for your many ideas and suggestions, as well as for encouraging me to pursue my interests.

I have also had the pleasure of being co-supervised by Dominic Horsman and Nicholas Chancellor. Thank you Dom for introducing me to the field of quantum error correction and proposing the project that eventually became the main topic of this thesis. Many thanks to Nick for always taking time to explain new concepts to me, as well as for suggesting various fruitful avenues for research. I wish both of you the best of luck as you start your new groups in Grenoble and Durham, and look forward to collaborating in the future.

I have enjoyed productive collaborations during my PhD. Many thanks to Stefan Zohren for hosting me in Oxford for a week to discuss CPC codes and other research ideas. Thanks also to Aleks Kissinger for providing an alternative way of representing CPC codes using the ZX-calculus. Finally, I am grateful to have been able to work with David Headley on circuit compilation problems.

During my time at Durham I have been part of the Atomic and Molecular research group (recently re-branded to Quantum Light and Matter). Many thanks to everyone

I have shared an office with, in particular Ben Beswick and Rob Bettles, who have ensured a pleasant working environment from day one. I have fond memories of doing the ATMOL graduate course with the other members of my cohort: Will Hamlyn, Ryan Hanley, Prosenjit Majumder, Ginny Marshall, Mew Ratkata, Dominic Reed and Oliver Wales. Other highlights of my time in ATMOL have included the daily coffee breaks which have involved many entertaining conversations about physics, football, politics and all the other important issues of our times. I must also mention the weekly Friday Evening Seminar, and thank all those involved in its organisation.

I would like to thank the people of Durham City for providing such a friendly environment to live in. After my three years here, I agree wholeheartedly with Bill Bryson's assertion that Durham is the best little city in the world with its unrivalled setting and architecture. Also unrivalled is Durham's community spirit, as exemplified by events such as the annual Big Meeting. Special thanks must go to the members of Durham City Vélo Cycling Club, with whom I have spent hundreds of hours exploring Durham and its beautiful environs. There is no better way to explore a region than on a bike!

My time in Durham would not have been the same without all the friends I have made outside of the Physics Department. Thanks in particular to my housemate Dominic Charrier, as well as everyone I have met through Ustinov College. Finally I would like to thank my mother Christine, my father David and my brother Theo, for providing support and encouragement throughout every stage of my education.

Joschka Roffe
April 29, 2019

Declaration

I confirm that no part of the material offered has previously been submitted by myself for a degree in this or any other university. Where material has been generated through joint work, the contribution of others has been indicated.

Joschka Roffe
Durham, April 29, 2019

Publications

I carried out the work for this thesis under the supervision of Viv Kendon, Dominic Horsman and Nicholas Chancellor. The material covered in this thesis is based on the following three publications and preprints:

- [1] Nicholas Chancellor, Aleks Kissinger, Joschka Roffe, Stefan Zohren, and Dominic Horsman. Graphical structures for design and verification of quantum error correction. *arXiv:1611.08012*, 2016.
- [2] Joschka Roffe, David Headley, Nicholas Chancellor, Dominic Horsman, and Viv Kendon. Protecting quantum memories using coherent parity check codes. *Quantum Science and Technology, 3(3):035010*, 2018.
- [3] Joschka Roffe, Stefan Zohren, Dominic Horsman, and Nicholas Chancellor. Quantum codes from classical graphical models. *arXiv:1804.07653*, 2018.

Chapter 4 is based on work carried out in [1] and [2]. The CPC framework was first introduced in these papers using two different methods: [1] uses techniques from the ZX-calculus, whereas [2] uses conventional quantum circuit notation. The ZX-calculus presentation was predominantly developed by Dominic Horsman and Aleks Kissinger. I outlined an alternative presentation using conventional circuit notation in [2]. As such, I adopt the quantum-circuit notation in this thesis. Sections 4.5 and 4.6 cover the tripartite structure for CPC codes and their relation to CSS codes. This work extends and improves upon the results originally developed by Dominic Horsman and Aleks Kissinger in [1].

The results outlined in chapter 5 were originally published in [2]. I carried out all the work for this experiment.

Chapter 6 is based on results published in [2]. I carried out the work for this chapter in collaboration with David Headley.

Chapter 7 is based on work I carried out in [2]. I collaborated with David Headley to develop the SWAP gate compilation strategies outlined in section 7.2.3.

Chapter 8 is based on work carried out in [3]. I carried out this work jointly with Nicholas Chancellor and Stefan Zohren.

Section 9.2.2 in chapter 8.6 outlines a method for the implementing a maximum entropy decoding technique on a CPC code. This is based on work currently in preparation [4].

Notation

Throughout the thesis I attempt to adhere to the following convention of notation:

Pauli matrices

$$\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{0.0.1}$$

For an overview of the Pauli group, see appendix A.

The binary addition operator

The '$\oplus$' operator is used to represent addition modulo 2. eg.

$$1 \oplus 1 = (1 + 1) \bmod 2 = 0 \tag{0.0.2}$$

This notation will be used for all the binary arithmetic in this thesis.

Quantum circuit notation

We adopt standard quantum circuit notation as commonly presented in the literature, for example as seen in [5] or [6]. Any quantum gates that are unique to this work are defined in the text.

Chapter 1

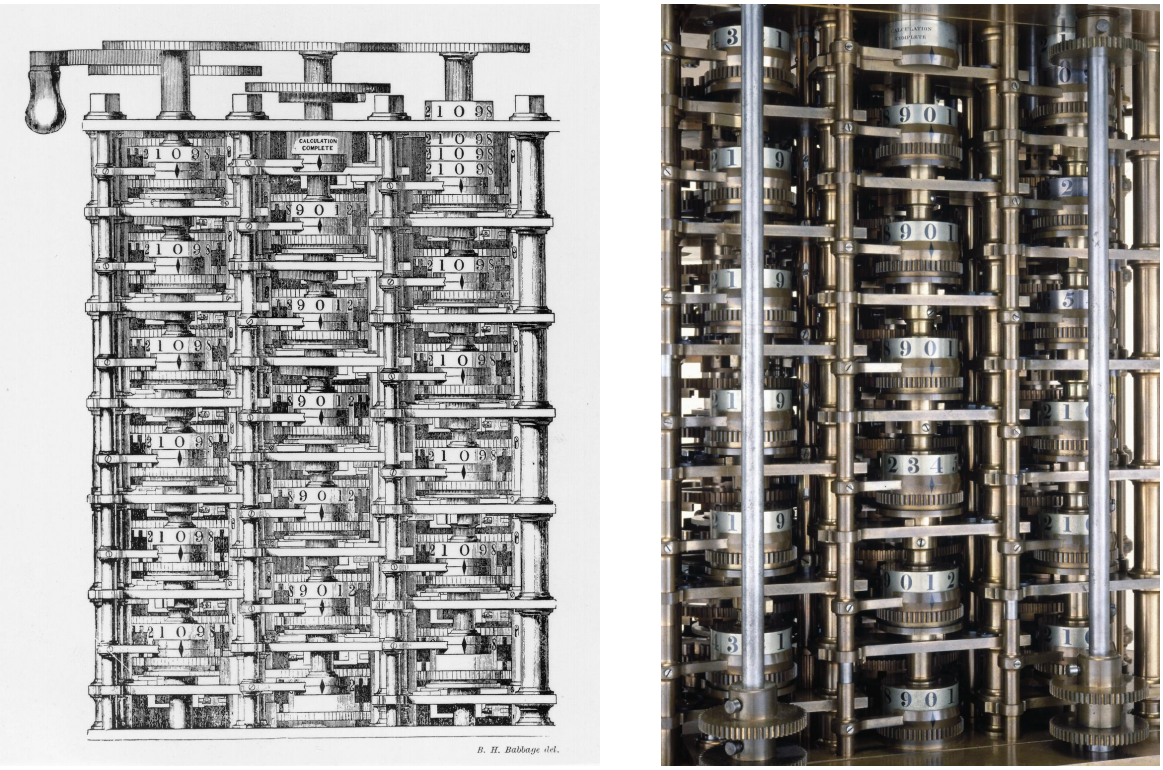
Introduction

1.1 Computing through the ages

The development of the digital computer is the defining technological advancement of the past century. Whilst the industrial revolution of the Victorian era mechanised many aspects of manufacturing, agriculture and transport, the modern computer revolution has mechanised the storage, manipulation and distribution of information. Computers now permeate every part of our lives; they assume an indispensable role across all aspects of industry, business, science and media. Furthermore, the spread of the Internet has left us more connected than ever before. And our dependence on computers is set to rise: the impact of advanced computing methods such as machine learning and artificial intelligence is ever increasing. This is apparent, for example, in the development of technologies such as self-driving cars [7].

Primitive computing devices have existed for millennia. The most widely adopted of these early devices was the abacus, the first reported use of which dates back to the period 2700-2300BC [8]. Remarkably, the abacus remained the leading calculating device for millennia. In fact, it was not until the discovery of logarithms, and the invention of the slide rule in the 1620s [9], that its performance was superseded.

The modern era of computing can trace its roots to the nineteenth century engineer



(a) (b)

Figure 1.1: Left: Charles Babbage's plan for the Difference Engine. Right: A modern replica of the Difference Engine built for the London Science Museum showing the gears with which the decimal numbers are realised. *Image source (for both images): The Board of Trustees of the Science Museum, London. Image released under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 Licence.*

Charles Babbage, who proposed a machine to automate the calculation of artillery tables [10, 11]. This device, known as the Analytical Engine, incorporated all of the fundamental elements of a modern computer: an input, a memory, a processor and an output. In contrast to previous calculators, which required a human operator, the Analytical Engine was designed to run automatically from a set of instructions known as a program. The plans of the Analytical Engine were studied by the mathematician Ada Lovelace, who was the first to realise that the device might have applications beyond artillery table calculation [12]. As an example, Lovelace wrote an algorithm for the Analytical Engine to calculate Bernoulli numbers [13], and in doing so, became

the world's first computer scientist.[a]

Unfortunately, Babbage never succeeded in building a full-scale Analytical Engine. In his first attempt, after securing funding from the British government in 1823, he began the construction of a prototype, known as the Difference Engine [11]. This device, like the Analytical Engine, operated using decimal digital logic (numbers 0 to 9) [15]. From a human perspective, this was a sensible choice, as we are familiar with the decimal number system from our everyday lives. However, from a technological perspective, decimal systems can be difficult to implement. In Babbage's devices, the decimal numbers were realised as the positions of a mechanical gear [10]. A challenge with this approach was that any under (or over) rotation of the gear could result in the desired decimal number being misrepresented, leading to errors which would compromise the computation. This problem was compounded by the fact that the Difference Engine required thousands of gears, each of which had to milled to the utmost precision to prevent the occurrence of errors. This ultimately proved to be an insurmountable engineering challenge[b], and led to the failure of the project. Babbage's proposed device was decades ahead of its time, but the technology available to him could not match his vision.

The first computer algorithm outlined by Lovelace was special purpose, designed specifically to be run on the Analytical Engine hardware. Nearly a century later, in 1935, the notion of a universal computing device, capable of running any conceivable algorithm, was proposed by Alan Turing [16]. Such devices, now referred to as Turing machines, were first studied as abstract mathematical models. However, in the Second World War, prototype Turing machines were soon built to perform calculations for military research, most notably for the Enigma code breaking effort and the Manhattan Project [17]. During this time, the Hungarian physicist John Von Neumann proposed a design for a Turing Machine that would become the basis of computing architectures to the

[a] Ada Lovelace was the daughter of the Anne Isabella Milbanke and the poet Lord Byron. Ada's mother's family were based in Seaham, a seaside town in County Durham, UK. Given her contributions to the field of computer science, and her connection to the local area, it has been proposed that 'Ada Lovelace College' would be good name for the 17th Durham College, which is currently under construction. For more details see this `change.org` campaign [14]

[b] This problem was not helped by the fact that Babbage had a notoriously bad working relationship with his chief engineer Joseph Clements.

modern day [18]. Such Von Neumann architectures were designed to be arbitrarily reprogrammable via stored programs. In contrast, earlier devices required physical hardware intervention – rewiring etc – every time the algorithm was to be changed.

The first example of a computer based on the Von Neumann architecture was the 1948 Manchester Baby, built by Frederic Williams, Tom Kilburn, and Geoff Tootill [19]. The Manchester Baby employed binary digital logic (numbers 0 or 1). The advantage of the binary basis for computation is that there are only two states to differentiate between. Crucially, these binary basis states can easily be realised as the *on/off* positions of a switch. Switches are easy systems to engineer: compare, for example, the gears in Babbage's Difference Engine, which needed to be able to differentiate between ten intermediate positions in order to realise decimal numbers. As such, computers based on binary logic can be much more reliable.

In the Manchester Baby device, the binary switches were realised using vacuum tubes. These were bulky, power hungry components that would often break down. However, the invention of the transistor (by John Bardeen, Walter Brattain and William Shockley [20]), soon provided a much more robust and efficient way of realising binary logic. The first of these transistor-based computers was built in 1953 at the University of Manchester by the same research group who built the Manchester Baby [21].

Computer development to the modern day has involved the successive miniaturisation of the transistor-based computer, first realised in Manchester, and inspired by the initial blueprints set out by Turing and Von Neumann. In the simplest terms, the 'power' of a computer can be related to the density of transistors in its processing unit. In 1965, Gordon Moore, a co-founder of the Intel Corporation, wrote a paper predicting that transistor densities would double approximately every two years [22]. This observation, now known as Moore's Law, has since become the de-facto yardstick for monitoring improvements in computing power.

Over the past fifty years, advances in micro-computer technology have roughly followed the trend predicted by Moore. Current state-of-the-art (as of October 2018) computer chips have transistors as small as 7 nm. A 1 nm transistor has been successfully demonstrated in a laboratory [23]. However, Moore scaling cannot continue indefinitely,

as eventually this would lead to transistors at the subatomic scale. This problem is recognised, with the manufacture of micro-computer chips moving increasingly to parallel architectures to gain a computational advantage. With the effect of Moore's Law plateauing, it is unlikely we will see further gains in computing power on the scale we have experienced over the past half-century. To replicate such advances in the next century, we will need a new paradigm in computing.

1.2 Quantum computing

Moore scaling has allowed the development of modern computers capable of performing large simulations and detailed real-time graphics rendering. However, there remain certain classes of problem that are unlikely to ever be efficiently solvable using current computing technology. As an example, consider the problem of simulating the dynamics of a multi-body quantum system. The wavefunction for a simple two-level system, ϕ, is parametrised by a vector containing two complex numbers, $\phi = (\alpha_1, \alpha_2)^T$, making it trivial to simulate. However, as the the number of quantum systems, n, in the simulation is increased the size of the state space scales 2^n such that $\phi_n = (\alpha_1, \alpha_2, \alpha_3, ..., \alpha_{2^n})^T$. As a result, efficiently simulating large ensembles of quantum particles on conventional computing hardware is impractical in real time.

Speaking at a keynote address in 1981 on the topic of 'Simulating physics with computers' [24], Richard Feynman argued that accurately modelling quantum systems would require machinery from the quantum world: his suggestion was that the exponentially scaling nature of quantum systems should be seen not as a hindrance, but as an opportunity to develop a new paradigm in computing based on quantum principles. The new generation of quantum computers he proposed would replace the bits in a conventional computer with quantum bits (now universally referred to as qubits), allowing vastly greater quantities of information to be encoded within the resultant state space of the vector.

To contextualise the incentives for computing in the quantum realm, it is instructive to compare classical bits with qubits [5, 6]. We will first consider a classical input state,

b_{in}, consisting of a single bit initialised deterministically in one of the two states of the computational basis

$$\mathcal{B} = \{0, 1\}, \tag{1.2.1}$$

where 0 represents the 'off' state and 1 the 'on' state. A classical computation, C, is implemented by performing a gate operation

$$b_{\text{in}} \xrightarrow{\text{Classical computation: } C} b_{\text{out}}, \tag{1.2.2}$$

where the output state, b_{out}, is also from the set $\mathcal{B}$. The key difference for a qubit, $|\psi\rangle$, is that it can be initialised as a quantum superposition of both basis states

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{1.2.3}$$

where the probability amplitudes, α and β, are complex numbers that satisfy the relation $|\alpha|^2 + |\beta|^2 = 1$. A quantum computation can then be performed on both basis states of the qubit in parallel by evolving the wavefunction via a unitary operation, U, as shown below

$$|\psi\rangle \xrightarrow{\text{Quantum computation: } U} U\,|\psi\rangle = \alpha\,U\,|0\rangle + \beta\,U\,|1\rangle\,. \tag{1.2.4}$$

For a single qubit, the Hilbert space available for computation is therefore of dimension $\dim(\mathcal{H}_2) = 2$, compared to a single classical bit where $\dim(\mathcal{B}) = 1$.

As the number of qubits in the register is increased, the size of the Hilbert space grows $\dim(\mathcal{H}_n) = 2^n$. At first glance, it may seem that this provides quantum computers with access to exponentially scaling parallelism. However, there is a catch. Whilst unitary operations do transform all basis states in parallel, the output of the computation is also a quantum state. Extracting information from a quantum state requires us to perform a basis measurement, which will cause the wavefunction to collapse to a single state of the chosen basis. The challenge in designing quantum algorithms therefore lies in finding ways to exploit additional quantum phenomena, such as interference and entanglement, to ensure that the eventual measurement collapses to the desired state with high probability [6].

The first useful quantum algorithm to promise a definitive advantage over the best-known classical techniques was developed by Peter Shor in 1994. Shor's algorithm uses a technique called the quantum Fourier transform to provide an efficient method for finding the prime factors of large integers [25]. The publication of Shor's algorithm was particularly significant in that it posed a threat to the widely used RSA cryptography protocol which relies upon the assumed intractability of factoring large numbers [26]. As the first practical application for quantum computation, the Shor algorithm is often credited for initiating the world-wide research effort to build a working quantum device.

1.3 Quantum error correction

As mentioned previously, the bits in a conventional (classical) computer are realised as the *on/off* states of a transistor switch. The two states of a transistor are differentiated by billions of electrons, providing the encoded bits with an inbuilt redundancy that all-but-eradicates logical errors. As such, any frustrations with modern computers are almost always due to software errors rather than physical errors (assuming the hardware is well looked after!).

Unfortunately, the situation is not so reassuring for quantum computers. To date, qubits have been realised in a range of experimental settings. These include trapped ions [27, 28, 29], superconducting circuits [30, 31, 32], photonic systems and spins in semiconductors [33, 34, 35] amongst others (for a review of approaches to experimental quantum computing see [36]). However, a challenge faced by all of these technologies is that the qubits are realised as single quantum systems: there is no inbuilt redundancy as is the case with the transistor switches in classical devices. Furthermore, quantum systems are extremely difficult to control, with even the slightest environmental disturbance sufficient to introduce decoherence. For circuit-model quantum computers, qubit errors are therefore not only disastrous for the computation, but also inevitable.

At first, it was thought that qubit errors would pose an insurmountable challenge to the realisation of a working quantum computer; as with Babbage's Analytical Engine, the theoretical idea was beyond the reach of the available hardware at the time. However,

unlike Babbage, physicists working on early quantum computing theory had access to an established field of expertise in classical error correction. First formalised by Claude Shannon in 1948 [37], classical coding theory allows information to be communicated and processed using imperfect hardware. Classical coding protocols can be thought of as a first layer of software for the device, and are designed to detect and correct errors when they occur. Driven by the demands of high-performance communication networks, these methods have been extensively optimised [38, 39].

Implementing error correction in a quantum setting is not straightforward. A major hurdle is the No-Cloning theorem, which prevents quantum data from being arbitrarily duplicated [40]. However, a breakthrough was reached in 1995 by Peter Shor, who published a paper proposing a method for quantum error correction [41]. His protocol involved distributing the information of a single-qubit across an entangled multi-qubit space. This method allowed for redundant encoding without cloning, and afforded the system extra degrees of freedom that can be exploited to detect and correct errors. With this result, the field of quantum computing moved from a theoretical curiosity to a practical possibility.

In the following years, Shor's work was expanded and a variety of quantum error correction codes were proposed [42, 43]. Furthermore, it was soon shown that quantum codes can be designed in a fault tolerant manner so that errors do not propagate uncontrollably through the circuit: this is an essential requirement for the realisation of quantum error correction codes on real hardware [44, 45, 46]. In [47, 48], stabilizer methods were introduced by Daniel Gottesman as a general framework for the construction of quantum error correction codes. Stabilizer codes are described in terms of non-disturbing operators on the encoded quantum information, and have the advantage that they can be efficiently classically simulated (subject to certain constraints on the types of quantum computation being simulated) [49, 50, 51]. These early developments in quantum error correction eventually led to the proof of the threshold theorem, which states that a quantum computer can be arbitrarily scaled provided the error rates of the individual components remain below a certain level [52, 53, 54, 55].

Currently, the most widely pursued error correction protocol for experiment is the surface code [56, 57]. This falls within a family of stabilizer codes that take advantage

of topological methods to encode information [58]. The specific advantage of the surface code is its high threshold (of approx. 1% [59]), combined with the fact that it only requires nearest-neighbour interactions between qubits. However, there are drawbacks, most notably the fact that it has poor encoding density. A minimum of thirteen physical qubits are required per logical qubit in a surface code [60]. In practice, however, it is thought this number will actually be of the order of thousands of qubits when a surface code is implemented for the first time [57]. Recently, efforts have been made to develop quantum codes with higher encoding densities, inspired in part by classical Low Density Parity Check (LDPC) codes [61, 62, 63]. However, these newer protocols often come with complications, such as the requirement for long range interactions between qubits.

Another hurdle to be overcome for fault tolerant quantum computation is finding a way to realise universal encoded quantum logic. For many codes, it is possible to implement a subset of encoded quantum gates via transversal application of gates in a qubit-wise manner between code blocks. However, a no-go theorem exists that prohibits the implementation of a full universal gate set in this way on a quantum computer [64]. As such, alternative techniques are required to perform universal encoded logic. Various methods have been proposed [65, 66, 67, 68, 69], but these typically impose a high cost in terms of the amount of additional qubits required. For example, the surface code can realise a universal gate set using a technique called magic state injection. However, estimates suggest that the preparation of such magic states could consume as much as 90% of physical qubits in a quantum computer [57].

In the quest to build a circuit model quantum computer there are many challenges to be overcome. At the hardware level, the methods for the realisation and control of qubits need to be improved. In addition to this, a major theoretical challenge lies in finding better ways to achieve fault tolerance. These two problems are being approached in parallel, with advances in either influencing the direction of the other.

1.4 The coherent parity check (CPC) framework

The study of many modern schemes for quantum error correction relies upon a detailed understanding of advanced mathematical themes such as group theory and topology. As such, the development of methods for quantum error correction has often occurred in isolation from the wide body of existing expertise in classical coding theory. In response to this concern, the question we set out to answer when I began my PhD was:

How can we make better contact between classical and quantum coding theory?

Our solution has involved the development of the new formalism for quantum error correction that we call the coherent parity check (CPC) framework. First introduced in our papers [1, 2], the CPC framework is built around a fundamental gadget that allows any sequence of parity checks to be turned into a stabilizer code. We will see that this freedom in the choice of parity checks affords the CPC framework many advantages over existing methods for code design, in particular with regard to adapting classical techniques for quantum codes.

As an example, the CPC framework provides a direct way of re-purposing distance-three classical codes for use in quantum error correction. This is possible via simple structural templates that separately interpret the classical codes as the parity checking sequences for the bit- and phase-checking stages of the CPC code respectively [1, 2]. In contrast to other methods [43], CPC codes impose comparatively few restrictions on the form of classical codes that can be used.

The fact that the CPC framework allows any sequence of parity checks to be turned into a stabilizer code essentially takes away the quantum mechanics from the code design process. In [3], we showed how this observation can be leveraged to develop a mapping that allows CPC codes to be represented as a factor graph of the type commonly seen in classical error correction and machine learning. Once this mapping has been performed, the original quantum code is in a form which allows existing classical techniques to be readily applied. This has potential applications for the design or decoding of quantum codes using well-developed classical techniques.

1.5 Thesis structure

In this thesis, I provide a detailed outline of the CPC framework for quantum error correction, first introduced in [1, 2], and further developed in [3]. The content of each chapter in this thesis is outlined below.

Chapters 2 and 3 cover the necessary background in classical and quantum coding theory required to understand the CPC framework. Where possible, I have tried to introduce concepts in the simplest possible way, for example, by introducing coding concepts using detection codes rather than full correction codes. Prior knowledge of linear algebra, elementary quantum mechanics and the circuit model for quantum computation is assumed. For those unfamiliar with this background, 'An Introduction to Quantum Physics' by David Griffiths is a good resource for learning basic quantum mechanics [70], and David McMahon's 'Quantum Computing Explained' is an accessible introduction to both linear algebra and the circuit model for quantum computing [5].

Chapter 4 introduces the CPC framework, starting with an outline of the operation of the fundamental CPC gadget. Following this, I explain how CPC codes can be constructed by combining multiple such gadgets via a template called the canonical structure for CPC codes. Methods for calculating stabilizers and the logical Pauli operators of a CPC code are then described. The chapter ends with descriptions of various methods for the construction of distance-three CPC codes based on classical codes. This includes a CPC method for the construction of the well-known class of Calderbank, Shor and Steane (CSS) codes.

In chapter 5, I outline the methodology and results of an experiment we ran to test $[[4, 2, 2]]$ CPC codes on the IBM 5Q device. Our results show that syndromes of such a code can be used to improve the fidelity of the circuit output, relative to the case where the syndromes are ignored. This is an interesting result, as it demonstrates the benefits of a quantum error detection protocol on an existing device with high error rates.

Chapter 6 outlines how CPC codes can be represented in terms of any realistic max-

imally entangling native gate. The theoretical study of quantum error correction circuits usually involves the use of idealised gates such as CNOT gates. However, the native gates of a given quantum computing technology will typically be of a different form. Our result allows CPC code design to be performed directly with the device's native gate. This will facilitate the process of translating quantum codes from their theoretical representation to the hardware compiled.

In chapter 7, I explore how the CPC framework can be used to set up machine search routines for code discovery. As a proof-of-concept example, I show how exhaustive search techniques can be used to discover a large set of $[[7, 3, 3]]$ codes. I then explain how the discovered set can be searched to find the code that best matches the requirements of an idealised seven-qubit ion trap. Such a design approach, combined with our native gates result outlined in chapter 6, demonstrates that the CPC framework is a useful tool for the construction of bespoke quantum error correction codes tailored to the needs of a given device.

In chapter 8, I explain how CPC codes have an equivalent representation as classical factor graphs. This mapping is achieved via an intermediate graphical language called the operational representation. The role of the operational representation is to take away any quantum-mechanical behaviour of the code, such as indirect propagation, so that it can be represented as a classical code. Once mapped to the factor graph, further modification of the CPC code can proceed using classical intuition and techniques. As an example, I demonstrate how a $[[5, 1, 3]]$ CPC code can be constructed by modifying the factor graph representation of a CPC detection code.

In chapter 8.6, I summarise the main results of the thesis. The thesis concludes with ideas for future work to extend the CPC framework and its applications. This includes a discussion of the potential links between the CPC framework and flag syndrome extraction techniques. I also discuss how the CPC factor graph mapping could be used as a tool to enable a classical decoding technique to be applied to a quantum code.

Chapter 2

Classical Error Correction

Information is stored, communicated and processed using physical systems that are susceptible to error. Modern processors operate with negligibly small error rates. However, the same cannot said for storage [71] and communication protocols [72]. Error mitigation strategies therefore play a vital role in ensuring reliable operation of such information technologies [73]. The field of information theory deals with the development, analysis and benchmarking of error mitigation techniques. In this chapter, we provide a brief overview of the construction of classical error correction codes which enable information to be reliably stored and communicated using fault-prone hardware. The material in this chapter is designed to serve as background for the understanding of quantum error correction codes and the eventual presentation of the coherent parity check (CPC) framework, the main topic of this thesis.

To set the scene, consider a simple situation in which we wish to store a bit of information for a set amount of time t. Possible techniques for achieving this include writing the bit-value on a piece of paper, or saving it on an electronic storage device such as a hard drive. However, for any such data storage method, there will be a finite probability p_t that the device suffers a fault that will corrupt the data. For example, in the case where the bit-value is written on a piece of paper, the ink could smudge during the storage time causing its value to be misread.

In information theory, scenarios such as the one above are considered in terms of the

communication of a message (the initial data) over a noisy channel[a]. The channel describes the probability of the constituent message bits being subject to an error. An example of an error channel (and the one we will use throughout this chapter) is the *binary symmetric channel*, which stipulates that each bit is flipped with probability p_f. The mathematical action of the binary symmetric channel, for a single input bit b_{in}, is described as follows

$$b_{\text{in}} \xrightarrow{\text{binary symmetric channel}} b_{\text{out}} = \begin{cases} b_{\text{in}}, & \text{with probability } 1 - p_f, \\ b_{\text{in}} \oplus 1, & \text{with probability } p_f \end{cases} , \tag{2.0.1}$$

where b_{out} is the output bit [73]. In general, a message B_{in} will have the form $B_{\text{in}} = b_1 b_2 ... b_n$ where each b_i is subject to the mapping in equation 2.0.1. In the the design of any error correction protocol, the goal is to find a recovery operation $\mathcal{R}$ so that that $B_{\text{in}} = \mathcal{R} \cdot B_{\text{out}}$ with high probability. The hardware solution to this challenge is to improve the quality of the physical device so that p_f becomes negligibly small and no recovery operation is needed (ie. $\mathcal{R} = \mathbb{1}$). However, in practice, this is not always possible. Error correction protocols provide a systems approach to the problem by enabling reliable transmission using faulty hardware. The fundamental principle underpinning such an approach is that the message is redundantly encoded, so that the amount of resources used to transmit it over the noisy channel is increased beyond the theoretical minimum. In the following sections we explain how redundant encoding allows for errors to be detected and corrected.

2.1 Redundant encoding & repetition codes

Redundant encoding involves distributing the information content of an initial message across an expanded space of bits. The exact way in which this is achieved is specified by a set of instructions referred to as an *error correction code*. One of the simplest examples is the three-bit repetition code which duplicates each bit in the initial message

[a]Note that 'communication' in this context does not exclusively refer to the movement of information from one place to another. In a storage device, for example, the information is physically static. In this case, the message can be thought of as being communicated over time. Similarly, this is also the case for computation.

three times. The action of the encoder for a three-bit repetition code on each bit b_i in the initial message is as follows

$$b_i \xrightarrow{\text{encoder}} \mathbf{b_L} = b_i b_i b_i. \tag{2.1.1}$$

Following the encoding, the resultant three-bit string $\mathbf{b_L} = b_i b_i b_i$ is called a logical codeword. For the three-bit code, the two possible logical codewords are

$$\mathcal{B}_L = \{0_L, 1_L\} = \{000, 111\} \tag{2.1.2}$$

The logical codeword is passed through the noisy channel, after which it can be decoded using a majority vote by the receiver. For example, if the initial bit value is set to $b = 0$ then the resultant codeword is $\mathbf{b_L} = 000$. In this case, the receiver will be able to correctly decode the message if a single-error occurs on any of the bits. For example, if the received logical codeword is read as $\mathbf{b_{L'}} = 010$, the receiver can deduce from a majority vote that the initial bit-value was $b = 0$. However, if errors occur on two or more of the bits, the majority vote will not correctly decode the codeword.

When a raw bit is transmitted through a binary symmetric channel, the logical error rate (ie. the probability of corrupted transmission) is $p_L = p_f$. If the message is encoded with the three-bit code, however, incorrect transmission will only occur if two or more bits are errored meaning the logical error rate becomes $p_L \approx p_f^2$ to leading order (assuming that the probability p_f is small). Therefore, provided the additional resources (in terms of bit number and time) are available, encoding via the three-bit code works as a strategy for increasing the reliability of transmission.

In this thesis error correction codes are labelled using the $[n, k, d]$ notation, where n is the total number of bits (also referred to as the 'block length') and k is the number of encoded (logical) bits. The code distance d is defined as the minimum weight of an error required to change one codeword into another [74]. The code distance is related to the maximum number of correctable errors t via the relation

$$d = 2t + 1. \tag{2.1.3}$$

Under this notation, the three-bit repetition code is labelled $[n = 3, k = 1, d = 3]$ as

each bit of information is distributed across three bits in total. The code distance is $d = 3$, because three bit-flips are required to change $0_L = 000$ into $1_L = 111$.

We have now shown how the three-bit repetition code can suppress the logical error rate for transmission through a binary symmetric channel from $p_L = p_f$ to $p_L \approx p_f^2$ to leading order. For repetition codes, the logical error rate can be further suppressed simply by increasing the number of duplications in the initial encoding. The $[5, 1, 5]$ repetition code, for example, is capable of correcting up to $t = 2$ errors (calculated using equation 2.1.3) and has a logical error rate of $p_L \approx p_f^3$ to leading order. In principle, the logical error rate of the protocol can be arbitrarily reduced by increasing the length of the repetition code. However, this appraoch comes at the expense of the transmission rate, R, defined as the ratio of logical bits to physical bits

$$R = \frac{k}{n}. \tag{2.1.4}$$

A general n-bit repetition code has the parameters $[n, k = 1, d = n]$, resulting in a transmission rate $R_{\text{rep}} = 1/n$. Decreasing the logical error rate of the code (by increasing the value of n) therefore results in a reduction in the transmission rate. Because of this diminishing transmission rate, repetition codes are not commonly used in practice. Instead, most classical error correction protocols are based on a family of codes known as parity check codes. Parity check codes can exceed the performance of repetition codes – in terms of both transmission rate and logical error rate – and are introduced in the following section.

2.2 Parity check codes

Repetition codes encode a single bit per logical block so that $k = 1$, resulting in a vanishing rate $R_{\text{rep}} = 1/n$ as the code length is increased. In contrast, parity check codes encode two or more logical states per block so that $k \geq 2$. Rather than duplicating the information in each bit, as is done for repetition codes, parity check codes work by measuring correlations between the code's bits and keeping track of them over time [73, 74]. It is therefore possible to construct parity-check codes with better trans-

mission rate scaling than repetition codes. We now outline the general operational principles for such parity check codes [73, 74].

The bits in a parity check code are partitioned into two types: data bits and parity bits. The role of the parity bits is to store the results of parity check measurements on the data bits. We now outline the simplest possible example of a parity check code. We start by considering a two-bit data register $\mathbf{r} = d_1 d_2$, where d_1 and d_2 are the data bits. In the encode stage of the parity check cycle, an extra bit is introduced to measure and store the parity check of the data bits, which is calculated as the binary sum $q = d_1 \oplus d_2$. The resultant three-bit codeword is then given by $\mathbf{c} = d_1 d_2 q$, giving four possible logical states

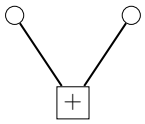
$$\mathcal{B}_L = \{00_L, 01_L, 10_L, 11_L\} = \{000, 011, 101, 110\}. \tag{2.2.1}$$

A single error on any of the three bits can be detected by comparing the value of the parity checks at times $q(t_0)$ and $q(t_1)$ before and after transmission through the error channel. A single bit-flip error occurring during transmission will cause the value of the parity to change so that $q(t_0) \neq q(t_1)$. Under the $[n, k, d]$ labelling convention, this parity check cycle is a $[3, 2, 2]$ code. Note that the distance is $d = 2$ as two bit-flips are required to change one codeword into another. Using equation 2.1.3, we can see that the number of errors that can be corrected with a $[3, 2, 2]$ code is $t = 0$. This means that the $[3, 2, 2]$ code is a detection code, rather than a full correction code. Parity check codes that can both detect and localise errors (ie. with distance $d \geq 3$) require multiple overlapping parity checks. We will explore examples of such constructions later in this chapter.

2.3 Factor graphs and the generator matrix formalism for classical codes

A factor graph is a tool designed to provide a visualisation of the relationship between the data and parity bits in a given classical code. The factor graph for the $[3, 2, 2]$

detection code, outlined in the previous section, is shown to the left below



$$, \quad A_{[3,2,2]} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \tag{2.3.1}$$

where the round nodes represent the data bits and the crossed square nodes the parity bits [73]. The edges indicate which bits are involved in each check. The adjacency matrix corresponding to this factor graph is given by $A_{[3,2,2]}$, where the rows represent the data bits and the columns the parity bits. For any parity check code, we can define a generator matrix that relates the data register $\mathbf{r}$ to the codewords $\mathbf{c}$ as follows

$$\vec{\mathbf{c}} = G^T \cdot \vec{\mathbf{r}}, \tag{2.3.2}$$

where the matrix multiplication is performed modulo 2, and $\vec{\mathbf{c}}$ and $\vec{\mathbf{r}}$ are the vector representations of $\mathbf{r}$ and $\mathbf{c}$ respectively.[b] The generator matrix can be written in terms of the code's adjacency matrix as follows

$$G_{[n,k,d]} = \begin{bmatrix} \mathbb{1}_k & A_{[n,k,d]} \end{bmatrix}. \tag{2.3.3}$$

Using the above equation, the generator matrix for the $[3, 2, 2]$ code is written as

$$G_{[3,2,2]} = \begin{bmatrix} \mathbb{1}_2 & A_{[3,2,2]} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{2.3.4}$$

The codewords for $[3, 2, 2]$ code can then be calculated using equation 2.3.2 to give

$$\vec{\mathbf{c}} = G^T_{[3,2,2]} \cdot \vec{\mathbf{r}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_1 \oplus d_2 \end{bmatrix}. \tag{2.3.5}$$

Another useful matrix in the study of coding theory is the parity check matrix, which is defined as follows

$$H_{[n,k,d]} = \begin{bmatrix} A^T & \mathbb{1}_{n-k} \end{bmatrix}. \tag{2.3.6}$$

[b] Note on the notation. Here we are using bold letters to refer to strings, eg. $\mathbf{r} = 000$. The vector representation of this string is then given by $\vec{\mathbf{r}} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$.

The parity check matrix satisfies the property $H \cdot \vec{\mathbf{c}} = \mathbf{0}$ for all the codewords $\mathbf{c}$. The parity check matrix for the $[3, 2, 2]$ code is $H_{[3,2,2]} = [\ 1\ 1\ 1\ ]$. If the codeword suffers an error $\mathbf{e}$ during transmission, the received message will have the form $\vec{\mathbf{m}}' = \vec{\mathbf{c}} \oplus \vec{\mathbf{e}}$ where $\vec{\mathbf{e}}$ is the vector representation of $\mathbf{e}$. The action of the parity check matrix on the received message vector gives the following

$$H \cdot (\vec{\mathbf{c}} \oplus \vec{\mathbf{e}}) = H \cdot \vec{\mathbf{e}} = \vec{\mathbf{s}}, \tag{2.3.7}$$

where $\vec{\mathbf{s}}$ is a vector of the parity check measurements which we refer to as the syndrome. For a distance $d \geq 3$ code, each single-bit error will produce a unique syndrome.

