IOP Publishing Quantum Sci. Technol. 8 (2023) 045028 https://doi.org/10.1088/2058-9565/acf157

Quantum Science and Technology

OPEN ACCESS

RECEIVED
1 November 2022

REVISED
26 June 2023

ACCEPTED FOR PUBLICATION
17 August 2023

PUBLISHED
22 September 2023

PAPER

Graphical structures for design and verification of quantum error correction

Nicholas Chancellor[1], **Aleks Kissinger**[2,3], **Stefan Zohren**[3], **Joschka Roffe**[1,4,5,*] and **Dominic Horsman**[1,2]

[1] Department of Physics, Durham University, Durham, United Kingdom
[2] Department of Computer Science, University of Oxford, Oxford, United Kingdom
[3] Department of Engineering Science, University of Oxford, Oxford, United Kingdom
[4] Dahlem Center for Complex Quantum Systems, Freie Universität Berlin, Berlin, Germany
[5] Department of Physics and Astronomy, University of Sheffield, Sheffield, United Kingdom
[*] Author to whom any correspondence should be addressed.

E-mail: joschka@roffe.eu

Keywords: quantum computing, quantum error correction, *ZX* calculus

Abstract

We introduce a high-level graphical framework for designing and analysing quantum error correcting codes, centred on what we term the coherent parity check (CPC). The graphical formulation is based on the diagrammatic tools of the *ZX*-calculus of quantum observables. The resulting framework leads to a construction for stabilizer codes that allows us to design and verify a broad range of quantum codes based on classical ones, and that gives a means of discovering large classes of codes using both analytical and numerical methods. We focus in particular on the smaller codes that will be the first used by near-term devices. We show how CSS codes form a subset of CPC codes and, more generally, how to compute stabilizers for a CPC code. As an explicit example of this framework, we give a method for turning almost any pair of classical $[n, k, 3]$ codes into a $[[2n-k+2, k, 3]]$ CPC code. Further, we give a simple technique for machine search which yields thousands of potential codes, and demonstrate its operation for distance 3 and 5 codes. Finally, we use the graphical tools to demonstrate how Clifford computation can be performed within CPC codes. As our framework gives a new tool for constructing small- to medium-sized codes with relatively high code rates, it provides a new source for codes that could be suitable for emerging devices, while its *ZX*-calculus foundations enable natural integration of error correction with graphical compiler toolchains. It also provides a powerful framework for reasoning about all stabilizer quantum error correction codes of any size.

1. Preliminaries

1.1. Introduction

Quantum computers of an appreciable size that run for any significant amount of time will need to be error corrected [1, 2]. Devices are beginning to be fabricated that approach the low error needed for error correction to work, and recent experiments have shown proof-of-concept of a number of key elements, e.g. error detection [3, 4], repeated error correction cycles [5–7], and operations on encoded qubits [8–10]. Excitingly, we also now have a direct experimental implementation of an error correction system where the logical errors decrease as the code size increases [11]. Quantum error correction (QEC) expands the Hilbert space in which logical qubits live by adding more physical resources to make a larger, typically entangled state. The additional degrees of freedom are used to detect and correct errors, without disturbing the logical information held non-locally in the larger state. One leading form of error correction includes topological codes such as the surface code [12]. Each block of physical qubits contains a single logical qubit, and higher error tolerances are obtained by expanding the size of the block. These codes are well-studied, conceptually straightforward, flexible, and have high thresholds (the maximum error rate of the underlying components

that can be tolerated—for surface codes, around 1% [13]). Such codes are powerful, but need too many physical qubits to support a single logical qubit to make them viable for the first generation of quantum computers currently being developed.

More efficient use of qubit resources can be gained by using either non-topological Calderbank-Shor-Steane (CSS) codes [14, 15], or else quantum low density parity check (LDPC) codes (inspired by high-performance classical error correction protocols) [16–19]. Recent advances have proved for LDPC codes in particular that such codes can be designed with constant rate and linear distance-to-block-length scaling [20]. Furthermore, efficient decoding methods have been developed making quantum LDPC codes useful in a practical setting [20–22]. The greater efficiency of both non-topological CSS and LDPC codes is particularly important in the near term as, while we are at the point where devices are near or below error thresholds for QEC (for example superconducting [11] and transmon [23, 24] qubits), the overheads associated with QEC will form a large barrier to its useful operation.

The added efficiency of such codes, however, comes at the expense of losing the high-level structure which makes topological codes so appealing, such as localised stabilizer measurements, efficient decoding algorithms, and the ability to implement fault-tolerant computations via topological manipulations [25–28]. Finding the best code for a given hardware device, which is also easy to work with conceptually (as the topological codes are) but efficient in terms of qubit resources is a hard problem. What is needed is a high-level language for stabilizer codes that enables them to be constructed intuitively and easily. With a flexible construction, codes can be tailored to the needs of different devices, enabling e.g. automated search for codes that are implementable with certain constraints on qubit connectivity.

In this paper we introduce the coherent parity check (CPC) construction for quantum stabilizer codes, as a framework that enables such flexible development of QEC protocols, in particular for near-term devices. The CPC framework gives a new way of interpreting classical error correcting codes as quantum codes. Rather than re-interpreting classical parity checks as stabilizer measurements (as in e.g. Calderbank, Shor and Steane (CSS) codes [14, 15]), they are interpreted as a direct description of the encoder (or equivalently, decoder) circuit. For our first family of codes, called *tripartite* CPC codes, we do this by making an explicit partition into data, bit-check, and phase-check qubits. Then, a pair of classical error correcting codes are used to determine how the bit- and phase-check qubits interact with the data qubits, respectively. The classical codes can be arbitrary, and need not, for example, yield commuting stabilizers. There is a price to pay for this extra flexibility: such an encoder may yield a quantum code with a lower code distance than its classical constituents. To correct this, a third, *cross-check* matrix is employed to enable bit- and phase-check qubits to 'check each other' for otherwise undetectable errors.

As an integral part of the techniques in this construction, we also present an associated graphical toolkit for constructing and reasoning about CPC codes, based on the *ZX-calculus*. This tensor-network-based language originated as a means of studying the interaction of complementary quantum observables [29], but also gives a very powerful tool for representing and transforming circuits [30]. For example, it has been shown that any two Clifford circuits describe the same unitary if and only they can be transformed into each other using the four core rules of the *ZX*-calculus [30, 31]. By considering extensions to the calculus, this has been extended to Clifford+T circuits [32] and an exact-universal family of circuits [33, 34]. In the present paper, we show how the *ZX*-calculus enables a visual representation of CPC codes and, through re-writing, the generation of error syndrome and stabilizer tables. The *ZX*-calculus has a history of use with error correction, e.g. [35–38]. It is also now being used in a number of places in the technology ecosystem, both by academic and industrial parties, including for compilation by Quantinuum (formerly Cambridge Quantum Computing) [39], and for surface codes by PsiQuantum [40] and Google [41]. This current paper builds on these foundations to enable QEC to be integrated directly and natively within a *ZX* compiler toolchain.

After giving an explicit construction of a [[11,3,3]] tripartite CPC code, we give two more general constructions for distance-3 codes: one that turns any $[n, k, 3]$ Hamming code into a $[[2n-k+1, k-1, 3]]$ code, and another that turns almost any pair of $[n, k, d \geqslant 3]$ codes into a $[[2n-k+2, k, 3]]$ code, subject to the relatively minor restriction that the codes must not have a 'global' parity check. That is, they admit a standard-form generator matrix $[\mathbb{1}|A]$ where A does not contain a row of all 1's.