2.4 Hamming codes

The $[3, 2, 2]$ code is a detection code as its syndrome vector conveys insufficient information to pinpoint errors. Intuitively, this is obvious, as the length of the $[3, 2, 2]$ syndrome vector is $|\vec{\mathbf{s}}_{[3,2,2]}| = 1$ meaning there are only two possible syndrome strings. It is therefore impossible to uniquely map a syndrome to each of the three possible single-bit errors that can occur in a $[3, 2, 2]$ encoding. Similar counting arguments can be made to derive a bound which dictates the maximum theoretically possible rate for an $[n, k, d]$ code. This bound is called the Hamming bound and is defined as follows

$$\sum_{t=0}^{(d-1)/2} \binom{n}{t} \leq 2^{n-k} \tag{2.4.1}$$

where $n - k$ is the length of the syndrome vector [74]. The left-hand-side of the Hamming bound gives the total number of errors that can occur for the specified code distance, and the right-hand-side the maximum number distinct syndromes of length $n - k$. In a correction code, each error must map to a unique syndrome, meaning the left-hand-side of the Hamming bound needs to be less than or equal to the right-hand-side. Any classical error correction code must have n, k and d parameters that satisfy the Hamming bound.

The Hamming codes [74] are a family of codes which satisfy the Hamming bound for

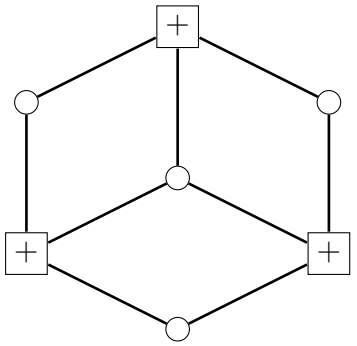
distance $d = 3$. The parity check matrix of a Hamming code is constructed by writing a matrix with columns given by all the binary strings of length $j \geq 3$ excluding the zero vector. As an example, the parity check matrix for the Hamming code with $j = 3$ is given by

$$H_{j=3} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{2.4.2}$$

We can now use equations 2.3.6 and 2.3.3 to write the generator matrix for the code as follows

$$G_{j=3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{2.4.3}$$

The syndrome table for single bit errors for this code is given in table 2.1. From this, it can be seen that each single-bit error produces a unique syndrome, meaning the code has distance $d = 3$. Consequently the Hamming code obtained by setting $j = 3$ is a $[7, 4, 3]$ code. The factor graph for the $[7, 4, 3]$ code is shown below



. (2.4.4)

Parity check matrices constructed with other values of j will produce larger Hamming codes that satisfy the Hamming bound. In chapter 4, we outline methods for constructing quantum Hamming codes using the CPC framework.

Single-bit error, bit number	**Syndrome, s**
1	101
2	110
3	011
4	111
5	100
6	010
7	001

Table 2.1: The syndrome table for the $[7, 4, 3]$ Hamming code. Each single-bit error maps to a unique syndrome.

2.5 High performance classical codes and decoding

We have now introduced two classes of classical error correction code: repetition codes and Hamming codes. Repetition codes add redundancy by simple duplication of the initial data. The logical error rate of a repetition code can be arbitrarily reduced by increasing the number of duplications, but this comes at the expense of transmission rate. Hamming codes, based on the parity check family of codes, can achieve transmission rates that saturate the Hamming bound, but have fixed distance $d = 3$ meaning the logical error rate does not necessarily decrease with increased block size. It was initially believed that there would always be a trade-off between logical error rate and transmission rate for communication over a noisy channel, so that achieving near-zero logical error rate $p_L \to 0$ would also require a near-zero transmission rate $R \to 0$. However, in a pioneering paper written in 1948, Claude Shannon showed that this is not always the case [37]. His remarkable result, known as the Shannon noisy-channel coding theorem, proves that for any channel, there exist codes with non-zero rate that have a vanishingly small logical error probability. The theorem also defines a bound that stipulates the maximum possible rate for lossless transmission along a given noisy channel. This bound is commonly referred to as a the channel capacity or the Shannon

limit [38].

Whilst the Shannon noisy-channel coding theorem proves the existence of good codes, it does not guarantee that these codes are practical to implement. A major implementation challenge for large codes is finding efficient strategies for decoding; this is necessary so that appropriate recovery operations can be deduced in real-time from the syndromes. For the Hamming $[7, 4, 3]$ code, which has seven code bits, it is possible to exhaustively list all the error syndromes in a lookup table, as shown in table 2.1. However, for codes that perform at close to the Shannon limit, block lengths of 1000 or more bits are common, for which the lookup table approach is impractical. For codes of this size, decoding is treated as a maximum posterior inference problem. The basic setup for this can be summarised with a Bayesian relation of the form

$$P(\mathbf{m}|\mathbf{s}) = \frac{P(\mathbf{s}|\mathbf{m})P(\mathbf{m})}{P(\mathbf{s})}, \tag{2.5.1}$$

where $P(\mathbf{m}|\mathbf{s})$ is the probability of the decoded message being $\mathbf{m}$ given a syndrome $\mathbf{s}$, $P(\mathbf{m})$ is the initial message probability and $P(\mathbf{s})$ is a normalisation constant related to the probability of measuring a certain syndrome. The $P(\mathbf{s}|\mathbf{m})$ term is called the likelihood and can be computed using existing information from the error channel. The decoding task typically involves finding the value of $\mathbf{m}$ which maximises the likelihood. Doing this exactly is computationally hard for larger codes, but efficient approximate inference algorithms are known, such as belief propagation [75]. State-of-the-art modern error correction protocols, such as low density parity check codes [38, 76] and turbo codes [39, 77], have been shown to perform at close to the Shannon limit, by employing probabilistic inference methods for their decoding.

Chapter 3

Quantum error correction

In this chapter we introduce the basic concepts behind the construction of quantum error correction codes. In classical information and computing, information is realised as bits that take on values 0 and 1. For quantum computers, the corresponding unit of information is the *qubit* which has the general state

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{3.0.1}$$

where α and β are complex numbers that satisfy the condition $|\alpha|^2 + |\beta|^2 = 1$. Qubits can encode information in a coherent superposition of their basis states, meaning quantum computers have access to a computational space that scales 2^n where n is the total number of qubits [5, 6]. By exploiting superposition, in combination with other quantum effects such as entanglement, it is possible to construct quantum algorithms that can in principle outperform their classical counterparts. Existing examples of such algorithms include methods for factoring [25] and search [78]. However, if these algorithms are ever to be realised on current or near-future quantum hardware, it will be necessary for the qubits to be error corrected [79, 80, 81, 82, 83].

There are a number of complications that prevent techniques from classical coding theory being ported directly to quantum computers. The first of these is the No-Cloning theory for quantum states, which dictates that it is not possible to construct

a unitary operator U_{clone} which performs the following operation

$$U_{\text{clone}}(|\psi\rangle \otimes |s\rangle) \rightarrow |\psi\rangle \otimes |\psi\rangle \,, \tag{3.0.2}$$

where $|\psi\rangle$ is the state to be cloned and $|s\rangle$ is a blank state [6, 40, 79]. In contrast, classical codes work under the assumption that data can be arbitrarily duplicated. For quantum coding, it is therefore necessary to find alternative ways of adding redundancy to the system [41].

A further complication that arises when translating error correction codes to the quantum regime is the increased complexity of quantum error channels. In a classical bit-based computer, errors can occur can occur via a variety of physical processes. However, at the logical level, these processes can be understood in terms of a single error-type, the bit-flip, which takes $0 \rightarrow 1$ and vice-versa [79]. An example of such a channel, outlined in chapter 2, is the binary symmetric channel.

In contrast to classical bits, qubits are continuously parametrised between their basis states, $|0\rangle$ and $|1\rangle$, and are therefore susceptible to a continuum of errors. At first glance, it would therefore seem that quantum error correction protocols should closely resemble techniques from classical analogue computation, for which the theory of error correction is not well developed [17]. Surprisingly, however, it turns out that this continuum of quantum errors can be discretised so that the ability to correct for a discrete set of errors is sufficient to correct for any error [42]. To illustrate this, consider a single-qubit error process E, which acts coherently as follows

$$|\psi\rangle \xrightarrow{\text{coherent error}} E\,|\psi\rangle \,. \tag{3.0.3}$$

Single-qubit error processes of this type are represented by a unitary two-dimensional matrix, which can be expanded in the Pauli basis $\{\mathbb{1}, X, Y, Z\}$. Equation 3.0.3 can therefore be rewritten as

$$E\,|\psi\rangle = \alpha_I \mathbb{1}\,|\psi\rangle + \alpha_X X\,|\psi\rangle + \alpha_Z Z\,|\psi\rangle + \alpha_Y Y\,|\psi\rangle \,, \tag{3.0.4}$$

where $\alpha_{I,X,Z,Y}$ are the expansion coefficients [6]. The Y-operator is equivalent (up to phase) to the simultaneous occurrence of an X- and Z-error. Coherent noise processes

can therefore be discretised to linear combinations of bit-flips (also referred to as X-errors) and phase-flips (also referred to as Z-errors). The extraction of syndromes in a quantum error correction protocol involves making a projective measurement that collapses the quantum state to one of the terms in equation 3.0.4. Consequently, a quantum code with the ability to correct for single X- and Z-errors can resolve the entire continuum of coherent errors possible for a single qubit [42, 79]. Furthermore, this result generalises to arbitrary quantum channels, including those that describe incoherent processes. For the latter, the same result can be proved by a following a similar line of reasoning in the operator sum representation for quantum noise processes [6].

The third complication that arises when constructing quantum codes is the problem of wavefunction collapse. For classical codes, it is possible to perform arbitrary parity checks without risk of comprising the encoded data. For quantum codes, however, any parity checks must be carefully chosen so not to decohere the encoded information. In the following sections, we describe how a family of codes known as stabilizer codes can overcome these hurdles [47].

3.1 Quantum redundancy and the two-qubit code

As outlined in the previous section, quantum error correction is complicated by the No-Cloning theorem and the existence of a uniquely quantum error-type, the phase-flip. So, faced with these challenges, how is redundancy added to a quantum system to allow errors to be corrected in real time? Classical repetition codes work by increasing the resources used to encode the data beyond the theoretical minimum. Analogously, in quantum codes redundancy is added by expanding the Hilbert space in which the qubits are encoded [41, 79]. To see how this is achieved in practice, we now describe the two-qubit code, a prototypical quantum code designed to detect a single-bit flip error. The encode stage of the two-qubit code, acting on the general state $|\psi\rangle$, has the following action

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \xrightarrow{\text{two-qubit encoder}} |\psi\rangle_L = \alpha\,|00\rangle + \beta\,|11\rangle = \alpha\,|0\rangle_L + \beta\,|1\rangle_L\,, \quad (3.1.1)$$

where after encoding the logical basis states are $|0\rangle_L = |00\rangle$ and $|1\rangle_L = |11\rangle$. Note that this does not correspond to cloning the state as

$$|\psi\rangle_L = \alpha\,|00\rangle + \beta\,|11\rangle \neq |\psi\rangle \otimes |\psi\rangle\,. \tag{3.1.2}$$

The effect of the encoding operation is to distribute the quantum information in the initial state $|\psi\rangle$ across the entangled two-party logical state $|\psi\rangle_L$. This introduces redundancy to the encoding that can be exploited for error detection. To understand exactly how this works, it is instructive to consider the computational Hilbert spaces before and after encoding. Prior to encoding, the single qubit is parametrised within a two-dimensional Hilbert space $|\psi\rangle \in \mathcal{H}_2 = \text{span}\{|0\rangle, |1\rangle\}$. After encoding the logical qubit occupies a four-dimensional Hilbert space

$$|\psi\rangle \in \mathcal{H}_4 = \text{span}\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}. \tag{3.1.3}$$

More specifically the logical qubit spans a two-dimensional subspace of this expanded Hilbert space

$$|\psi\rangle_L \in \mathcal{C} = \text{span}\{|00\rangle, |11\rangle\} \subset \mathcal{H}_4, \tag{3.1.4}$$

where $\mathcal{C}$ is called the codespace. Now, imagine that the logical qubit is subject a bit-flip error on the first qubit resulting in the state

$$X_1\,|\psi\rangle_L = \alpha\,|10\rangle + \beta\,|01\rangle\,, \tag{3.1.5}$$

where X_1 is a bit-flip error acting on the first qubit. The resultant state is rotated into a new subspace

$$X_1\,|\psi\rangle_L \in \mathcal{F} \subset \mathcal{H}_4, \tag{3.1.6}$$

where we call $\mathcal{F}$ the error subspace. Notice that an X_2-error will also rotate the logical state into the $\mathcal{F}$ subspace. If the logical state $|\psi\rangle_L$ is uncorrupted, it occupies the codespace $\mathcal{C}$, whereas if it has been subject to a single-qubit bit-flip, it occupies the error space $\mathcal{F}$. As the $\mathcal{C}$ and $\mathcal{F}$ subspaces are mutually orthogonal, it is possible to distinguish which subspace the logical qubit occupies via a projective measurement [79]. In the context of quantum coding, measurements of this type are called stabilizer measurements, and can be thought of as quantum parity checks designed to leave the

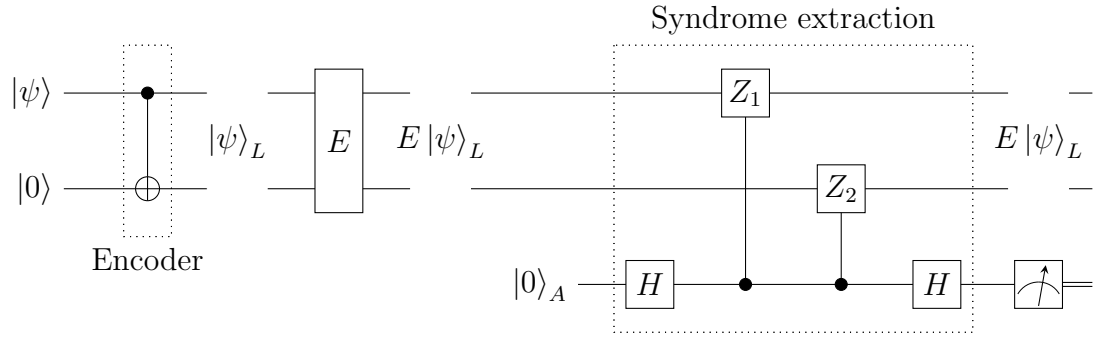


Figure 3.1: Circuit diagram for the two qubit code. Encode stage: the information contained in $|\psi\rangle$ is entangled with a blank redundancy qubit $|0\rangle$ to create a logical state $|\psi\rangle_L$. Error stage: during the error window (shown by the circuit element E), the two code qubits are potentially subject to bit-flip errors. Syndrome extraction stage: the Z_1Z_2 operator is measured on the code qubits and the result is copied to ancilla qubit A. The subsequent measurement of the ancilla gives the code syndrome.

encoded quantum information intact [46].

For the purposes of differentiating between the codespace $\mathcal{C}$ and the error space $\mathcal{F}$, a stabilizer measurement of the form Z_1Z_2 is sufficient. The Z_1Z_2 parity check operator has the following effect on the logical state of the two-qubit code

$$Z_1Z_2|\psi\rangle_L = Z_1Z_2(\alpha|00\rangle + \beta|11\rangle) = (+1)|\psi\rangle_L . \tag{3.1.7}$$

The measurement is said to 'stabilize' the logical qubit $|\psi\rangle_L$ as it leaves it unchanged by projecting it onto the $(+1)$ eigenspace. Conversely, the Z_1Z_2 operator projects the errored states, $X_1|\psi\rangle_L$ and $X_2|\psi\rangle_L$, onto the (-1) eigenspace. Notice that for either outcome, the information encoded in the α and β coefficients of the logical state remains undisturbed [46, 79].

Figure 3.1 shows the circuit implementation of the two-qubit code. In the encode stage, a CNOT gate is used to entangle the $|\psi\rangle$ state with a blank redundancy qubit to create the logical state $|\psi\rangle_L$. Following this, we assume the logical qubit is subject to a bit-flip error channel, the action of which is represented by the circuit element E in the quantum circuit diagram. Following the error stage, an ancilla qubit $|0\rangle_A$ is introduced

Error, E	Syndrome, s
$I_1 I_2$	0
$X_1 I_2$	1
$I_1 X_2$	1
$X_1 X_2$	0

Table 3.1: The syndrome table for the two-qubit code.

to perform the measurement of the $Z_1 Z_2$ stabilizer. The syndrome extraction stage of the circuit transforms the quantum state as follows

$$E\left|\psi\right\rangle_L\left|0\right\rangle_A \xrightarrow{\text{syndrome extraction}} (\mathbb{1}+Z_1Z_2)E\left|\psi\right\rangle_L\left|0\right\rangle_A + (\mathbb{1}-Z_1Z_2)E\left|\psi\right\rangle_L\left|1\right\rangle_A, \quad (3.1.8)$$

where E is an error from the set $\{\mathbb{1}, X_1, X_2, X_1X_2\}$. From the above we see that if the logical state is in the codespace (I.e., if $E = \{\mathbb{1}, X_1X_2\}$)) then the ancilla is measured deterministically as '0'. Likewise, if the logical state is in the error subspace (I.e., if $E = \{X_1, X_2\}$) then the ancilla is measured deterministically as '1'. The stabilizer construction depicted in figure 3.1 therefore gives a syndrome that indicates whether or not the logical state has been subject to an error. The syndromes for all bit-flip error types in the two-qubit code are shown in table 3.1.

The effectiveness of a quantum code can be assessed via a fidelity analysis [6, 79, 82]. The fidelity F gives the overlap between an initial state $|\psi\rangle$ and an evolved state $|\psi(t_f)\rangle$, and is calculated as follows

$$F(t_f) = |\left\langle\psi(t_f)|\psi\right\rangle|^2. \quad (3.1.9)$$

The first step in the fidelity analysis of a quantum code is to calculate the raw fidelity of a single-qubit prior to encoding. The raw fidelity provides a benchmark that the quantum code must 'beat' in order for the protocol to be considered worthwhile. For the purposes of our analysis of the two-qubit code, we assume that the code qubits are independently subject to an error process described by the quantum bit-flip channel. In a single timestep t_f, the quantum bit-flip channel transforms the density matrix of

a single qubit state $\rho = |\psi\rangle\langle\psi|$ as follows

$$\rho(t_f) = \xrightarrow{\text{bit-flip channel}} (1-p)\rho + pX\rho X, \tag{3.1.10}$$

where p is the probability of a bit-flip error [6]. The fidelity of an unencoded single qubit can then be computed to give

$$\begin{aligned} F(t_f) &= |\langle\psi(t_f)|\psi\rangle|^2 = \langle\psi|\rho(t_f)|\psi\rangle = \langle\psi|[(1-p)\rho + pX\rho X]|\psi\rangle \\ &= \langle\psi|[(1-p)|\psi\rangle\langle\psi| + pX|\psi\rangle\langle\psi|X]|\psi\rangle = (1-p) + p(\langle\psi|X|\psi\rangle)^2. \end{aligned} \tag{3.1.11}$$

The raw fidelity is defined as the lower bound of this fidelity so that $F(t_f)_{\text{raw}} = \textbf{min}\ F(t_f)$. By inspection, it is clear that $F(t_f)$ is minimised when the $X|\psi\rangle$ term is orthogonal to $|\psi\rangle$. This gives the raw fidelity below

$$F(t_f)_{\text{raw}} = 1 - p. \tag{3.1.12}$$

Our aim is now to show that $F(t_f)_L > F(t_F)_{\text{raw}}$, where $F(t_f)_L$ is the mininum fidelity of the output of the two-qubit code. The quantum bit-flip channel transforms the density matrix $\rho_L = |\psi\rangle_L\langle\psi|_L$ of the logical state of the two-qubit code as follows

$$p_L(t_f) \xrightarrow{\text{bit-flip channel}} (1-p)^2\rho_L + p(1-p)X_1\rho_L X_1 + p(1-p)X_2\rho_L X_2 + p^2 X_1X_2\rho_L X_1X_2. \tag{3.1.13}$$

From table 3.1, we see that the two-qubit code triggers a syndrome $s = 1$ for all single-qubit errors (I.e., X_1 or X_2). This syndrome information can be used to post-select the output of the code and improve the fidelity. Post-selection involves excluding the terms in equation 3.1.13 that correspond to the single-qubit errors. The post-selected density matrix $\rho_L(t_f)_{s=0}$ then has the form

$$\rho_L(t_f)_{s=0} = \frac{(1-p)^2\rho_L + p^2 X_1X_2\rho_L X_1X_2}{(1-p)^2 + p^2}, \tag{3.1.14}$$

where the denominator ensures normalisation. The lower bound of the fidelity[a] of the

[a]The lower bound occurs when the $p^2 X_1X_2\rho_L X_1X_2$ term in equation 3.1.14 is equal to zero. This is the case when $X_1X_2|\psi\rangle_L$ is orthogonal to $|\psi\rangle_L$.

output of the two-qubit code $F(t_f)_L$ can now be computed to give

$$F(t_f)_L = \langle\psi_L|\rho_L(t_f)_{s=0}|\psi_L\rangle = \frac{(1-p)^2}{(1-p)^2+p^2} \approx 1-p^2, \tag{3.1.15}$$

where the approximation assumes that p is small. From this, it can be seen that the two-qubit code satisfies the requirement that $F(t_f)_L > F(t_f)_{\text{raw}}$. In a single application of the two-qubit code, if no error is detected, the leading order in the error is suppressed from p to p^2. The fidelity analysis has therefore demonstrated that quantum redundancy can be harnessed to create useful error detection protocols. In the following sections, we explore the construct of stabilizer codes that can both detect and correct errors.

3.2 The quantum Hamming bound

The two-qubit code works by de-localising the information in a single qubit across two qubits. The resultant logical state is encoded in a two-dimensional subspace of the expanded Hilbert space. If a single-qubit error then occurs, the logical state is rotated to an orthogonal subspace, an event that can be detected via a projective measurement that does not collapse the encoded quantum information. However, as the two-qubit code is a detection code, its syndromes do not provide sufficient information to both detect and localise errors. In order to create an error correction code that does this, a more complicated partitioning of the Hilbert space is necessary. More specifically, we require that every error rotates the logical state to a unique space. Mathematically, this requirement is expressed as follows

$$E_i\,|\psi\rangle_L \in \mathcal{F}_i \;\forall\; E_i \in E = \{E_1, E_2, ..., E_N\}, \tag{3.2.1}$$

where E is the set of errors E_i and $\mathcal{F}_i$ is a unique error space. For a quantum error correction code encoding k qubits, we require a 2^k-dimensional subspace for the codespace $\mathcal{C}$ in addition to a 2^k-dimensional subspace for each error space $\mathcal{F}_i$. These requirements lead to a bound on the resources required to realise a quantum error correction code

[6, 84]. This bound is called the quantum Hamming bound and is defined as follows

$$\sum_{j=0}^{(d-1)/2} \binom{n}{j} |\mathcal{E}|^j \leq 2^{n-k}, \tag{3.2.2}$$

where $\mathcal{E}$ lists the error-types that the qubits are suspectible to, n is the total number of physical qubits in the code, k is the number of encoded qubits and d is the distance of the quantum code.[b] The term to the left of the above equation gives the total number of errors that need to be corrected for a distance d code (I.e., the number of distinct syndromes that are required). The quantum Hamming bound is similar to the Hamming bound for classical codes, defined in equation 2.4.1 in section 2.4, but includes the extra term $|\mathcal{E}|$ to account for the fact that quantum codes are susceptible to more than one error type. In this thesis, we use the $[[n, k, d]]$ labelling convention for quantum codes.[c] Any non-degenerate quantum error correction code must have n, k and d parameters that satisfy the quantum Hamming bound. As an example, consider the case in which we wish to construct an $[[n, 1, 3]]$ quantum code. What is the minimum number of physical qubits required? If we assume that our error channel is susceptible to X-, Z- and Y-errors, then $\mathcal{E} = \{X, Z, Y\}$ and $|\mathcal{E}| = 3$. Plugging this into equation 3.2.2, we can deduce that the smallest possible $[[n, 1, 3]]$ code is a $[[5, 1, 3]]$ code.

3.3 Stabilizer codes

In the two-qubit code shown in figure 3.1, a projective measurement of the Z_1Z_2 stabilizer is performed on the logical state. The resultant measurement outcome is called a syndrome, and tells us whether or not an error has occurred. To construct a quantum error correction code, multiple overlapping stabilizer checks are required. We now describe the general construction for such $[[n, k, d]]$ stabilizer codes [46].

[b] The distance for a quantum code is defined in the same way as the distance for a classical code: the minimum weight of an error operator required to change one codeword to another.

[c] Note the use of double brackets. This allows quantum codes to be be differentiated from classical codes which are labelled $[n, k, d]$.

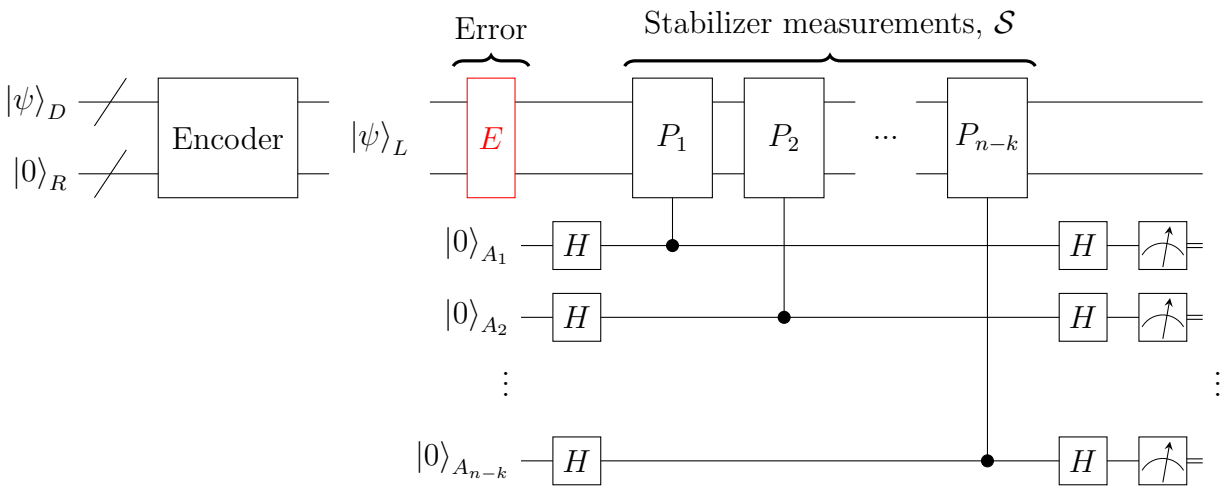


Figure 3.2: Circuit illustrating the structure of an $[[n, k, d]]$ stabilizer code. A quantum data register $|\psi\rangle_D = |\psi_1\psi_2...\psi_k\rangle$ is entangled with redundancy qubits $|0\rangle_R = |0_1 0_2 ... 0_{n-k}\rangle$ via an encoding operation to create a logical qubit $|\psi\rangle_L$. After encoding, a sequence of $n-k$ stabilizer checks P_i are performed on the register to determine whether an error has occurred.

The left-hand side of the quantum Hamming bound (equation 3.2.2) gives the number of errors that need to be uniquely detected in an $[[n, k, d]]$ quantum code. Taking the base-2 logarithm of the right-hand side of equation 3.2.2 gives $n-k$, which is the length of the syndrome vector required for there to be a unique syndrome for each of these errors. As each bit in the syndrome vector is the result of a stabilizer measurement, $n-k$ stabilizer measurements are required in total.

The circuit in figure 3.2 shows the basic structure of an $[[n, k, d]]$ stabilizer code. A register of k data qubits, $|\psi\rangle_D$, is entangled with $n-k$ blank redundancy qubits, $|0\rangle_R$, via an encoding operation to create a logical qubit $|\psi\rangle_L$. At this stage, the data previously stored solely in $|\psi\rangle_D$ is distributed across the combined Hilbert space of data and redundancy qubits. Errors can then be detected by performing $n-k$ stabilizer measurements P_i as shown to the right of figure 3.2.

In the circuit in figure 3.2, each of the stabilizers are measured using the same syndrome extraction method that was used for the two qubit code in figure 3.1. For each stabilizer

P_i, the syndrome extraction circuit maps the logical state as follows

$$E\left|\psi\right\rangle_L\left|0\right\rangle_A \xrightarrow{\text{syndrome extraction}} (\mathbb{1}+P_i)E\left|\psi\right\rangle_L\left|0\right\rangle_A + (\mathbb{1}-P_i)E\left|\psi\right\rangle_L\left|1\right\rangle_A . \tag{3.3.1}$$

If the stabilizer P_i commutes with an error E the measurement returns '0'. If the stabilizer P_i anti-commutes with an error E the measurement returns '1'. The task of creating a stabilizer code therefore involves finding stabilizers that anti-commute with one or more of the errors to be detected.

In order to ensure that the stabilizers of the code can be measured simultaneously (or in a way that is independent of their ordering), it is necessary that they commute with one another [79]. The general requirements for the stabilizers $\mathcal{S}$ of a quantum code can be summarised mathematically as follows

$$\mathcal{S} = \{P_i\left|\psi\right\rangle_L = \left|\psi\right\rangle_L ,\ [P_i, P_j] = 0, \forall(i,j)\}. \tag{3.3.2}$$

In addition to stabilizers, each code will have $2k$ Pauli-logical operators that allow for switching between the encoded basis states. For each logical qubit i, there is a logical $\bar{X}_i$ Pauli operator and logical $\bar{Z}_i$. The logical Pauli operators, $\bar{X}_i$ and $\bar{Z}_i$, commute with all the stabilizers in $\mathcal{S}$, but anti-commute with one another.

The challenges of constructing stabilizer quantum codes are twofold. First, an appropriate encoding operation must be built to create the logical qubit. Second, a compatible set of stabilizer parity checks needs to be discovered so that errors can be checked without compromising the encoded quantum data. As a result of these challenges, it is not easy to re-purpose classical parity checking sequences for use in a stabilizer codes. In the next section, we describe the Calderbank, Shor and Steane (CSS) construction which provides methods by which good stablizer codes can be derived from classical codes which satisfy certain properties [42, 43]. The CPC framework, the main topic of this thesis, provides an alternative construction that allows stabilizer codes to be derived from the starting point of any classical code.

3.4 Calderbank, Shor and Steane (CSS) codes

Calderbank, Shor and Steane (CSS) codes are a family of stabilizer codes that can be derived by combining pairs of classical codes [42, 43]. Their general construction can be summarised as follows:

1. Choose two classical codes C_1 and C_2 that satisfy the condition that $C_2 \subset C_1$. This condition means that the codewords of C_2 are also codewords of C_1.

2. If C_1 is an $[n, k_1, d_1]$ code and C_2 is a $[n, k_2, d_2]$ code, an $[[n, k_1 - k_2, \min(d_1, d_2)]]$ CSS code can be constructed. This CSS code will have quantum codewords given by
$$|x\rangle_L = |x \oplus C_2\rangle_L = \frac{1}{\sqrt{|C_2|}} \sum_{y \in C_2} |x \oplus y\rangle \,, \tag{3.4.1}$$
where $x \in C_1$ is a codeword of C_1.

3. CSS codes have the property that each of their stabilizers consists of only Z-Pauli operators or only X-Pauli operators. I.e., there are no mixed stabilizers of the form $P_X = Z_i X_j$ in a CSS code.

4. The stabilizers of a CSS code can be deduced from the C_1 and C_2 as follows. The rows of the parity check matrix $H(C_1)$ give the Z-stabilizers. The rows of the generator matrix of $G(C_2)$ give the X-stabilizers.

5. A useful class of CSS codes are those in which C_2 is defined as $C_2 = C_1^{\perp} \subset C_1$, where $C_1^{\perp}$ is the dual of C_1. The dual of a code C is the code $C^{\perp}$ in which the role of the generator and parity checks matrices are reversed. A code C_1 which satisfies the requirement that $C_1^{\perp} \subset C_1$ is called a weakly self-dual code.

We now demonstrate the construction of a CSS code by means of an example. We start by choosing C_1 to be a $[4, 3, 2]$ detection code with the following generator and

parity check matrices

$$G(C_1) = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad H(C_1) = [1 \quad 1 \quad 1 \quad 1]. \tag{3.4.2}$$

The binary codewords of C_1 are given by

$$C_1 = \{0000, 1001, 0101, 0011, 1100, 1010, 0110, 1111\}. \tag{3.4.3}$$

We now define $C_2 = C_1^{\perp}$, which gives a $[4, 1, 3]$ code with the following generator and parity check matrices

$$G(C_2) = H(C_1) = [1 \quad 1 \quad 1 \quad 1], \quad \mathrm{H(C_2) = G(C_1)} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{3.4.4}$$

The codewords of C_2 are then given by

$$C_2 = \{0000, 1111\}. \tag{3.4.5}$$

From the above, it can be seen that the requirement that $C_2 \subset C_1$ is satisfied. We can therefore use equation 3.4.1 to construct the codewords of a CSS code from C_1 and C_2

$$CSS(C_1, C_2) = \begin{bmatrix} |00\rangle_L = \frac{1}{\sqrt{2}}(|0000\rangle + |1111\rangle), \\ |01\rangle_L = \frac{1}{\sqrt{2}}(|0110\rangle + |1001\rangle), \\ |10\rangle_L = \frac{1}{\sqrt{2}}(|1010\rangle + |0101\rangle), \\ |11\rangle_L = \frac{1}{\sqrt{2}}(|1100\rangle + |0011\rangle) \end{bmatrix}. \tag{3.4.6}$$

The above codewords are the logical basis states of a $[[4, 2, 2]]$ detection code. The $[[4, 2, 2]]$ detection code is the smallest code capable of detecting a full quantum error set $\mathcal{E} = \{X, Y, Z\}$. The stabilizers of the $[[4, 2, 2]]$ CSS code can be read off from $H(C_1)$ and $G(C_2)$ respectively, and are given by

$$\mathcal{S}_{[[4,2,2]]} = \{Z_1 Z_2 Z_3 Z_4, X_1 X_2 X_3 X_4\}. \tag{3.4.7}$$

3.5 The finite geometry representation of stabilizer codes

The error operators, logical operators and stabilizers of quantum codes are elements of the n-qubit Pauli group $\mathcal{G}_N$ (for an outline of the Pauli group see appendix A). We now explain the finite geometry representation for elements of $\mathcal{G}_N$ [6, 46, 80, 81], an alternative representation that can simplify the study of quantum codes and their properties. A Pauli group operator $g \in G_N$, acting on N qubits, is mapped to a binary row-vector of length $2N$ in the finite geometry representation as follows

$$g \xrightarrow{\text{finite geometry mapping}} G_{XZ}(g) = (b_1 \quad b_2 \quad ... \quad b_N \quad | \quad b_{N+1} \quad b_{N+2} \quad ... \quad b_{2N}) \tag{3.5.1}$$

where b_i are binary variables. If g contains an X_i-operator acting on qubit i, this is represented as a '1' at position i in $G_{XZ}(g)$. Likewise, if g contains a Z_j-operator acting on qubit j, this is represented as a '1' at position $j+N$ in $G_{XZ}(g)$. Finally a Y_i operator at position m maps to a '1' at position m and $m+N$ in $G_{XZ}(g)$. As an example, consider an 4-qubit operator of the form $g_4{=}X_1\mathbb{1}_2Z_3Y_4$. In the finite geometry representation g_4 maps to

$$G_{XZ}(g_4) = (1 \quad 0 \quad 0 \quad 1 \quad | \quad 0 \quad 0 \quad 1 \quad 1). \tag{3.5.2}$$

The finite geometry representation can be used to construct a quantum parity check matrix, which plays a role similar to the parity check matrices of a classical code. The quantum parity check matrix is constructed by stacking the finite geometry representations of the stabilizers of a code. For example, the quantum parity check matrix for the stabilizers of the $[[4,2,2]]$ code (which are defined in equation 4.4.6), is given by

$$G_{XZ}(\mathcal{S}_{[[4,2,2]]}) = \left(\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right). \tag{3.5.3}$$

For CSS codes, the quantum parity check matrix has the form

$$G_{XZ}(\mathcal{S}_{CSS(C_1,C_2)}) = \left(\begin{array}{c|c} G(C_1) & \mathbf{0} \\ \mathbf{0} & H(C_2) \end{array}\right), \tag{3.5.4}$$

where $G(C_1)$ and $H(C_2)$ are the generator and parity check matrices of C_1 and C_2 respectively. From this, the separation between the X-containing stabilizers and Z-containing stabilizers can clearly be seen. The quantum parity check matrix for CSS codes also demonstrates how the finite geometry representation can help to make contact between classical and quantum codes.

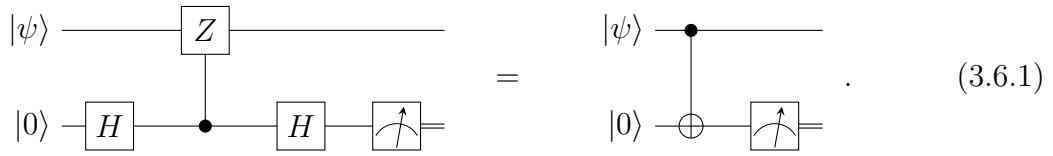
Once a quantum parity check matrix $G_{XZ}(\mathcal{S})$ has been constructed for a code with stabilizers $\mathcal{S}$, the syndromes $\vec{S}_E$ for each error E be computed as follows

$$\vec{S}_E = G_{XZ}(\mathcal{S}) \cdot (G_{ZX}(E))^T, \tag{3.5.5}$$

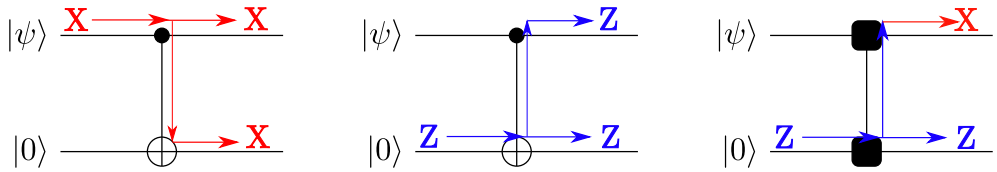
where $G_{ZX}(E)$ is the flipped finite geometry representation of E. Note that the positions of the X and Z elements in $G_{ZX}(E)$ have been flipped.