We show how any CSS code can be represented as a tripartite CPC code, and furthermore how to compute logical operators and stabilizers for tripartite CPC codes. We generalise tripartite codes to *mixed CPC codes*, which enable qubits to act as mixed bit- and phase- parity checks. This in turn allows for encoding and numerical search for both cross-check matrices and optimisation of codes. By search we are able to find many thousands of small quantum codes. Optimising over parameters such as circuit depth then enables us to find codes optimised to potential devices. These include a structurally straightforward [[11,3,3]] code, and a dense [[9,3,3]] code. We have also used machine search to identify distance-5 codes, giving explicit check matrices for [[18,3,5]] and [[20,3,5]] codes. For codes of this size the machine search is

often very quick; for example, using a simple search program of < 100 lines on a single core of a desktop machine we can generate around 140 [[9,3,3]] codes in ten minutes.

Finally, we describe initial investigations into performing computation as well as memory tasks in these codes. As many logical data qubits are located on the same space of physical qubits, operations between them can be performed within the code block by altering the exact configuration of the encoder. The *ZX* graphical tools enable the configuration of the modified encoder to be found easily for Clifford-group gates, using the automated diagram re-writing tool Quantomatic [42].

Both the CPC framework, and the associated graphical tools, provide us with a new understanding of the construction of QEC codes. As well as providing a deeper insight into the theoretical foundations of all error correction procedures, this work also lends itself to the practical development of new codes that can be tailored to specific quantum architectures. Indeed, this work has already been used by the present authors to provide a number of follow-on results. In [43] this construction is used to implement a $[[4,2,2]]$ CPC detection code on the IBM Quantum System One device. Furthermore, that paper also provides an alternative exposition of CPC codes, using standard circuit notation, thus broadening even further the general applicability of the framework we present in the present paper. In [44] we use this framework to derive an Ising model mapping for decoding general QEC codes. This can then be imported for use on a quantum annealing co-processor, for example the D-Wave device. Finally in [43], we find another graphical model for quantum codes based on the classical factor graph formalism, enabling the CPC framework in specialised cases to be used with even less knowledge of quantum mechanics and QEC than is needed for the *ZX*-calculus. In all, the CPC framework gives a powerful graphical formalism for reasoning about QEC codes that interfaces with other important areas of quantum computing research.

1.2. Quantum and classical error correction

The job of error correction is to detect that an error has occurred, pinpoint which data carriers have become errored, and correct the error back to the original state. In general this is done using probabilistic inference: measurements on the data give the most likely error, which is then corrected for. Error correction protocols expand the number of data carriers, with the extra degrees of freedom used to perform the error correction. Exactly how a message (or a computation) is re-written into the larger space defines the particular *error correction code*.

In classical error correction codes, a message string of n bits communicated over a channel. Errors are considered as changes to bit values: a 0 can flip to a 1, and vice versa. To detect if this has occurred, different bit values in the string are compared to each other at the start of the communication. These measurements are then communicated along with the string, and the comparisons performed again. If there are changes, then a bit value has changed during transit. With suitable choice of which bit-value comparisons are sent, the position of the error can be found.

QEC differs from classical error correction in two important respects. First, quantum data (qubits) can suffer more than one form of error. Even on the simplest error model, both bit- and phase- values of a qubit can flip during transit: $|0\rangle \leftrightarrow |1\rangle$ and $(\alpha|0\rangle + \beta|1\rangle) \leftrightarrow (\alpha|0\rangle - \beta|1\rangle)$. Second, measurement of qubits, unlike bits, generally disturbs the system, with the state after measurement being an eigenstate of the measurement operator rather than the original state. To compensate for this, the most typical method of QEC expands the qubit space so that the only operators that are measured are so-called 'stabilizers'. The expanded state is a joint eigenstate of these operators, and therefore measuring them will not disturb the state. The particular stabilizer subspaces of the expanded state give the QEC code.

The difference can be seen most straightforwardly in basic three-system examples. In the classical case, consider the basic parity check of figure 1. A and B are the 'data' bits, and P is a parity checking bit. At the beginning of the protocol, the joint bit-parity of A, B is measured and stored in P: $[[P(0)]] = [[A(0)]] \oplus [[B(0)]]$. After a time in which errors can occur, the procedure is repeated: $[[P(t)]] = [[A(0)]] \oplus [[B(0)]] \oplus [[A(t)]] \oplus [[B(t)]]$. If there were no errors then $[[P(t)]] = 0$. An outcome 1 shows that an error has occurred (but not, at this stage, where).

Now we consider the quantum case, figure 2. A single data qubit $|A\rangle = a|0\rangle + b|1\rangle$ is supplemented with two additional qubits for the code, P, Q, initialised in the state $|0\rangle$. The three are entangled using the encoder given, creating the three-qubit state $a|000\rangle + b|111\rangle$. The state is now in an eigenstate of the two Pauli operators $S_1 = Z_A \otimes Z_P$ and $S_2 = Z_A \otimes Z_Q$. These can therefore be measured without disturbing the data encoded in the state. If at a subsequent point the operators are measured and found not to return the value $+1$ then an error has occurred. More specifically, a bit-flip error has occurred; this encoding detects only a single type of error. Unlike the classical case, this is an error correction code as the two 'syndromes' (outcomes ± 1 of measuring the two stabilizers S_1 and S_2) give enough information to pinpoint the source of the error: if $S_1(S_2)$ flips to -1 then $P(Q)$ has an error, and if both are measured as -1 then it is A that is errored.

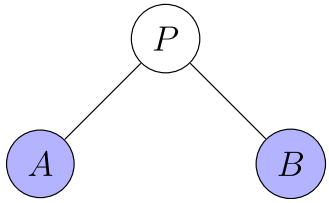


Figure 1. A classical three-bit error detection code: two bits of data, A and B, have their mutual parity encoded into the bit-value of P.

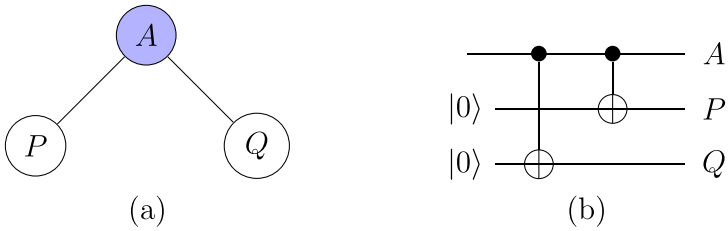


Figure 2. Quantum three-qubit code: (a) A single qubit of data $|A\rangle = a|0\rangle + b|1\rangle$ is supplemented by two additional code qubits P, Q; (b) Encoding circuit, resulting a single logical qubit supported on all three physical qubits, $a|000\rangle + b|111\rangle$.

The use of additional 'code' qubits in the quantum case therefore serve a dual purpose. Firstly they expand the space so that some of the operators that stabilize it are known. These then can be measured without disturbing the encoded data. Secondly, the pattern of these measurements needs to be such that, as in the classical case, it gives enough information to decode whether there is an error and (if it is a correction code not just a detection one) where it has occurred. Note that, normally in QEC all the 'code' qubits are called 'data' qubits. Their stabilizers are measured fault-tolerantly by bringing in additional 'syndrome' qubits, which are not generally included in the count of the number of qubits in a code. Here, we will distinguish 'data' and 'code' (and later 'parity') qubits, all of which are simply called 'data' qubits standardly.

More additional qubits are needed if a quantum code is to correct both phase- and bit- errors. One method is to concatenate, nesting a bit-correction code in a phase-correction code (the three-qubit code can be concatenated into a nine-qubit code capable of detecting and correcting one of both types of error). Another way is used in CSS quantum codes: two classical codes, sharing a property of duality, are used together, one correcting bit- and one correcting phase- information. For example, the CSS Steane code is formed from two copies of the classical Hamming code, encoding one qubit of information with six additional code qubits. The stabilizers are (with the tensor product understood):

$$Z_1Z_3Z_5Z_7 \;\; Z_2Z_3Z_6Z_7 \;\; Z_4Z_5Z_6Z_7$$
$$X_1X_3X_5X_7 \;\; X_2X_3X_6X_7 \;\; X_4X_5X_6X_7.$$

Quantum codes are often described using $[[n,k,d]]$ terminology: k qubits of information are carried using n total qubits, with the code capable of correcting $(d-1)/2$ Pauli errors. For example, the Steane code is a $[[7,1,3]]$ code.

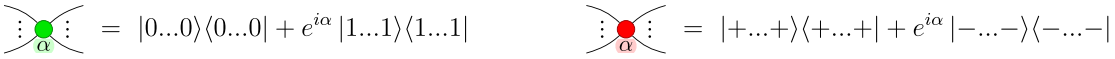
By comparison with classical codes, not many quantum codes are known. The various constraints in terms of specifying stabilizer subspaces, error decoding, and (in the case of CSS codes) finding dual classical codes, give significant challenges for identifying good codes for various use-cases. The most flexible in terms of expanding easily to any desired distance are the topological codes. However, they have huge overheads in terms of qubit resources compared to the more information-dense CSS codes. This would make CSS codes seem the obvious choice, in particular for first-generation quantum technologies where the efficient use of qubit resources is paramount. However, CSS codes often lack the desirable properties of topological codes, such as scalability, sparsity, and efficient decoding algorithms.

1.3. The *ZX*-calculus

The *ZX*-calculus is a language for reasoning about quantum systems which generalises quantum circuits. It was originally developed to study the interaction of mutually unbiased bases [29, 30], and takes its name from the Pauli Z and X observables whose respective bases of eigenstates define the primitive components of *ZX*-diagrams. Unlike quantum gates, these components exhibit a well-understood algebraic structure (based

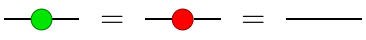
on so-called 'commutative Frobenius algebras') which enable one to easily prove many identities between *ZX*-diagrams. In particular, equality of *ZX*-diagrams is captured by a small number of diagrammatic equations (i.e. equations between certain small, equivalent tensor networks). Thus, reasoning about equality for *ZX*-diagrams becomes an exercise in diagram transformation.

As with circuit diagrams, *ZX*-diagrams consist of compositions and tensor products of linear maps. Plugging two diagrams together represents composition and putting them side-by-side represents tensor product. The primitive components in a *ZX*-diagram are called *spiders*. These are linear maps with m input wires and n output wires, labelled by a phase angle $\alpha \in [0, 2\pi]$:



Omitted phase angles are assumed to be 0. Note that spiders need not be unitary, but in the special case of $m = n = 1$, they are unitary and equal to the usual Z and X phase gates:

In particular, if $\alpha = \pi$, these capture the Pauli Z and X gates, respectively. If $\alpha = 0$, these are equal to the identity operator:

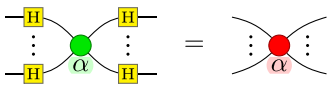


(1)

Similarly, spiders with $\alpha = 0$ and two output or input wires are equal to the (unnormalised) Bell state or effect, respectively:

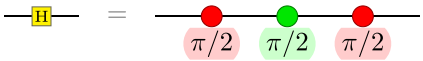
(2)

In addition to the two colours of spiders, we also include Hadamard gates, which flip the colour:



(3)

We treat this a derived operation, as we can build it out of spiders using the Euler decomposition (see [30, 9.4.4]):

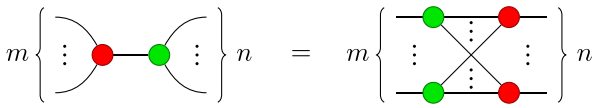


(4)

The most important rule of the *ZX*-calculus is the *spider fusion* law, which says that if two spiders of the same colour are connected, they fuse together into one bigger spider:

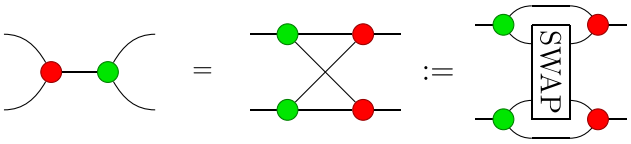
(5)

The second most important rule is *strong complementarity*, which is expressed as follows:



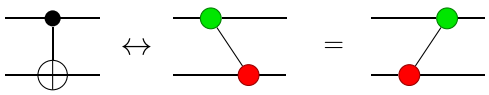
(6)

where the RHS is a totally connected bipartite graph. That is, each of the m green spiders on the left is connected to each of the n red spiders on the right. We freely use SWAP gates to account for 'wire crossings'. For example, in the case of $m = n = 2$, we have:

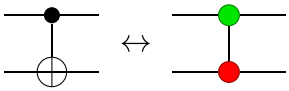


This can always be done without ambiguity. Alternatively (and equivalently), we can treat *ZX*-diagrams simply as a graphical depiction of a tensor network, *àla* Penrose [45]. For more details, see [30, sections 3.1.3 and 5.2.4].

An interesting class of *ZX*-diagrams are the *Clifford ZX-diagrams*, where we restrict the angles on spiders to be multiples of $\frac{\pi}{2}$. This is a superset of the set of Clifford circuits. We have already seen the construction of Hadamard gates in (4). We already saw the construction of $Z(\pi/2) = S$ gates and Hadamard gates. CNOT gates can be built out of spiders as follows:



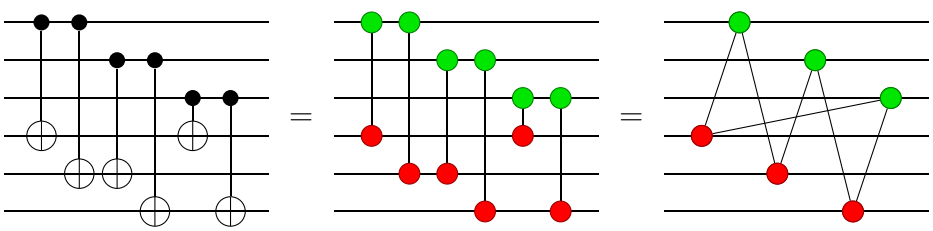
The equation above, combined with (5) lets us reverse the direction of any wire in a *ZX*-diagram, hence we will in general treat them as undirected. For example, the equation above enables us to write the following without ambiguity:



Thanks to spider fusion, we can compactly represent circuits which compute more general parities of computational basis states in a way that closely resembles the associated Tanner graph. For example, the unitary:

$$U :: |a,b,c,d,e,f\rangle \mapsto |a,b,c,d \oplus a \oplus c, e \oplus a \oplus b, f \oplus b \oplus c\rangle$$

can be captured as:



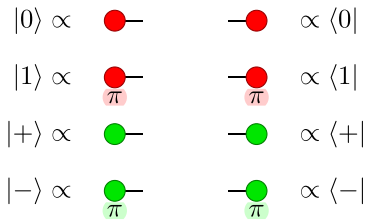
We will exploit this fact, and introduce some new notation that makes an explicit connection with parity check matrices, in the next section.