3.6 Gates for stabilizer codes

The syndrome extraction process for a stabilizer code involves making Pauli measurements on the logical state to detect X- and Z-errors as shown in figure 3.2. We now introduce the circuit notation we will use for the depiction of quantum codes in this thesis. Bit-flip check measurements on the logical state are represented by CNOT gates via the following mapping



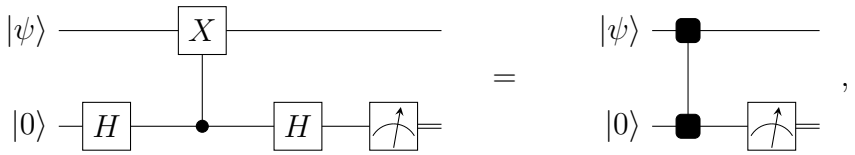
. (3.6.1)



(a) (b) (c)

Figure 3.3: Propagation rules for single-qubit X- and Z-errors through CNOT and conjugate-propagator gates.

Phase-flip check measurements are represented as follows



, (3.6.2)

where the symmetric gate with black-square controls is called a conjugate-propagator gate. Note that the conjugate-propagator gate is a new gate we have defined for use with the coherent parity check framework. In addition to making circuits more compact, representing parity check operations in terms of CNOT and conjugate-propagator gates allows for easy visualisation of the propagation of errors through the circuit. For example, CNOT gates propagate Z-errors as follows

. (3.6.3)

The above propagation rule is calculated by evolving the error via the following matrix relation $\text{CNOT} \cdot Z_2 \cdot \text{CNOT} = Z_1 Z_2$. The propagation rules for X-errors can be similarly calculated. The complete set of propagation rules for both CNOT and conjugate propagator gates is shown in figure 3.3.

3.7 Fault tolerant circuit design

In the discussion of stabilizer codes so far, we have assumed that errors only occur in certain locations in the circuit. For example, in the circuit diagram for the two-qubit code (figure 3.1), errors are restricted to the region marked by the multi-qubit gate E. In this circuit, it is assumed that all of the machinery associated with encoding and syndrome extraction is error-free. However, on many quantum technology platforms, two-qubit gates are a dominant source of error [85, 86]: it is therefore unrealistic to assume that syndrome extraction will occur perfectly.

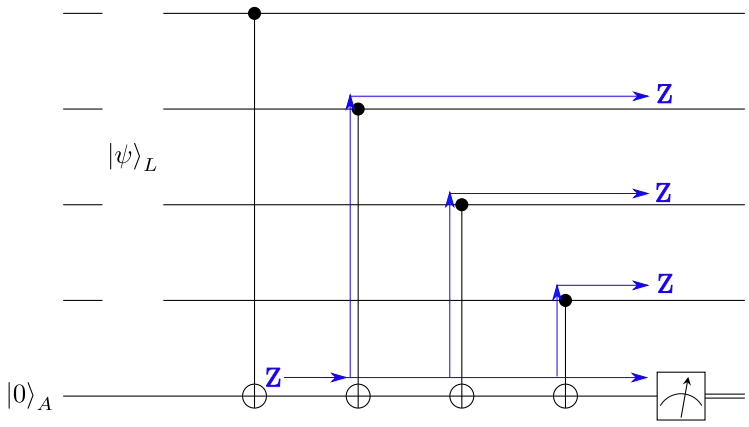


Figure 3.4: The extraction circuit for the $Z_1Z_2Z_3Z_4$ stabilizer in the $[[4, 2, 2]]$ code. Errors that occur on the ancilla qubit A can propagate to the code qubits. As an example, the propagation resulting from a Z-error that occurs after the first CNOT gate on the ancilla qubit is shown by the blue arrows.

To contextualise the problem, consider the extraction circuit for the $Z_1Z_2Z_3Z_4$ stabilizer of the $[[4, 2, 2]]$ code, shown in figure 3.4. From the figure, we see that if a Z-error occurs after the first CNOT gate on the ancilla qubit A, a $Z_2Z_3Z_4$-error is propagated to the register. The $[[4, 2, 2]]$ code has distance $d = 2$, meaning the $Z_2Z_3Z_4$ error goes undetected. Quantum codes must therefore be carefully designed to stop errors cascading through the circuit in this way. A quantum code that can account for errors (up to a specified distance) at any location in its circuit is said to be fault tolerant.

Various techniques for achieving fault tolerance are known [44, 45, 87, 88], but these typically increase code overheads in both qubit-number and time. Consequently, fault tolerance is an important consideration when assessing the viability of a quantum code.

Chapter 4

Coherent Parity Check Codes

The coherent parity check (CPC) framework for quantum error correction is designed to provide an alternative approach to the design and understanding of stabilizer codes. The signature feature of the framework is the ability to construct a stabilizer code from the starting point of any sequence of parity checks. This is possible due to a fail-safe CPC code structure that guarantees that the quantum mechanical requirements of the code, such as stabilizer-commutativity, are automatically satisfied. As a result of this freedom in the choice of parity checks, the CPC framework facilitates the re-purposing of good classical codes for use in quantum error correction. Furthermore, under the CPC framework, the problem of discovering new quantum codes can be set up in a way that is amenable to machine search.

In this chapter, I outline the CPC framework and provide examples of how it can be used to construct distance-three codes. The CPC framework was first introduced in Chancellor et al. [1], using tools from the ZX-calculus [89]. In [2], I provided an alternative introduction using conventional quantum circuit notation. I will also use the circuit notation presentation in this thesis.

All CPC codes are built around a fundamental gadget with a symmetric *encode-error-decode* structure that amounts to an extended measurement of the identity operator. I prove that the CPC gadget is inherently non-disturbing and can be implemented using any parity checking sequence. Following this, I show how multiple CPC gadgets

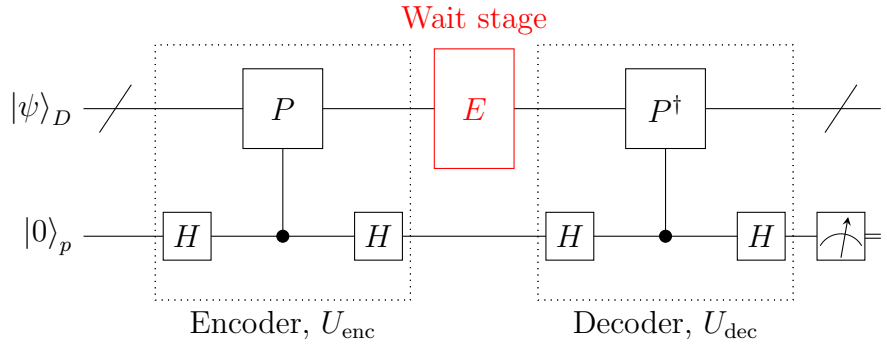


Figure 4.1: The fundamental CPC gadget illustrating the symmetric *encode-error-decode* structure. Encode stage: a parity check P, controlled by the parity qubit, is applied to the multi-qubit register $|\psi\rangle_D = |\psi_1\psi_2...\psi_k\rangle$ and the result is copied to the parity qubit p. The parity qubit is kept coherent throughout the wait stage, during which an error E can occur on the register. Decode stage: the register is disentangled from the parity qubit via the application of the unitary inverse of the first parity check $P^\dagger$. The final syndrome measurement of qubit p tells us whether the results of the two parity checks differ. For appropriately chosen parity checks, this information can be used to detect errors.

can be combined to create quantum codes. After demonstrating the construction of a $[[4, 2, 2]]$ CPC detection code, I establish a canonical CPC code structure for the design of general $[[n, k, d]]$ stabilizer codes. I then prove that, under this canonical CPC structure, the encoder always produces a valid stabilizer code.

This chapter ends by outlining CPC methods for the derivation of distance-3 stabilizer codes. I provide an explicit construction for turning any $[n, k, 3]$ classical Hamming code into a $[[2n - k, k, 3]]$ CPC code. This result extends a method originally outlined in [1]. Finally, I describe how CPC methods can used as an alternative way of deriving distance-three CSS codes from the starting point of any distance-three classical code.

4.1 The fundamental CPC gadget

The fundamental CPC gadget, shown in figure 4.1, is the building block upon which all CPC codes are based [1]. The basic premise behind the CPC gadget is that the parity of the quantum register is never explicitly measured. Instead, parity information is stored coherently as quantum data and compared over time. This is made possible by the gadget's symmetric *encode-error-decode* structure.

4.1.1 Error detection with any parity check

The CPC gadget takes a multi-qubit register $|\psi\rangle_D$ and a parity qubit p, prepared in the state $|0\rangle_p$, as its input. The action of the encode stage of the gadget, labelled U_{enc} in figure 4.1, is to apply the parity operator P to the register and record the outcome in parity qubit p. Rather than measuring the syndrome immediately, the parity qubit is kept coherent during a wait stage in which the register is potentially subject to an error E. Note that we are not yet considering errors that occur on the parity qubit. In section 4.2, we outline how multiple CPC gadgets can be combined to allow for error detection on the combined system of register and parity qubits.

Following the wait stage, the parity qubit is disentangled from the register via a decoding operation, labelled U_{dec} in figure 4.1, which is the unitary inverse of the encoder. The encoder applies the parity operator P to the register and the decoder applies its inverse $P^\dagger$. The final syndrome measurement of parity qubit p tells us whether the results of these two parity checks differ. For an appropriately chosen parity check, this syndrome information can indicate whether an error occurred during the wait stage.

To prove its error detection capabilities, it is convenient to rearrange the circuit for the CPC gadget into the form shown in figure 4.2. This rewrite is achieved by moving the error operator E through the parity check operator P. Both the error gate and the parity check gate are Pauli group operations. A property of the Pauli group is that its elements either commute or anti-commute with one another. Consequently, the effect of pushing the error operator to the front of the circuit is to introduce a global phase

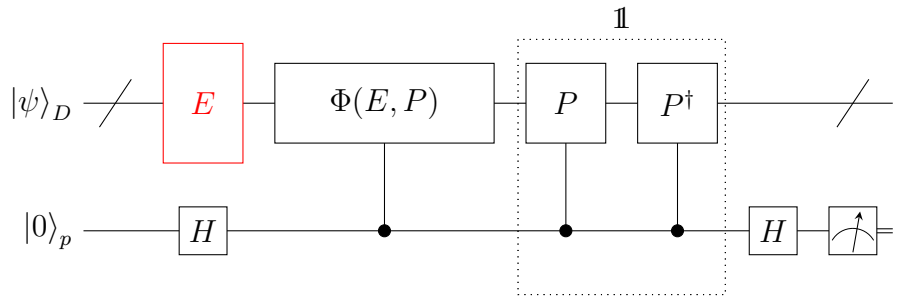


Figure 4.2: To prove the non-disturbing nature of the fundamental CPC gadget, it is useful to rearrange the circuit by moving the error operator E through the first parity check P. Following this rewrite, the controlled parity check operators are adjacent and cancel. In this form, the CPC gadget can be viewed as a measurement of the $\Phi(E,P) = \pm\mathbb{1}_D$ operator on the data register. The value of the final syndrome measurement depends only upon whether E commutes with P.

$\Phi(E,P)$ on the register which is controlled by the parity qubit. This global phase is dependent upon both the parity check and the error operator, and is defined as follows,

$$\Phi(E,p) = \begin{cases} (+1)\mathbb{1}_D, & \text{if } [E,P] = 0 \\ (-1)\mathbb{1}_D, & \text{if } [E,P] \neq 0, \end{cases} \tag{4.1.1}$$

where $\mathbb{1}_D$ is the identity operator on the data register and the commutator is given by $[E,P] = E \cdot P - P \cdot E$. Note that, after the rewrite, the controlled parity-check operators are adjacent to each other and cancel. The full mathematical action of the CPC circuit U_{CPC}, can now be expressed as follows,

$$U_{\text{CPC}} |\psi\rangle_D |0\rangle_p = (\mathbb{1} + \Phi(E,P)) E |\psi\rangle_D |0\rangle_p + (\mathbb{1} - \Phi(E,P)) E |\psi\rangle_D |1\rangle_p . \tag{4.1.2}$$

Using the definition of the global phase operator $\Phi(E,P)$ given in equation (4.1.1), the output of the CPC gadget simplifies to

$$U_{\text{CPC}} |\psi\rangle_D |0\rangle_p = \begin{cases} E |\psi\rangle_D |0\rangle_p , & \text{if } [E,P] = 0 \\ E |\psi\rangle_D |1\rangle_p , & \text{if } [E,P] \neq 0. \end{cases} \tag{4.1.3}$$

From the above we can see that eventual measurement of parity qubit p depends only

upon whether P commutes with E. If no error occurs during the wait stage, then $E = \mathbb{1}_D$ and the syndrome is measured deterministically as '0'. Likewise, if an error does occur, but it commutes with the parity operator, $[E, P] = 0$, then the syndrome is also '0'. Finally, if the error anti-commutes with the parity check, $[E, P] \neq 0$, then the syndrome is measured as '1'. A quantum error detection protocol can therefore be constructed from the CPC gadget by selecting a parity check that anti-commutes with the error to be identified.

The CPC gadget can be thought of as an extended measurement of the $\pm\mathbb{1}_D$ operator on the data register, where the sign depends upon the commutation relation between P and E. As the $\pm\mathbb{1}_D$ operator is trivially non-disturbing for all quantum states, the parity qubit p will always be completely disentangled from the register at the end of the cycle. Consequently, the CPC gadget provides a construction whereby any parity check operator can be applied to the register in a failsafe manner. Therefore, the only restriction on the form of P is is that it is a Pauli group operator $P \in \mathcal{G}_k$.

4.1.2 Multi-check CPC gadgets

Figure 4.3 shows how multiple parity check operators can be merged into a single CPC cycle. As is the case with the fundamental CPC gadget, the *encode-error-decode* structure of this multi-check CPC cycle ensures that its overall action amounts to an extended measurement of the $\pm\mathbb{1}_D$ operator on the data register. The CPC construction therefore makes it possible to implement arbitrary sequences of parity checks on a quantum data register. As a result, new CPC codes can be discovered by randomly generating new parity sequences or by inheriting them from existing classical codes. In the following sections, we outline general methods for constructing CPC codes that allow for the detection of a full quantum error set.

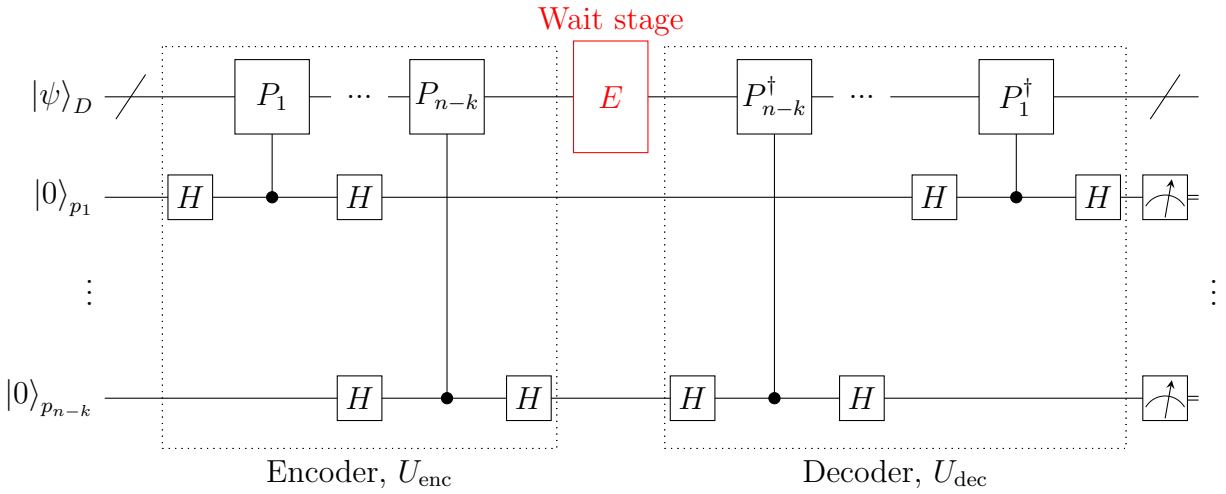


Figure 4.3: A CPC gadget which applies multiple parity checks $\mathcal{P} = \{P_1, ..., P_k\}$ to the register in a single cycle. A multi-check CPC code of this form retains the *encode-error-decode* structure, ensuring that any sequence of parity checks $\mathcal{P}$ can be used without risk of decohering the register.

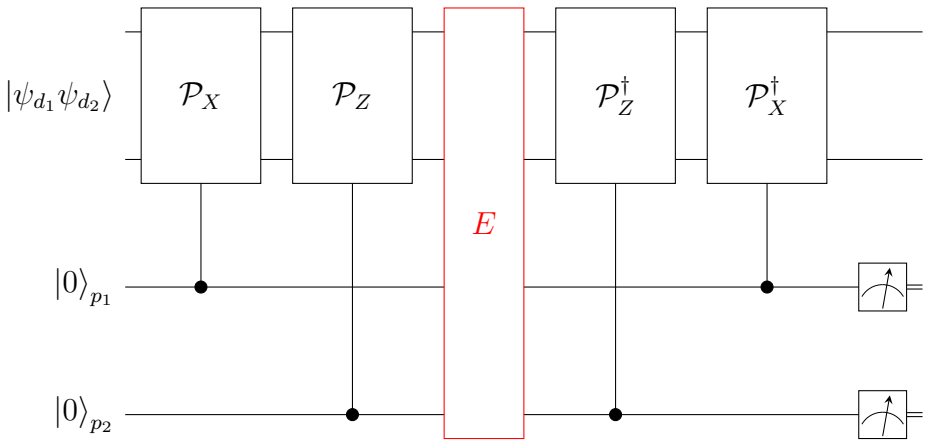


Figure 4.4: A preliminary CPC code formed by partitioning the encoder/decoder into two parity checking stages. The stage labelled $\mathcal{P}_X$ detects X-errors on the register and the stage labelled $\mathcal{P}_Z$ detects Z-errors on the register.

4.2 Construction of a $[[4, 2, 2]]$ CPC detection code

The $[[4, 2, 2]]$ detection code is the simplest quantum protocol to offer protection against a full depolarizing noise channel with single X-, Y- and Z-errors. In section 3.4, we demonstrated how the $[[4, 2, 2]]$ code can be derived via the CSS construction from the starting point of a classical $[4, 3, 2]$ detection code. Here, we show how an equivalent $[[4, 2, 2]]$ code can be constructed in the CPC picture by combining two classical $[3, 2, 2]$ codes.

4.2.1 Translating classical parity checking sequences to a CPC code

Figure 4.4 shows the preliminary *encode-error-decode* CPC setup for a code involving two data qubits $|\psi_{d_1,d_2}\rangle$ and two parity qubits p_1 and p_2. The encoder and decoder have been partitioned into two stages, corresponding to the bit- and phase-checking parts of the code respectively. We now show how the specific form of the parity check sequences, $\mathcal{P}_X$ and $\mathcal{P}_Z$, can be formed from the adjacency matrix of a $[3, 2, 2]$ code.

The classical $[3, 2, 2]$ code was introduced in section 2.2. The code performs a single parity check on two data qubits, and has a generator matrix $G_{[3,2,2]}$ and parity matrix $H_{[3,2,2]}$ given by

$$G_{[3,2,2]} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad H_{[3,2,2]} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}. \tag{4.2.1}$$

A $[3, 2, 2]$ code of this form cannot be used to create a quantum code with with the CSS construction as it does not satisfy the condition of weak self duality[a]. However, as described in the previous section, there are no such restrictions for CPC codes as the *encode-error-decode* structure ensures any parity checking sequence can be implemented

[a]The dual of the $[3, 2, 2]$ code is a $[3, 1, 3]$ code with generator $G_{[3,1,3]} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ and $H_{[3,1,3]} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$. The codewords of this code are given by $\mathcal{C}_{[3,1,3]} = \{000, 111\}$. The codewords of the $[3, 2, 2]$ code are given by $\mathcal{C} = \{000, 011, 101, 110\}$. From this we can see that the $\mathcal{C}^{\perp}_{[3,2,2]} = \mathcal{C}_{[3,1,3]} \not\subset \mathcal{C}_{[3,2,2]}$. The $[3, 2, 2]$ code is therefore not weakly self dual.

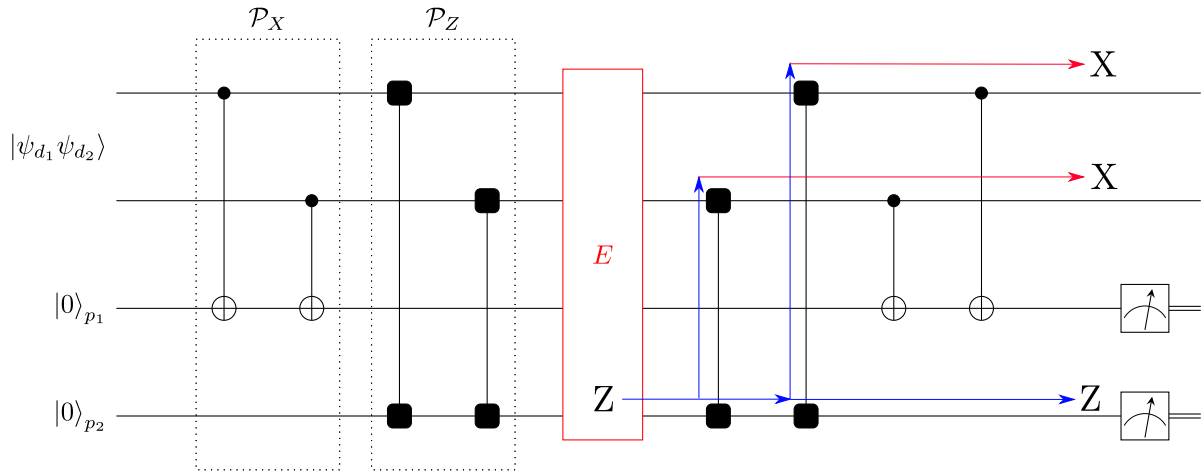
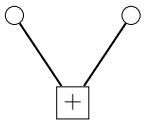


Figure 4.5: The preliminary $[[4, 2, 1]]$ CPC detection code formed by setting the bit and phase checking stages of the encoder/decoder to $\mathcal{P}_X = \{Z_{d_1} Z_{d_2}\}$ and $\mathcal{P}_Z = \{X_{d_1} X_{d_2}\}$. The square gates in this circuit correspond to conjugate-propagator gates that are defined in section 3.6. The arrows show how a Z_{p_2}-error on qubit p_2 is propagated through the decoder. The conjugate-propagator gates in the phase-check stage of the decoder propagate the Z_{p_2}-error as a multi-qubit $X_{d_1} X_{d_2}$-error to the register. The Z_{p_2}-error does not trigger a syndrome in the M_{p_2} measurement at the end of the circuit, as this measurement is performed in the computational basis. Consequently, the Z_{p_2}-error goes undetected, meaning the preliminary code does not correspond to a fully functional detection code. It therefore has distance $d = 1$.

on the register. For the $[3, 2, 2]$ code, the parity checking sequence is described by the adjacency matrix A, obtained from the generator and parity matrices via the relations $G = [\mathbb{1}_{n-k}, A]$ and $H = \left[A^T, \mathbb{1}_k\right]$. The adjacency matrix for the $[3, 2, 2]$ code is therefore given by

$$A_{[3,2,1]} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad (4.2.2)$$



where the factor graph illustrates the connectivity between parity and data bits described by the adjacency matrix. From the above, we see that the parity bit stores the parity of the two data bits. For the purposes of constructing a CPC detection code, we interpret the $[3, 2, 2]$ code's adjacency matrix as a description of the parity checking

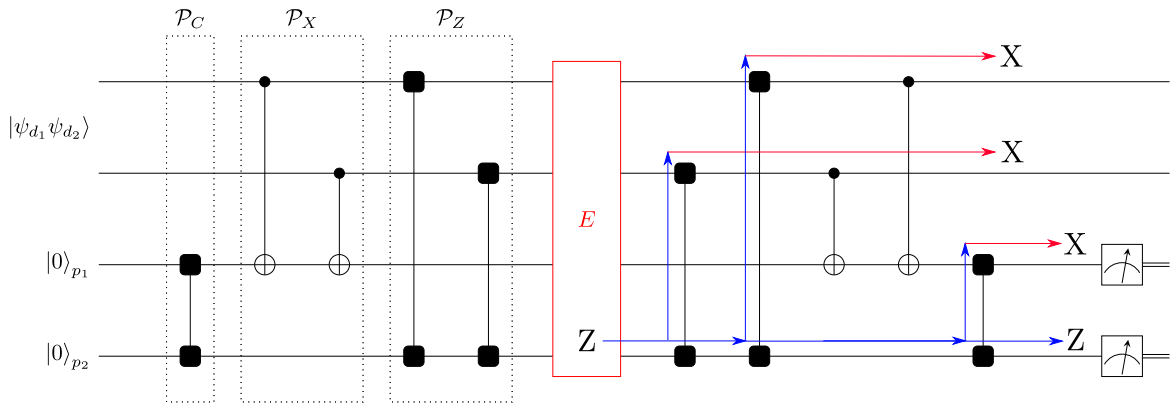


Figure 4.6: The [[4, 2, 2]] CPC code obtained by adding an additional cross-check stage $\mathcal{P}_C$ to the encoder and decoder. The arrows show how the previously considered Z_{p_2}-error is now detected by the cross-check operator in $\mathcal{P}_C$. The Z_{p_2}-error is copied as a X_{p_1}-error to parity qubit p_1 by the conjugate-propagator gate in $\mathcal{P}_C$. This X_{p_1}-error then triggers a '1' syndrome in the M_{p_1} measurement.

sequences $\mathcal{P}_X$ and $\mathcal{P}_Z$ so that

$$\mathcal{P}_X = \{Z_{d_1} Z_{d_2}\}, \quad \mathcal{P}_Z = \{X_{d_1} X_{d_2}\}. \tag{4.2.3}$$

Figure 4.5 shows a CPC code cycle implemented with the parity checking sequences given by $\mathcal{P}_X$ and $\mathcal{P}_Z$. It can be seen that this preliminary circuit corresponds to two implementations of a $[3, 2, 2]$ classical code on the same register, the first to detect bit-flips and the second to detect phase-flips.

4.2.2 Adding cross-check operators

The operation of the preliminary CPC code depicted in figure 4.5 can be analysed by considering the error propagation rules outlined in section 3.6. In order to highlight the shortcomings of this circuit, we consider the specific case of a Z_{p_2}-error on parity qubit p_2. The arrows depict the propagation of this error through the decoder, and show that conjugate propagator gates in the phase-check stage of the decoder propagate the Z_{p_2}-error to the register as a multi-qubit $X_{d_1} X_{d_2}$-error. The Z_{p_2}-error also propagates

Error	**Syndrome**
I	$0_{p_1}0_{p_2}$
$X_A, X_B, X_{p_1}, Z_{p_2}$	$1_{p_1}0_{p_2}$
$Z_A, Z_B, Z_{p_1}, X_{p_2}$	$0_{p_1}1_{p_2}$
$Y_A, Y_B, Y_{p_1}, Y_{p_2}$	$1_{p_1}1_{p_2}$

Table 4.1: The syndrome table for the $[[4, 2, 2]]$ quantum error detection code. If no errors occur, the code returns a '00' syndrome. If a single X, Y or Z error occurs on any of the four qubits, a non-zero syndrome is returned.

to the measurement operator M_{p_2} at the end of the circuit. However, as all syndrome measurements in a CPC circuit are performed in the computational basis, the Z_{p_2}-error does not trigger a '1' syndrome in the M_{p_2} measurement. As a result, we are left with a situation in which a multi-qubit error is propagated to the register in an undetected way. The preliminary CPC circuit in figure 4.5 is therefore a $[[4, 2, 1]]$ code with distance $d = 1$. We now show how the addition of a 'cross-check' operator between the parity qubits can close this undetected error propagation pathway to promote the the circuit to the desired distance of $d = 2$.

Figure 4.6 shows the previously considered $[[4, 2, 1]]$ circuit augmented by an additional conjugate-propagator gate between parity qubits p_1 and p_2. The role of this cross-check stage, labelled $\mathcal{P}_C$ in figure 4.6, is to provide a detection pathway for phase-errors that occur on the parity bits. The arrows in figure 4.6 illustrate how the previously considered Z_{p_2}-error is detected in the new version of the circuit. The cross-check operator copies the Z_{p_2}-error to a X_{p_1}-error on the parity qubit p_2. This X_{p_1} then triggers a '1' syndrome in the M_{p_1} measurement. As the cross-check operator is added to the encoder as well as the decoder, the new circuit retains the *encode-error-decode* CPC code structure. The operation of the code can be verified by computing the syndromes, which are listed in table 4.1. It can be seen that the code can detect all single qubit errors from the set $\{X, Y, Z\}$, and therefore has distance $d = 2$. The circuit depicted in figure 4.6 therefore corresponds to a fully functional $[[4, 2, 2]]$ quantum error detection code.

We have now outlined the basic operation of the [[4, 2, 2]] code, the simplest CPC code capable of detecting errors from a full quantum $\{X, Y, Z\}$ error set. The CPC construction for this code begins by combining two classical $[3, 2, 2]$ codes to detect bit- and phase-errors on the register respectively. The resultant preliminary circuit, however, has a reduced code distance due to undetectable phase-errors on the parity bits. This shortcoming is addressed through the addition of cross-checks which promote the code to distance $d = 2$. The three-stages of the CPC encoder – bit-checks, phase-checks and cross-checks – can be compactly described in terms of CPC adjacency matrices. For example, the [[4, 2, 2]] CPC code has the following adjacency matrices

$$m_b = \begin{matrix} \\ [d_1] \\ [d_2] \end{matrix} \begin{matrix} \begin{matrix} [p1] & [p2] \end{matrix} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \end{matrix}, \quad m_p = \begin{matrix} \\ [d_1] \\ [d_2] \end{matrix} \begin{matrix} \begin{matrix} [p1] & [p2] \end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \end{matrix}, \quad m_c = \begin{matrix} \\ [p1] \\ [p2] \end{matrix} \begin{matrix} \begin{matrix} [p1] & [p2] \end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{matrix}, \tag{4.2.4}$$

where m_b represents the bit-checks, m_p the phase-checks and m_c the cross-checks. For the bit-flip and phase-flip adjacency matrices, m_b and m_p, the rows refer to the data qubits and the columns the parity qubits. Looking at the bit-flip matrix, we can see that both register qubits connect to parity qubit p_1 via CNOT gates in accordance with the circuit in figure 4.6. Likewise, matrix m_P tells us that both register qubits are connected to parity qubit p_2 via conjugate-propagator gates. Finally, from matrix m_c, we see that there is a single cross-check between parity qubits p_1 and p_2. For this [[4, 2, 2]] CPC code, we can make contact between its adjacency matrix representation in equation 4.2.4 and the initial adjacency matrix of the $[3, 2, 2]$ classical code, given by equation 4.2.2. This can be seen by expressing the m_b and m_p adjacency matrices as follows

$$m_b = (A_{[3,2,2]} \mid 0) = \begin{matrix} \\ [d_1] \\ [d_2] \end{matrix} \begin{matrix} \begin{matrix} [p1] & [p2] \end{matrix} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \end{matrix}, \quad m_p = (0 \mid A_{[3,2,2]}) = \begin{matrix} \\ [d_1] \\ [d_2] \end{matrix} \begin{matrix} \begin{matrix} [p1] & [p2] \end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \end{matrix}. \tag{4.2.5}$$

In the above form, it is easy to see how the initial adjacency matrix of the $[3, 2, 2]$ code is used to form the encoder of the [[4, 2, 2]] CPC code.

4.3 The canonical form of CPC codes

The [[4,2,2]] quantum error detection code illustrates the basic principles behind the operation of a CPC code. The encoder of the $[[4,2,2]]$ code involves successive stages of cross-checks, bit-checks and phase-checks. We now generalise this three-part CPC structure to provide a canonical form for the construction of arbitrary $[[n,k,d]]$ CPC codes.

The canonical form of an [[n,k,d]] CPC code is shown in figure 4.7. Such codes have k data qubits, $|\psi\rangle_D = |\psi_{d_1}\psi_{d_2}...\psi_{d_k}\rangle$, and $m = n - k$ parity qubits, $|0\rangle_P = |0_{p_1}0_{p_2}...0_{p_m}\rangle$. As with the $[[4,2,2]]$ detection scheme, the encode stage of a general CPC code involves successive rounds of cross-checks, bit-checks and then phase-checks. The cross-checks are realised as conjugate-propagator gates between the parity qubits (represented by the green gate in figure 4.7). The bit-checks are realised as CNOT gates where the register qubits act as the control and the parity qubits the target (represented by the blue gate in figure 4.7). Finally, the phase-checks are realised as conjugate-propagator gates between the data qubits and parity qubits (represented by the red gates in figure 4.7). The sequence of gates within each stage of the encoder can be compactly described in terms of adjacency matrices of the form

$$m_b = \begin{array}{c} \\ [D_1] \\ [D_2] \\ [...] \\ [D_k] \end{array} \begin{array}{c} \begin{array}{cccc} [p_1] & [p_2] & [...] & [p_m] \end{array} \\ \begin{pmatrix} b_{11} & b_{12} & ... & b_1 \\ b_{21} & b_{22} & ... & b_{2m} \\ ... & ... & ... & ... \\ b_{k1} & b_{k2} & ... & b_{km} \end{pmatrix} \end{array}, \quad \mathrm{m_p} = \begin{array}{c} \\ [D_1] \\ [D_2] \\ [...] \\ [D_k] \end{array} \begin{array}{c} \begin{array}{cccc} [p_1] & [p_2] & [...] & [p_m] \end{array} \\ \begin{pmatrix} h_{11} & h_{12} & ... & h_{1m} \\ h_{21} & h_{22} & ... & h_{2m} \\ ... & ... & ... & ... \\ h_{k1} & h_{k2} & ... & h_{km} \end{pmatrix} \end{array},$$

$$m_c = \begin{array}{c} \\ [p_1] \\ [p_2] \\ [...] \\ [p_{(m-1)}] \\ [p_m] \end{array} \begin{array}{c} \begin{array}{ccccc} [p_1] & [p_2] & [...] & [p_{m-1}] & [p_m] \end{array} \\ \begin{pmatrix} 0 & c_{12} & ... & c_{1(m-1)} & c_{1m} \\ c_{12} & 0 & ... & c_{2(m-1)} & c_{2m} \\ ... & ... & ... & ... & ... \\ c_{1(m-1)} & c_{2(m-1)} & ... & 0 & c_{(m-1)m} \\ c_{1m} & c_{2m} & ... & c_{(m-1)m} & 0 \end{pmatrix} \end{array}, \tag{4.3.1}$$

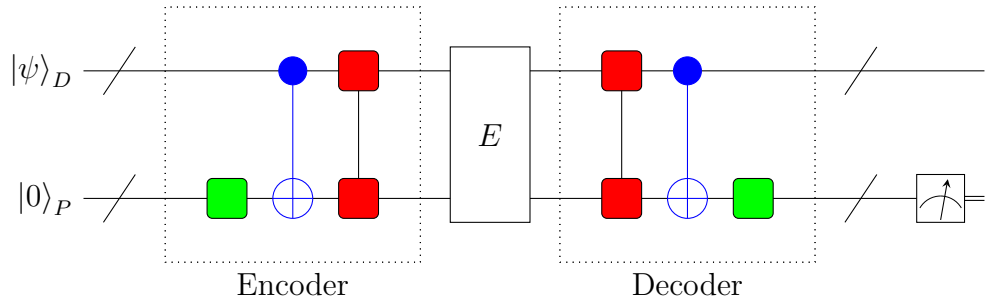


Figure 4.7: The canonical form of CPC codes, showing the symmetric *encode-error-decode* structure. In an $[[n, k, d]]$ CPC code the qubits are split into two distinct types: k data qubits, $|\psi\rangle_D = |\psi_{d_1}\psi_{d_2}...\psi_{d_k}\rangle$, and $n-k$ parity qubits, $|0\rangle_P = |0_{p_1}0_{p_2}...0_{p_{n-k}}\rangle$. The encoder involves successive rounds of cross-checks (green), bit-checks (blue) and phase-checks (red). The decoder is simply the unitary inverse of the encoder.

where b_{xy}, h_{xy}, c_{xy} are binary values. The cross-check matrix m_c is always symmetric about the diagonal.

4.4 Computing the stabilizers & Pauli logical operators of a CPC code

The CPC framework provides a fail-safe code structure that enables a stabilizer code to be derived from the starting point of any sequence of parity checks. In this section, we explain how the code stabilizers and Pauli logical operators can be computed for a CPC code expressed in the canonical form described in section 4.3.

4.4.1 Computing the stabilizers of a CPC code

For an $[[n, k, d]]$ CPC code, the code qubits are separated into k data qubits $|\psi_{d_1}...\psi_{d_k}\rangle$ and $n-k$ parity bits $|0_{p_1}...0_{p_{n-k}}\rangle$. As the parity bits are always initialised deterministically in the $|0_{p_i}\rangle$ state, they correspond to a constrained 2^{n-k} subspace of the input. Conversely, the register corresponds to the 2^k dimensional subspace in which the data

is initially contained. The encoder is split into three parts corresponding to the cross-checking, bit-checking and phase-checking stages respectively. The encoder U_{CPC} can therefore be described as follows

$$U_{\text{CPC}} = U_P \cdot U_B \cdot U_C, \tag{4.4.1}$$

where U_C, U_B and U_P are the unitary operators corresponding to the three stages of the encoder. Each stage of the encoder is composed of either conjugate-propagator gates (for U_C and U_P) or CNOT gates (for U_B), the exact sequence of which is specified by the adjacency matrices m_b, m_p and m_c. As a result, the encoder will always be a Clifford operation which transforms stabilizer states to stabilizer states (for an outline of the Clifford group see appendix B). The stabilizers of a CPC code can therefore be computed simply by conjugating the stabilizers of the constrained qubits of the input. For a CPC code adhering to the canonical structure, the constrained qubits are the parity bits which are prepared in the $|0_{p_i} 0_{p_2}, ..., 0_{p_{n-k}}\rangle$ state, and are stabilized by

$$\mathcal{S}_{\text{input}} = \{S_i, S_2, ..., S_{n-k}\} = \{Z_{p_1}, Z_{p_2}, ..., Z_{p_{n-k}}\} \tag{4.4.2}$$

It is immediately clear that the initial set of constrained stabilizers $\mathcal{S}_{\text{input}}$ are mutually commuting, as each act on separate qubits. The stabilizers of a CPC code $\mathcal{S}_{\text{CPC}} = \{S_i, S_2, ..., S_{n-k}\} = \{Z_{p_1}, Z_{p_2}, ..., Z_{p_{n-k}}\}$ are computed as follows

$$S_i = U_{\text{CPC}} \cdot Z_{p_i} \cdot U_{\text{CPC}}^\dagger \tag{4.4.3}$$

As required, the resultant set of stabilizers $\mathcal{S}_{\text{CPC}}$ will be mutually commuting. This follows from the fact that the stabilizers of the input $\mathcal{S}_{\text{input}}$ are mutually commuting, and that commutation relations are preserved when the stabilizers are evolved by a unitary operator[b].