In addition to this connection with Clifford circuits, the family of Clifford *ZX*-diagrams are interesting because it is very easy to give a *complete* set of equations for them. Namely, if any two Clifford *ZX*-diagrams

yield the same linear map (up to renormalisation), one can be transformed into the other efficiently using just equations (1)–(6) above [30, 31]. In particular, equality between Clifford circuits and stabiliser states can be decided using (1)–(6) as a special cases. Given that the these rules can also prove equalities for some non-stabiliser states and operations, the *ZX*-calculus can be seen as a 'beefed up' version of the stabiliser formalism.

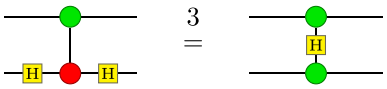
It was furthermore shown that by adding 3 additional equations to the *ZX*-calculus, one can decide equality between pairs of *arbitrary ZX*-diagrams [32–34, 46], and hence arbitrary universal quantum circuits, though the intermediate diagrams may grow exponentially large for the non-Clifford case.

We now briefly summarise the derived operations and rules from the *ZX*-calculus that will be used in this paper. First, note that we can express bras and kets associated with the Pauli *Z* and *X* eigenstates as spiders:

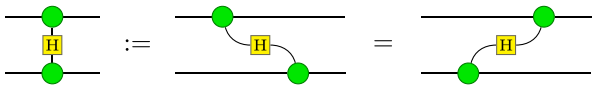


(7)

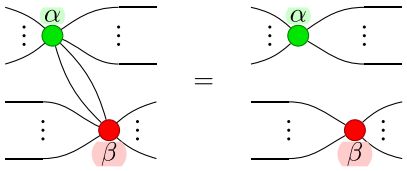
We already saw how to construct Z, X, and CNOT gates. We can thus also construct CZ gates and simplify the diagram using (3):



As in the case of CNOT gates, the drawing the wire with the H-gate horizontally is unambiguous because:



The strong complementarity law implies the simpler *complementarity law*, which enables pairs of edges between spiders of opposite colours to be removed:



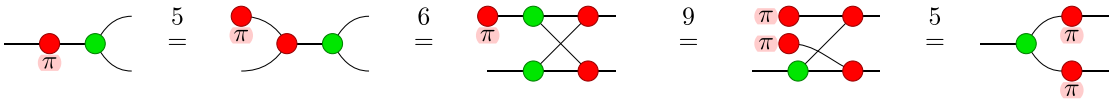
(8)

This equation holds whenever the ONBs associated to a pair of spiders are complementary (a.k.a. mutually unbiased) with respect to each other [29]. Note that this can disconnect previously connected diagrams, so this has quite a different character from (5), which dictates what happens when spiders of the same colour meet.

Using the *ZX*-calculus, one can show that green spiders copy Z-basis states and red spiders copy *X*-basis states. That is, for $k \in \{0, 1\}$, we have:

(9)

Combining this with strong complementarity, this implies that Pauli X gates copy through green spiders:



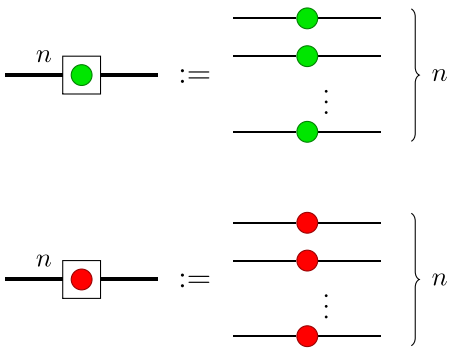
Similarly, we can show that Pauli Z gates copy through red spiders. Furthermore, it is straightforward to show more generally that:

(10)

for any n. This fact will be used throughout the paper to propagate Pauli X and Z gates (a.k.a. bit and phase errors, respectively) through ZX diagrams.

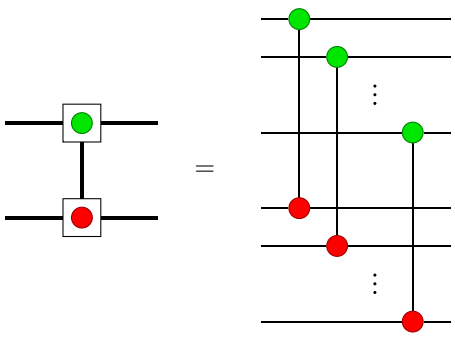
1.4. Scalable notation for the *ZX* calculus

We now introduce a higher-level notation for ZX-diagrams which makes a direct connection with parity-check matrices. First, we define a 'spider box' on a collection of n nodes as follows:



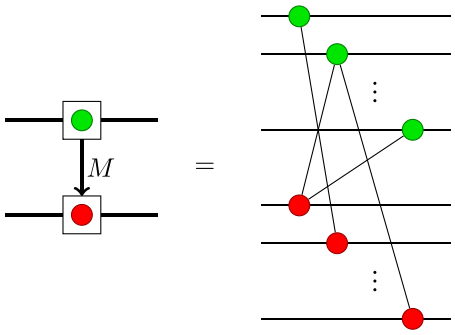
We will typically suppress the 'n' on thick wires if it is clear from the context.

Spider boxes can be joined by edges to either single nodes or to other spider boxes. An unlabelled edge from a spider box to a spider box denotes an edge from the ith wire of the first box to the ith wire of the second (the spider boxes must therefore be of the same size, that is contain the same number of nodes):



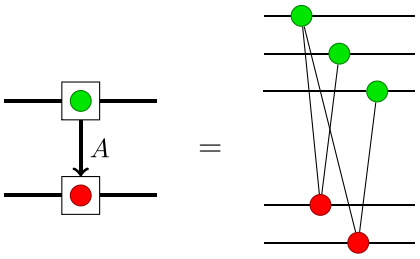
In other words, it represents a sequence of n CNOT gates, where the ith qubit with a green spider serves as a control for the ith qubit with a red spider.

We can make this notation more expressive by allowing an edge between spider boxes (of different colours) to be associated with an adjacency matrix. That is, two spider boxes can be joined by a directed edge labelled by matrix M with entries in $\{0,1\}$, where $M_{ij}=1$ indicates the presence of a wire connecting the jth input node to the ith output node:



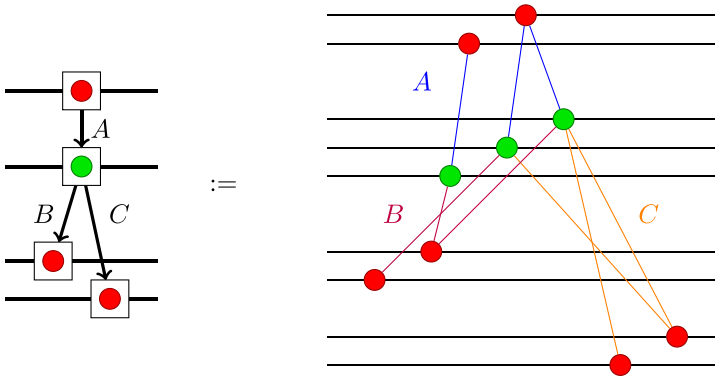
Note that M does not need to be a symmetric matrix, hence the need for indicating the direction. It also does not need to be a square matrix, hence there can be different numbers of input spiders and output spiders, as in the following example:

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \longrightarrow$$

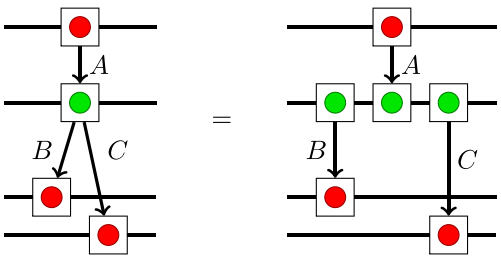


(11)

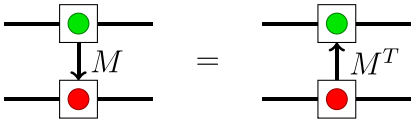
This notation extends in the obvious way for multiple spider-boxes connected by adjacency matrices, e.g.