[b]Theorem: Unitary operators preserve commutation relations between stabilizers.
Proof: Consider two commuting stabilizers s_1 and s_2 such that $[s_1, s_2] = 0$. If these stabilizers are evolved by a unitary operator U, the resultant states are given by $Us_1U^\dagger$ and $Us_2U^\dagger$. The new commutation relation between the evolved stabilizers can then be computed as $[Us_1U^\dagger, Us_2U^\dagger] = Us_1U^\dagger Us_2U^\dagger - Us_2U^\dagger Us_1U^\dagger = U[s_1s_2 - s_2s_1]U^\dagger = U[s_1, s_2]U^\dagger = 0$. Unitary evolution therefore preserves the commutation relations between stabilizers.

4.4.2 Computing the Pauli logical operators of a CPC code

The Pauli logical operators for a CPC code can be computed via a similar method as the stabilizers. Prior to encoding, each data bit in the register has two Pauli logical operators of form

$$\mathcal{L}_{\text{initial}} = \{X_{d_1}, Z_{d_1}, ..., X_{d_k}, Z_{d_k}\} \tag{4.4.4}$$

Note that the initial Pauli logical operator pairs X_{d_i} and Z_{d_i} anti-commute with one another by definition. As they are acting on different qubits, the initial set of Pauli logical operators $\mathcal{L}_{\text{initial}}$ commutes with $\mathcal{S}_{\text{initial}}$. The encoded logical operators $\bar{X}_{d_i}$ and $\bar{Z}_{d_i}$ can be calculated by evolving the initial Pauli logical operators as follows

$$\bar{Z}_{d_i} = U_{\text{CPC}} \cdot Z_{d_i} \cdot U_{\text{CPC}}^{\dagger}, \quad \bar{X}_{d_i} = U_{\text{CPC}} \cdot X_{d_i} \cdot U_{\text{CPC}}^{\dagger}. \tag{4.4.5}$$

The resultant logical operators of the CPC code $\mathcal{L}_{\text{CPC}} = \{\bar{X}_{d_i}, \bar{Z}_{d_i}, ..., \bar{X}_{d_k}, \bar{Z}_{d_k}\}$ then commute with the encoded stabilizers $\mathcal{S}_{\text{CPC}}$. This again follows from the fact that unitary operators preserve commutation relations betwen stabilizer operators. Similar reasoning can be applied to demonstrate that the relation $[\bar{X}_{d_i}, \bar{Z}_{d_i}] = 1$ holds for the encoded logical operators.

4.4.3 Example: Computing the stabilizers and logical Pauli operators of the $[[4, 2, 2]]$ CPC detection code

The $[[4, 2, 2]]$ CPC detection code was introduced in section 4.2, and the corresponding circuit is illustrated in figure 4.6. We now show how the stabilizers and logical operators of the this code can be computed using equations 4.4.3 and 4.4.5.

The $[[4, 2, 2]]$ code has two data qubits and two parity qubits. The initial stabilizers of the code, prior to encoding, are therefore given by $\mathcal{S}_{\text{initial}} = \{I_1 I_2 Z_3 I_4, I_1 I_2 I_3 Z_4\}$. Likewise, the initial logical operators are given by $\mathcal{L}_{\text{initial}} = \{X_1 I_2 I_3 I_4, I_1 X_2 I_3 I_4, Z_1 I_2 I_3 I_4, I_1 Z_2 I_3 I_4\}$. The corresponding encoded versions of these sets can then be calculated using equations 4.4.3 and 4.4.5. This can be achieved by hand, or using a stabilizer simulator such as [49, 50]. The resultant encoded stabilizers

and logical Pauli operators are then given by

$$\mathcal{S}_{[[4,2,2]]} = \{Z_1Z_2Z_3X_4, X_1X_2X_3Z_4\} \tag{4.4.6}$$

$$\mathcal{L}_{[[4,2,2]]} = \{X_1I_2X_3I_4, I_1X_2X_3I_4, Z_1I_2I_3X_4, I_1Z_2I_3X_4\} \tag{4.4.7}$$

Up to a Hadamard gate on the qubit p_2, the stablizers of the CPC $[[4,2,2]]$ code are equivalent to the stabilizers of the $[[4,2,2]]$ code derived using CSS methods in section 3.4. By inspection of the logical operators, it can also be verified that this code has distance $d = 2$.

4.4.4 Construction of stabilizer tables and Pauli logical operators from the CPC adjacency matrices

The finite geometry notation for quantum stabilizer codes was introduced in section 3.5. We now show how, using the finite geometry formulation, the stabilizer tables and Pauli logical operators of a CPC code can be computed directly from its adjacency matrices. For a CPC code with adjacency matrices m_b, m_p and m_c, the quantum parity check matrix is given by

$$G_{XZ}(\mathcal{S}_{\text{CPC}}) = \begin{array}{c} \overset{[d_1,\ldots,d_k]}{m_p^T} \quad \overset{[p_1,\ldots,p_{n-k}]}{m_b^T \cdot m_p \oplus m_c} \;\Big|\; \overset{[d_1,\ldots,d_k]}{m_b^T} \quad \overset{[p_1,\ldots,p_{n-k}]}{\mathbb{1}_{n-k}} \end{array}, \tag{4.4.8}$$

where each row corresponds to a finite geometry representation of a stabilizer from the set $\mathcal{S}_{\text{CPC}}$. Similarly the Pauli logical operators can be written as follows

$$G_{XZ}(\mathcal{L}_{\text{CPC}}) = \begin{array}{c} \\ \mathcal{L}_X\{ \\ \mathcal{L}_Z\{ \end{array} \begin{array}{c} \overset{[d_1,\ldots,d_k]}{\mathbb{1}_k} \quad \overset{[p_1,\ldots,p_{n-k}]}{m_b} \;\Big|\; \overset{[d_1,\ldots,d_k]}{\mathbf{0}} \quad \overset{[p_1,\ldots,p_{n-k}]}{\mathbf{0}} \\ \mathbf{0} \quad m_p \;\Big|\; \mathbb{1}_k \quad \mathbf{0} \end{array}, \tag{4.4.9}$$

where the first line corresponds to the Pauli-X logical operators and the second the Pauli-Z logical operators.

As an example, we again consider the case of the $[[4, 2, 2]]$ code with the adjacency matrices defined in equation 4.2.5. Substituting equation 4.2.5 into equation 4.4.8 gives the following quantum parity check matrix

$$G_{XZ}(\mathcal{S}_{[[4,2,2]]}) = \left(\begin{array}{cccc|cccc} \overset{[d_1]}{0} & \overset{[d_2]}{0} & \overset{[p_1]}{0} & \overset{[p_2]}{1} & \overset{[d_1]}{1} & \overset{[d_2]}{1} & \overset{[p_1]}{1} & \overset{[p_2]}{0} \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right). \tag{4.4.10}$$

It can easily be verified that two rows of the above matrix describe the same stabilizers as defined in equation 4.4.6. Similarly, the Pauli logical operators of the $[[4, 2, 2]]$ code can be found by substituting equation 4.2.5 into equation 4.4.9 to give

$$G_{XZ}(\mathcal{L}_{[[4,2,2]]}) = \left(\begin{array}{cccc|cccc} \overset{[d_1]}{1} & \overset{[d_2]}{0} & \overset{[p_1]}{1} & \overset{[p_2]}{0} & \overset{[d_1]}{0} & \overset{[d_2]}{0} & \overset{[p_1]}{0} & \overset{[p_2]}{0} \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right). \tag{4.4.11}$$

where the first two rows corresponds to the Pauli-X logical operators $\mathcal{L}_X$ and the final two rows the Pauli-Z logical operators $\mathcal{L}_Z$. Again, it can be seen that the above matrix is the finite geometry representation of the Pauli logical operators defined in 4.4.7.

4.4.5 Efficient calculation of the CPC code syndromes

The finite geometry representation of the CPC code stabilizers, defined in equation 4.4.3, can be used to efficiently calculate the code syndromes. In section 3.5, it was shown that code syndromes can be calculated via the relation

$$\vec{S}_{\text{CPC}}(E) = G_{ZX}(E) \cdot (G_{XZ}(\mathcal{S}_{FG}))^T, \tag{4.4.12}$$

where $G_{XZ}(\mathcal{S}_{\text{CPC}})$ is the finite geometry representation of the stabilizers and $G_{ZX}(E)$ is the finite geometry representation of the error vector E. Note that for $G_{ZX}(E)$ the

positions of the X and Z components of the array have been switched. For a CPC code the error vector term can be written as follows

$$G_{ZX}(E) = \begin{array}{c} \overset{[d_1,\ldots,d_k]}{} \quad \overset{[p_1,\ldots,p_{n-k}]}{} \quad \overset{[d_1,\ldots,d_k]}{} \quad \overset{[p_1,\ldots,p_{n-k}]}{} \\ \left(\begin{array}{cc|cc} \mathcal{E}_{dz} & \mathcal{E}_{pz} & \mathcal{E}_{dx} & \mathcal{E}_{pz} \end{array} \right) \end{array}, \tag{4.4.13}$$

where the four terms correspond to the different components on the error vector acting on the data and parity qubits. As an example, consider the case of a two-qubit error of form $E = X_{d_1} I_{d_2} Z_{p_1} I_{p_2}$. In this case $\mathcal{E}_{dx} = (1 \quad 0)$ and $\mathcal{E}_{pz} = (1 \quad 0)$, so that $G_{XZ}(E) = (0 \quad 0 \quad 1 \quad 0 \quad | \quad 1 \quad 0 \quad 0 \quad 0)$. Substituting equation 4.4.13 into equation 4.4.12 gives the following general expression for the syndrome of a CPC code

$$\begin{aligned} \vec{S}_{\text{CPC}}(E) &= G_{ZX}(E) \cdot (G_{XZ}(\mathcal{S}_{\text{CPC}}))^T \\ &= (\mathcal{E}_{dz} \quad \mathcal{E}_{pz} \quad | \quad \mathcal{E}_{dx} \quad \mathcal{E}_{px})(m_p^T \quad m_b^T \cdot m_p \oplus m_c \quad | \quad m_b^T \quad \mathbb{1}_{n-k})^T \\ &= \left(\mathcal{E}_{dz} \cdot m_p + \mathcal{E}_{pz} \cdot m_p^T \cdot m_b \oplus m_c + \mathcal{E}_{dx} \cdot m_b + \mathcal{E}_{px}\right) \bmod 2. \end{aligned} \tag{4.4.14}$$

The above expression allows syndrome tables to be quickly calculated, bypassing the need to carry out a full-matrix or stabilizer circuit simulation. For example, the single-error syndrome table for set depolarizing Pauli noise channel $\{X, Y, Z\}$ can be easily constructed from the adjacency matrices as follows

$$\begin{array}{c} \overset{[p_1,\ldots,p_{n-k}]}{} \\ \begin{array}{l} [E_{dx}] \\ [E_{dz}] \\ [E_{dy}] \\ [E_{px}] \\ [E_{pz}] \\ [E_{py}] \end{array} \begin{bmatrix} m_b \\ m_p \\ m_b \oplus m_p \\ \mathbb{1}_{n-k} \\ m_p^T \cdot m_b \oplus m_c \\ m_p^T \cdot m_b \oplus m_c \oplus \mathbb{1}_{n-k} \end{bmatrix} \end{array}. \tag{4.4.15}$$

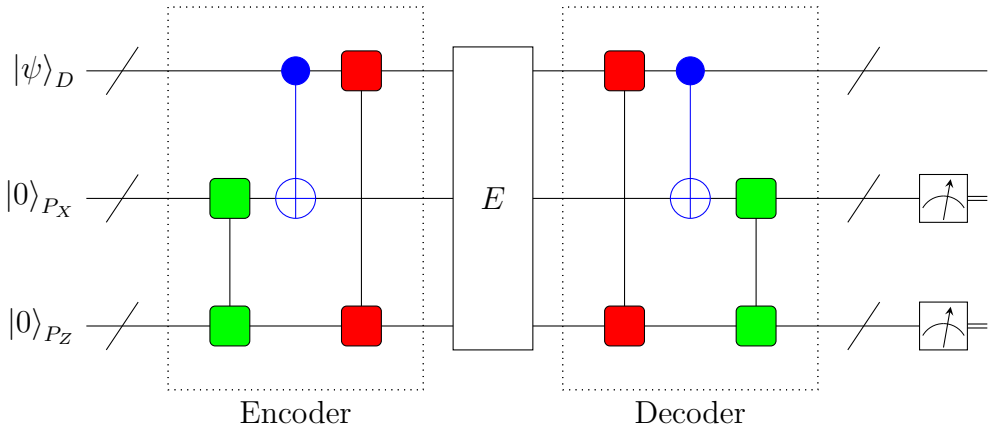


Figure 4.8: The tripartite construction for CPC codes. The parity check qubits are split into two blocks P_X and P_Z. The X-errors on the register are detected by implementing sequence of bit checks (depicted by the blue gate) between the register and block P_X. Likewise, Z-errors are detected by implementing phase-checks (depicted by the red gate) that are copied to P_Z. Cross-checks (depicted by the green gate) are also applied to ensure that errors on the parity qubits themselves can be detected. *This figure was originally published in* [2].

4.5 Tripartite CPC codes

In this section we outline methods for the construction of distance-three CPC codes. The codes we introduce are based on a tripartite structure in which the code qubits are separated into blocks of data qubits, bit-parity qubits and phase-parity qubits. The structure of tripartite CPC codes is illustrated in figure 4.8, where it can be seen that the bit-check stage interacts only with the parity qubit block P_X and the phase-check stage only with the parity qubit block P_Z. In accordance with the canonical structure outlined in section 4.3, tripartite CPC codes also include a cross-check stage to ensure that phase-errors on the parity bits can be detected. The specific advantage of adopting a tripartite structure is that it provides a setting in which pairs of classical codes can easily be combined to form a quantum CPC code. The $[[4, 2, 2]]$ CPC code outlined in section 4.2 is an example of a tripartite distance-two code constructed in this way. We now show how this approach can be generalised to distance-three codes capable of detecting and localising single-qubit errors. Note that the tripartite construction

presented here differs slightly from the version first outlined in [1]. The changes are designed to make the tripartite structure more flexible in the types of classical code it can accept.

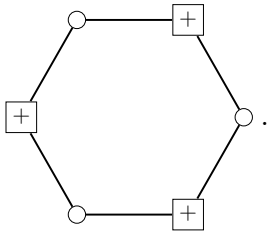
4.5.1 Construction of a $[[9,3,3]]$ tripartite CPC code

In this section we show how a $[[9,3,3]]$ tripartite CPC code can be constructed by combining two copies of a classical ring code. The ring code we consider is the $[6,3,3]$ code with the following generator and parity check matrices

$$G_{[6,3,3]} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \quad H_{[6,3,3]} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{4.5.1}$$

Using the relation $G = [\mathbb{1}_{n-k}, A]$, the adjacency matrix and corresponding factor graph for the $[6,3,3]$ ring code can be obtained as

$$A_{[6,3,3]} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad$$



. (4.5.2)

To create a tripartite CPC code from $A_{[3,2,2]}$, we chose m_b and m_p CPC adjacency matrices of the form

$$m_b = \left(A_{[3,2,2]} \mid \mathbf{0}\right) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}, \tag{4.5.3}$$

$$m_p = \left(\mathbf{0} \mid A_{[3,2,2]}\right) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}, \tag{4.5.4}$$

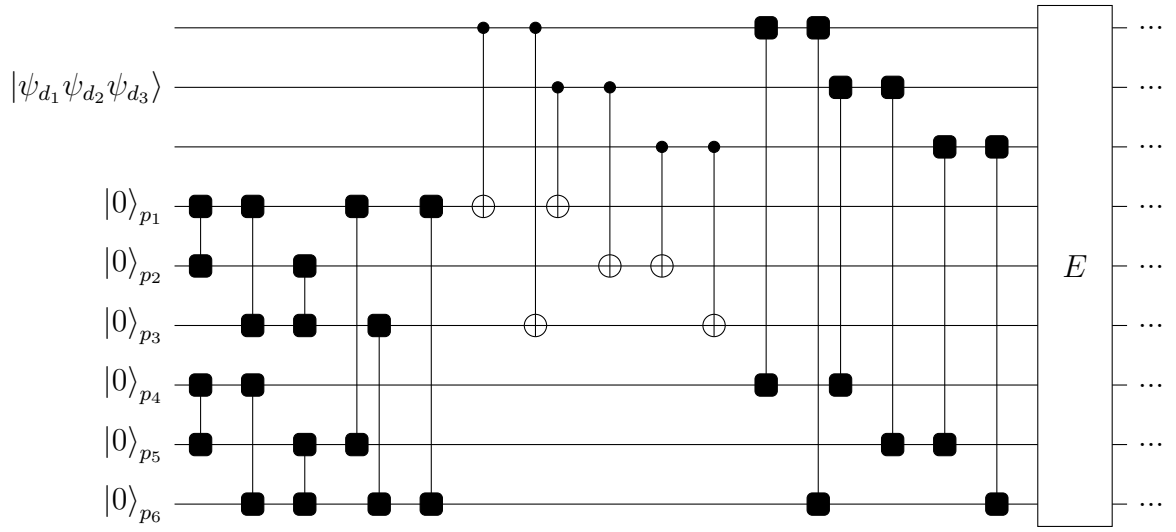


Figure 4.9: The encoder for the $[[9, 3, 3]]$ tripartite CPC code. The adjacency matrices for this encoder are given by equations 4.5.3, 4.5.4 and 4.5.5.

where the CNOT gates specified by m_b interact with the first three parity qubits and the conjugate-propagator gates specified by m_p interact with the final three parity qubits. The final step in constructing a CPC code is to choose a cross-check matrix that fixes the code distance to $d = 3$. For this particular example, we select a cross-check matrix of the form

$$m_c = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}, \tag{4.5.5}$$

to give the encoder shown in figure 4.9. The syndromes for this code can be calculated using equation 4.4.12 (or any other quantum circuit simulation technique) and are shown in table 4.2. By inspection, it can easily be verified that there are unique syndromes for single qubit X-, Z- and Y-errors on all nine of the code qubits. Consequently, the tripartite code with the encoder depicted in figure 4.9 is a $[[9, 3, 3]]$ CPC code.

		X-error syndromes m_b	Z-error syndromes m_p	Y-error syndromes $m_b \oplus m_p$
data qubits	d_1	101000	000101	101101
	d_2	110000	000110	110110
	d_3	011000	000011	011011
		X-error syndromes $\mathbb{1}_6$	Z-error syndromes $m_p^T \cdot m_b \oplus m_c$	Y-error syndromes $\mathbb{1}_6 \oplus m_p^T \cdot m_b \oplus m_c$
parity qubits	p_1	100000	011010	111010
	p_2	010000	101001	111001
	p_3	001000	110100	111100
	p_4	000100	010011	010111
	p_5	000010	001101	001111
	p_6	000001	100110	100111

Table 4.2: The syndrome table for the $[[9, 3, 3]]$ tripartite CPC code. Each syndrome consists of a string of six bits. The first three bits, coloured red, represent measurement outcomes from the parity qubits in block P_X. The final three bits, coloured blue, represent measurement outcomes from the parity qubits in block P_Z.

4.5.2 Tripartite CPC Hamming codes

In this section we provide a general method for translating any classical $[N, K, 3]$ Hamming code into a $[[2N-K, K, 3]]$ quantum CPC code. This method makes use of the the tripartite structure depicted in figure 4.8, and provides a standard form for the adjacency matrix m_c. Our construction is encapsulated by the following theorem:

Theorem 1. *Let L_k be a classical $[N, K, 3]$ Hamming code with adjacency matrix A_{L_K}. For any such L_k it is always possible to construct a tripartite $[[n = 2N-K, k = K, d = 3]]$ quantum CPC code defined by bit-check matrix $m_b = (A_{L_K} \mid \mathbf{0})$, phase-check matrix $m_p = (\mathbf{0} \mid A_{L_K})$ and cross-check matrix*

$$m_c = \begin{array}{c} \\ [P_X] \\ [P_Z] \end{array} \begin{array}{c} \begin{array}{cc} [P_X] & [P_Z] \end{array} \\ \left(\begin{array}{c|c} \mathbf{1}_{m/2} \oplus \mathbb{1}_{m/2} & \mathbf{w}_{m/2} \\ \hline (\mathbf{w}_{m/2})^T & \mathbf{1}_{m/2} \oplus \mathbb{1}_{m/2} \end{array} \right) \end{array}, \tag{4.5.6}$$

where $m = n - k$ and $\mathbf{1}_{m/2}$ is a square matrix of ones of dimension $m/2$. The permutation matrix $\mathbf{w}_{m/2}$ is constructed by shifting each row in the identity matrix $\mathbb{1}_{m/2}$ by one column to the right.

Proof. To prove the above theorem, we consider the syndromes for single X-, Z- and Y- errors on the data qubits and the parity qubits. Our aim is to show that distinct syndromes are produced for each single-qubit error so that the code distance is $d = 3$. In the following we write each syndrome measurement in the form

$$\vec{S}_i^q = (s_{X_i} \mid s_{Z_i}), \tag{4.5.7}$$

where the index i corresponds to the qubit label and q the error-type. The s_{X_i} component describes the portion of the syndrome measured in bit parity check block P_X, and s_{Z_i} the component measured in phase parity check block P_Z. This partition of the parity check qubits is described in figure 4.8 which shows the tripartite CPC structure. In this proof we will also consider syndrome matrices S^q constructed by stacking individual syndromes $\vec{S}_i^q$.

Data qubits: X- and Z-errors. Bit and phase errors on the the data qubits of the tripartite CPC Hamming codes are propagated to the parity qubits by the gates described by the check matrices $m_b = (A_{L_K} \mid \mathbf{0})$ and $m_p = (\mathbf{0} \mid A_{L_K})$. The adjacency matrix A_{L_K} is derived from a classical Hamming code and as such will provide a unique syndrome for errors on the data bits. The general construction for classical Hamming codes is outlined in section 2.4. The syndrome for a X-error on data qubit i corresponds to row i of the matrix m_b, and will therefore have the form $\vec{S}_i^{dx} = (A_{L_K} \mid \mathbf{0})_i$. Likewise Z-errors on the data qubits will produce syndromes of the form $\vec{S}_i^{dz} = (\mathbf{0} \mid A_{L_K})_i$. As they are detected in different blocks of the parity qubits, there is no overlap between the syndromes for X- and Z- errors on the data qubits. The final property to ascertain about the error syndromes is their weight. By construction, the rows of the Hamming code adjacency matrix have weight $W(A_{L_K}) \geq 2$. The X- and Z- errors on the data qubits in the CPC code will therefore also have weight greater than or equal to two. These syndrome properties are summarised in table 4.3.

Data qubits: Y-errors. Y-errors can be thought of as the simultaneous occurrence of an X-error and a Z-error on the same qubit. As a result, the syndrome for a Y-error on the data qubits of a tripartite CPC Hamming code will have the form $\vec{S}_i^{dy} = (A_{L_K} \mid A_{L_K})_i$. They are therefore distinguishable from the previously considered X- and Z-errors because they are detected in both the P_X and P_Z parity qubit blocks. The weight of these Y-errors syndromes will be greater than or equal to 4.

Parity qubits: X errors. Single X-errors on the parity qubits produce a syndrome of weight one of the form $\vec{S}_i^{px} = \mathbb{1}_{m_i}$. It is immediately clear that these syndromes will be distinct from one another, as well as from the syndromes for the previously considered errors.

Parity qubits: Z errors. Z-errors on the parity qubits of a tripartite CPC Hamming code are detected by other parity bits via the relation $m_p^T \cdot m_b \oplus m_c$. Substituting in the expressions for m_b, m_p and m_c specified in theorem 1, we get the following syndrome

		X-error syndromes m_b	Z-error syndromes m_p	Y-error syndromes $m_b \oplus m_p$
data qubits	$\vec{S}_i^d = (s_{X_i}\|s_{Z_i})$	$(A_{L_k} \mid \mathbf{0})_i$	$(\mathbf{0} \mid A_{L_k})_i$	$(A_{L_k} \mid A_{L_k})_i$
	$W(s_{X_i})$	≥ 2	0	≥ 2
	$W(s_{Z_i})$	0	≥ 2	≥ 2
		X-error syndromes $\mathbb{1}_m$	Z-error syndromes $m_p^T \cdot m_b \oplus m_c$	Y-error syndromes $\mathbb{1}_m \oplus m_p^T \cdot m_b \oplus m_c$
parity qubits	$\vec{S}_i^p = (s_{X_i}\|s_{Z_i})$	$\vec{S}_i^{px} = (\mathbb{1}_m)_i$	$\vec{S}_i^{pz}$	$\vec{S}_i^{py} = \vec{S}_i^{px} \oplus \vec{S}_i^{pz}$
	$W(s_{X_i})$	$\begin{cases} 1, & \text{if } i \leq m/2 \\ 0, & \text{if } i > m/2 \end{cases}$	$\begin{cases} m/2-1, & \text{if } i \leq m/2 \\ 2, & \text{if } i > m/2 \end{cases}$	$\begin{cases} m/2, & \text{if } i \leq m/2 \\ 2, & \text{if } i > m/2 \end{cases}$
	$W(s_{Z_i})$	$\begin{cases} 0, & \text{if } i \leq m/2 \\ 1, & \text{if } i > m/2 \end{cases}$	$\begin{cases} 1, & \text{if } i \leq m/2 \\ m/2-1, & \text{if } i > m/2 \end{cases}$	$\begin{cases} 1, & \text{if } i \leq m/2 \\ m/2, & \text{if } i > m/2 \end{cases}$

Table 4.3: Table showing the structure and weights of the error signatures resulting from different error types in a $[[n, k, d]]$ tripartite CPC Hamming code. Each syndrome $\vec{S}_i = (s_{X_i}|s_{Z_i})$ consists of a string of bits of length $m = n - k$. The first part s_{X_i} corresponds to the measurement outcomes from parity qubit block P_X. The second part s_{Z_i} corresponds to the measurement outcomes from parity qubit block P_Z. The syndrome vectors $\vec{S}_i^{pz}$ and $\vec{S}_i^{py}$ are rows of the syndrome matrices defined in equations 4.5.10 and 4.5.12 respectively.

matrix for Z-errors on the parity qubits

$$\begin{aligned} S^{pz} &= m_p^T \cdot m_b \oplus m_c \\ &= \begin{matrix} [P_X] \\ [P_Z] \end{matrix} \left(\begin{array}{c|c} \overset{[P_X]}{\mathbf{1}_{m/2} \oplus \mathbb{1}_{m/2}} & \overset{[P_Z]}{\mathbf{w}_{m/2}} \\ \hline (A_{L_K})^T \cdot A_{L_K} \oplus (\mathbf{w}_{m/2})^T & \mathbf{1}_{m/2} \oplus \mathbb{1}_{m/2} \end{array} \right). \end{aligned} \tag{4.5.8}$$

For an $[N, K, 3]$ Hamming code with adjacency matrix A_{L_K}, the following relation holds $(A_{L_K})^T \cdot A_{L_K} = \mathbb{1}_{N-K}$. This follows from the fact that the Hamming code parity check matrix H_L is orthogonal such that $H \cdot H^T = \mathbf{0}$ [90]. By noting that the parity check

matrix has the form $H = [A^T|\mathbb{1}]$, the Hamming code code orthogonality condition can be expressed as follows

$$H_{L_K} = \Big[(A_{L_K})^T \quad | \quad \mathbb{1}_{N-K}\Big] \left[\begin{array}{c} A_{L_K} \\ \hline \mathbb{1}_{N-K} \end{array}\right] = (A_{L_K})^T \cdot A_{L_K} \oplus \mathbb{1}_{N-K} = \mathbf{0}. \tag{4.5.9}$$

The above expression can then be rearranged to give the desired equality $(A_{L_K})^T \cdot A_{L_K} = \mathbb{1}_{N-K}$. Substituting this into equation 4.5.8 gives the final form of the syndrome matrix for Z-which occur on the parity qubits of a tripartite CPC Hamming code

$$S^{pz} = \begin{array}{c} \\ {\scriptstyle [P_X]} \\ {\scriptstyle [P_Z]} \end{array} \begin{array}{c} \begin{array}{cc} {\scriptstyle [P_X]} & {\scriptstyle [P_Z]} \end{array} \\ \left(\begin{array}{c|c} \mathbf{1}_{m/2} \oplus \mathbb{1}_{m/2} & \mathbf{w}_{m/2} \\ \hline \mathbb{1}_{m/2} \oplus (\mathbf{w}_{m/2})^T & \mathbf{1}_{m/2} \oplus \mathbb{1}_{m/2} \end{array}\right) \end{array}, \tag{4.5.10}$$

where we make use of the fact $m/2 = N - K$. The term $\mathbf{1}_{m/2} \oplus \mathbb{1}_{m/2}$ has unique rows, and from this it follows the matrix S has unique rows representing distinct syndromes for Z-errors on the parity bits. We now need to verify that the form of these syndromes is distinct compared to the previously considered errors. This can again be achieved by examining the syndrome weights and their distributions across the two parity check blocks P_X and P_Z. Calculating the weights of the rows in each block of the syndrome matrix in equation 4.5.10 gives

$$W(S^{pz}) = \begin{array}{c} \\ {\scriptstyle [P_X]} \\ {\scriptstyle [P_Z]} \end{array} \begin{array}{c} \begin{array}{cc} {\scriptstyle [P_X]} & {\scriptstyle [P_Z]} \end{array} \\ \left(\begin{array}{c|c} m/2 - 1 & 1 \\ \hline 2 & m/2 - 1 \end{array}\right) \end{array}. \tag{4.5.11}$$

The structure of these syndromes is different to those previously seen. Therefore the code produces unique syndromes for all single-qubit X- and Z- errors on the code qubits.

Parity bits: Y-errors. The final error type to consider for the tripartite CPC Hamming code construction are Y-errors on the parity qubits. The syndrome matrix for this error

matrix is constructed as follows

$$S^{py} = S^{px} \oplus S^{pz} = \begin{matrix} \\ [P_X] \\ [P_Z] \end{matrix} \begin{pmatrix} \begin{matrix} [P_X] \\ \mathbf{1}_{m/2} \end{matrix} & \begin{matrix} [P_Z] \\ \mathbf{w}_{m/2} \end{matrix} \\ \mathbb{1}_{m/2} \oplus (\mathbf{w}_{m/2})^T & \mathbf{1}_{m/2} \end{pmatrix}. \tag{4.5.12}$$

Both the $\mathbb{1}_{m/2} \oplus (\mathbf{w}_{m/2})^T$ and $\mathbf{w}_{m/2}$ components of this matrix have unique rows, meaning the above matrix admits distinct syndromes for all single-qubit Y-errors on the parity bits. The weights of the syndromes in S^{py} are described by matrix

$$W(S^{py}) = \begin{matrix} \\ [P_X] \\ [P_Z] \end{matrix} \begin{pmatrix} \begin{matrix} [P_X] \\ m/2 \end{matrix} & \begin{matrix} [P_Z] \\ 1 \end{matrix} \\ 2 & m/2 \end{pmatrix}. \tag{4.5.13}$$

The above matrix shows that the Y-errors have a syndrome structure that is distinct from the other error types. We have now shown that the construction for tripartite CPC Hamming codes described in theorem 1 produces unique syndromes for X-, Z- and Y-errors on all of the code qubits. The structures of these syndromes for each error type are summarised in table 4.3. Therefore, the tripartite CPC code constructed by combining two copies of a Hamming code L_k with a cross-check matrix of the form given in equation 4.5.6 will always give a code of distance 3. This concludes the proof. □

4.6 CPC encoders for CSS codes

In this section we explore the correspondence between CSS codes and the CPC framework. We begin by proving that all CSS codes can be expressed as a tripartite CPC codes, and give an explicit example for the seven-qubit Steane code. Following this, we outline a CPC method that allows (almost) any pair of $[N, K, 3]$ classical codes to be turned into a $[[n = 2N - K + 2, k = K, d = 3]]$ CSS code.

4.6.1 The CPC representation of CSS codes

We now show how CSS codes can be described as CPC codes. Our result is encapsulated by the following theorem.

Theorem 2. *Let $\mathcal{C}_{\text{CSS}}$ be an $[[n, k, d]]$ CSS code with a quantum parity check matrix of the form*

$$G_{\text{CSS}} = \left(\begin{array}{c|c} H_X & \boldsymbol{0} \\ \hline \boldsymbol{0} & H_Z \end{array}\right), \tag{4.6.1}$$

where H_X and H_Z are the parity check matrices of two classical codes with rank R_X and R_Z respectively. For any such $\mathcal{C}_{\text{CSS}}$ it is possible to construct a CPC encoder with adjacency matrices of the form

$$m_b = (\alpha \mid \boldsymbol{0}), \quad m_p = (\boldsymbol{0} \mid \beta), \quad m_c = \begin{pmatrix} \boldsymbol{0} & \gamma \\ \gamma^T & \boldsymbol{0} \end{pmatrix}, \tag{4.6.2}$$

where α is a $(k \times R_X)$ binary matrix, β is a $(k \times R_Z)$ binary matrix and γ is a $(R_X \times R_Z)$ binary matrix.

Proof. For a CPC code, the quantum parity check matrix is given by

$$G_{XZ}(\mathcal{S}_{\text{CPC}}) = \left(m_p^T \quad m_b^T \cdot m_p \oplus m_c \;\middle|\; m_b^T \quad \mathbb{1}_{n-k}\right), \tag{4.6.3}$$

where m_b, m_p and m_c are the CPC adjacency matrices. For the purposes of proving theorem 2, we need to show that equation 4.6.1 can be rewritten into the form of equation 4.6.3. This will allow us to derive relations for the CPC adjacency matrices for the CSS code.

We begin by noting that, as both H_X and H_Z are classical parity check matrices,

equation 4.6.1 can be rewritten as follows

$$G_{\text{CSS}} = \left(\begin{array}{cc|cc} A_X^T & \mathbb{1}_{n-k} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & A_Z^T & \mathbb{1}_{n-k} \end{array} \right), \tag{4.6.4}$$

where A_X and A_Z are the adjacency matrices of the two classical codes H_X and H_Z. Once in this form, Gaussian elimination can be performed to further rewrite G_{CSS} into the form

$$G_{\text{CSS}} = \left(\begin{array}{ccc|ccc} \overbrace{J}^{k} & \overbrace{\mathbb{1}_{R_X}}^{R_X} & \overbrace{K}^{R_Z} & \overbrace{\mathbf{0}}^{k} & \overbrace{\mathbf{0}}^{R_X} & \overbrace{\mathbf{0}}^{R_Z} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & L & M & \mathbb{1}_{R_Z} \end{array} \right), \tag{4.6.5}$$

where the numbers above the braces indicate the width of each block of the matrix, and R_Z and R_X are the matrix ranks of H_X and H_Z respectively. The next step in the proof is to swap the two column blocks of width R_X to obtain the parity check matrix

$$G'_{\text{CSS}} = \left(\begin{array}{ccc|ccc} \overbrace{J}^{k} & \overbrace{\mathbf{0}}^{R_X} & \overbrace{K}^{R_Z} & \overbrace{\mathbf{0}}^{k} & \overbrace{\mathbb{1}_{R_X}}^{R_X} & \overbrace{\mathbf{0}}^{R_Z} \\ \hline \mathbf{0} & M & \mathbf{0} & L & \mathbf{0} & \mathbb{1}_{R_Z} \end{array} \right). \tag{4.6.6}$$

Note that the previous step corresponds to performing Hadamard gates on the affected qubits, meaning the modified parity check matrix G'_{CSS} is locally equivalent to the original. We now take advantage of the fact that a quantum parity check matrix $G = (G_X \mid G_Z)$ is self-orthogonal so that

$$G_X \cdot G_Z^T + G_Z \cdot G_X^T = \mathbf{0}, \tag{4.6.7}$$

to find an expression for M in terms of J, K and L

$$\begin{aligned} & G^X_{\text{CSS}} \cdot \left(G^Z_{\text{CSS}}\right)^T + G^Z_{\text{CSS}} \cdot \left(G^X_{\text{CSS}}\right)^T \\ &= \begin{pmatrix} J & \mathbf{0} & K \\ \mathbf{0} & M & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{0} & L^T \\ \mathbb{1}_{R_X} & \mathbf{0} \\ \mathbf{0} & \mathbb{1}_{R_Z} \end{pmatrix} + \begin{pmatrix} \mathbf{0} & \mathbb{1}_{R_X} & \mathbf{0} \\ L & \mathbf{0} & \mathbb{1}_{R_Z} \end{pmatrix} \begin{pmatrix} J^T & \mathbf{0} \\ \mathbf{0} & M^T \\ K^T & \mathbf{0} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{0} & J \cdot L^T + K + M^T \\ L \cdot J^T + K^T + M & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}. \end{aligned} \tag{4.6.8}$$

From the above, we see that $M = L \cdot J^T + K^T$. By substituting this into equation 4.6.9, we can write G'_{CSS} as

$$G'_{\text{CSS}} = \left(\begin{array}{ccc|ccc} \overbrace{J}^{k} & \overbrace{\mathbf{0}}^{R_X} & \overbrace{K}^{R_Z} & \overbrace{\mathbf{0}}^{k} & \overbrace{\mathbb{1}_{R_X}}^{R_X} & \overbrace{\mathbf{0}}^{R_Z} \\ \mathbf{0} & L \cdot J^T + K^T & \mathbf{0} & L & \mathbf{0} & \mathbb{1}_{R_Z} \end{array} \right). \tag{4.6.9}$$

We are now in a position to make contact between G'_{CSS} and the target form of the CPC parity check matrix given by equation 4.6.3. We do this by setting

$$m_b = \left(J^T \mid \mathbf{0}\right), \quad m_p = \left(\mathbf{0} \mid L^T\right), \quad m_c = \begin{pmatrix} \mathbf{0} & K \\ K^T & \mathbf{0} \end{pmatrix}. \tag{4.6.10}$$

It can be seen that the above adjacency matrices have the form specified in equation 4.6.2 when the following substitutions are made: $\alpha = J^T$, $\beta = L^T$ and $\gamma = K$. A CPC encoder can therefore be constructed for any CSS code. This concludes the proof. □

4.6.2 Example: A CPC encoder for the Steane $[[7, 1, 3]]$ code

As an example of theorem 2, we now show how a CPC encoder can be constructed for the famous seven-qubit Steane CSS code [42]. The Steane $[[7, 1, 3]]$ code has the

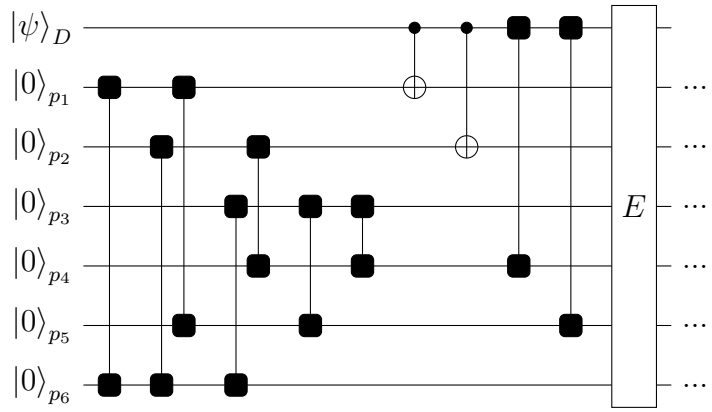


Figure 4.10: The CPC encoder for the $[[7, 1, 3]]$ Steane code.

following parity check matrix

$$G_{\text{Steane}} = \left(\begin{array}{c|c} H_{[7,4,3]} & 0 \\ \hline 0 & H_{[7,4,3]} \end{array} \right), \tag{4.6.11}$$

where $H_{[7,4,3]}$ is the parity check matrix for classical $[7, 4, 3]$ Hamming code defined in section 2.4. Following the procedure outlined in the proof of theorem 2, the above quantum parity check matrix can be rewritten into the form

$$G'_{\text{Steane}} = \left(\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right). \tag{4.6.12}$$

Reading off the above, we obtain the following CPC adjacency matrices

$$m_b = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad m_p = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix},$$
$$m_c = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{4.6.13}$$

The CPC encoder for the Steane code, with the above adjacency matrices, is shown in figure 4.10.