Just like with normal spiders, spider-boxes fuse together along (un-labelled) edges:



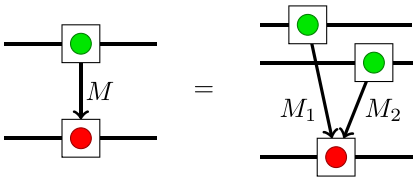
Also, note that the direction of the edge may be reversed by transposing the adjacency matrix:



(12)

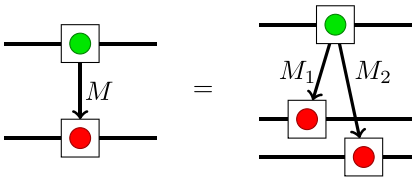
and spider-boxes may be split or combined using block matrices. For example, a row of block matrices yields:

$$M = \left(\begin{array}{c|c} M_1 & M_2 \end{array} \right) \quad \Longleftrightarrow$$



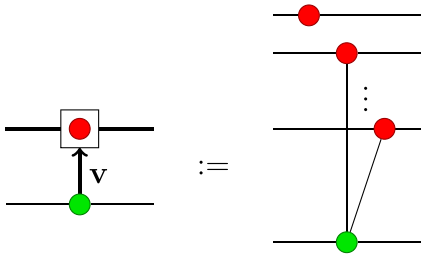
and a column of block matrices yields:

$$M = \left(\begin{array}{c} M_1 \\ \hline M_2 \end{array} \right) \quad \Longleftrightarrow$$



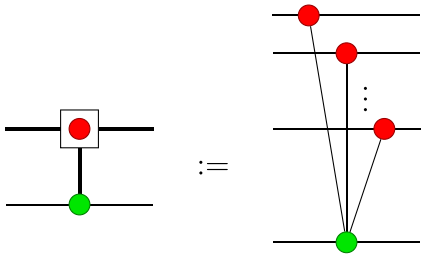
(13)

In the case where one spider box has size $n = 1$, this reduces to a single spider on a wire. In such a case, we depict the $n = 1$ spider-box simply as a spider:



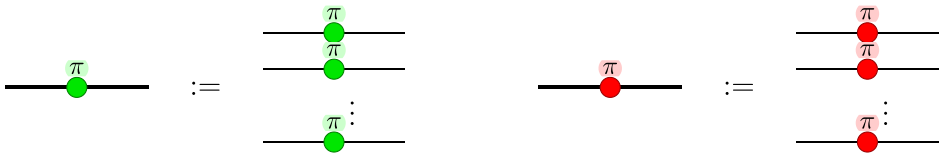
(14)

It behaves exactly like the case of general spider-boxes connected by an adjacency matrix (which in this case is a vector), with only one exception: a box connected to a node by an un-labelled edge stands for an edge connecting *every* spider in the spider-box to the single spider:



(15)

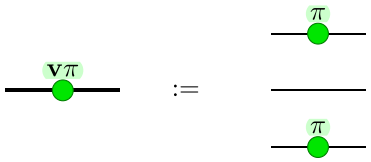
This notation is useful for studying the flow of errors through a ZX-diagram. We represent Pauli errors appearing on multiple wires using bit-vectors. On a thick wire, a single green or red spider labelled by a π-phase indicates the presence of a Pauli Z or Pauli X gate on all n qubits, respectively:



(16)

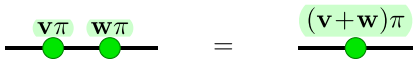
More generally, for a vector $\mathbf{v}$ with entries $v_i \in \{0, 1\}$, a spider labelled $\mathbf{v}\pi$ indicates the presence of a π phase on the ith wire if and only if $v_i = 1$. For example:

$$\mathbf{v} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \longrightarrow$$



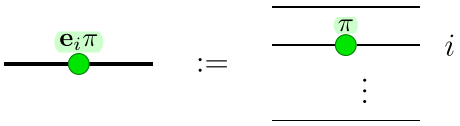
(17)

Since π-phases combine modulo-2, error vectors add:



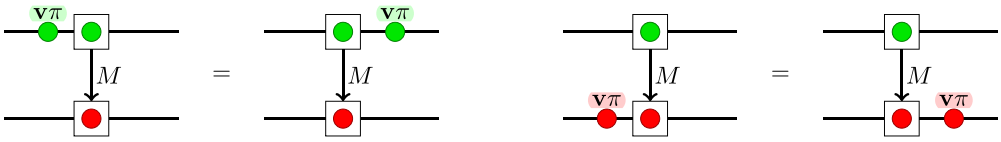
where the sum $\mathbf{v}+\mathbf{w}$ is taken over GF(2).

Often it will be useful to study the case of a single error, in which case we can use the unit vector $\mathbf{e}_i$, which has a 1 in the ith position and zeroes elsewhere:



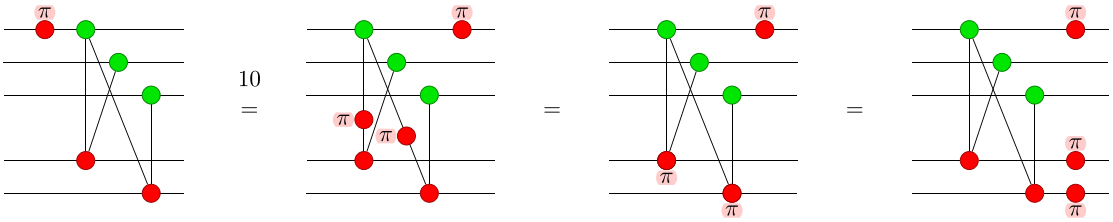
(18)

Thanks to spider-fusion, errors will commute through spider boxes of the same colour:

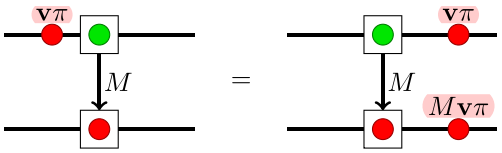


(19)

On the other hand, if errors of a different colour meet a spider, we can apply (10) to copy the errors through. In that case, errors will not only pass through a spider, but also propagate to the neighbours of that spider (modulo-2):

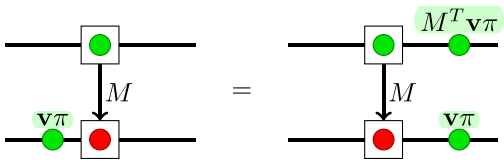


We can capture this behaviour succinctly in terms of adjacency matrices using matrix multiplication over GF(2). A $\mathbf{v}$-labelled error propagates *forward* along an adjacency matrix M to become an $M\mathbf{v}$-labelled error:



(20)

Combining this with (12), we can see that a **v**-labelled error propagates *backwards* across an M-labelled edge to become an $M^T\mathbf{v}$-labelled error:



(21)

By symmetry, equations (20) and (21) also hold with the colours reversed.
This notation can be fully formalised within the *ZX* environment as a PROP [47].

2. The CPC construction

The construction for QEC that we introduce in this paper is based around the process of *coherent parity checking*. A CPC is a procedure for checking for errors on pairs (or more) of qubits over time. It is analogous with parity checking in classical error correction in a more direct way than standard presentations of QEC.

In this section we introduce the basic gadget on two qubits, showing how it checks for errors while being non-disturbing. After introducing it in terms of circuit notation and Dirac notion, we give the operation in terms of the *ZX*-calculus, and show how the graphical formalism simplifies calculations. We begin to make contact with the usual format for QEC by showing how the gadget constructs stabilizers across qubits, and deal graphically with errors propagating in the gadget. We end the section by constructing a first example CPC code, with two gadgets checking two data qubits for bit and phase errors. We demonstrate in this example what becomes the central issue of CPC codes: finding cross-checks between the parity-checking qubits to remove undetectable errors. With this cross check in place for two data qubits, we reproduce a known example in the CPC formalism: the [[4,2,2]] error detection code.

By examining this example we identify a common structure to CPC codes that will carry through into the rest of the paper, where bit- and phase- parity checks are identified separately within the code. This section concerns only a handful of qubits and error detection; in subsequent sections we look at using the basic structure of a CPC code in order to develop scalable building procedures for codes in the construction.

2.1. Coherent parity checking

The basic CPC is a three-stage circuit on three qubits that detects an error of a single type (Pauli X or Z) on one of the qubits. As with classical parity checking, figure 1, we use one of the qubits (a 'parity qubit') to detect errors, and the other two ('data qubits') to store information[6]. The circuit for the basic operation is shown in figure 3. The data qubits A and B are in the state $|\psi_{AB}\rangle = \sum_{ij} a_{ij}|ij\rangle$ where $i,j \in 0,1$. The parity qubit P starts in the state $|0\rangle$, and then is entangled with the data qubits through two CNOT gates. Measuring the parity qubit now gives a measurement of the Pauli $Z_A \otimes Z_B$ operator—the joint bit parity of A and B. In qubits, as we noted in the Preliminaries, such a measurement would be disturbing. We therefore do not measure P but let the system evolve.

Using a simple error model in which error ε occurs in a specific time window t during which the system is evolving, we then repeat the encoding step at the end of the gadget to unentangle the parity check qubit from the data qubits. By measuring the parity qubit, it is possible to deduce whether an error has occurred during time t on either A, B, or P, while not disturbing the information $|\psi_{AB}\rangle$ held in AB. Importantly, nothing needs be known about the state of A or B—it does not have to be a stabilizer state.

To see how this simple gadget works, we walk through its mathematical action on the three qubit system $|\psi_{AB}\rangle \otimes |0_P\rangle$. To this end, it useful to re-express the CPC circuit in the form shown in figure 4 by making the substitution

$$\mathrm{CNOT}_{a,b} \to H_a \circ \mathrm{CZ}_{a,b} \circ H_a. \tag{22}$$

where H_i is the single-qubit Hadamard operator, and '$\circ$' denotes sequential gate composition. In this form, it can be seen that the action of the encoder is to perform the parity check Z_AZ_B on the data register, conditional on the value of the parity check qubit which is prepared in the conjugate basis by a Hadamard gate.

[6] It is worth noting at the outset that in the CPC construction, the term 'data qubits' refers to a subset of what are termed data qubits in standard presentations of stabilizer codes. In the CPC framework, both data and parity qubits together make up what are in other presentations called simply data qubits. We will deal with syndrome qubits later on.

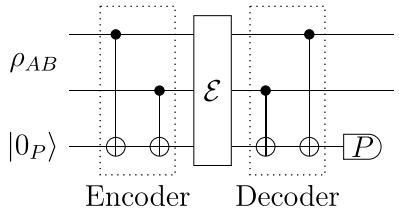


Figure 3. The fundamental coherent parity check. A bit-flip error on any of the three qubits is picked up by the measurement P.

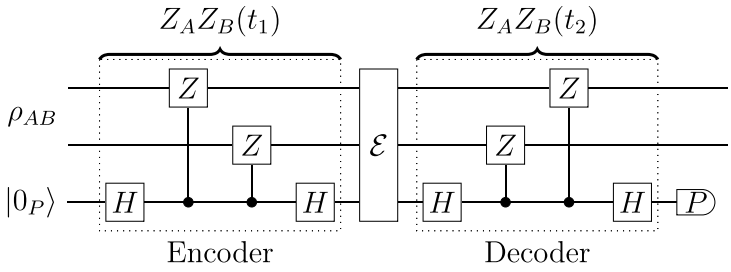


Figure 4. The fundamental CPC gadget re-written as controlled-phase operations. The CPC can be viewed as coherently comparing the operator Z_AZ_B at time t_1 and t_2 (before and after any error). Note that a standard Z_AZ_B syndrome measurement would measure out the parity-check qubit at time t_1 as well as at t_2, whereas the CPC parity check keeps the parity-check qubit unmeasured until t_2.

Following the encode stage, the state of the three-qubit system is given by

$$U_{\text{encode}}\left|\psi_{AB}\right\rangle\left|0_P\right\rangle=\frac{1}{2}\left(I+Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|0_P\right\rangle+\frac{1}{2}\left(I-Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|1_P\right\rangle. \tag{23}$$

We now have a three-party entangled state, where the two terms of the superposition correspond to the $+1$ and -1 eigenstates of the Z_AZ_B operator respectively.

During the wait stage, the system is subject to a single-qubit operation from the set $\mathcal{E}=\{I,X_A,X_B,X_P\}$. The state of the CPC gadget is then given by

$$\mathcal{E}U_{\text{encode}}\left|\psi_{AB}\right\rangle\left|0_P\right\rangle=\frac{1}{2}\mathcal{E}\left(I+Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|0_P\right\rangle+\frac{1}{2}\mathcal{E}\left(I-Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|1_P\right\rangle. \tag{24}$$

Following the wait stage, the parity qubit is disentangled from the register by the decoder. The decoder, U_{decode}, is the unitary inverse of the encoder and transforms the system as follows,

$$\begin{aligned}U_{\text{decode}}\mathcal{E}U_{\text{encode}}\left|\psi_{AB}\right\rangle\left|0_P\right\rangle=&\frac{1}{4}\left(I+Z_AZ_B\right)\mathcal{E}\left(I+Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|0_P\right\rangle+\frac{1}{4}\left(I-Z_AZ_B\right)\mathcal{E}\left(I+Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|1_P\right\rangle\\&+\frac{1}{4}\left(I-Z_AZ_B\right)\mathcal{E}\left(I-Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|0_P\right\rangle+\frac{1}{4}\left(I+Z_AZ_B\right)\mathcal{E}\left(I-Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|1_P\right\rangle.\end{aligned} \tag{25}$$

The above simplifies to

$$U_{\text{decode}}\mathcal{E}U_{\text{encode}}\left|\psi_{AB}\right\rangle\left|0_P\right\rangle=\frac{1}{2}\left(\mathcal{E}+Z_AZ_B\mathcal{E}Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|0_P\right\rangle+\frac{1}{2}\left(\mathcal{E}-Z_AZ_B\mathcal{E}Z_AZ_B\right)\left|\psi_{AB}\right\rangle\left|1_P\right\rangle. \tag{26}$$