4.6.3 A CPC method for constructing CSS codes

In this section we introduce a CPC method for the construction of CSS codes from almost any pair of classical $[N, K, 3]$ codes. Our method adopts the CPC structure depicted in figure 4.11. This is similar to the tripartite structure introduced in section 4.5, but includes two additional parity qubits P'_X and P'_Z. These qubits, which we refer to as 'second-tier' parity qubits, interact only with other parity qubits. We will see that the second-tier parity qubits play an important role in ensuring that codes with the structure given by figure 4.11 are CSS.

The method begins with an $[N, K, 3]$ classical code which we label L. Our procedure allows (almost) any such classical code to be repurposed in the construction of a CPC encoder that describes a CSS code with parameters $[[n = 2N - K + 2, k = K, d = 3]]$. The first step is to define m_b and m_p adjacency matrices with the following form

$$m_b = \begin{array}{c} \begin{matrix} [P_X] & [P'_X] & [P_Z] & [P'_Z] \end{matrix} \\ \left(\begin{array}{cc|cc} A_L & 0_c & \mathbf{0} & 0_c \end{array}\right) \end{array}, \quad m_p = \begin{array}{c} \begin{matrix} [P_X] & [P'_X] & [P_Z] & [P'_Z] \end{matrix} \\ \left(\begin{array}{cc|cc} \mathbf{0} & 0_c & A_L & 0_c \end{array}\right) \end{array} \tag{4.6.14}$$

where A_L is the adjacency matrix for the code L and 0_c is a column of zeros. The column labels correspond to the different blocks of parity qubits, as arranged in the

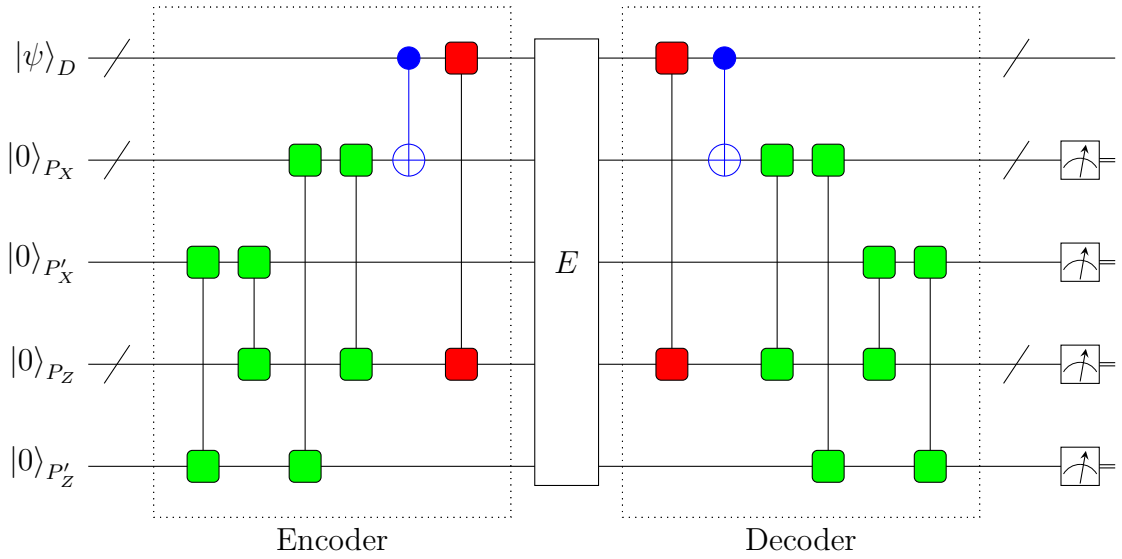


Figure 4.11: The tripartite CPC structure for CSS codes. As with tripartite codes, the parity qubits are split into bit- and phase-checking blocks P_X and P_Z. Each block includes a second-tier qubit, labelled P'_X and P'_Z, which does not interact directly with the data register and serves as a global check on the parity qubits in the other block. This means that P'_X checks all the parity qubits in P_Z (including P'_Z), whilst P'_Z checks all the parity qubits in P_X. The general forms for the cross-checks (green gates), bit-checks (blue gates) and phase-checks (red gates) in this circuit are defined by the adjacency matrices in equations 4.6.14 and 4.6.15.

structure depicted in figure 4.11. Note that the above form ensures that the data qubits do not interact directly with the second-tier parity qubits P'_X and P'_Z. Next, we define a cross-check matrix m_c with the following form

$$m_c = \begin{array}{c} \\ [P_X] \\ [P'_X] \\ [P_Z] \\ [P'_Z] \end{array} \begin{array}{c} \begin{array}{cccc} [P_X] & [P'_X] & [P_Z] & [P'_Z] \end{array} \\ \left(\begin{array}{cc|cc} \mathbf{0} & 0_c & Q & 1_c \\ 0_r & 0 & 1_r & 1 \\ \hline Q^T & 1_c & \mathbf{0} & 0_c \\ 1_r & 1 & 0_r & 0 \end{array} \right), \end{array} \tag{4.6.15}$$

where Q is an invertible square matrix that does not contain a row of all ones, 1_c is

a column of ones, 1_r is a row of ones and 0_r is a row of zeros. For most classical codes L, it is possible to find an appropriate invertible matrix Q such that the CPC code obtained from the adjacency matrices defined in equations 4.6.14 and 4.6.15 is a distance-three CSS code.

In order to demonstrate the validity of the above method for the construction of CSS codes, we first need to show that the CPC codes that result from the adjacency matrices of the form described in equations 4.6.14 and 4.6.15 produce unique syndromes for all single-qubit errors. Following this, we will show that the quantum parity check matrices for these codes are CSS.

Table 4.4 summarises the structure of the syndromes for CPC codes of the type described by equations 4.6.14 and 4.6.15. From the table, it can be seen that errors on the data qubits will always be distinguishable from one another as L is a valid classical code in its own right. Furthermore, errors that occur on the data qubits do not interact with the second-tier parity qubits P'_X and P'_Z. This means they can always be distinguished from Z-errors on the parity qubits which trigger the second-tier measurements. The syndrome matrix for Z-errors on the parity qubits is given by

$$S^{pz} = \begin{array}{c} \\ [P_X] \\ [P'_X] \\ [P_Z] \\ [P'_Z] \end{array} \begin{array}{c} \begin{array}{cccc} [P_X] & [P'_X] & [P_Z] & [P'_Z] \end{array} \\ \left(\begin{array}{cc|cc} \mathbf{0} & 0_c & Q & 1_c \\ 0_r & 0 & 1_r & 1 \\ \hline A_L^T \cdot A_L \oplus Q^T & 1_c & \mathbf{0} & 0_c \\ 1_r & 1 & 0_r & 0 \end{array} \right) \end{array}, \tag{4.6.16}$$

An appropriate matrix Q therefore needs to be found in order to ensure that there are unique syndromes for single-qubit Z-errors on the parity qubits. To show that this is (almost) always possible, we first note that the term $A_L^T \cdot A_L$ corresponds to a square matrix. We then make use of a theorem from linear algebra that states that any square matrix M can be expressed as a sum $M = T_1 + T_2$ where T_1 and T_2 are invertible matrices [91]. If the value of Q is chosen as an invertible matrix, we can write the following expression $A_L^T \cdot A_L \oplus Q^T = F$ where F is another invertible matrix. Invertible matrices have unique rows. Therefore, choosing Q as an appropriate invertible matrix will result in a unique set of syndromes in S^{pz} in most cases. The

		X-error syndromes m_b	Z-error syndromes m_p	Y-error syndromes $m_b \oplus m_p$
data qubits	$\vec{S}_i^d = (s_{X_i}\ s_{X_i'} \vert s_{Z_i}\ s'_{Z_i})$	$(A_L\ 0_c \mid \mathbf{0}\ 0)_i$	$(\mathbf{0}\ 0_c \mid A_L\ 0)_i$	$(A_L\ 0_c \mid A_L\ 0)_i$
	$W(s_{X_i})$	≥ 2	0	≥ 2
	$W(s_{Z_i})$	0	≥ 2	≥ 2
	$W(s'_{X_i})$	0	0	0
	$W(s'_{Z_i})$	0	0	0
		X-error syndromes $\mathbb{1}_m$	Z-error syndromes $m_p^T \cdot m_b \oplus m_c$	Y-error syndromes $\mathbb{1}_m \oplus m_p^T \cdot m_b \oplus m_c$
parity qubits	$\vec{S}_i^p = (s_{X_i}\ s_{X_i'} \vert s_{Z_i}\ s'_{Z_i})$	$\vec{S}_i^{px} = (\mathbb{1}_m)_i$	$\vec{S}_i^{pz}$	$\vec{S}_i^{py} = \vec{S}_i^{pz} \oplus \vec{S}_i^{px}$
	$W(s_{X_i})$	$\begin{cases} 1, & \text{if } i \leq m/2 \\ 0, & \text{if } i > m/2 \end{cases}$	$\begin{cases} 0, & \text{if } i \leq m/2 \\ > 1, & \text{if } i > m/2 \end{cases}$	$\begin{cases} 1, & \text{if } i \leq m/2 \\ > 1, & \text{if } i > m/2 \end{cases}$
	$W(s_{Z_i})$	$\begin{cases} 0, & \text{if } i \leq m/2 \\ 1, & \text{if } i > m/2 \end{cases}$	$\begin{cases} > 1, & \text{if } i \leq m/2 \\ 0, & \text{if } i > m/2 \end{cases}$	$\begin{cases} > 1, & \text{if } i \leq m/2 \\ 1, & \text{if } i > m/2 \end{cases}$
	$W(s'_{X_i})$	1	1	1
	$W(s'_{Z_i})$	1	1	1

Table 4.4: Table showing the structure and weights of the error signatures resulting form different error types in the $[[n, k, d]]$ CPC codes described by theorem **??**. Each syndrome $\vec{S}_i = (s_{X_i}\ s_{X_i'} \vert s_{Z_i}\ s'_{Z_i})$ consists of a string of bits of length $m = n - k$. The parts of the syndrome correspond to the the different blocks of parity qubits as partitioned in figure 4.11. The syndrome vector $\vec{S}_i^{px}$ is a row of the syndrome matrix defined in equation 4.6.16.

caveat is due to fact that Q must not have a row of all '1's, as this would cause a clash with the syndrome for Z-errors that occur on the second-tier parity qubits. The consequence of this is that may not be possible to construct a CSS code for certain classical codes L.

We have now shown that is (almost) always possible to construct a distance-three CPC code with the structure outlined in figure 4.11 from the starting point of a classical code L. We now show that such codes are CSS. To this end, we first use equation 4.6.3 to write the quantum parity check matrix for the CPC code

$$G_{\text{CPC}} = \left(\begin{array}{ccccccccccc} \scriptstyle[D] & & \scriptstyle[P_X] & \scriptstyle[P'_X] & \scriptstyle[P_Z] & \scriptstyle[P'_Z] & \scriptstyle[D] & \scriptstyle[P_X] & \scriptstyle[P'_X] & \scriptstyle[P_Z] & \scriptstyle[P'_Z] \\ A_L^T & & \mathbf{0} & 0_c & Q & 1_c & \mathbf{0} & \mathbb{1} & 0_c & \mathbf{0} & 0_c \\ 0_r & & 0_r & 0 & 1_r & 1 & 0_r & 0_r & 1 & 0_r & 0 \\ \hline \mathbf{0} & & A_L^T \cdot A_L \oplus Q^T & 1_c & \mathbf{0} & 0_c & A_L^T & \mathbf{0} & 0_c & \mathbb{1} & 0_c \\ 0_r & & 1_r & 1 & 0_r & 0 & 0 & 0_r & 0 & 0_r & 1 \end{array}\right). \tag{4.6.17}$$

By applying Hadamard gates to the qubits in blocks P_X and P'_X the above transforms to

$$G'_{\text{CPC}} = \left(\begin{array}{ccccccccccc} \scriptstyle[D] & \scriptstyle[P_X] & \scriptstyle[P'_X] & \scriptstyle[P_Z] & \scriptstyle[P'_Z] & \scriptstyle[D] & & \scriptstyle[P_X] & \scriptstyle[P'_X] & \scriptstyle[P_Z] & \scriptstyle[P'_Z] \\ A_L^T & \mathbb{1} & 0_c & Q & 1_c & \mathbf{0} & & \mathbf{0} & 0_c & \mathbf{0} & 0_c \\ 0_r & 0_r & 1 & 1_r & 1 & 0_r & & 0_r & 0 & 0_r & 0 \\ \hline \mathbf{0} & \mathbf{0} & 0_c & \mathbf{0} & 0_c & A_L^T & & A_L^T \cdot A_L \oplus Q^T & 1_c & \mathbb{1} & 0_c \\ 0_r & 0_r & 0 & 0_r & 0 & 0 & & 1_r & 1 & 0_r & 1 \end{array}\right). \tag{4.6.18}$$

The above quantum parity check matrix has no overlapping X and Z stabilizers and is therefore CSS.

4.6.4 Example: Construction of a $[[11, 3, 3]]$ CSS code from the classical ring code

In section 4.5.1 we showed how a $[[9, 3, 3]]$ CPC code can be constructed by combining two copies of the classical ring code into a tripartite structure. However, this code is not CSS due to the fact that cross-checks are performed between qubits in the same parity checking block. We now show how an $[[11, 3, 3]]$ CSS code can be constructed from the ring code using the result proved in theorem **??**.

The adjacency matrix for the $[6, 3, 3]$ ring code is given by

$$A_{[6,3,3]} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}. \tag{4.6.19}$$

Substituting this into equations 4.6.14 we get the following CPC adjacency matrices m_b and m_p

$$m_b = \overset{\begin{matrix}[P_X]\end{matrix}}{\left(A_{[6,3,3]}\right.} \quad \overset{\begin{matrix}[P'_X]\end{matrix}}{0_c} \;\middle|\; \overset{\begin{matrix}[P_Z]\end{matrix}}{\mathbf{0}} \quad \overset{\begin{matrix}[P'_Z]\end{matrix}}{\left.0_c\right)} = \left(\begin{array}{cccc|cccc} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right), \tag{4.6.20}$$

$$m_p = \overset{\begin{matrix}[P_X]\end{matrix}}{\left(\mathbf{0}\right.} \quad \overset{\begin{matrix}[P'_X]\end{matrix}}{0_c} \;\middle|\; \overset{\begin{matrix}[P_Z]\end{matrix}}{A_{[6,3,3]}} \quad \overset{\begin{matrix}[P'_Z]\end{matrix}}{\left.0_c\right)} = \left(\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right) \tag{4.6.21}$$

The general form for the cross-check $m_c(Q)$ is given by equation 4.6.15. For this particular example, we set the invertible matrix Q as

$$Q = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}. \tag{4.6.22}$$

It can easily be verified, using equation 4.4.12 or otherwise, that the $[[11, 3, 3]]$ CPC

code defined by m_b, m_p and m_c produces unique syndromes for single-qubit errors. By inserting equations 4.6.22 and 4.6.19 into equation 4.6.18, we obtain the following CSS quantum parity check matrix for the $[[11, 3, 3]]$ code

$$G'_{\text{CPC}} = \left(\begin{array}{ccc|ccc|c|ccc|c||ccc|ccc|c|ccc|c}
\multicolumn{3}{c}{[D]} & \multicolumn{3}{c}{[P_X]} & [P'_X] & \multicolumn{3}{c}{[P_Z]} & [P'_Z] & \multicolumn{3}{c}{[D]} & \multicolumn{3}{c}{[P_X]} & [P'_X] & \multicolumn{3}{c}{[P_Z]} & [P'_Z] \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1
\end{array}\right) \tag{4.6.23}$$

where the labels show the different blocks of qubits.

4.7 Summary and discussion

To summarise, the CPC framework is built around a fundamental gadget in which the parity information of a quantum register is stored coherently and compared over time. Under the CPC code structure, the parity information of the quantum register is never explicitly measured out. A consequence of this is that there are no restrictions on the form of the parity checks, a freedom that can exploited to facilitate the mapping of classical codes to quantum codes.

CPC codes have a symmetric structure involving an encode stage, an error stage and a decode stage. The encoder is the unitary inverse of the decoder, meaning that the overall action of the CPC circuit is an identity operation if no errors occur. The decode (and encode stage) of the CPC circuit involve three rounds of parity checks: bit-checks, phase-checks and cross-checks. The bit- and phase-checks detect X- and Z-errors on the register respectively. The cross-checks are performed between the parity qubits themselves, and are designed to detect errors that are indirectly propagated through

the circuit. The three rounds of parity checks can be compactly represented in terms of three adjacency matrices.

The encoded stabilizers of a CPC code can be calculated by pushing the initial stabilizers through the encode stage of the circuit. Pauli X- and Z- logical operators can be calculated via a similar technique. An interesting thing to note is that any combination of adjacency matrices will result in a stabilizer code when fed into the template provided by the CPC structure. The only task that then remains to verify the 'usefulness' of the code is to measure its code distance. We make use of this feature of the CPC framework in chapter 7, where we show how new CPC codes can be discovered via automated methods.

Tripartite CPC codes separate the parity qubits into two blocks to measure bit- and phase-flip parity measurements separately. In this chapter, we have proved a general method for turning any pair of $[n, k, 3]$ Hamming codes into a $[[2n - k, k, 3]]$ tripartite CPC code.

CSS codes can always be represented as CPC codes. Furthermore, we have outlined a CPC method for turning almost any pair of classical $[n, k, 3]$ codes into a $[[2n - k + 2, k, 3]]$ CSS code. As an example, we showed how an $[[11, 3, 3]]$ CSS code can be constructed by combining two copies of the classical $[6, 3, 3]$ ring code. In contrast, using the traditional method for CSS codes, the same code is derived by combining an $[11, 7, 3]$ code with an $[11, 4, 3]$ subcode. The advantage of the CPC framework for CSS code construction is that any code can be used as the starting point, rather than having to find a pair of codes that satisfy the requirements imposed by the conventional CSS approach.

In this chapter we have not considered encoded computation for CPC codes beyond the Pauli X and Z logical operators. In [1], steps were made to find ways of representing encoded CNOT operations in a CPC circuit. As all CPC codes are stabilizer codes, existing techniques for achieving universal quantum computation [69] will be applicable.

The CPC constructions for tripartite codes and CSS codes are not optimal in the sense that they do not saturate the quantum Hamming bound. In chapter 7, we explore how automated search methods can be used to discover denser code on the quantum

Hamming bound.

So far, we have limited our study of CPC codes to codes with distance $d = 3$. However, as the number of available qubits increases, it will become essential to find codes with larger distances to ensure reliable operation. As the number of possible error locations grows exponentially with code distance, it is not easy to find general constructions for codes with high distance. Possible avenues for resolving these problems are discussed in chapter 8.6.

Chapter 5

Implementation of a $[[4, 2, 2]]$ CPC code on the IBM 5Q device

In this chapter, I outline an experiment in which I prepared and ran a $[[4, 2, 2]]$ CPC detection code on the IBM 5Q superconducting qubit device. The results of this experiment were originally published in [2].

The IBM 5Q is a small-scale quantum computer, built and maintained by IBM Quantum [92]. The device has five programmable superconducting transmon qubits, and is accessible to the public via the Internet. Whilst the individual gate-error rates on the IBM 5Q are too high to suppress the error rate for a quantum memory, our results show that the syndrome information from a full encode-decode cycle of the $[[4, 2, 2]]$ CPC code can be used to improve the output state fidelity by post-selection.

In general, it should be noted that the *encode-error-decode* structure of a CPC code is not intended as a description of how the corresponding stabilizer code should be implemented in practice. CPC codes, as presented in the canonical form described in section 4.3, are not fault tolerant. Therefore, the CPC setup should be treated as a tool for code discovery rather than code implementation. In this section, however, we consider the exception that proves the rule. For the case of a $[[4, 2, 2]]$ CPC code – the simplest possible CPC code for a quantum error model – the *encode-error-decode* structure can be leveraged as a useful code implementation strategy. We show that

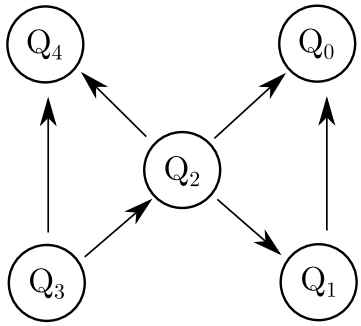
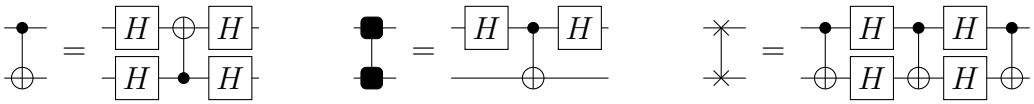


Figure 5.1: The connectivity map of the IBMQX4 version of the IBM 5Q quantum computer illustrating the 'bowtie' layout. The arrows indicate the allowed CNOT operations and their preferred directions. *This figure originally published in* [2].

for the limited case of a [[4, 2, 2]] code, prepared with certain input states, a CPC circuit can be compiled that is hardened to errors. We then show that the syndrome information this circuit produces can be used to improve the fidelity of the output state.

5.1 Experimental overview and conditions for success

Ultimately, the condition for success for a quantum code is to test whether the encoded protocol has a lower logical error rate than the equivalent circuit before encoding [93]. In the case of a CPC quantum memory, the circuit that is encoded is simply an extended identity operation. The usefulness of the [[4, 2, 2]] code could therefore be assessed by comparing the fidelity of the encoded output to the equivalent output of an unprotected two-qubit data register. However, the gate error rates on the IBM 5Q hardware are so high that such a comparison is unlikely to produce a positive result. This problem is compounded by the fact that the IBM hardware limits the experiment to a single encode-decode cycle, meaning certain regions of the [[4, 2, 2]] circuit – before the encoder and after the decoder – are left unprotected. Consequently, the aim of the experiment presented here is restricted to demonstrating that, whilst not suppressing the logical error rate, the [[4, 2, 2]] CPC code does detect errors. We now describe the method by



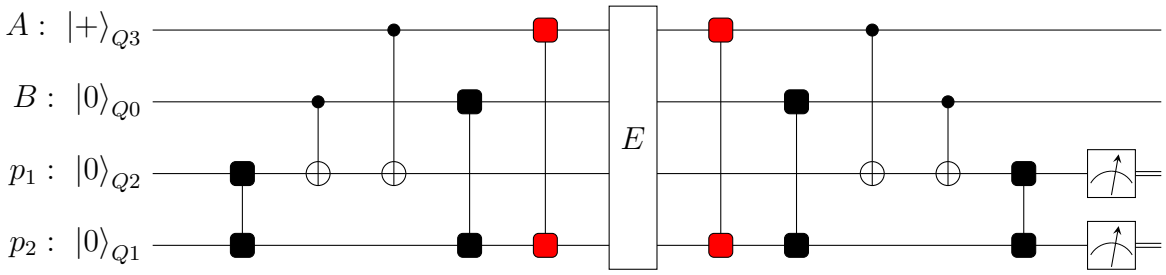
(a) (b) (c)

Figure 5.2: (a) The direction of a CNOT operation on the IBMQX4 can be reversed via the addition of Hadamard gates to the inputs and outputs. (b) The conjugate propagator gate expressed in terms of a CNOT gate. (c) Realisation of a SWAP gate using three CNOT operations. *This figure was originally published in* [2].

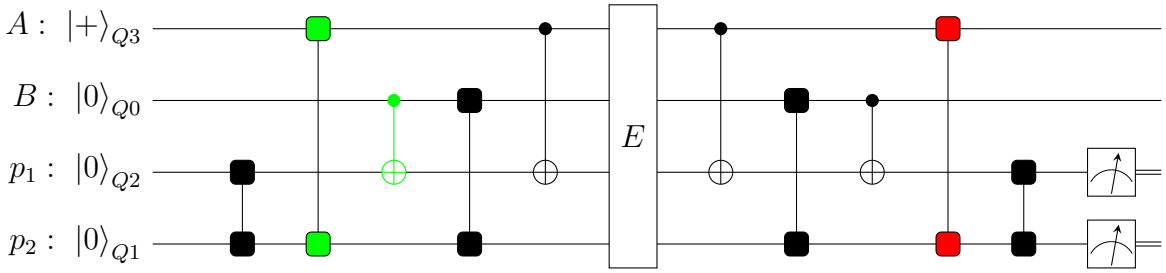
which this is achieved.

Our experiment on the IBM 5Q encodes a single input state $|\psi_{AB}\rangle = |+_A 0_B\rangle$ using a $[[4, 2, 2]]$ CPC quantum memory of the type described in section 4.2. The $|+_A 0_B\rangle$ state is an easy-to-prepare quantum state that is susceptible to both bit- and phase-flip errors, and therefore provides a suitable test of the $[[4, 2, 2]]$ CPC code as a quantum memory.

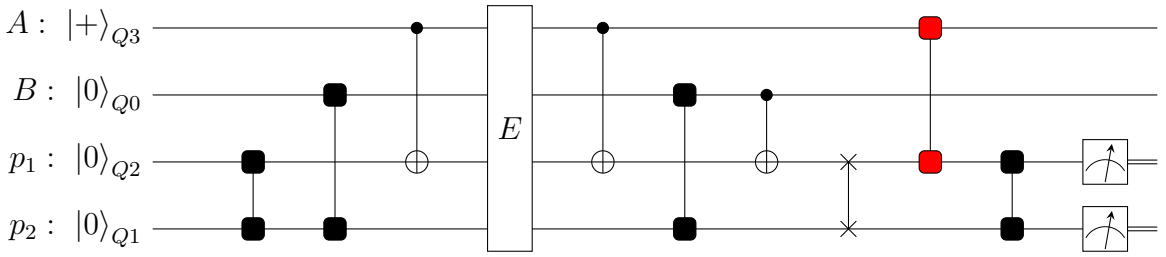
The compiled $[[4, 2, 2]]$ CPC code is run multiple times with the input state $|\psi_{AB}\rangle = |+_A 0_B\rangle$ on the IBM 5Q hardware. At the end of each CPC code cycle, the parity qubits are measured to provide a syndrome designed to indicate if an error has occurred. An approximation to the output state of the register is reconstructed from the experimental data using quantum state tomography. The quality of this output is quantified by calculating its fidelity relative to the input state $|+_A 0_B\rangle$. In this experiment we compare the output fidelity of the $[[4, 2, 2]]$ protocol before and after post-selection. In the former, the syndrome information is ignored, whereas in the latter it is used to determine which experimental runs are discarded during post-selection. The condition for success is that the post-selection should improve the output fidelity. If this is the case, it will demonstrate that the $[[4, 2, 2]]$ CPC code is detecting errors and produces useful syndrome information.



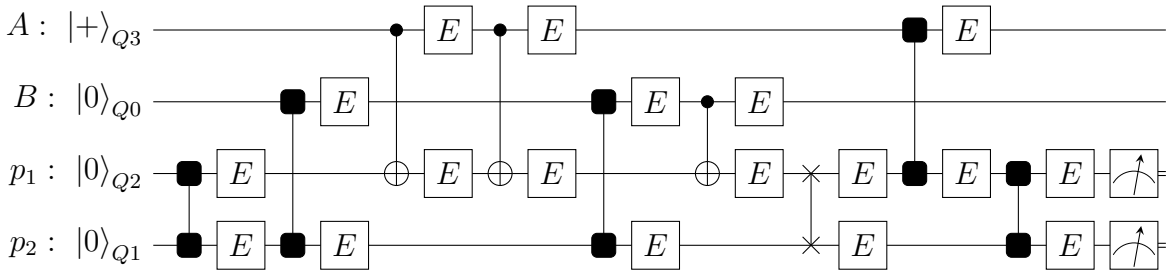
(a) The [[4,2,2]] CPC code with a $|\psi_{AB}\rangle = |+_A 0_B\rangle$ input mapped onto the IMBQX4 chip. The red conjugate propagator gates are not possible according to the connectivity map for the IBMQX4 shown in figure 5.1.



(b) A modified version of the [[4,2,2]] circuit in which the order of gates in the encoder and decoder has be rearranged. In this new form, the circuit can be simplified by noting that the action of the gates marked in green is the identity.



(c) A SWAP gate can be added to the [[4,2,2]] circuit to exchange the p_1 and p_2 parity qubits. This allows the 'illegal' operation marked in red in the decoder to be performed via a nearest-neighbour interaction.



(d) When running the [[4,2,2]] code on a real device, it can no longer be assumed that errors occur only in the wait-stage between the encoder and decoder. For the [[4,2,2]] circuit in question, a single-qubit fault E after any gate will not propagate a multi-qubit error to the register without triggering a syndrome.

Figure 5.3: Steps for compiling the [[4,2,2]] CPC quantum memory onto the IBMQX4 chip. *This figure was originally published in* [2].

5.2 Compiling a [[4, 2, 2]] CPC circuit onto the IBM 5Q

Our experiment is run on the IBMQX4 version of the IBM 5Q, the technical details for which can be found in [94]. Figure 5.1 depicts the 'bow tie' layout of the chip. The arrows represent the allowed CNOT operations between qubits. The direction of the arrow indicates the preferred CNOT direction, but the operation can be reversed via the circuit transformation shown in figure 5.2a.

The [[4, 2, 2]] code, as outlined in section 4.2, has two data qubits $\{A, B\}$ and two parity qubits $\{p_1, p_2\}$. In this experiment, the code qubits are mapped onto the physical qubits of the IBMQX4 device as follows: $\{A \to Q_3,\ B \to Q_0,\ p_1 \to Q_2,\ p_2 \to Q_1\}$. The input state becomes $|\psi_{AB}\rangle = |+\rangle_{Q_3} \otimes |0\rangle_{Q_0}$, and the resultant circuit is shown in figure 5.3a. The two conjugate propagator gates marked in red are not possible on the IBMQX4, as there is no connectivity between qubits Q_3 and Q_1 (see figure 5.1). The [[4, 2, 2]] CPC circuit must therefore be modified to accommodate this hardware constraint.

The first step in compiling the [[4, 2, 2]] circuit for the IBMQX4 is to rearrange the gates into the order shown in figure 5.3b. This is a departure from the canonical form of CPC codes outlined in section 4.3. However, it can easily be checked that the modified circuit remains a functional [[4, 2, 2]] CPC code capable of detecting single X- and Z-errors on any of the qubits during the wait-stage.

In the rearranged form of the circuit in figure 5.3b, and when the input state is $|\psi_{AB}\rangle = |+\rangle_{Q_3} \otimes |0\rangle_{Q_0}$, it can be seen that the action of the gates highlighted in green is the identity. The green gates can therefore be omitted from the circuit without affecting the function of the quantum memory. Following this simplification, the only operation that remains prohibited by the IMBQX4's connectivity constraints is the red conjugate propagator gate between Q_1 and Q_3 in the decoder. One way of resolving this problem is to perform a SWAP operation between Q_2 and Q_1, as shown in figure 5.3c. The SWAP gate exchanges the positions of the p_1 and p_2 parity check qubits, enabling the red conjugate propagator gate to be performed via a nearest-neighbour interaction. A SWAP gate is achieved via the application of three CNOT gates (see figure 5.2c), and

is therefore an expensive operation that should be used sparingly. In section 7.2.3 in chapter 7, we explore how the CPC code design process can be used to minimise the SWAP gate count when compiling larger codes onto quantum hardware.

5.3 A note on fault tolerance for the $[[4, 2, 2]]$ circuit

So far, we have considered a simplified model of CPC code operation in which it is assumed errors only occur during the wait-stage between the encoder and the decoder. However, we have observed that the error rates for CNOT operations and readout on the IBMQX4 are of the order 10^{-2} (daily calibration data can be obtained from the IBM Q website [92]). This realistically means that any quantum code must be designed to detect errors that occur at any point in the circuit. To this end, figure 5.3d shows the IMBQX4-compiled $[[4, 2, 2]]$ circuit under a more general error model.

Fault tolerant circuit construction ordinarily necessitates the introduction of additional qubits [44, 45]. However, in this particular instance of the $[[4, 2, 2]]$ CPC code with a known $|+\rangle_{Q_3} \otimes |0\rangle_{Q_0}$ input, it can be verified that a single fault at any of the locations marked on figure 5.3d will not propagate a multi-qubit error to the register without triggering a syndrome. The circuit can therefore be considered to have been hardened against single-qubit errors in the encode and decode stages of the circuit. It should be noted, however, that this does not extend the circuit to full fault tolerance when implemented on the IBMQX4 chip. State preparation and measurement (SPAM) errors are not accounted for, nor is the $[[4, 2, 2]]$ code capable of detecting correlated two-qubit errors that might occur after a CNOT gate. Another issue is that the circuit allows certain single-qubit errors to propagate to the register in an undetectable way. It is not currently possible to measure, then reset a qubit on the IBMQX4 via the public API. As a result, our implementation is restricted to a single encode-decode cycle, meaning the undetected single-qubit errors will reduce the output fidelity. However, as outlined in [1], CPC codes can be expressed in terms of stabilizer codes. Adopting this approach allows CPC codes to be implemented using existing syndrome extraction techniques, and enables errors to be decoded over multiple cycles. Assuming access to hardware that allows qubit reset, CPC codes implemented in this way would be

tolerant of the single-qubit errors that propagate to the register.

5.4 Experimental data reconstruction methods

The IBM Quantum Information Software Kit (QISKIT) [95] was used to prepare the $[[4,2,2]]$ experiment for quantum state tomography on the output qubits Q_0 and Q_3. QISKIT quantum tomography tools were used to create a set of nine circuits from the original $[[4,2,2]]$ circuit (depicted in figure 5.3c), each of which was designed to measure the output qubits Q_0 and Q_3 in a different measurement basis from the list $\{XX,\ XY,\ XZ,\ YX,\ YY,\ YZ,\ ZX,\ ZY,\ ZZ\}$. These quantum tomography circuits were then run multiple times to create a distribution of results that could be used reconstruct an approximation to the density matrix of the output state. The QISKIT method used for state reconstruction from the experimental data was the fast maximum likelihood method for quantum tomography, a description of which can be found in [96].

The quantum tomography circuits for the $[[4,2,2]]$ memory were run in batches of 8192 shots. After each batch, the QISKIT maximum likelihood method was used to reconstruct the density matrix ρ_{dd} of the directly decoded output before post-selection. The syndrome qubits were then inspected to determine which of the shots in the batch should be discarded during post-selection. State reconstruction was then performed again on the reduced set to obtain a post-selected density matrix ρ_{ps}. The quality of the directly decoded and post-selected output state for each batch was quantified by calculating the fidelity, $F(\rho) = \left(\mathrm{Tr}\left[\sqrt{\rho^{1/2}\sigma\rho^{1/2}}\right]\right)^2$, where σ is the target density matrix. For the chosen input state $|\psi_{AB}\rangle = |+\rangle_{Q_3} \otimes |0\rangle_{Q_0}$, the target density matrix is given by

$$\sigma = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{5.4.1}$$

The purity of the density matrices, defined by $P(\rho) = \mathrm{Tr}\left[\rho^2\right]$, was also calculated to provide a coherence measure for the output states.

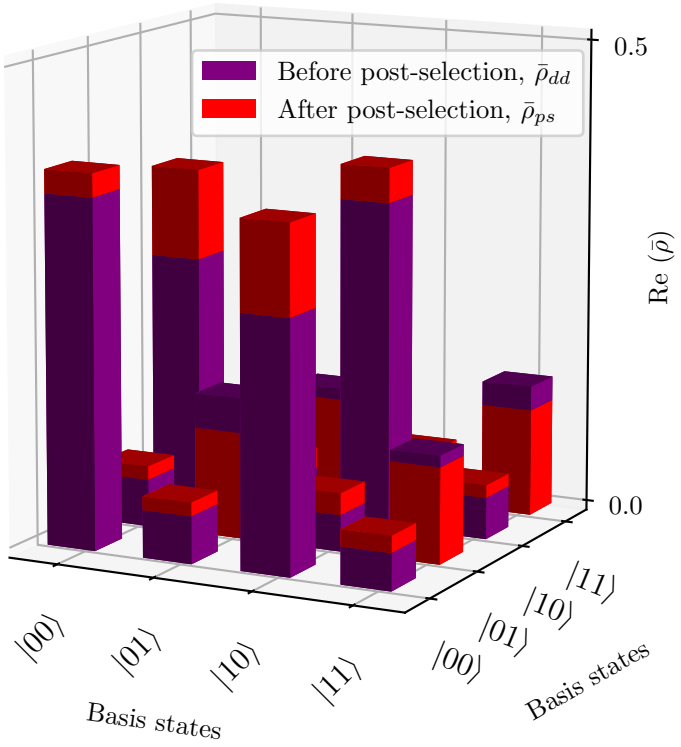


Figure 5.4: Plot of the real components of the density matrices $\bar{\rho}_{dd}$ and $\bar{\rho}_{ps}$ corresponding to the experimental output state. The height from the origin of the purple part of each bar shows the strength of the corresponding density matrix element before post-selection. The height from the origin of the red part of each bar shows the strength of the corresponding density matrix element after post-selection. *This figure was originally published in* [2].

5.5 Experimental results

The [[4, 2, 2]] CPC quantum memory circuit, depicted in figure 5.3c, was run on the IBMQX4 device between the 25$^{\text{th}}$ and 27$^{\text{th}}$ November 2017. A summary of the experimental results for state purity, fidelity and yield can be found in table 5.1. Error bars were calculated as one standard deviation of a single run value consisting of 8192 experimental executions of the quantum tomography circuit set. The standard error of the mean over all 154 runs was too small to be visible on our plots. Calibration data

	Purity, $P(\rho)$	**Fidelity**, $F(\rho)$	**Yield**
Before post-selection, $\bar{\rho}_{dd}$	0.52 ± 0.02	0.62 ± 0.03	100%
After post-selection, $\bar{\rho}_{ps}$	0.74 ± 0.03	0.75 ± 0.04	$(54 \pm 2)\%$
no. runs: 154 batches of 8192 shots			

Table 5.1: Quality metrics for the reconstructed density matrices before and after post-selection. The fidelity is calculated relative the target density matrix σ which is defined in equation (5.4.1). The yield is the proportion of shots per batch that are retained during the post-selection process. The errors are calculated as one standard deviation of a single run value consisting of 8192 experimental shots.

for the device on each of the three days of the experiment can be found in appendix C.

A total of 154 batches of 8192 shots were run over the course of the experiment. Figure 5.4 shows a plot of the real components of the elements of $\bar{\rho}_{dd}$ and $\bar{\rho}_{ps}$ averaged across the 154 batches. It is immediately clear that the post-selected density matrix $\bar{\rho}_{ps}$ better preserves the four target elements, which we identify as the non-zero elements in the target state σ given by equation (5.4.1). The bar-chart in figure 5.5 shows these target elements in isolation, from which it is apparent that post-selection has the biggest impact in preserving the strength of the off-diagonal coherences. This can also be seen when comparing the purity values, shown in table 5.1, for $\bar{\rho}_{dd}$ and $\bar{\rho}_{ps}$. The directly-decoded density matrix $\bar{\rho}_{dd}$ has a purity of $P(\bar{\rho}_{dd}) = 0.52 \pm 0.02$, implying it represents a near-fully mixed classical ensemble with a purity of 0.5. In contrast, the post-selected density matrix $\bar{\rho}_{ps}$ has a purity of $P(\bar{\rho}_{ps}) = 0.74 \pm 0.03$, suggesting it has undergone only partial decoherence.

The fidelities of $\bar{\rho}_{dd}$ and $\bar{\rho}_{ps}$ relative to the target state are $F(\bar{\rho}_{dd}) = 0.62 \pm 0.03$ and $F(\bar{\rho}_{ps}) = 0.75 \pm 0.04$ respectively. The fidelity of the post-selected state is therefore greater than the directly-decoded state with a confidence level of three standard deviations. From this we can conclude that the $[[4, 2, 2]]$ quantum memory produces useful syndrome information for protecting a $|\psi_{AB}\rangle = |+_A 0_B\rangle$ state. A consideration, however, is that the average yield (the proportion of results retained after post-selection) was $(54 \pm 2)\%$ averaged over the 154 batches.

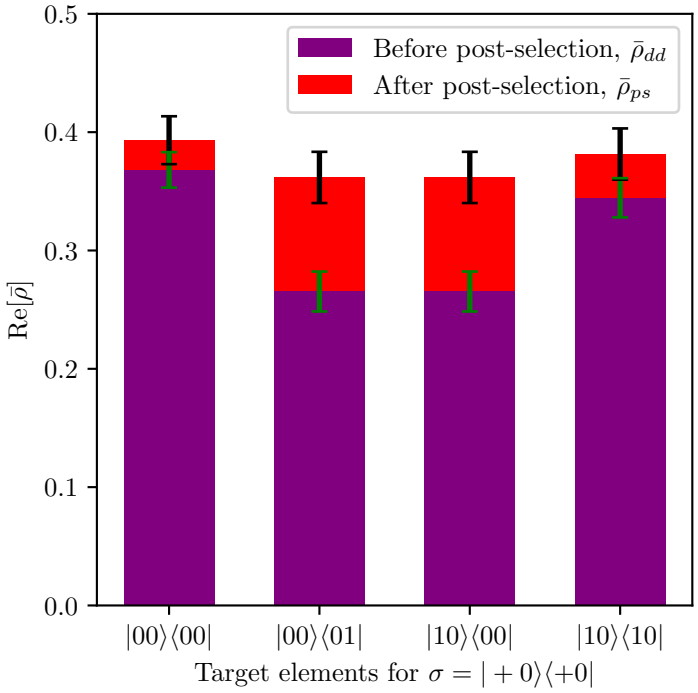


Figure 5.5: Plot of the target elements for ρ_{dd} and ρ_{ps}. The target elements are the non-zero elements in the density matrix σ given in equation (5.4.1). The errors are calculated as one standard deviation of a single run value consisting of 8192 experimental shots. The height from the origin of the purple part of each bar shows the strength of the corresponding density matrix element before post-selection. The height from the origin of the red part of each bar shows the strength of the corresponding density matrix element after post-selection. *This figure was originally published in* [2].

5.6 Summary of the IBM 5Q experiment

Our experiment on the IBM 5Q device showed that the syndrome information produced by a [[4, 2, 2]] CPC quantum memory can be used to improve the fidelity and purity of the code output. This is an interesting result, as it demonstrates the benefits of a quantum error detection protocol on an existing device with high error rates. Our method involved running a single round of the CPC protocol. The logical state was

then decoded, and analysed using quantum state tomography. The advantage of this approach was that syndrome information could be gathered without having to introduce additional ancilla qubits. However, once the error rates of quantum devices such as the IBM 5Q improve, it will become beneficial to implement multiple rounds of error detection. Our experiment should therefore be viewed as a proof-of-concept test for the hardware in its current state, rather than a complete solution for future devices.

In addition to our work, several other experiments have been performed on IBM Quantum devices to test versions of the $[[4, 2, 2]]$ detection code. In [97], it was shown that certain encoded gates can transform logical states with a higher fidelity than the equivalent circuit on raw qubits. Similarly, in [98], randomised benchmarking techniques were used to analyse the operation of encoded gates in the $[[4, 2, 2]]$ codespace. Again, the results showed an improvement in performance compared to the equivalent gates before encoding. The advantage of using randomised benchmarking is that it provides a way of measuring gate fidelity that is independent of state preparation and measurement (SPAM) errors [98].

Chapter 6

Native gates for CPC codes

In our discussion of the CPC framework so far, quantum codes have been expressed in terms of CNOT and conjugate-propagator gates. The use of such 'ideal' gates allows for intuitive visualisation of the propagation of errors through the decoder, and simplifies the calculation of syndrome tables via the techniques described in section 4.4.5. However, in practice, the native two-qubit entangling interaction of a given experiment will be of a different form. As a result, when compiling a QEC code, additional operations are required to allow CNOT and conjugate-propagator gates to be synthesised from the native interaction. If the native interaction is maximally entangling, this will involve the addition of single-qubit corrections. Consequently, it can be useful to approach the problem of code design from the starting point of the native-gate set. This simplifies the process of translating the code from its theoretical representation to its hardware-compiled form.

In this chapter, based on work first published in [2], I explore how the symmetric *encode-error-decode* structure of the CPC framework allows the operation of stabilizer codes to be easily studied using a native gates representation rather than CNOT and conjugate-propagator gates. I first show how the $[[4, 2, 2]]$ CPC code, introduced in section 4.2, can be efficiently compiled in terms of an ion trap native gate. Following this, I demonstrate that this result generalises any realistic maximally entangling native gate. The examples in this chapter outline native gate compilation strategies for the

non fault tolerant CPC representation of quantum codes. However, the simplification procedures I outline could equally be applied to codes implemented fault tolerantly.

6.1 An ion trap native gate

Ion traps are considered one the leading platforms for quantum computation. Ion-based qubits have long coherence times and high-fidelity two-qubit gates [27, 86]. Furthermore, they can be read out with near 100% efficiency [99]. It has also been proposed that multiple ion-trap cells could be networked via auxillary qubit systems to create larger hybrid quantum computers [100]. In such a hybrid networked architecture, good QEC codes will be vital to ensure the quantum data in each ion trap is protected.

In this chapter we consider the construction of CPC codes using an ion trap native gate. Specifically, we consider an ion trap with a two-qubit entangling native gate that gives rise to a unitary matrix of the form

$$U = \exp\left(-\mathrm{i}\frac{\pi}{2}\left[Z \otimes Z\right] t\right) = e^{\mathrm{i}\pi t/2}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-\mathrm{i}\pi t} & 0 & 0 \\ 0 & 0 & e^{-\mathrm{i}\pi t} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6.1.1}$$

where t is a tuning parameter. Such interactions can be realised via geometric phase gate procedures [101, 102, 103]. In this paper, we consider the symmetrised phase (SP) gate, which is one of the simplest possible maximally entangling gates that arises from the above ion trap unitary matrix [104]. The SP native gate is realised by setting the tuning parameter in equation (6.1.1) to $t_{\mathrm{SP}} = 1/2$. Up to a global phase, the gate can then be described as a matrix, F, of the form

$$F_{q_2}^{q_1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathrm{i} & 0 & 0 \\ 0 & 0 & \mathrm{i} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{6.1.2}$$

where q_1 and q_2 are the input qubits to the gate. In the following section, we explicitly show how a $[[4, 2, 2]]$ detection code can be efficiently compiled with the SP native gate of equation (6.1.2). Building on this example, we then demonstrate how efficient compilation is in principle possible for any experimentally realistic maximally entangling native gate.

6.2 Compiling the $[[4, 2, 2]]$ CPC detection with an ion trap native gate

Here we show that the $[[4, 2, 2]]$ CPC detection code, introduced in section 4.2, can be efficiently compiled with an ion trap native gate. For the purposes of this example, we adopt an ion trap with a SP native gate as introduced in equation (6.1.2) in section 6.1. The SP native gate can be transformed into a CNOT via the application of local unitary operations to its inputs and outputs. A possible mapping, in matrix equation form, is given by

$$\mathrm{CNOT}_{q_2}^{q_1} = (I_{q_1} \otimes H_{q_2}) \cdot (P_{q_1} \otimes P_{q_2}) \cdot F_{q_2}^{q_1} \cdot (I_{q_1} \otimes H_{q_2}) \,, \tag{6.2.1}$$

where $F_{q_2}^{q_1}$ is the matrix representation of the SP gate defined in equation (6.1.1), and P is a phase gate defined as $P = \mathrm{diag}\,(1, -\mathrm{i})$. Realising a CNOT gate on ion trap hardware, via the above mapping, requires the application of the native gate combined with four single-qubit gates, as shown in figure 6.1a. Likewise, figure 6.1b shows how the conjugate-propagator gate can be constructed from the native gate via the addition of six single-qubit operations. We will see that, when the native gates are compiled into a CPC circuit, constructive simplifications become possible to reduce the total number of single-qubit gates required.

Figure 6.2 illustrates the steps involved in the compilation and simplification of the $[[4, 2, 2]]$ CPC code with the SP native gate. The original circuit, expressed in terms of CNOT and conjugate-propagator gates, is shown in figure 6.2a. The first step of compilation involves substituting the CNOT and conjugate-propagator gates with the SP native interaction, via the circuit rewrites rules defined in figure 6.1. The resultant

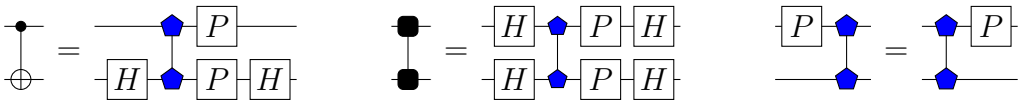


Figure 6.1: (a) A CNOT gate expressed in terms of the SP native gate, which is represented by the connected blue pentagons. The matrix form of the SP native gate is given in equation (6.1.2). (b) A conjugate-propagator gate expressed in terms of the SP native gate. (c) Both the phase gate P and the SP native gate are represented as diagonal matrices in the computational basis. As a result, the P gate can be moved freely through the SP native gate. *This figure was originally published in* [2].

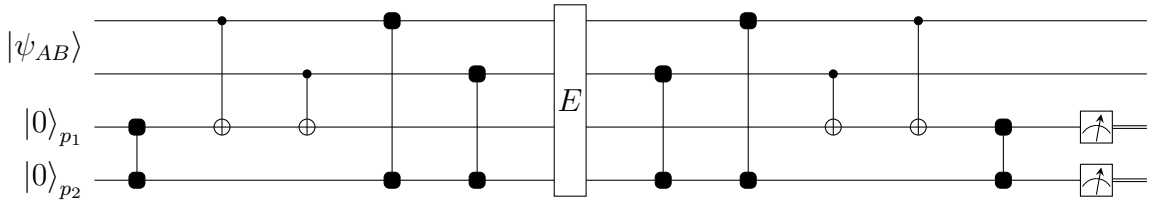
circuit is shown in figure 6.2b.

Now that the circuit is written in terms of the native gate, circuit simplifications can be applied to reduce the single-qubit gate count. In figure 6.2b, pairs of H gates that cancel to the identity are labelled in red. In the encoder, the H gates labelled in blue are paired with their counterparts from the decoder. We can now exploit the symmetry of the CPC code to further reduce the gate-count. The effect of the blue H gates around the wait-stage is to transform X errors into Z errors and vice-versa, as described by the following matrix transformations

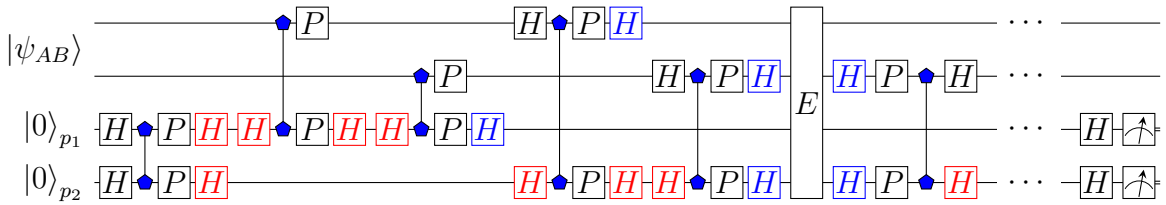
$$\begin{aligned} H \cdot (E = X) \cdot H &= H \cdot X \cdot H = Z, \\ H \cdot (E = Z) \cdot H &= H \cdot Z \cdot H = X, \end{aligned} \tag{6.2.2}$$

where E represents the error that occurs in the wait stage. The $[[4, 2, 2]]$ code can detect both X and Z errors, as shown in syndrome table 4.1 in section 4.2. As a result, the blue H gates do not change the errors into a form that cannot be detected. The blue H gates can therefore be discarded without affecting the operation of the $[[4, 2, 2]]$ code.

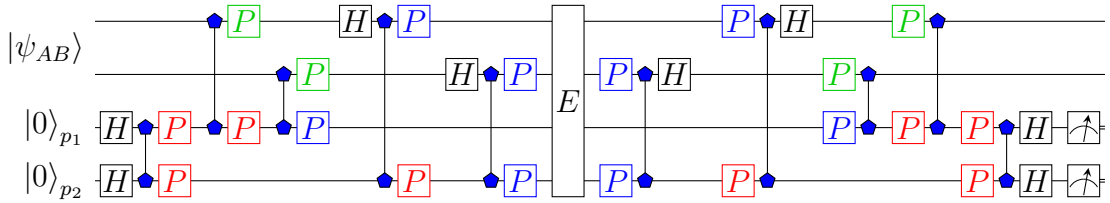
Figure 6.2c shows the compiled $[[4, 2, 2]]$ code following the removal of the unnecessary H gates. Notice that both the P gate and the SP gate are described by diagonal matrices in the computational basis. As a result, we have the freedom to move P gates



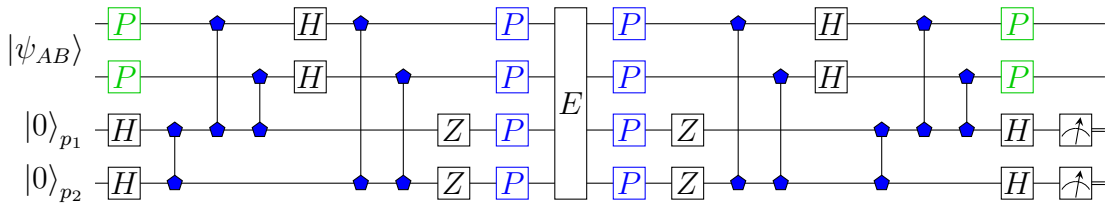
(a) The original [[4,2,2]] CPC error detection code expressed in terms of CNOT and conjugate propagator gates.



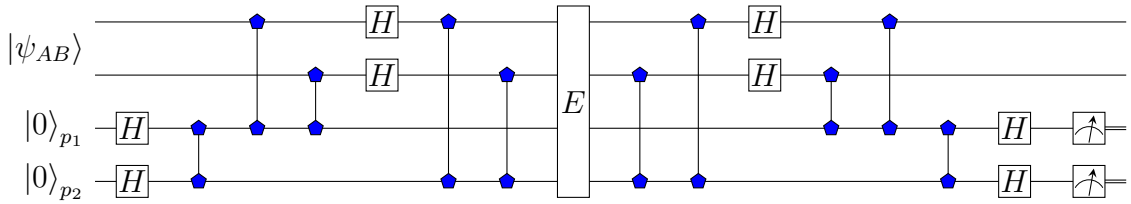
(b) The compiled circuit prior to simplification. Note that parts of the decoder have been hidden to save space. The pairs of Hadamards, labelled red, cancel to the identity. The blue H gates can also be discarded without affecting the operation of the code.



(c) The circuit following removal of the H gates. The pairs of red P gates combine to form Z gates.



(d) The Z gates and blue P gates can be moved freely through the SP gates to the centre of the circuit. Due to their symmetry about the error window, the P and Z gates can be omitted from the code. The green P gates do not affect circuit operation and are also discarded.



(e) The compiled [[4,2,2]] CPC detection code following circuit simplification. Only four single-qubit gates remain in the encoder.

Figure 6.2: Compiling the $[[4, 2, 2]]$ detection code with the SP native gate. *This figure was originally published in* [2].

through the SP native gate as shown in figure 6.1c. Two P gates combine to form a Z gate as follows $P \cdot P = Z$. In the circuit in figure 6.2c, pairs of P gates are highlighted in red. As Z gates are diagonal in the computational basis, they can also be moved through the SP gates.

In the circuit in figure 6.2d, the Z gates and blue P gates have been pushed to the centre of the circuit. In the event that no error occurs, these P gates combine to form a Z-error via the relation $Z = P \cdot P$. However, the locations of these errors are known, and they can therefore be accounted for in post-processing. If an error does occur, the effect of symmetric P gates about the wait stage, E, is to transform X-errors into Y-errors and vice versa, as described by the following matrix transformation rules

$$\begin{aligned} P \cdot (E = X) \cdot P &= P \cdot X \cdot P = (-\mathrm{i})Y, \\ P \cdot (E = Y) \cdot P &= P \cdot Y \cdot P = (-\mathrm{i})X, \end{aligned} \tag{6.2.3}$$

where the $(-\mathrm{i})$ global phase does not affect the syndrome measurement. These transformations are unproblematic as $[[4, 2, 2]]$ code can detect both X and Y errors (see syndrome table 4.1). As the effect of the blue P gates can be described in terms of single-qubit Clifford operations on the output, they can be removed from the circuit and accounted for in post-processing. There are also P gates highlighted in green, located on the register qubits at the beginning and end of each error cycle. These gates occur before the first round of CPC checks, and can therefore be removed from the circuit without affecting the final syndrome readout. Finally, the Z gates located symmetrically about the wait-stage introduce a global phase to the errors. This global phase does not affect the propagation of errors through the circuit, meaning the Z gates can be removed. It should be noted that the above simplifications will result in a modified syndrome table. However, the *no-error* case will remain unique meaning the function of the code is maintained.

The final simplified form of $[[4, 2, 2]]$ CPC code compiled with the SP native gate is shown in figure 6.2e. The single-qubit gate count in the encoder has been lowered from 26 gates in the original compiled circuit (figure 6.2b), to 4 gates in final circuit (figure 6.2e).

6.3 Requirements for CPC gates

We have now shown that the $[[4, 2, 2]]$ code can be efficiently compiled with the SP native gate. Most of the single-qubit corrections can be eliminated, either by direct cancellation between adjacent Hadamards, or by moving P gates through the circuit. We now show that efficient CPC code translation, from the idealised CNOT version to the hardware-compiled version, is possible for a range of native gate types. We begin by outlining the general requirements for two-qubit gates in a CPC circuit.

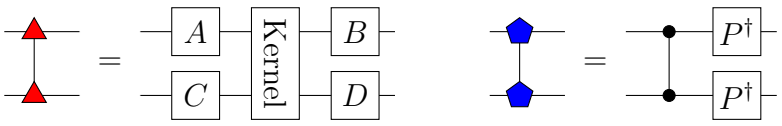
In a CPC code, the role of two-qubit interaction gates is to distribute error information from the register to the parity qubits. For example, CNOT gates propagate bit-errors from their control to target via the rule in equation (3.6.3). More generally, we require that the two-qubit CPC gate, $\Omega_{q_2}^{q_1}$, has the ability to change the weight of an error operator, $E_{q_1}^i \otimes \mathbb{1}_{q_2}$, such that

$$\Omega_{q_2}^{q_1} \cdot \left(E_{q_1}^i \otimes \mathbb{1}_{q_2}\right) \cdot \left(\Omega_{q_2}^{q_1}\right)^\dagger = \left(E_{q_1}^j \otimes E_{q_2}^k\right), \tag{6.3.1}$$

where q_1 and q_2 are the control and target qubits respectively, and $E^{i,j,k}$ are non-identity elements of the single-qubit Pauli group. As both $E_{q_1}^i \otimes \mathbb{1}_{q_2}$ and $E_{q_1}^j \otimes E_{q_2}^k$ are Pauli group operators, we see that $\Omega_{q_2}^{q_1}$ must be a Clifford gate (for an overview of the Clifford group see appendix B). CPC quantum memories can be described entirely in terms of Clifford gates, as their operations are restricted to manipulating stabilizer states.

Another way of thinking about the CPC interaction gates is in terms of entanglement. In equation (6.3.1), it can be seen that the general CPC gate de-localises error information from the control to the target, suggesting the operation has the potential to entangle states. Furthermore, we know that elements of the two-qubit stabilizer states are either maximally entangled or separable. Any Clifford entangling gate that maps between these states, and therefore any CPC interaction, is a maximally entangling operation.

We have now established that CPC gates must be maximally entangling Clifford operations. However, many experiments will have native gates that do not satisfy these



(a) (b)

Figure 6.3: Left: A general maximally entangling Clifford native gate (red triangles) can be expressed in terms of a kernel supplemented by local corrections on its inputs and outputs. The kernel will always be of the form CZ or CZ-SWAP . The local corrections, $\{A, B, C, D\}$, are single-qubit Clifford gates and can expressed as products of H and P gates. Right: The SP native gate expressed in terms of its CZ kernel. *This figure was originally published in* [2].

requirements. For example, several qubit technologies have a native interaction of the form $\sqrt{\text{SWAP}}$ [105], which is only partially entangling. In these circumstances, multiple applications of the native gate, in addition to local operations, are required to synthesise the desired maximally entangling behaviour. It is often the case that quantum computing experiments will have different error rates for single-qubit and two-qubit gates [28]. Circuit compilation strategies should therefore aim to minimise the gate-type with the highest error rate. In the case of ion traps, for example, the two-qubit gates have lower fidelities than single-qubit gates [27, 86].

6.4 Circuit simplification with any maximally entangling Clifford gate

We will now outline general CPC circuit simplification procedures for maximally entangling Clifford gates. It can be shown that all Clifford entangling gates are local Clifford equivalent to either the CZ or the CZ-SWAP interaction [106, 107]. With this knowledge, we can write all maximally entangling Clifford gates in terms of a central kernel, containing either a CZ or CZ-SWAP interaction, supplemented by local Clifford gates (see figure 6.3a).

The single-qubit Clifford group is generated by P and H gates. Any native gate can

therefore be constructed from its by kernel via the addition of local gates generated from combinations of P and H. The P gates can be trivially pushed through the CZ kernel. Likewise, it is possible to push P gates through the CZ -SWAP kernels, although the effect of the SWAP gate must be taken into account.

In section 7.2.4 we demonstrated the compilation of a CPC code using the SP native gate, which is local Clifford equivalent to a CZ gate. The exact transformation from CZ kernel to SP gate is shown in figure 6.3b. As the local operations in this case consist of $P^\dagger$ gates, we have the freedom move P gates through the SP native gate. Hadamard gates H, however, restrict movement, but in many cases will cancel when the native gate is compiled into a CPC circuit.

The general procedure for compiling a CPC code with a given native gate can now be written as follows. First, eliminate any unnecessary H gates by identifying cancellations between adjacent CPC gates. Second, determine the behaviour when P gates are pushed through the native gate. As all maximally entangling Clifford gates have either a CZ or CZ-SWAP kernel, it is often possible to trivially move P gates through each block of the encoder. Once these simplifications rules have been established, they can be applied systematically to substantially reduce the CPC circuit gate count.

Chapter 7

Automated design of CPC codes

In chapter 4 I described how the CPC framework provides new approaches to the design of stabilizer codes. The symmetric *encode-error-decode* structure of CPC codes ensures that a commuting set of stabilizers is always obtained, regardless of the form of the parity checks upon which the code is based. The only task that then remains to verify whether the resultant CPC circuit 'works' is to measure the code distance for a given error model. Therefore, the CPC framework essentially reduces quantum code design to a classical decoding problem. In this chapter, I take advantage of this feature of the CPC framework to show how good quantum codes can be discovered via simple machine search techniques. As a case study, I describe a design process for the discovery of CPC code optimised for the hardware and layout demands of a seven-qubit ion trap. The methods and results outlined in this chapter were first published in [2].

7.1 Machine search for CPC code discovery

A CPC code which encodes k logical qubits in n physical qubits can be compactly described in terms of three adjacency matrices. These matrices – m_b, m_p and m_c – were defined in section 4.3 and describe the parity checking sequences in the bit, phase and cross-checking stages of the encoder respectively. Owing to the fail-safe CPC code structure, every combination of these three adjacency matrices will correspond to an

$[[n, k, d =?]]$ stabilizer code where d is the code distance. As a result, new CPC codes can be discovered simply by generating new permutations of the adjacency matrices and selecting the combinations that have the desired code distance. The CPC approach to code design allows machine search routines to be set up over a space of stabilizer codes. This search can be performed using exhaustive, random or more sophisticated strategies to select the adjacency matrices.

The maximum possible encoding density of a non-degenerate quantum error correction code is constrained by the quantum Hamming bound, defined in equation 3.2.2 in chapter 3. In section 4.5, a method was outlined for converting pairs of classical Hamming codes to a quantum CPC code. This approach relied on a tripartite structure, in which the code qubits were separated into data, bit-checking and phase-checking blocks. Whilst the tripartite CPC structure is useful for proving general code properties, it does not always allow for the discovery of codes that saturate the Quantum Hamming bound. In this section, we show how exhaustive search methods can be used to discover CPC codes with optimal rates.

7.2 Case study: Designing a CPC code for a seven-qubit ion trap

The first generation of quantum computers will be limited in size to no more than a couple of hundred qubits [92, 108]. In this section I outline a design process for constructing hardware-optimised quantum codes with the CPC framework. To illustrate this design process, we outline its application for a idealised seven-qubit linear ion trap. For simplicity, we limit this example to the discovery of a non-fault tolerant CPC code for a basic quantum error model. Whilst this first example is proof-of-concept, the techniques we introduce will also be applicable to the discovery of codes designed for full fault tolerant implementation.

Our CPC design process is split into three stages. 1) **CPC code discovery**: numerical search techniques are used to find CPC codes that maximise the quantum encoding density for a seven qubit register. 2) **SWAP gate compilation**: the best CPC codes

from the discovered set are identified by analysing which ones have the lowest two-qubit count when implemented on a linear nearest-neighbour architecture. 3) **Native gate compilation**: further optimisations are made by identifying CPC circuits with efficient translations from the CNOT version of the code to the native gate version, using the circuit simplification strategies outlined in chapter 6.

7.2.1 Overview of the ion trap model

In our first demonstration of the CPC design process, we consider an idealised linear ion-trap with seven application qubits. This scheme has been chosen because several existing ion trap experiments have a similar size and layout [27, 29, 109, 110]. We assume that this ion trap has a native two qubit entangling gate of the symmetrised phase gate variety introduced in section 6.1 and defined by equation 6.1.2.

In the design of a CPC code for this seven-qubit ion trap, we assume that during the wait stage the ion trap qubits are subject to a biased depolarizing noise channel of the form

$$\mathcal{E}[\rho] = (1 - p_x - p_z - p_x p_z)\rho + p_x X\rho X + p_x p_z Y\rho Y + p_z Z\rho Z, \tag{7.2.1}$$

where ρ is the single-qubit density matrix, and p_x and p_z are the probabilities of X- and Z-errors respectively. This error model assumes the ion trap has independent error mechanisms for X- and Z-errors, but Y-errors occur only as a result of successive single-qubit errors of the form XZ and ZX [a]. Similar error models have recently been considered in [111, 112]. For the purposes of our ion trap model, we assume that the error probabilities p_x and p_z are low enough that the probability of Y-errors becomes negligibly small. The effective error model can then be written as

$$\mathcal{E}[\rho] \approx (1 - p_x - p_z)\rho + p_x X\rho X + p_z Z\rho Z. \tag{7.2.2}$$

Under the above error model for the idle ion trap qubits, the CPC quantum memory

[a] Note that the ion trap we consider is an idealised proof-of-concept model, and does not correspond to a specific ion trap experiment.

only needs to correct X- and Z-errors. We choose this noise model for our proof-of-concept outline of the CPC design process, as it corresponds to the simplest possible non-classical error model. Our aim is to discover non-degenerate quantum codes which produce a unique syndrome for single X- and Z-errors on any of the seven qubits in the trap.

When optimising codes for a specific device, it is important to consider the connectivity between qubits. For our ion trap model, we assume that arbitrary single-qubit operations can be performed on any of the ions in the register. On ion trap hardware, it is in principle possible to implement interactions between spatially separated qubits, for example, by exploiting the collective vibrational modes of the ions as a quantum bus [101]. In practice, however, the fidelity of two-qubit interactions decreases with separation [113]. For this reason, in our idealised model ion trap, two-qubit gates are limited to nearest-neighbour interactions. Under nearest-neighbour constraints, interactions between spatially separated qubits are achieved by performing SWAP operations to move quantum information around the trap. These SWAP operations can be realised either by physically shuttling qubits between zones of the trap [114], or by synthesising SWAP gates from CNOT interactions [113]. In stage 2 of the CPC design process, we show how CPC codes can be compiled with SWAP gates to allow for implementation with only nearest neighbour interactions.

7.2.2 Stage 1: CPC code discovery

The first stage in the CPC design process involves setting up and implementing a routine to discover codes appropriate for the target hardware. For the seven-qubit ion trap subject to the error model defined in equation 7.2.2, the quantum Hamming bound (equation 3.2.2) tells us that the maximum number of data qubits that can be encoded in seven physical qubits is $k_{\max} = 3$ at distance $d = 3$. The optimal seven-qubit CPC code will therefore be of the form $[[n = 7, k = 3, d = 3]]$. Note that as they are distance $d = 3$, the $[[7, 3, 3]]$ codes will be able to correct one error per CPC cycle.

An advantage of the CPC framework lies in the fact that new instances of such optimal codes can be discovered numerically, either using brute-force or more sophisticated

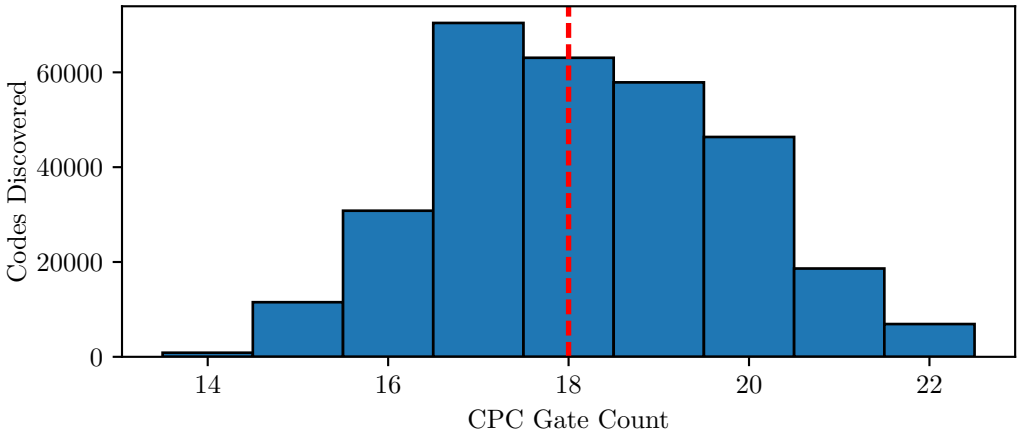


Figure 7.1: A histogram showing all of the $[[7, 3, 3]]$ CPC codes discovered in stage 1 of the CPC design process, binned by encoder length. The CPC gate count is the combined total of CNOT and conjugate propagator gates in the encoder. The median length, marked in red, is 18. In comparison, the shortest $[[7, 3, 3]]$ circuits have a CPC gate count of 14. *This figure was originally published in* [2].

optimisation techniques. We now demonstrate these strategies in practice, by showing how optimal $[[7, 3, 3]]$ CPC codes can be discovered via exhaustive search.

A $[[7, 3, 3]]$ CPC code has $k = 3$ data qubits and $n-k = 4$ parity qubits. The adjacency matrices therefore have the form

$$m_b = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{pmatrix}, \quad m_p = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \end{pmatrix},$$
$$m_c = \begin{pmatrix} 0 & c_{12} & c_{13} & c_{14} \\ c_{12} & 0 & c_{23} & c_{24} \\ c_{13} & c_{23} & 0 & c_{34} \\ c_{14} & c_{24} & c_{34} & 0 \end{pmatrix}, \tag{7.2.3}$$

where b_{xy}, h_{xy}, c_{xy} are binary values. New CPC circuits can be made by generating

Encoder gate length (number of CPC gates)		
Minimum	**Median**	**Depth reduction**
14 (864 discovered)	18	22%
Size of $[[7, 3, d =?]]$ search space: 1.07×10^9 circuits		
Number of $[[7, 3, 3]]$ codes discovered: $306,480$ (0.03% of search space)		
Number of symmetry-inequivalent $[[7, 3, 3]]$ codes: 2190		

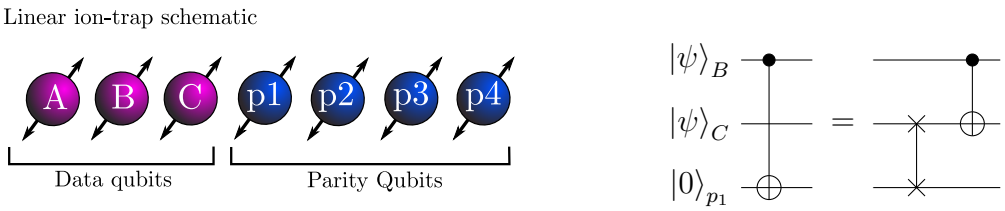
Table 7.1: Summary of the exhaustive search for $[[7, 3, 3]]$ CPC codes. The number of CPC gates is defined as the combined total of CNOT + conjugate propagator gates in the encoder. The depth reduction is calculated as the percentage decrease in the encoder length of the smallest circuit relative to the median.

different instances of these matrices. The syndrome matrix for single-qubit X- and Z-errors is then given by

$$\begin{array}{c} \\ [E_{dx}] \\ [E_{dz}] \\ [E_{px}] \\ [E_{pz}] \end{array} \begin{array}{c} \overset{[p_1,\ldots,p_4]}{} \\ \left[\begin{array}{c} m_b \\ m_p \\ \mathbb{1}_4 \\ m_p^T \cdot m_b \oplus m_c \end{array}\right] \end{array}. \tag{7.2.4}$$

The code search task involves finding combinations of the matrices m_b, m_p and m_c that result in a syndrome matrix with unique rows. These combinations describe the encoders of $[[7, 3, 3]]$ CPC codes.

The number of possible combinations of the adjacency matrices for CPC circuits of type $[[7, 3, d =?]]$ is 2^{30}. By an exhaustive search, we have discovered that $306,480$ of these permutations (0.03% of the search space) are working $[[7, 3, 3]]$ codes. These codes have distance $d = 3$, and produce unique syndromes for all single-qubit X- and Z-errors across the seven qubits. Of the discovered set, there are 2190 symmetry-inequivalent codes that cannot be transformed from one to another by rearranging the qubit order. However, some permutations of the same code are more amenable to circuit optimisation than others. We will therefore continue to consider the entire set



(a) (b)

Figure 7.2: (a) A schematic of the seven-qubit linear ion trap. Three of the qubits have been labelled as data qubits and four as parity qubits as required by the $[[7, 3, 3]]$ code. It is assumed that entangling gates can only be performed between nearest-neighbour qubits. (b) A CNOT gate between spatially separated qubits can be implemented using only nearest-neighbour interactions through the addition of SWAP gates. *This figure was originally published in* [2].

of $306,480$ codes in the CPC design process.

Now that a set of $[[7, 3, 3]]$ circuits has been found, the next stages in the CPC design process involves analysis to determine which one of the $306,480$ codes is best suited for implementation on the ion trap device. Figure 7.1 shows a histogram of the discovered $[[7, 3, 3]]$ codes, binned by the combined number of CNOT gates and conjugate propagator gates in their encoder. This quantity will be referred to as the CPC gate count.

In ion trap hardware, inter-qubit operations are typically the most expensive gate-type in terms of their potential to introduce errors [27, 86]. As a result, the CPC circuits with the lowest CPC gate count are most desirable. In the set of $[[7, 3, 3]]$ codes, the shortest circuit encoders have 14 CPC gates. This is a 22% reduction in circuit depth compared to the median gate count of 18. The number of $[[7, 3, 3]]$ circuits with the minimum encoder depth of 14 is 864 of which 245 are symmetry inequivalent. Further work is therefore necessary to narrow down the code set, and find the optimum quantum memory for the ion trap device.