The final step in the CPC gadget is to measure the parity qubit P. In the event that no error occurred, $\mathcal{E}=I$, the second term in the above goes to zero and the measured syndrome is 0. Intuitively we would expect this as the encoder is the unitary inverse of the decoder and

$$U_{\text{decode}}\mathcal{E}U_{\text{encode}}\left|\psi_{AB}\right\rangle\left|0_P\right\rangle=I\left|\psi_{AB}\right\rangle\left|0_P\right\rangle, \tag{27}$$

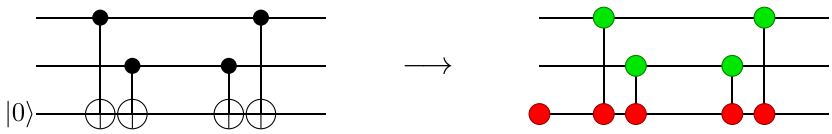
when $\mathcal{E}=I$. If a bit-flip error did occur, $\mathcal{E}\in\{X_A,X_B,X_P\}$, the first term goes to zero and the measured syndrome is 1. More generally, the output of the CPC gadget can be written as follows

$$U_{\text{decode}}\mathcal{E}U_{\text{encode}}\left|\psi_{AB}\right\rangle\left|0_P\right\rangle=\begin{cases}\left|\psi_{AB}\right\rangle\left|0_P\right\rangle, & \text{if } [\mathcal{E},Z_AZ_BZ_P]=0\\ \left|\psi_{AB}\right\rangle\left|1_P\right\rangle, & \text{if } [\mathcal{E},Z_AZ_BZ_P]\neq 0.\end{cases} \tag{28}$$

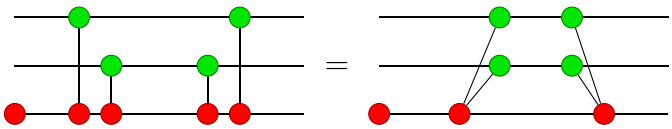
From the above we see that the parity qubit is no longer entangled with the register at the end of the CPC cycle. The final syndrome measurement will therefore not decohere the register: the output depends only upon whether the error operator $\mathcal{E}$ commutes with the parity check operator $Z_A Z_B Z_P$.

This elementary operation is a very simple error detection code (there is not yet enough information to correct the error) for a single error of a single qubit. In appendix A we give the full $|\psi_{AB}\rangle$ analysis showing how many errors it can detect, and we calculate the error suppression to be $\varepsilon^2 \to \varepsilon^4$. The result also generalises to other parity checks. For example, replacing the $Z_A Z_B$ parity check with $X_A X_B$ gives a CPC gadget that can detect phase-flip errors.

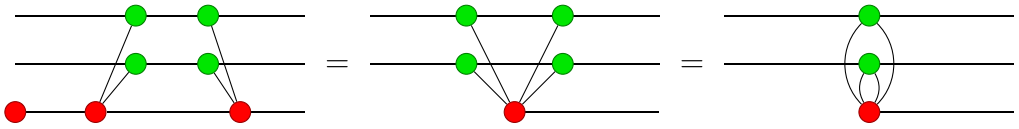
The action of the CPC is even clearer when considered diagrammatically. We first translate the encode and decode circuits, along with the preparation of the parity-check qubit, into the *ZX*-calculus:



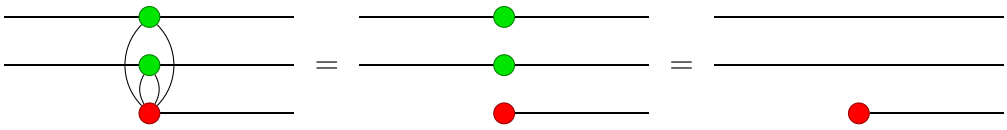
First, we can represent the encoder and the decoder more compactly by fusing together spiders of the same colour:



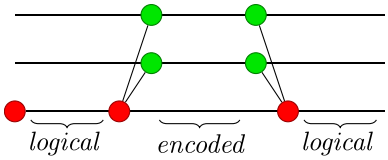
As we just saw, the decoder should undo the action of the encoder, leaving the parity qubit in the $+1$ eigenstate of the Z basis and leaving the first two qubits unchanged. We can show this using the *ZX*-calculus as follows. First, fuse the matching spiders in the encoder and decoder together:



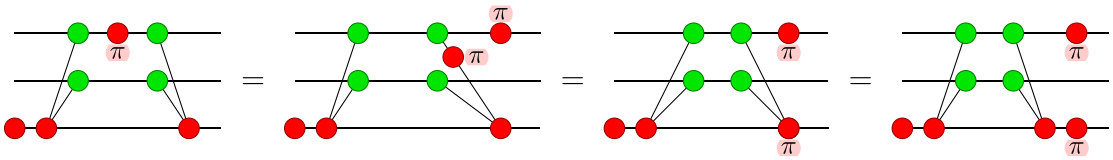
Then, by the complementarity rule, pairs of edges between red and green spiders vanish, giving us the result:



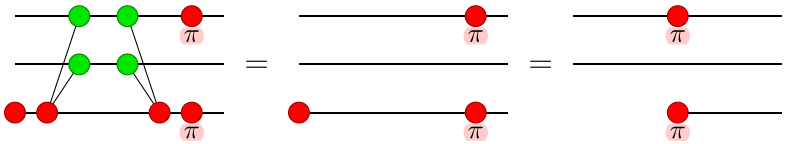
Now, lets see what happens when an error occurs between the encoder and the decoder. First, note that there are two kinds of 'regions' in the CPC gadget, the *logical* region, where the data qubits are not entangled with parity qubits, and an *encoded* region, where the data qubits are entangled with the parity qubits:



If a Pauli error occurs in the encoded region, we can push it forward (or backwards) into the logical region. For example, a bit (i.e. Pauli X) error on the first data qubit can be pushed forward across the decoder, using the copy law and spider fusion laws:

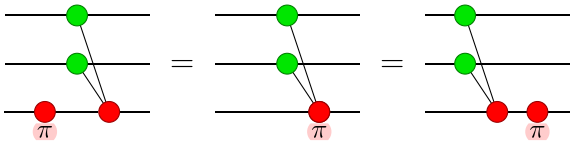


Then as before, the encoder and decoder cancel each other out, leaving the parity qubit in the -1 eigenstate of the Z measurement:



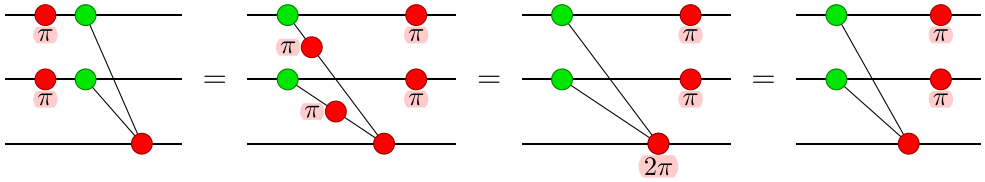
If we measure the parity qubit, we will detect that a Pauli-X error occurred somewhere, and indeed the Pauli-X error remains on the first qubit after decoding. A similar thing happens if an error occurs on the second data qubit.