The encoder length statistics for the $[[7, 3, 3]]$ CPC codes are summarised in table 7.1. Note that in this simple first analysis, we have not accounted for any of the constraints

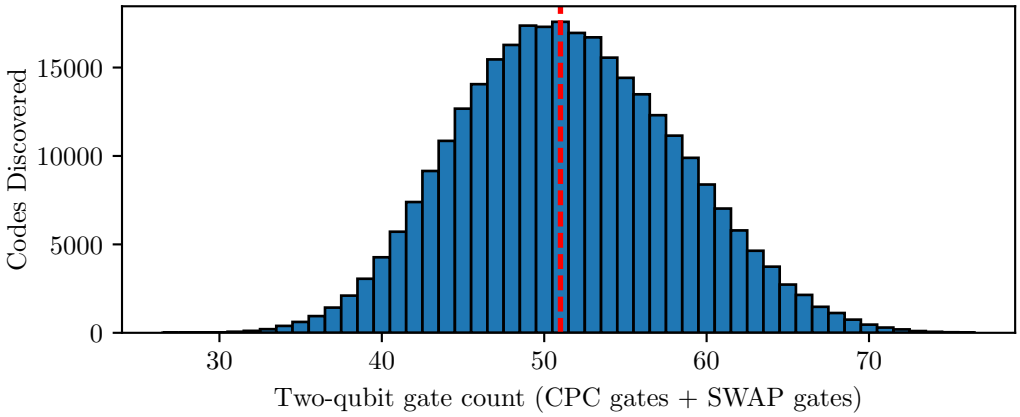


Figure 7.3: The distribution of $[[7, 3, 3]]$ CPC codes binned by two-qubit gate count after compilation onto a nearest-neighbour architecture. The total gate count is defined as the number of CPC gates + SWAP gates in the encode stage of the circuit. *This figure was originally published in* [2].

imposed by the ion-trap's nearest-neighbour requirement for two-qubit operations. In the next section, we outline how the $[[7, 3, 3]]$ codes can be compiled in such a setting through the introduction of additional SWAP gates.

The results in this section demonstrate that the CPC framework provides constructive tools for discovery of optimal $[[7, 3, 3]]$ codes that saturate the quantum Hamming bound. Furthermore, the search was performed using a simple brute-force technique that requires only a basic knowledge of the CPC code structure to implement. The Python script used to perform the code search is approximately 200 lines long, and required approximately 4 days to run on a CPU clocked at 3.2GHz with 8Gb of RAM.

7.2.3 Stage 2: SWAP gate compilation

The second stage of the CPC design process involves selecting codes to meet the demands of the chosen quantum hardware and its qubit layout. Figure 7.2a shows the idealised model ion trap under consideration, labelled with 3 data qubits and 4 parity qubits as required by the $[[7, 3, 3]]$ code. Under the restriction of nearest-neighbour

Encoder gate length (number of two-qubit gates)		
Minimum	**Median**	**Depth reduction**
27 (1 discovered)	51	47%
Optimum code:		
Encoder length=27 gates; no. CPC gates=14; no. SWAP gates=13		

Table 7.2: Summary of the gate-count statistics for the set of $[[7, 3, 3]]$ codes following the SWAP gate compilation. The two-qubit gate count is defined as the number of CPC gates + SWAP gates. The depth reduction is the percentage decrease in gate-count of the smallest circuit relative to the median.

connectivity, interactions between spatially separated qubits can still be realised by performing SWAP operations. For example, interacting qubit B with p_1 would first require a SWAP gate between qubits B and C, or qubits C and p_1. In general, circuits with fewer long range interactions will require fewer SWAP gates, and will therefore have a reduced two-qubit gate count.

There are a number of strategies for calculating the sequences of SWAP operations required to compile a CPC circuit on a nearest-neighbour architecture. Here we adopt a simple approach in which qubits are always swapped in the upwards direction in the diagram. As an example of this, in figure 7.2b, qubit p_1 is swapped upwards, instead of qubit B being swapped downwards. More advanced SWAP compilation strategies, that combine upwards and downwards moves, can yield circuits with lower SWAP counts. However, such analysis is computationally expensive, and can impose a bottleneck in the CPC design process. By restricting our approach to upwards SWAP moves only, an exhaustive search of the $[[7, 3, 3]]$ CPC codes remains possible, and is sufficient for this case study.

Figure 7.3 shows the histogram of the SWAP compiled $[[7, 3, 3]]$ codes distributed by the total two-qubit gate count (CPC gates + SWAP gates). The optimum $[[7, 3, 3]]$ CPC code with the shortest encoder is shown in figure 7.4b. The encoder for this circuit includes 14 CPC gates, and requires an additional 13 SWAP operations to be implemented on a linear, nearest-neighbour architecture. The depth of the encoder,

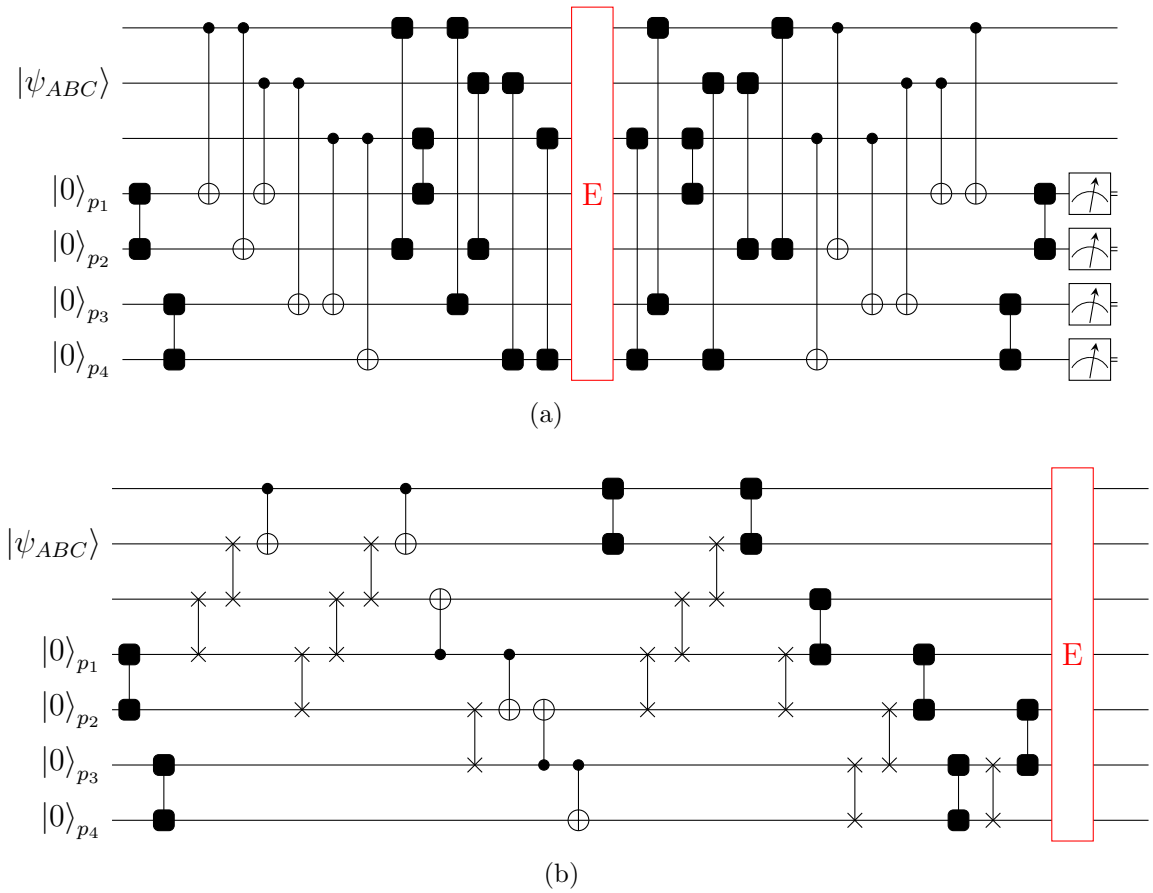


Figure 7.4: Circuit diagrams demonstrating SWAP gate compilation for a nearest-neighbour architecture. (a) The $[[7, 3, 3]]$ code with the smallest two-qubit gate count prior to the addition of SWAP gates. (b) The encoder for the same circuit, with SWAP gates included. *This figure was originally published in* [2].

	Local gate count (number of single-qubit gates)	
	Mininum	**Median**
Before simplification	72 (1 discovered)	92
After simplification	7 (1 discovered)	10
% **change**	90%	89%

Table 7.3: Summary of the local gate-count for the $[[7, 3, 3]]$ code following compilation with the SP native gate.

in terms of the number of two qubit gates, is therefore 27. For comparison, the uncompilied version of this $[[7, 3, 3]]$ code is shown in figure 7.4a.

The results of two-qubit gate count analysis for the $[[7, 3, 3]]$ codes, following compilation onto the nearest-neighbour hardware, are summarised in table 7.2. The optimum circuit has an encoder length of 27, compared to the median of 51, a 47% reduction in circuit gate count. Only one CPC code was discovered with the minimum encoder length. The CPC circuit optimisation, with regards to qubit layout, can therefore be considered complete.

7.2.4 Stage 3: Native gate compilation

The ion trap under consideration has a native gate that resembles the symmetrised phase (SP) gate introduced in section 6.1. The final stage of the CPC design process involves systematically applying the SP simplification procedures described in section 6.2 to each of the 306,480 discovered CPC codes. The compilation efficiency of a given code can be quantified by counting the number of local gates that remain in the simplified circuit. The optimal code for the ion trap device is then identified as the circuit with the shortest total encoder length, defined by

$$\mathcal{L}_{\text{CPC}} = |\text{CPC}| + |\text{SWAP}| + |\text{LOCAL}|, \tag{7.2.5}$$

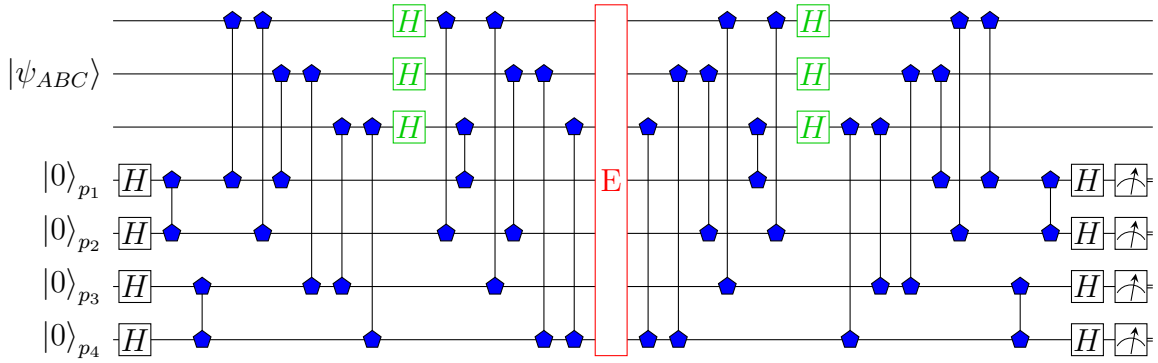


Figure 7.5: The native gate compiled form of the $[[7, 3, 3]]$ CPC code with the lowest total gate count. Note that the SWAP operations have been omitted to save space. *This figure was originally published in* [2].

where $|\text{CPC}|$ is the CPC gate count, $|\text{SWAP}|$ is the SWAP gate count and $|\text{LOCAL}|$ is the local gate count.

Table 7.3 summarises the simplification statistics for the local gate counts when the $[[7, 3, 3]]$ CPC codes are compiled with the SP native gate. Without applying any simplifications, the median local gate count is 92. After applying the simplification routine, the median is 10, an 89% reduction in gate count.

The compiled $[[7, 3, 3]]$ CPC circuit with the lowest local gate count after simplification is shown in figure 7.5. This circuit is compiled from the CPC code with the lowest two-qubit gate count, as discovered in the last section and depicted in figure 7.4. We can therefore identify the compiled $[[7, 3, 3]]$ code in figure 7.5 as the optimum code for our device with a total gate count of $\mathcal{L}_{\text{CPC}} = 34$. For comparison, the median total gate count across all $306,480$ CPC codes was $\mathcal{L}_{\text{CPC}} = 61$. The total reduction in circuit depth for the optimised circuit relative to the median is therefore 44%. Note that it will not always be the case that the circuit with the lowest two-qubit gate count will also be the circuit that compiles most efficiently. For this reason, the entire discovered set of $[[7, 3, 3]]$ CPC codes were considered in stage 3 of the design process, rather than restricting the analysis to the single code identified in stage 2.

7.3 Extending the CPC design process

In this chapter we have outlined a design process for the automated discovery and optimisation of CPC codes by applying it to a seven qubit ion trap device. In the first stage of the design process, exhaustive code-search methods were used to find $[[7, 3, 3]]$ CPC codes that saturate the quantum Hamming bound for seven qubits. These circuits were then modified through the addition of SWAP gates to allow them to be implemented on a nearest-neighbour architecture. Finally, the circuits were compiled with a SP native gate. At the end of the design process, the optimum hardware-ready code with the lowest gate count of $\mathcal{L}_{\text{CPC}} = 34$ was identified.

The design process outlined for ion traps will be adaptable to other qubit technologies. In chapter 6 we demonstrated that the symmetric *encode-error-decode* structure of CPC codes allows for efficient compilation with any realistic maximally entangling Clifford gate. This result means that simplification routines, similar to those seen with the ion trap SP gate, will be possible for a broad range of native gates from different quantum hardware.

The final circuit in the outline of the CPC design process, drawn in figure 7.5, shows the best CPC code in terms of total gate count. Here it was assumed, however, that each gate type – CPC, SWAP and local – are equal in terms of the overhead they impose on the code implementation. In practice, however, some types of operations will be more expensive than others. For example, in an ion trap setting, it is typically the case that two-qubit interactions have a lower fidelity than single-qubit operations [27, 86]. When implementing the CPC design process, such considerations should be taken into account for choosing the optimum code for the given device. For example, each CPC code could be assigned a weighted total gate count, $\mathcal{R}_{\text{CPC}}$, given by

$$\mathcal{R}_{\text{CPC}} = \gamma_1|\text{CPC}| + \gamma_2|\text{SWAP }| + \gamma_3|\text{LOCAL}|, \tag{7.3.1}$$

where $|\text{CPC}|$, $|\text{SWAP}|$ and $|\text{LOCAL}|$ are the counts for CPC gates, SWAP gates and local gates respectively. The count for each gate-type is weighted by a penalty strength γ which is based on the gate count.

In the code discovery stage of the CPC design process for the ion trap device, the aim was to find working $[[7, 3, 3]]$ codes that saturate the quantum Hamming bound for seven qubits. This involved calculating the code distance for all possible permutations of $[[7, 3, d =?]]$ CPC codes, a total of 2^{30} circuits. It was possible to exhaustively analyse all the circuits in less than a week on a desktop computer. In total, the search yielded $306,480$ working $[[7, 3, 3]]$ codes (0.03% of the search space).

For a CPC code with 4 data qubits, the quantum Hamming bound tells us that the optimal CPC code is of type $[[9, 4, 3]]$. However, there are 2^{50} permutations of this circuits of the form $[[9, 4, d =?]]$, which is an impractical search space for exhaustive methods. In the original CPC paper, it was shown that $[[9, 4, 3]]$ codes can be discovered simply by randomised search [1]. In future work, more sophisticated techniques, such as simulated annealing or parallel tempering, could be employed to more efficiently search for CPC codes.

When searching for large CPC codes, the number of circuits in the search space could be reduced by considering hardware constraints in advance. For example, for a nearest-neighbour device, each circuit permutation could be assigned a score on the basis of how many long-range interactions it contains. The code distance would then only be measured for the circuits with fewer long range interactions. Another optimisation parameter that could be considered is the weight of the code's stabilizers, a parameter that is useful to minimise when constructing fault tolerant circuits. Exhaustive and random search strategies for quantum code discovery have also been studied in [115, 116]. The particular strength of the CPC framework is that the symmetric *encode-error-decode* structure ensures the search is constrained to a space of non-disturbing codes. Investigating whether optimised CPC search strategies provide a higher density of good codes compared to other code search techniques would be an interesting area for future research.

Chapter 8

CPC codes as classical graphical models

In previous chapters I have shown that the CPC framework provides useful features for the design of quantum error correction codes. One of the key properties of CPC codes is that their structure guarantees that the quantum mechanical requirements of the code, such as stabilizer commutativity, are automatically satisfied. As a result, the CPC framework in effect reduces the process of quantum code design to a classical design problem. In this chapter, I expand upon this feature of the framework by introducing a mapping to allow CPC codes to be expressed in terms of factor graphs of the type commonly seen in classical information theory and machine learning. As an example, I demonstrate how this mapping can be used to derive a $[[5, 1, 3]]$ CPC code using classical code design tools. This can be done without reference to the underlying quantum mechanics. The ultimate aim of the factor graph representation is to make it easier to adapt well-developed techniques from classical information theory for quantum error correction. The work in this chapter is based upon work carried out in [3].

8.1 Overview of the mapping procedure

The factor graph representation for classical codes was introduced in section 2.3 in chapter 2. An advantage of this representation is that it gives access to a host of probabilistic graphical methods for decoding. For example, decoding strategies based on belief propagation for LDPC codes can achieve performance at close to the Shannon limit [38, 76].

While classical codes can be trivially expressed as factor graphs, the corresponding translation for quantum codes is somewhat more involved. The first complication stems from the fact that classical codes only need to account for one type of error, bit-flip errors, meaning there is only one edge type in a classical factor graph. This is problematic as qubits are susceptible to both bit- and phase-type errors. The second challenge in mapping CPC codes to factor graphs is that factor graph edges are unidirectional in that they only allow for the propagation of information from the data bits to parity bits. In contrast, quantum parity check operations are performed via unitary operations that act on the combined system of data plus parity qubits. A consequence of this unitarity is that the operations are bi-directional, meaning both qubits involved in the operation can change their state. The parity qubits themselves can therefore propagate errors to the register, leading to indirect propagation pathways which cannot be directly represented in a classical factor graph.

Our solution to the above problems involves first translating the CPC circuit to an intermediary graphical language which we call the *operational representation*. The operational representation serves as a quantum analogue of the classical factor graph, and includes multiple edge types to account for the different types of error propagation in a quantum code. In addition to this, we provide ways of annotating the operational representation to provide a visualisation of any indirect error propagation pathways. Once in the operational representation, the code can then be mapped directly to a factor graph via a set of graphical rules which we outline in the following sections. This factor graph corresponds to the classical code representation of the original CPC code. Classical methods can then be used to design or decode the circuit.

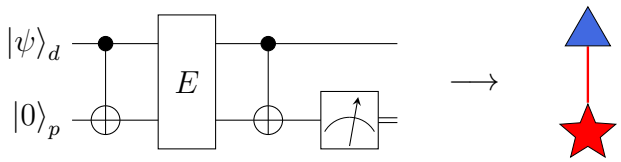
8.2 The operational representation for quantum codes

We now introduce an intermediary graphical representation we call the operational representation of CPC codes. This notation is designed to enable easy visualisation of the propagation of different error-types between qubits, and will serve as a stepping stone to our eventual presentation of CPC codes as classical factor graphs.

Graphs of the operational representation have two types of nodes:

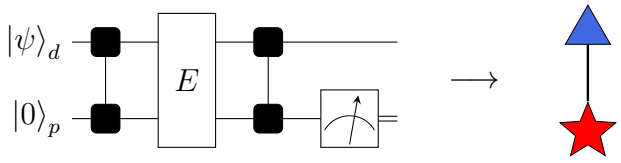
1. triangles (▲) representing data qubits and
2. stars (★) representing parity qubits.

The nodes are connected via edges that denote the parity check operations that are performed between qubits in a CPC code. In a CPC circuit there are three types of parity check operation (bit-checks, phase-checks and cross-checks), each of which needs its own edge-type in the operational representation. Bit-check operations, realised by CNOT gates, are represented by a red edge between a data node and a parity node. The mapping for such a bit-check edge, from circuit notation, is shown below



. (8.2.1)

Note that the red bit-check edge is undirected, as errors can propagate in both directions. Likewise, phase-check operations are represented by a undirected black edge between a data node and parity node. The mapping for a phase-check edge from CPC circuit notation to the operational representation is given by



. (8.2.2)

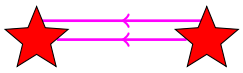
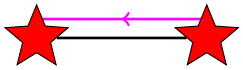
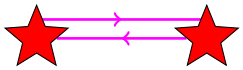
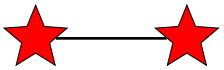
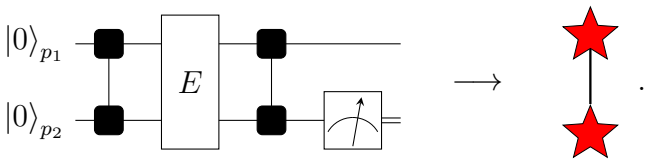
Rule	Before Rule	After Rule
Virtual Edge Cancellation		
Virtual Loop Cancellation		
Virtual Edge Reversal		
Virtual Edge Addition		

Table 8.1: Rules for simplifying annotated graphs in the operational representation.

Finally, cross-check edges are represented by an undirected black edge between two parity check nodes. For the cross-check edge, the mapping from the CPC circuit notation is written as follows

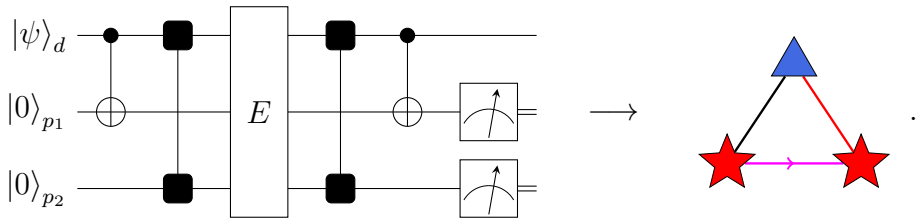


(8.2.3)

From equations 8.2.1-8.2.3 it can be seen that the nodes and edges of the operational representation give a graphical depiction of the parity checking sequences in the encoder/decoder of the corresponding CPC circuit. Each edge-type in the operational representation is mapped from an *encode-error-decode* CPC cycle in the circuit notation. We call this the operational representation, as the graph edges correspond to the physical operations performed between the qubits.

The canonical form for CPC codes, outlined in section 4.3, stipulates that the encoder is split into three blocks corresponding to the cross-check, bit-check and phase-checking stages respectively. Under this structure, phase-errors on the parity qubits can be detected via indirect propagation pathways. In the operational representation, these

indirect propagation pathways are represented by directed 'virtual edges'. As an example, consider the case shown below, where a data qubit is connected to one parity qubit via a bit-check edge and another via phase-check edge.



. (8.2.4)

A Z-error in the wait-stage on qubit p_2 will propagate to the register via the conjugate propagator gate, before being propagated to qubit p_2 by the CNOT gate. The Z-error on qubit p_2 is therefore detected by parity measurement p_1 via an indirect pathway. This is illustrated by the pink virtual edge between the two parity qubits. In general, the adjacency matrix m_v for virtual propagation in a CPC circuit can be calculated as follows

$$m_v = m_p^T \cdot m_b, \tag{8.2.5}$$

where m_b and m_p are the adjacency matrices for the bit- and phase-checking stages of the CPC code respectively. For example, in the circuit shown in equation 8.2.4, m_v is given by

$$m_b = \begin{pmatrix} 1 & 0 \end{pmatrix}, \quad m_p = \begin{pmatrix} 0 & 1 \end{pmatrix}, \quad m_v = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}. \tag{8.2.6}$$

The virtual edge therefore goes from qubit p_2 to qubit p_1 as shown in the operational representation in equation 8.2.4. If the CPC circuit contains cross-check operations, the adjacency matrix for the parity qubits in the operational representation is given by $m_v \oplus m_c$. If the adjacency matrix indicates there is propagation in both directions between two parity qubits, a cross-check edge is drawn between them. Conversely, if the propagation only occurs in one direction, a virtual edge is drawn. Simplifications can be applied directly to the operational representation itself. Table 8.1 lists the complete set of graphical simplifications for situations in which virtual edges and cross-check edges are combined.

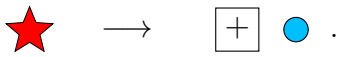
8.3 Translation rules mapping the operational representation to classical factor graphs

The graphical language of the operational representation, outlined in the previous section, allows quantum codes to be illustrated in terms of the physical operations connecting qubits. The operational representation can also be annotated to include virtual edges that highlight indirect propagation pathways for errors. We now show how the annotated operational representation can be mapped to an equivalent classical factor graph notation.

The data and parity nodes in the operational representation correspond to qubits that store both bit and phase error information. In a classical factor graph, a qubit can therefore be represented as two bits, one for each type of error as shown by the mapping below

(8.3.1)

As a convention, we choose to draw these bits side-by-side, with the node representing bit information on the left (coloured yellow) and the node representing phase information on the right (coloured blue). A parity qubit is also mapped to two bits in the classical factor graph notation via the following rule

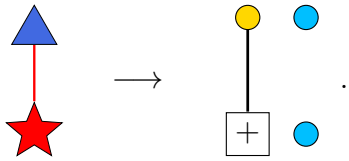


(8.3.2)

The bit information component of a qubit is used as a parity measurement, and is therefore drawn as a classical parity check node. The phase information of parity check qubit, however, cannot be directly measured (as all measurements in a CPC circuit are performed in the computational basis). As such, the phase-information component of the parity check qubit is mapped to an unmeasured classical data bit (shown in blue on the right).

We can now describe how the different edge types in the operational representation are drawn in a classical factor graph. Bit-check edges connect data qubits to parity qubits. Their action is to propagate bit information from the data qubit to the parity qubit

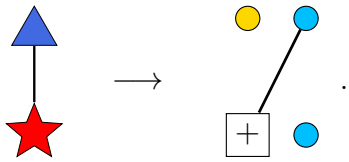
and phase information in the opposite direction. The mapping of a bit-check edge to classical factor graph notation is shown here:



(8.3.3)

An edge is drawn between the bit information component of the data qubit and the bit information component of the parity qubit. Notice, however, that there is no edge drawn between the phase-components of two qubits to indicate the propagation of phase-flip errors from the parity qubit to the data qubit. This is omitted, as there is no concept of indirect error propagation in a classical factor graph; the edges in a classical factor graph are only permitted between bit and factor nodes, and not between nodes of the same type. Instead, indirect propagation of errors is accounted for in classical factor graphs by placing edges in the place of the virtual edges in an annotated operational factor graph. An explicit example of this is shown later in this section.

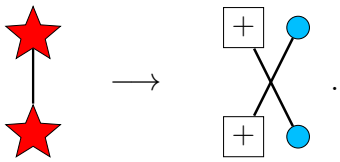
The classical factor graph representation of a phase-check edge reads:



(8.3.4)

The phase component of the data qubit is connected to the bit-component of the parity qubit via an edge. This shows that the phase-check edge propagates phase-errors on data qubits to bit-errors on the parity qubits. Recall that phase-edges are symmetric, and that error propagation also occurs in the reverse direction. The back-propagation of errors in this way is not shown in the classical factor graph. The reason for this is again that edges are only permitted between bit and factor nodes in a classical factor graph.

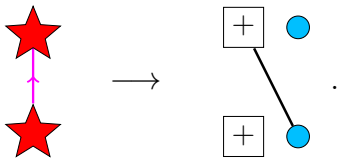
The classical factor graph representation of a cross-check edge reads:



(8.3.5)

The phase-component of each qubit is connected to the bit-component of the other. This reflects the expected error propagation behaviour for cross-check edges.

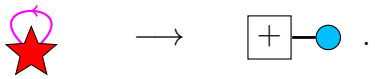
The next component to be mapped to classical factor graphs are the virtual edges that depict the indirect propagation of errors through the code. Virtual edges show how a phase-error on one parity qubit can be detected as a bit-flip error on another. The classical factor graph mapping of a virtual edge is given by



(8.3.6)

A virtual edge is directed meaning error information only propagates in one direction.

The final translation rule is for the virtual self loop edge. In the classical factor graph notation this edge is represented as follows:



(8.3.7)

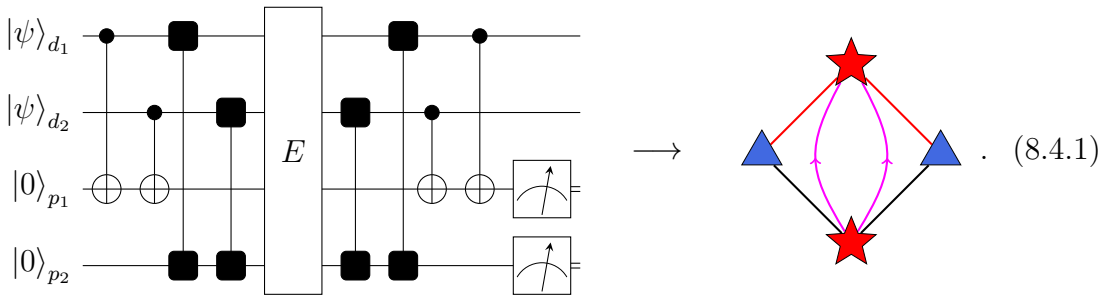
8.4 Example: Designing a [[4,2,2]] CPC code using the factor graph mapping

We have now outlined how to translate graphs of the operational representation to classical factor graphs. In this section, we describe how such mappings can be useful in the construction of a $[[4, 2, 2]]$ CPC detection code.

In section 4.2 we showed how a preliminary $[[4, 2, 1]]$ CPC code can be constructed

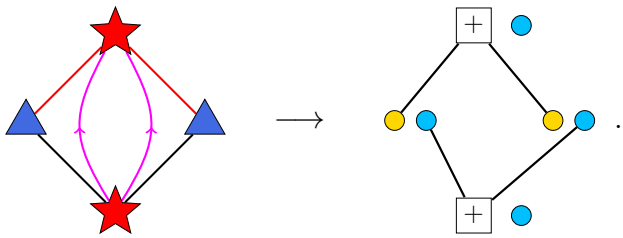
by combining bit-flip and phase-flip $[3, 2, 2]$ codes. One of the principal challenges in designing CPC codes is finding an appropriate set of cross-checks to achieve the desired code distance. For the case of a detection code, we require that $d = 2$. Whilst for small codes, such as the $[[4, 2, 2]]$ code, it is possible to find the cross-check operations by inspection, for larger codes this can become a difficult problem. We now show how the factor graph mapping allows classical code design techniques to be employed in the discovery of the cross-checks.

The $[[4, 2, 1]]$ CPC code, formed by combining two copies of a classical $[3, 2, 2]$ code, maps to the operational representation as follows



. (8.4.1)

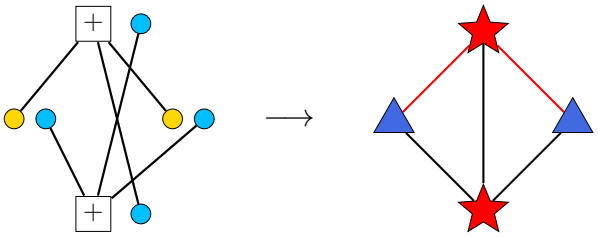
In the above, two virtual edges have been added between the parity qubits in accordance with the rule defined in equation 8.2.4. However, as these edges act in the same direction, they cancel each other out (see the simplification rule in table 8.1). Once the preliminary $[[4, 2, 1]]$ CPC code has been mapped to the operational representation, it can be further mapped to a factor graph following the rules outlined in section 8.3, as shown below.



. (8.4.2)

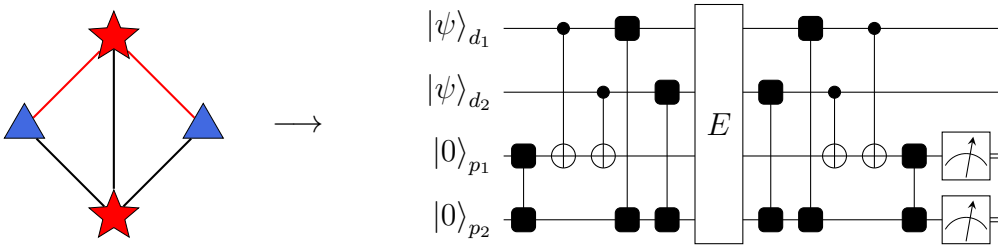
By inspection of the factor graph above, it is immediately clear that there are two bits that go unchecked, meaning the code has distance $d = 1$. This problem can be resolved by adding pairs of edges to the classical code until it has distance $d = 2$. An example

of how this can be achieved is shown to the left of the equation below



. (8.4.3)

Once the classical code has the desired properties, it can be mapped back to the operational representation as shown to the right of the equation above. For this specific example, the two edges that were added to the classical factor graph map to a cross-check between the two parity qubits in the operational representation. The circuit this operational representation corresponds to is given by



. (8.4.4)

The above circuit is a $[[4, 2, 2]]$ CPC code identical in form to the CPC code derived in section 4.2.

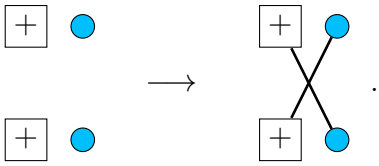
8.5 General rules for constructing tripartite CPC codes using classical factor graphs

In the previous section we outlined how the factor graph mapping can be used to diagnose and resolve the problems with the preliminary $[[4, 2, 1]]$ code to obtain a circuit with distance $d = 2$. Whilst this was a simple example, the factor graph formalism provides a highly general tool for the design of quantum error correction codes. We now outline a specific code design strategy for tripartite CPC codes that can be employed using classical factor graphs, without having to refer back to the operational form after

the initial mapping. Our approach enables quantum codes to be constructed using classical techniques and does not require detailed knowledge of quantum mechanics. The steps of this code design procedure can be summarised as follows:

1. Construct a preliminary code in the annotated operational representation by combining two classical codes using the tripartite structure described in section 4.5. Convert the resultant graph to a classical factor graph using the mappings described in section 8.3.
2. Calculate the distance of the preliminary code in its factor graph form.
3. Determine the form of the cross-checks that need to be added to fix the code distance to the desired length.
4. Map back to the operational representation to get the corresponding CPC circuit.

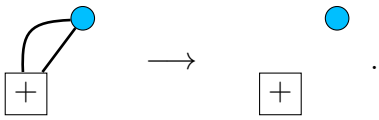
Following the initial mapping from the operational representation, the optimisation steps of the code design process (steps 2 and 3) are carried out entirely within the classical factor graph framework. This is advantageous as it provides a simple setting for code optimisation; only one error-type needs to be considered, and existing classical code design techniques can be employed. The reason that this method can be followed without reference to the operational form is that the addition of cross-checks does not lead to any indirect propagation of errors. In a classical factor graph, cross checks are added between parity qubits according to the following rule,



(8.5.1)

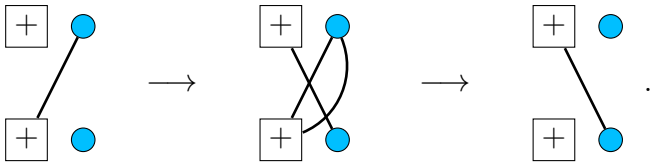
where a pair of edges link the phase component of one qubit to the bit component of the other. A situation which may be encountered when using this rule, occurs when a cross-check is applied between qubits that are already connected by one or more edges. For this case, we need to define a simplification rule for a double edge. Since

two successive bit-flip operations will cancel each other out, the following rule applies



(8.5.2)

As an example, consider the case, depicted below, in which a cross-check is added between a pair of qubits that are already connected by a virtual edge,

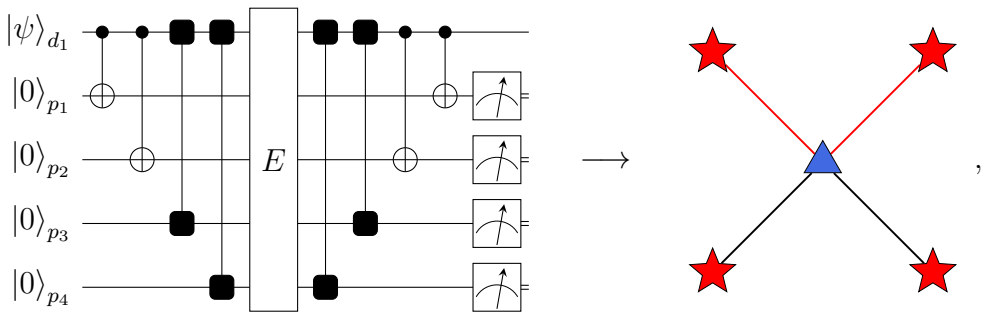


(8.5.3)

The double edge that is formed cancels to give the factor graph on the right. From this, we can deduce the rule that when a cross-check is added between a pair of qubits already connected by a virtual edge, the direction of the virtual edge is reversed.

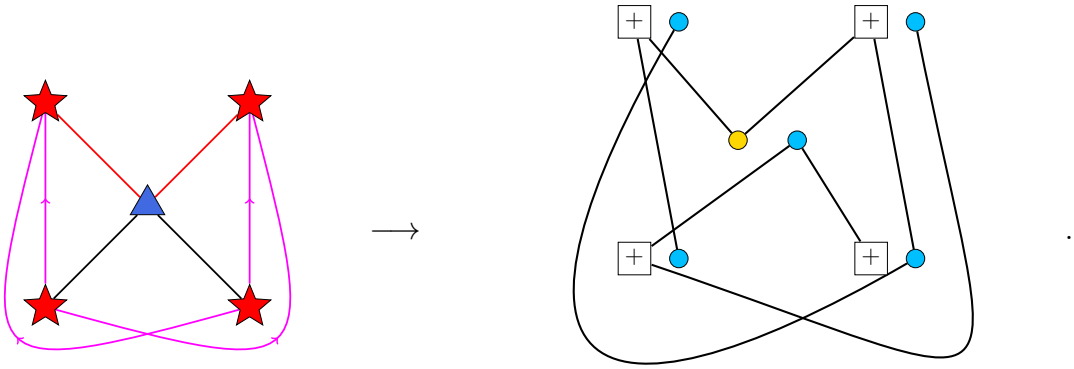
8.6 Example: Designing a $[[5, 1, 3]]$ CPC code using the factor graph mapping

As an example of the CPC factor graph design process, we now walk-through the construction of $[[5, 1, 3]]$ tripartite code. We start by combining two copies of a classical $[3, 1, 3]$ code into a tripartite CPC structure to create a $[[5, 1, 2]]$ preliminary code. The circuit for this code, along with the corresponding mapping to the operational representation, is shown below



, (8.6.1)

Note that the two parity qubits to the top of the diagram correspond to the bit-check block of the tripartite code, whilst the bottom two parity qubits are the phase-check block. The next step in the design process is to annotate the operational representation to depict any indirect propagation pathways. Once in this form, we can map to the factor graph representation as follows

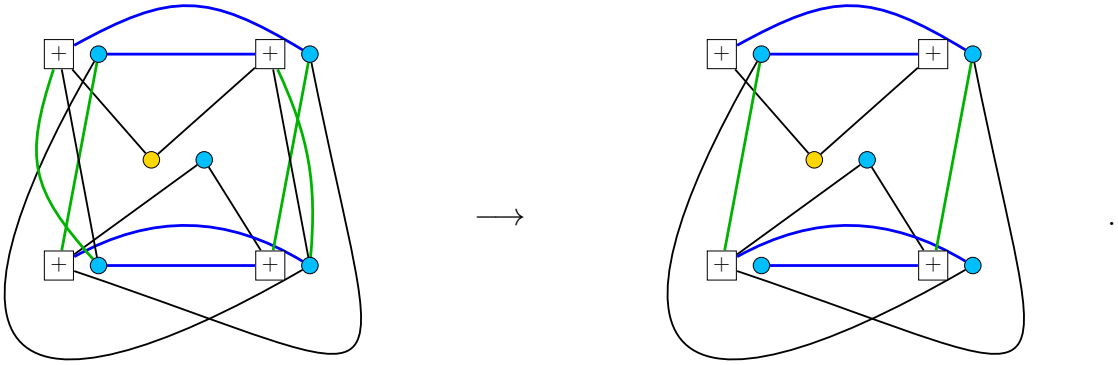


(8.6.2)

In the factor graph representation, we see that each bit is connected to at least one parity node so that the code has distance $d = 2$. However, our target distance is $d = 3$, meaning the code needs to be modified so that an error on any of its bits produces a unique syndrome. Our mapping translates each qubit in the operational representation to two data bits in the factor graph. Single-bit errors in the classical code therefore refer to either a X- or a Z-error in the CPC code. However, in quantum codes, it is usually also necessary to consider Y-errors. In the classical factor graph, such Y-errors can modelled as classical 'burst' errors affecting both the phase and bit components of a given qubit. Therefore, in order to verify that the classical code has $d = 3$, we need to check syndromes for all of the possible 'burst' errors in addition to the single-qubit errors.

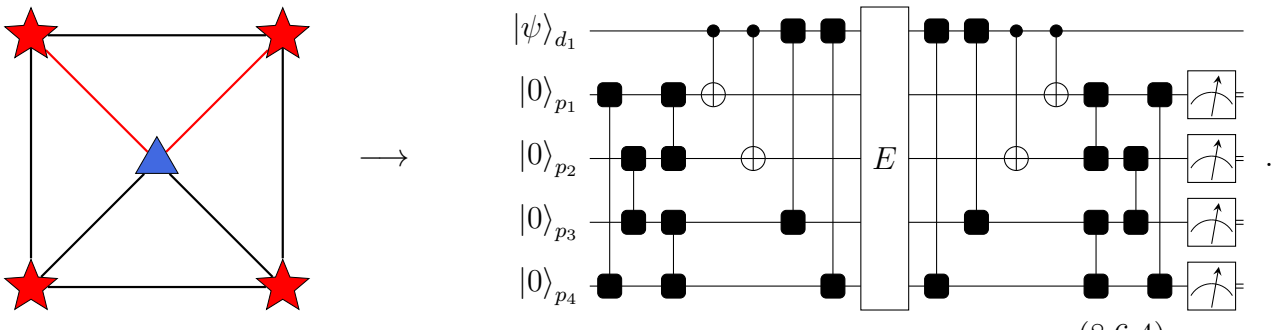
The next stage of code design involves modifying the classical code by adding cross-checks via the rule described in equation 8.5.1. For this particular example, we first apply cross-checks between the parity qubits in each block. These cross-checks are

marked in blue in the factor graph below



(8.6.3)

The second set of cross-checks are added between the two blocks, and are highlighted in green in the factor graph to the left of the above equation. Once the these cross-checks have been added, it can be confirmed that all single-bit and (weight-two) burst errors on the factor graph produce a unique syndrome. The code therefore has distance $d = 3$. The factor graph to the right of the above equation is the simplified form obtained by applying the rule defined in equation 8.5.3. Mapping back to the operational representation, we get the graph and CPC circuit shown below



(8.6.4)

The above circuit is a $[[5, 1, 3]]$ code capable of uniquely identifying single-qubit X-, Y- and Z-errors on any of the five qubits. Codes of this type are often referred to as perfect codes as they are the smallest possible distance three codes allowed by the quantum Hamming bound [117].

Chapter 9

Conclusions and outlook

In this thesis, I have introduced the Coherent Parity Check (CPC) framework as an alternative perspective on the design and analysis of stabilizer codes. Central to the CPC framework is a symmetric *encode-error-decode* structure that allows any sequence of parity checks to be turned into a stabilizer code. We have seen that this freedom in the choice of parity checks affords the CPC construction various advantages over existing approaches to code design. In this section, I summarise our main results and propose avenues for future work.

9.1 Summary

As a first use-case, we explored the application of CPC techniques for the mapping of classical codes to quantum codes. The CPC code structure allows any pair of classical codes to be interpreted as the bit- and phase-checking stages of a quantum code. The resultant circuit is guaranteed to be a stabilizer code, but will have a reduced code distance (relative to the original classical code) owing to indirect propagation of errors through the circuit. We showed that this problem can be resolved through the addition of cross-check operations between the parity qubits themselves. Following this, we introduced the tripartite structure for CPC codes, which provides a structural template to facilitate the mapping of classical codes to quantum codes. In particular, we

outlined a method that enables any $[n, k, 3]$ classical Hamming code to be re-purposed as an $[[2n - k, k, 3]]$ tripartite CPC code.

We showed that an existing family of quantum codes, the CSS codes, can be represented as CPC codes. Furthermore, we introduced a CPC method that enables almost *any* pair of classical $[n, k, 3]$ codes to be turned into a $[[2n - k + 2, k, 3]]$ CSS code. In contrast, the traditional method for the construction of CSS codes requires a pair of initial codes, C_1 and C_2, that satisfy the condition $C_2 \subset C_1$. As an example, we showed how an $[[11, 3, 3]]$ CSS code can be constructed from the starting point of two copies of the classical $[6, 1, 3]$ ring code. For comparison, the same code can be constructed via the conventional CSS method, but requires combining an $[11, 7, 3]$ code with an $[11, 4, 3]$ code. The CPC method is therefore more constructive in that the resultant quantum code closely resembles the operation of the classical code from which it was derived.

The CPC framework allows stabilizer code search problems to be formulated in a way that is particularly amenable to machine search. This is due to the CPC code structure that enables any sequence of parity checks to be turned into a stabilizer code; new codes can therefore be discovered simply by generating sequences of parity checks. The only task that then remains in order to assess the usefulness of the code is to measure the code distance. As a proof-of-concept example, we used exhaustive search to discover CPC codes for an idealised seven-qubit ion trap. A large set of circuits was discovered, from which we were able to select the one that best suited the needs of the device.

Quantum codes are usually represented in terms of idealised gates such as CNOT and conjugate-propagator gates. However, qubit technologies will often have native gates that are of a different form. We demonstrated that a strength of CPC codes is that they can be understood and analysed in terms of any realistic maximally entangling Clifford gate. This allows quantum codes to be designed directly with a device's native gate, bypassing the need for onerous – and often resource intensive – translation between the theoretical version of a quantum circuit and the hardware compiled version. This result, combined with the ability to discover new CPC codes via machine search, means the CPC framework is a powerful tool for the design of bespoke quantum error correction protocols tailored to the needs of a given device.

Small-scale quantum computers are currently in development. However, the qubit error rates of these devices are typically too low to realise large-scale error correction codes. Quantum detection codes offer the simplest protocols for implementation on early hardware. The smallest quantum detection protocol is the $[[4, 2, 2]]$ code, for which there is a convenient representation in the CPC framework. We demonstrated that such a $[[4, 2, 2]]$ CPC code can be specially compiled to be run on the IBM 5Q device. The code was specially modified so that it would be tolerant to some errors in the encode/decode stages of the CPC code. The output of the code was analysed using quantum state tomography. Our results showed that the code's syndrome measurements could be used to improve the fidelity of the circuit output, relative to the case in which the syndrome measurements were ignored.

The CPC framework provides a fail-safe template that guarantees a stabilizer code from the starting point of any sequence of parity checks. As such, we can think of the CPC framework as a tool that reduces the problem of discovering new codes to a classical design problem. Taking advantage of this feature, we demonstrated that CPC codes can be mapped to factor graphs of the type commonly seen in classical information theory and machine learning. Our mapping first involves translating the CPC circuits to a special-purpose intermediary graphical language called the operational representation. The role of this operational representation is to abstract away the quantum mechanical aspects of CPC codes, such as indirect propagation. Following this, the CPC code is in a form that can be directly mapped to classical code with a corresponding factor graph representation. Classical methods and intuition can then be used to complete the design of the code, without any reference to the underlying quantum mechanics. As an example, we showed how the factor graph representation can be used to find appropriate cross-check operations in the design of a $[[5, 1, 3]]$ CPC code. More generally, the expectation is that the factor graph mapping for CPC codes will make it easier to adapt techniques from classical information processing to the design of quantum error correction codes.

In summary, the CPC framework gives a new way to approach the design and analysis of quantum codes. CPC methods have been demonstrated for systematic code construction, as well as code discovery via automated search routines. Furthermore, the CPC factor graph mapping admits a representation of quantum codes that can

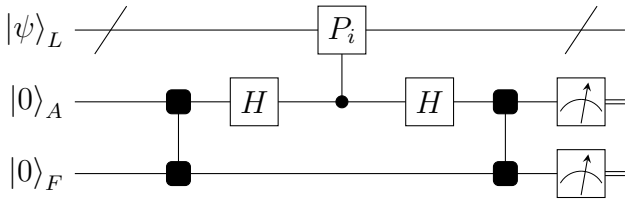


Figure 9.1: The flag method for syndrome extraction. A stabilizer operator P_i is applied to the register $|\psi\rangle_L$, and the result copied to an ancilla qubit A. Phase-flip errors on the ancilla are detected via two conjugate propagator gates that connect to a flag qubit F. The flag-check scheme is similar to the symmetric CPC gadgets we have covered in this thesis.

be understood without detailed understanding of quantum mechanics. This will provide access to large and established fields of expertise from classical information theory, which will prove particularly useful as quantum computers move closer to being realised in full.

9.2 Outlook

9.2.1 CPC methods for fault tolerant syndrome extraction

As discussed in section 3.7, when measuring the stabilizers of a code, fault tolerant syndrome extraction methods are required to ensure that errors do not spread to the code qubits in an uncontrolled way. However, such fault tolerant techniques can add considerable overhead in terms of the number of ancilla qubits required. For example, the Shor method for syndrome extraction [44], requires λ ancillas to measure a stabilizer P_i, where λ is defined as the weight of the stabilizer $\lambda = W(P_i)$. Recently, the so-called flag method for syndrome extraction was proposed by Chao and Reichardt [88, 118], and further developed by Chamberland and Beverland [119]. The flag scheme is a novel technique that allows a code stabilizer to be measured with only two ancilla qubits, regardless of the stabilizer's weight. The general construction for the flag syndrome extraction method is shown in figure 9.1. From this figure, the stabilizer is measured

and copied to ancilla qubit A, which is read out to give the syndrome. An additional flag qubit F is also included, so that any Z-errors on the ancilla qubit A can be detected. This flag-check is performed via two conjugate-propagator gates either side of the syndrome extraction stage of the circuit. It is clear from figure 9.1 that the flag-check is similar in form to the fundamental CPC gadget. As such, it would be interesting to investigate whether the CPC methods described in this thesis could be useful in the design of flag syndrome extraction circuits.

Recently, in [120], an alternative method for flag syndrome extraction was proposed in which all code stabilizers are measured in parallel. The idea is that flag-checks can be performed between the ancilla qubits themselves, instead of having to introduce a dedicated flag qubit for each stabilizer. In [120], examples of how this might be achieved were included for the $[[4, 2, 2]]$ detection code and the Steane $[[7, 1, 3]]$ code. These circuits were compiled manually, and no general construction was provided for larger codes. It is possible that our CPC auto-design methods could be used in the construction of such flag checking routines.

9.2.2 Maximum entropy decoding of quantum codes

As outlined in chapter 8, CPC codes can be mapped to factor graphs of the type commonly seen in classical error correction and machine learning. The resultant factor graph is a classical representation of the initial CPC code. The aim of the factor graph mapping is to provide an easy way of adapting classical techniques for quantum error correction. A potential use-case is to explore whether classical decoding methods can be applied to quantum codes. In a paper currently in preparation [121], we are investigating how a maximum entropy decoding technique for classical codes can be applied to quantum codes via a factor graph mapping.

It has been shown that classical error correction codes can be represented as Ising spin models [122]. This allows the error correction code to be described by an Ising Hamiltonian of the form $H_{\mathrm{EC}}(S)$. The ground-state of $H_{\mathrm{EC}}(S)$ then corresponds to the lowest weight error consistent with the syndrome S. Finding the ground-state of $H_{\mathrm{EC}}(S)$ is therefore equivalent to performing a maximum-likelihood decode of the error

correction code (for an explanation of maximum-likelihood decoding see section 2.5).

An alternative decoding strategy is the maximum entropy decoding method. In maximum entropy decoding, a Boltzmann distribution is first constructed from the Ising Hamiltonian. It has been shown that the temperature of this probability distribution can be precisely related to the error rate of the code bits [123]. The motivation for adopting a maximum entropy inference technique, is the idea that this prior information about error rates can be used to improve decoding performance relative to the maximum likelihood method. As a result, maximum entropy decoding is sometimes described as a method for decoding at finite temperature (and therefore also finite error rate) [124].

The error rates on a quantum computer can be measured using techniques such as randomised benchmarking [125]. Consequently, it is interesting to investigate whether a maximum entropy decoding strategy could improve the performance of a quantum code. The factor graph mapping for CPC codes, outlined in chapter 8, allows quantum codes to be represented as a classical factor graphs. We have shown that a maximum entropy inference strategy – after some modification – can be applied to these factor graphs to provide a way of decoding a quantum code. In [121], we will present benchmarks to compare the two decoding techniques for small quantum codes.

In [90], it was demonstrated that maximum entropy decoding strategy could be run on a programmable quantum annealing device. The results showed an improvement in decoding performance over a maximum-likelihood method. Ultimately our aim is to perform a similar experiment for decoding quantum codes. This would likely be the first instance of a quantum computing device being used to decode a quantum computer.

9.2.3 CPC codes with increased code distance

In this thesis, we have only considered CPC codes of distance $d = 3$. However, efficient codes with larger block sizes will require higher distances. This adds an extra layer of complexity to CPC design, as calculating the code distance is exponentially expensive

in d. One potential solution to this problem is to find an alternative metric for assessing the quality of a code. For example, in classical error correction, effective logical error rates are calculated for large codes using Monte Carlo methods [126]. Such techniques are sufficient to give a real-world indication of a code's performance.

Another challenge to be overcome with the design of CPC codes is the problem of finding cross-check operators. For the distance-three codes considered in this thesis, it was possible to find appropriate checks via direct inspection. However, as the distance of a code is increased, such an approach becomes more difficult as the number of error permutations to be considered scales exponentially. Recently, quantum low density parity check (LDPC) codes have become an active area of research [61]. These quantum LDPC codes are designed so that the number of encoded qubits k, and the code distance d, scales linearly with block size n. Various constructions have been proposed that come close to satisfying these requirements [62, 63, 127, 128], and it would be interesting to investigate the use of CPC methods in their further development.

Appendix

A The Pauli group

The Pauli group on a single-qubit, $\mathcal{G}_1$, is defined as the set of Pauli operators

$$\mathcal{G}_1 = \{\pm\mathbb{1}, \pm\mathrm{i}\mathbb{1}, \pm X, \pm\mathrm{i}X, \pm Y, \pm\mathrm{i}Y, \pm Z, \pm\mathrm{i}Z\}, \tag{A1}$$

where the ± 1 and $\pm\mathrm{i}$ terms are included to ensure $\mathcal{G}_1$ is closed under multiplication and thus forms a legitimate group [6]. In matrix form, the four Pauli operators are given by

$$\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{A2}$$

The general Pauli group, $\mathcal{G}$, consists of the set of all operators that are formed from tensor products of the matrices in $\mathcal{G}_1$. For example, the operator

$$\mathbb{1} \otimes X \otimes \mathbb{1} \otimes Y \in \mathcal{G} \tag{A3}$$

is an element of the four-qubit Pauli group. Note that for simplicity, in the context of quantum computing, the above operator would usually be expressed as X_2Y_4. The identity operators are omitted, and the remaining elements are subscripted with the label of the qubit they act on.

The elements of the Pauli group have eigenvalues $\{\pm 1, \pm\mathrm{i}\}$. Another useful property of

the Pauli group is that its elements either commute or anti-commute with one another.

B The Clifford group and stabilizer states

The Clifford group $\mathcal{C}$ is defined as the set of operators that normalise the Pauli group such that

$$U_{\mathrm{C}} \bullet P_i \bullet U_{\mathrm{C}}^{\dagger} = P_j, \quad U_{\mathrm{C}} \in \mathcal{C}, \quad \{P_i, P_j\} \in \mathcal{G} \; \forall \; \{i, j\}, \tag{B1}$$

where $U_{\mathrm{C}} \in \mathcal{C}$ is a Clifford operator and P_k are elements of the Pauli group. Clifford gates, $\mathcal{C}$, are generated by the set of three gates $\langle \mathrm{CNOT}, H, P \rangle$, such that $\mathcal{C} = \langle \mathrm{CNOT}, H, P \rangle$ [46]. Likewise, single-qubit Clifford gates, $\mathcal{C}_1$, are generated by the set $\langle H, P \rangle$, such that $\mathcal{C}_1 = \langle H, P \rangle$.

The stabilizer states are all the quantum states that can be reached from a blank register, $|0\rangle^{\otimes N}$, via the application of Clifford gates and computational basis measurements. Quantum circuits consisting only of Clifford gates acting on stabilizer states can be efficiently classically simulated. The proof of this is given by the Gottesman-Knill theorem [48]. Although the Clifford group is not a universal quantum gate set, it is sufficient for simulating many QEC circuits and all the quantum memories described in this paper.

C IMBQX4 calibration data

The experiment on the IBMQX4 outlined in chapter 5 was run over three days on 25$^{\mathrm{th}}$ November 2017, 26$^{\mathrm{th}}$ November and 27$^{\mathrm{th}}$ November 2017. The calibration data for each of these days can be found below:

```
{Date: 11-25-2017,
Single-qubit error rates (10^-3):
{Q0: 0.94, Q1: 0.60, Q2: 1.12, Q3: 1.37, Q4: 1.80},
Readout error rates (10^-2):
```

```
{Q0: 4.10, Q1: 5.70, Q2: 4.00, Q3: 3.30, Q4: 5.10},
Two-qubit error rates (10^-2):
{CX1_0: 1.88, CX2_0: 2.09, CX3_2: 1.97, CX2_1: 4.28, CX3_4: 2.15, CX2_4: 3.89}}

{Date: 11-26-2017,
 Single-qubit error rates (10^-3):
 {Q0: 0.86, Q1: 0.69, Q2: 1.12, Q3: 1.89, Q4: 2.06},
 Readout error rates (10^-2):
 {Q0: 4.10, Q1: 4.10, Q2: 4.30, Q3: 5.30, Q4: 7.10},
 Two-qubit error rates (10^-2):
 {CX1_0: 2.31, CX2_0: 2.22, CX3_2: 2.18, CX2_1: 4.80, CX3_4: 2.43, CX2_4: 3.93}}

{Date: 11-27-2017,
Single-qubit error rates (10^-3):
{Q0: 0.77, Q1: 0.43, Q2: 1.20, Q3: 1.72, Q4: 1.89},
Readout error rates (10^-2):
{Q0: 3.70, Q1: 4.90, Q2: 4.20, Q3: 5.30, Q4: 5.80},
Two-qubit error rates (10^-2):
{CX1_0: 2.01, CX2_0: 1.93, CX3_2: 2.75, CX2_1: 4.47, CX3_4: 2.28, CX2_4: 4.13}}
```

Additional Acknowledgements

I was supported by a Durham Doctoral Studentship (Faculty of Science) during my PhD studies. We acknowledge use of the IBM Quantum Experience for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Quantum Experience team. The quantum circuits in this paper were drawn using the QPIC package by Thomas Draper and Samuel Kutin [129].

Bibliography

[1] Nicholas Chancellor, Aleks Kissinger, Joschka Roffe, Stefan Zohren, and Dominic Horsman. Graphical structures for design and verification of quantum error correction. *arXiv:1611.08012*, 2016.

[2] Joschka Roffe, David Headley, Nicholas Chancellor, Dominic Horsman, and Viv Kendon. Protecting quantum memories using coherent parity check codes. *Quantum Science and Technology*, 3(3):035010, 2018.

[3] Joschka Roffe, Stefan Zohren, Dominic Horsman, and Nicholas Chancellor. Quantum codes from classical graphical models. *arXiv:1804.07653*, 2018.

[4] Joschka Roffe, Stefan Zohren, Dominic Horsman, and Nicholas Chancellor. Decoding quantum error correction with ising model hardware. *arXiv:1903.10254*, 2019.

[5] David McMahon. *Quantum Computing Explained*. Wiley-IEEE Computer Society Pr, 2007.

[6] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge, United Kingdom, 2010.

[7] Mike Daily, Swarup Medasani, Reinhold Behringer, and Mohan Trivedi. Self-driving cars. *Computer*, 50(12):18–23, 2017.

[8] Georges Ifrah. *The Universal History of Computing: From the Abacus to the Quantum Computer*. Wiley, 2000.

[9] Cliff Stoll. When slide rules ruled. *Scientific American*, 294(5):80–87, 2006.

[10] Chris Woodford. A brief history of computers. *www.explainthatstuff.com/historyofcomputers.html*, Accessed: 18/10/2018.

[11] Doron Swade and Charles Babbage. *Difference Engine: Charles Babbage and the Quest to Build the First Computer*. Viking Penguin, 2001.

[12] Benjamin Woolley. *The Bride of Science: Romance, Reason and Byron's Daughter*. Macmillan Pub Ltd, 2000.

[13] L. F. Menabrea and Ada Lovelace (English translator). Sketch of the Analytical Engine invented by Charles Babbage (with additional annotations by Ada Lovelace). *Scientific Memoirs, 3, 666-731, 1843. Originally publised in French in the Bibliotheque Universelle de Geneve, 1842.*

[14] Change.org. Change.org campaign: Name the next Durham college after notable Durham women. *www.change.org/p/durham-university-name-the-next-durham-colleges-after-notable-durham-women.*

[15] The Computer History Museum. The Babbage engines. *www.computerhistory.org/babbage/engines/*, Accessed: 18/10/2018.

[16] Alonzo Church and A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *The Journal of Symbolic Logic*, 2(1):42, 1937.

[17] Igor L. Markov. Limits on fundamental limits to computation. *Nature*, 512(7513):147–154, 2014.

[18] M.D. Godfrey and D.F. Hendry. The computer as Von Neumann planned it. *IEEE Annals of the History of Computing*, 15(1):11–21, 1993.

[19] C.P. Burton. Replicating the manchester baby: Motives, methods, and messages from the past. *IEEE Annals of the History of Computing*, 27(3):44–60, 2005.

[20] John Bardeen. Semiconductor research leading to the point contact transistor. *Nobel Lecture. NobelPrize.org*, 1956.

[21] David Anderson. Tom kilburn: A tale of five computers. *Commun. ACM*, 57(5):35–38, 2014.

[22] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, 2006.

[23] S. B. Desai, S. R. Madhvapathy, A. B. Sachid, J. P. Llinas, Q. Wang, G. H. Ahn, G. Pitner, M. J. Kim, J. Bokor, C. Hu, H.-S. P. Wong, and A. Javey. MoS2 transistors with 1-nanometer gate lengths. *Science*, 354(6308):99–102, 2016.

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

[25] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[26] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[27] C. J. Ballance, T. P. Harty, N. M. Linke, M. A. Sepiol, and D. M. Lucas. High-fidelity quantum logic gates using trapped-ion hyperfine qubits. *Physical Review Letters*, 117(6), 2016.

[28] Norbert M. Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A. Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305, 2017.

[29] M. F. Brandl, M. W. van Mourik, L. Postler, A. Nolf, K. Lakhmanskiy, R. R. Paiva, S. Möller, N. Daniilidis, H. Häffner, V. Kaushal, T. Ruster, C. Warschburger, H. Kaufmann, U. G. Poschinger, F. Schmidt-Kaler, P. Schindler, T. Monz, and R. Blatt. Cryogenic setup for trapped ion quantum computing. *Review of Scientific Instruments*, 87(11):113103, 2016.

[30] Yu Chen, C. Neill, P. Roushan, N. Leung, M. Fang, R. Barends, J. Kelly, B. Campbell, Z. Chen, B. Chiaro, A. Dunsworth, E. Jeffrey, A. Megrant, J. Y. Mutus, P. J. J. O'Malley, C. M. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, Michael R. Geller, A. N. Cleland, and John M. Martinis. Qubit architecture with high coherence and fast tunable coupling. *Phys. Rev. Lett.*, 113:220502, 2014.

[31] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and John M. Martinis. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

[32] Jerry M. Chow, Jay M. Gambetta, Easwar Magesan, David W. Abraham, Andrew W. Cross, B R Johnson, Nicholas A. Masluk, Colm A. Ryan, John A. Smolin, Srikanth J. Srinivasan, and M Steffen. Implementing a strand of a scalable fault-tolerant quantum computing fabric. *Nature Communications*, 5(1), 2014.

[33] Jarryd J. Pla, Kuan Y. Tan, Juan P. Dehollain, Wee H. Lim, John J. L. Morton, David N. Jamieson, Andrew S. Dzurak, and Andrea Morello. A single-atom electron spin qubit in silicon. *Nature*, 489(7417):541–545, 2012.

[34] M. Veldhorst, C. H. Yang, J. C. C. Hwang, W. Huang, J. P. Dehollain, J. T. Muhonen, S. Simmons, A. Laucht, F. E. Hudson, K. M. Itoh, A. Morello, and A. S. Dzurak. A two-qubit logic gate in silicon. *Nature*, 526(7573):410–414, 2015.

[35] Yu Wang, Chin-Yi Chen, Gerhard Klimeck, Michelle Y. Simmons, and Rajib Rahman. Characterizing si:p quantum dot qubits with spin resonance techniques. *Scientific Reports*, 6(1), 2016.

[36] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien. Quantum computers. *Nature*, 464(7285):45–53, 2010.

[37] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[38] D. J. C. MacKay and R. M. Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 32(18):1645, 1996.

[39] C. Berrou, A. Glavieux, and P Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. In *Proc. 1993 IEEE International Conf. on Communications, Geneva, Switzerland*, page 1064, 1993.

[40] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[41] P. W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52:R2493, 1995.

[42] A. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77:793–797, 1996.

[43] A. R. Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54:1098–1106, 1996.

[44] Peter Shor. Fault-tolerant quantum computation. *IEEE Comput. Soc. Press*, Proceedings of 37th Conference on Foundations of Computer Science.

[45] A. M. Steane. Active stabilization, quantum computation, and quantum state synthesis. *Physical Review Letters*, 78(11):2252, 1997.

[46] Daniel Gottesman. Theory of fault-tolerant quantum computation. *Physical Review A*, 57(1):127, 1998.

[47] Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv:quant-ph/9705052*, 1997.

[48] Daniel Gottesman. The Heisenberg representation of quantum computers. In *Group theoretical methods in physics. Proceedings, 22nd International Colloquium, Group22, ICGTMP'98, Hobart, Australia, July 13-17, 1998*, pages 32–43, 1998.

[49] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), 2004.

[50] Simon Anders and Hans Briegel. Fast simulation of stabilizer circuits using a graph-state representation. *Physical Review A*, 73(2), 2006.

[51] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. Simulation of quantum circuits by low-rank stabilizer decompositions. *arXiv:1808.00128*, 2018.

[52] John Preskill. Reliable quantum computers. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 454(1969):385–410, 1998.

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

[54] D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 176–188, New York, NY, USA, 1997. ACM.

[55] E. Knill. Resilient quantum computation. *Science*, 279(5349):342–345, 1998.

[56] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[57] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3), 2012.

[58] A. Y. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303, 2003.

[59] David S. Wang, Austin G. Fowler, and Lloyd C. L. Hollenberg. Surface code quantum computing with error rates over 1%. *Physical Review A*, 83(2), 2011.

[60] C. Horsman, A. Fowler, S. Devitt, and R. Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[61] Daniel Gottesman. Fault-tolerant quantum computation with constant overhead. *Quantum Info. Comput.*, 14(15-16):1338–1372, 2014.

[62] Nikolas P. Breuckmann and Barbara M. Terhal. Constructions and noise threshold of hyperbolic surface codes. *IEEE Transactions on Information Theory*, 62(6):3731, 2016.

[63] Anthony Leverrier, Jean-Pierre Tillich, and Gilles Zemor. Quantum expander codes. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE, 2015.

[64] Bryan Eastin and Emanuel Knill. Restrictions on transversal encoded quantum gate sets. *Physical Review Letters*, 102(11), 2009.

[65] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71(2), 2005.

[66] H. Bombin and M. A. Martin-Delgado. Statistical mechanical models and topological color codes. *Physical Review A*, 77(4), 2008.

[67] Theodore J. Yoder. Universal fault-tolerant quantum computation with bacon-shor codes. *arXiv:1705.01686*, 2017.

[68] Michael Vasmer and Dan E. Browne. Universal quantum computing with 3d surface codes. *arXiv:1801.04255*, 2018.

[69] Earl T. Campbell, Barbara M. Terhal, and Christophe Vuillot. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172–179, 2017.

[70] David J. Griffiths. *Introduction to Quantum Mechanics*. Cambridge University Press, 2016.

[71] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. *DRAM Errors in the Wild: A Large-Scale Field Study*. 2009.

[72] M. Jeruchim. Techniques for estimating the bit error rate in the simulation of digital communication systems. *IEEE Journal on Selected Areas in Communications*, 2(1):153–170, 1984.

[73] David MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, United Kingdom, 2003.

[74] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[75] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[76] D. J. C MacKay. Good error correcting codes based on very sparse matrices. *IEEE Trans. on Info. Theory*, 45(2):399, 1999.

[77] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Trans. on Communications*, 44:1261, 1996.

[78] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, 1996.

[79] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[80] *Quantum Information Processing and Quantum Error Correction: An Engineering Approach*. Elsevier, 2012.

[81] Frank Gaitan. *Quantum Error Correction and Fault Tolerant Quantum Computing*. CRC Press, 2008.

[82] Daniel Lidar and Todd Brun. *Quantum Error Correction*. Cambridge University Press, 2013.

[83] Steane A.M. A tutorial on quantum error correction. *Proceedings of the International School of Physics*, 162(Quantum Computers, Algorithms and Chaos):1–32, 2006.

[84] Daniel Gottesman. Class of quantum error-correcting codes saturating the quantum Hamming bound. *Physical Review A*, 54(3):1862, 1996.

[85] A.D. Córcoles, Easwar Magesan, Srikanth J. Srinivasan, Andrew W. Cross, M. Steffen, Jay M. Gambetta, and Jerry M. Chow. Demonstration of a quantum error detection code using a square lattice of four superconducting qubits. *Nature Communications*, 6:6979, 2015.

[86] T. P. Harty, D. T. C. Allcock, C. J. Ballance, L. Guidoni, H. A. Janacek, N. M. Linke, D. N. Stacey, and D. M. Lucas. High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit. *Physical Review Letters*, 113(22), 2014.

[87] David P. DiVincenzo and Panos Aliferis. Effective fault-tolerant quantum computation with slow measurements. *Physical Review Letters*, 98(2), 2007.

[88] Rui Chao and Ben W. Reichardt. Quantum error correction with only two extra qubits. *Physical Review Letters*, 121(5), 2018.

[89] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017.

[90] Nicholas Chancellor, Szilard Szoke, Walter Vinci, Gabriel Aeppli, and Paul A. Warburton. Maximum-entropy inference with a programmable annealer. *Scientific Reports*, 6(1), 2016.

[91] N. J. Lord. Matrices as sums of invertible matrices. *Mathematics Magazine*, 60(1):33–35, 1987.

[92] IBM Quantum Experience 2017. *https://quantumexperience.ng.bluemix.net/qx/community*, Accessed: 17/12/2017.

[93] Daniel Gottesman. Quantum fault tolerance in small experiments. *arXiv:1610.03507*, 2016.

[94] IBM. IBMQX4 technical specifications. *https://github.com/Qiskit/qiskit-backend-information/tree/master/backends/tenerife/V1*, Accessed: 24/10/2018.

[95] IBM. Quantum Information Softare Kit. *www.qiskit.org*, Accessed: 17/12/2017.

[96] John A. Smolin, Jay M. Gambetta, and Graeme Smith. Efficient method for computing the maximum-likelihood quantum state from measurements with additive Gaussian noise. *Physical Review Letters*, 108(7), 2012.

[97] Christophe Vuillot. Is error detection helpful on IBM 5Q chips? *Quantum Information & Computation*, 18(11&12):0949–0964, 2018.

[98] Robin Harper and Steven Flammia. Fault tolerance in the IBM q experience. *arXiv:1806.02359*, 2018.

[99] M Acton, K.-A. Brickman, P. C. Haljan, P. J. Lee, L. Deslauriers, and C. Monroe. Near-perfect simultaneous measurement of a qubit register. *Quantum Information and Computation*, 6:465, 2006.

[100] Naomi Nickerson, Ying Li, and Simon Benjamin. Topological quantum computing with a very noisy network and local error rates approaching one percent. *Nature Communications*, 4:1756, 2013.

[101] J. I. Cirac and P. Zoller. Quantum computations with cold trapped ions. *Physical Review Letters*, 74(20):4091, 1995.

[102] D. Leibfried, B. DeMarco, V. Meyer, D. Lucas, M. Barrett, J. Britton, W. M. Itano, B. Jelenković, C. Langer, T. Rosenband, and D. J. Wineland. Experimental demonstration of a robust, high-fidelity geometric two ion-qubit phase gate. *Nature*, 422(6930):412, 2003.

[103] Anders Sørensen and Klaus Mølmer. Quantum computation with ions in thermal motion. *Physical Review Letters*, 82(9):1971, 1999.

[104] Christopher Ballance. High-fidelity quantum logic in Ca^+. *Oxford University*, PhD thesis, 2014.

[105] Daniel Loss and David P. DiVincenzo. Quantum computation with quantum dots. *Physical Review A*, 57(1):120, 1998.

[106] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Local unitary versus local clifford equivalence of stabilizer states. *Phys. Rev. A*, 71:062323, Jun 2005.

[107] Zhengfeng Ji, Jianxin Chen, Zhaohui Wei, and Mingsheng Ying. The LU-LC conjecture is false. *Quantum Info. Comput.*, 10(1):97–108, January 2010.

[108] Naomi H Nickerson, Joseph F Fitzsimons, and Simon C Benjamin. Freely scalable quantum technologies using cells of 5-to-50 qubits with very lossy and noisy photonic links. *Physical Review X*, 4(4):041041, 2014.

[109] J. Randall, S. Weidt, E. D. Standing, K. Lake, S. C. Webster, D. F. Murgia, T. Navickas, K. Roth, and W. K. Hensinger. Efficient preparation and detection of microwave dressed-state qubits and qutrits with trapped ions. *Physical Review A*, 91(1), 2015.

[110] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63, 2016.

[111] Alan Robertson, Christopher Granade, Stephen D. Bartlett, and Steven T. Flammia. Tailored codes for small quantum memories. *Physical Review Applied*, 8(6), 2017.

[112] David K. Tuckett, Stephen D. Bartlett, and Steven T. Flammia. Ultrahigh error threshold for surface codes with biased noise. *Physical Review Letters*, 120(5), 2018.

[113] T. R. Tan, J. P. Gaebler, Y. Lin, Y. Wan, R. Bowler, D. Leibfried, and D. J. Wineland. Multi-element logic gates for trapped-ion qubits. *Nature*, 528(7582):380, 2015.

[114] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Physical Review A*, 89(2), 2014.

[115] Markus Grassl. *Searching for linear codes with large minimum distance*, volume 19 of *Algorithms and Computation in Mathematics*. Springer, Heidelberg, 2006.

[116] Winton Brown and Omar Fawzi. Short random circuits define good quantum error correcting codes. In *2013 IEEE International Symposium on Information Theory*. IEEE, 2013.

[117] Charles H. Bennett, David P. DiVincenzo, John A. Smolin, and William K. Wootters. Mixed-state entanglement and quantum error correction. *Phys. Rev. A*, 54:3824–3851, 1996.

[118] Rui Chao and Ben W. Reichardt. Fault-tolerant quantum computation with few qubits. *npj Quantum Information*, 4(1), 2018.

[119] Christopher Chamberland and Michael E. Beverland. Flag fault-tolerant error correction with arbitrary distance codes. *Quantum*, 2:53, 2018.

[120] Ben W. Reichardt. Fault-tolerant quantum error correction for Steane's seven-qubit color code with few or no extra qubits. *arXiv:1804.06995*, 2018.

[121] Nicholas Chancellor, Joschka Roffe, Dominic Horsman, and Stefan Zohren. Decoding quantum error correction with specialized hardware. *In preparation.*

[122] Nicolas Sourlas. Spin-glass models as error-correcting codes. *Nature*, 339:693, 1989.

[123] Hidetoshi Nishimori. *Statistical Physics of Spin Glasses and Information Processing.* Oxford University Press, 2001.

[124] Pál Ruján. Finite temperature error-correcting codes. *Physical Review Letters*, 70(19):2968–2971, 1993.

[125] Easwar Magesan, J. M. Gambetta, and Joseph Emerson. Scalable and robust randomized benchmarking of quantum processes. *Physical Review Letters*, 106(18), 2011.

[126] Matthew C. Davey and David J.C. MacKay. Monte carlo simulations of infinite low density parity check codes over gf(q). 1998.

[127] Sergey Bravyi and Matthew B. Hastings. Homological product codes. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 273–282, New York, NY, USA, 2014. ACM.

[128] Jean-Pierre Tillich and Gilles Zemor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193, 2014.

[129] Thomas Draper and Samuel Kutin. QPIC: Quantum circuit diagrams in latex. *https://github.com/qpic/qpic*, Accessed: 17/12/2017.