More generally, we can always compute the result of an error in the encoded region by pushing it forward across the decoder, and noting the presence of π-phases on parity qubits. For example, if an X error occurs on the parity qubit, it can be pushed through the decoder as follows:



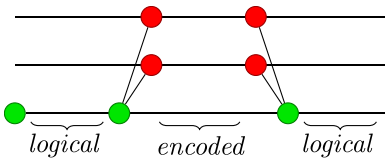
Hence, we will also observe a -1 outcome if we measure the parity qubit, even though no error occurred on the data qubits.

However, if two bit errors occur, either on both data qubits or on one data qubit and on the parity qubit, the π-phases cancel out in the logical region of the parity qubit, so the errors remain undetected. For example, the case of an error on both data qubits is computed as follows:

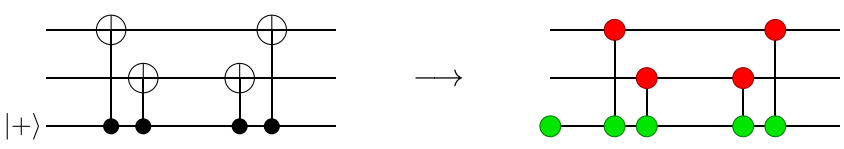


Hence, this CPC gadget is able to detect (but not yet correct) a single bit error.

Just as bit errors are represented by π-labelled red spiders, phase errors (i.e. Pauli Z errors) are represented by π-labelled *green* spiders. Hence, reversing all of the colours produces a CPC gadget that can detect a single phase error:



Explicitly, this can be realised using the same circuit as before, except we reverse the roles of the Z and X bases:



Since all of the rules of the ZX-calculus are colour-symmetric, the reasoning is identical to before.

2.2. Stabilizers from a CPC

We can now start to make contact between codes based on the CPC gadget, as presented here, and the usual understanding of QEC in terms of stabilizer subspaces and syndrome measurement. General stabilizer codes encode quantum information by 'spreading' the state of the data qubits in a non-specific way over a space of codewords. In contrast, CPC codes retain a clear distinction between qubits which encode data and qubits which encode parity information. To see this, consider two data qubits A and B which are in the state

$$|\psi_{AB}\rangle = \alpha_{00}|0_A0_B\rangle + \alpha_{01}|0_A1_B\rangle + \alpha_{10}|1_A0_B\rangle + \alpha_{11}|1_A1_B\rangle . \tag{29}$$

The action of the CPC encoder is to replicate the parity value given by the operator Z_AZ_B into a parity check qubit P such that,

$$|\psi_{ABP}\rangle_{\text{enc}} = U_{\text{encode}}|\psi_{AB}\rangle|0_P\rangle , \quad Z_AZ_B|\psi_{ABP}\rangle_{\text{enc}} = p_{AB}|\psi_{ABP}\rangle_{\text{enc}} , \tag{30}$$

$$Z_P|\psi_{AB}\rangle_{\text{enc}} = p_p|\psi_{ABP}\rangle_{\text{enc}} , \quad p_{AB} = p_p \;\forall\; \{A,B\} , \tag{31}$$

where $p_{\{AB,P\}} = \pm 1$ are the parity check outcomes. Applied to the two qubit state, the full output of the CPC encoder is therefore

$$|\psi_{ABP}\rangle_{\text{enc}} = \alpha_{00}|0_A0_B\rangle|0_P\rangle + \alpha_{01}|0_A1_B\rangle|1_P\rangle + \alpha_{10}|1_A0_B\rangle|1_P\rangle + \alpha_{11}|1_A1_B\rangle|0_P\rangle . \tag{32}$$

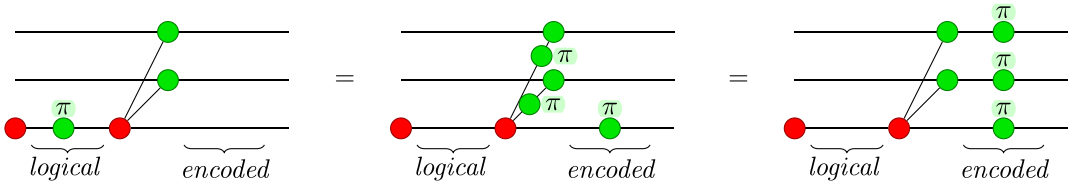
The encode stage projects the $|\psi_{AB}\rangle$ state into a $4D$ subspace of the expanded 3-qubit Hilbert space $\mathcal{H}_{ABP}$. In the language of conventional stabilizer codes, this partitioning of the Hilbert space can be thought of in terms of a code space $\mathcal{C}_{\text{code}}$ and an error space $\mathcal{C}_{\text{error}}$ as shown below

$$\mathcal{C}_{\text{code}} = \begin{bmatrix} |0_A0_B\rangle|0_P\rangle , \\ |0_A1_B\rangle|1_P\rangle , \\ |1_A0_B\rangle|1_P\rangle , \\ |1_A1_B\rangle|0_P\rangle \end{bmatrix} , \qquad \mathcal{C}_{\text{error}} = \begin{bmatrix} |0_A0_B\rangle|1_P\rangle , \\ |0_A1_B\rangle|0_P\rangle , \\ |1_A0_B\rangle|0_P\rangle , \\ |1_A1_B\rangle|1_P\rangle \end{bmatrix} . \tag{33}$$

In each of the four element of $\mathcal{C}_{\text{code}}$, the bit values of the first two qubits correspond to the basis states in unencoded state $\{|0_A0_B\rangle , |0_A1_B\rangle , |1_A0_B\rangle , |1_A1_B\rangle\}$. As a result it remains possible to distinguish qubits A and B as the data qubits even after encoding. Carrying the parity information forward coherently, in a qubit rather than a classical measurement outcome, allows arbitrary such joint parity measurements to be made, rather than having to measure a known stabilizer of the data qubits.

The duplication of parity information into the parity check qubits gives rise to stabilizers across the combined system of data+parity qubits. The code space of the CPC gadget $\mathcal{C}_{\text{code}}$, defined in equation (33), is stabilized by the operator $Z_AZ_BZ_P$. This is the case, regardless of the values of A and B, as the encoder ensures that $Z_AZ_B|\psi_{AB}\rangle_{\text{enc}} = Z_P|\psi_{AB}\rangle_{\text{enc}}$ and therefore $Z_AZ_BZ_P|\psi_{AB}\rangle_{\text{enc}} = (+1)|\psi_{AB}\rangle_{\text{enc}}$. The decode step of the CPC gadget can be viewed as measuring the $Z_AZ_BZ_P$ stabilizer. While the identification of stabilizers becomes more complicated as we move to CPC codes that detect both bit and phase errors, we will see that the conclusion carries through, and that CPC constructed codes are stabilizer codes.

While supporting this way of viewing how the CPC encoding constructs a stabilizer across the state, a ZX calculation gives a further insight into how the stabilizer is formed. To construct a stabilizer, we re-write from a known stabilizer at the start of the diagram. Before encoding, the parity qubit is initialised in the $|0\rangle$, which is represented in the ZX as a red spider with a single output wire. Hence, a Pauli Z operation (a green π in ZX notation) on the parity qubit does nothing to the unencoded state. Hence, we can we compute a stabiliser of the encoded state by graphically 'pushing' the green π through the encoder:



In doing so, we have translated the un-encoded stabiliser Z_P to the encoded stabiliser $U_{\text{encode}}Z_PU^{\dagger}_{\text{encode}} = Z_AZ_BZ_P$.

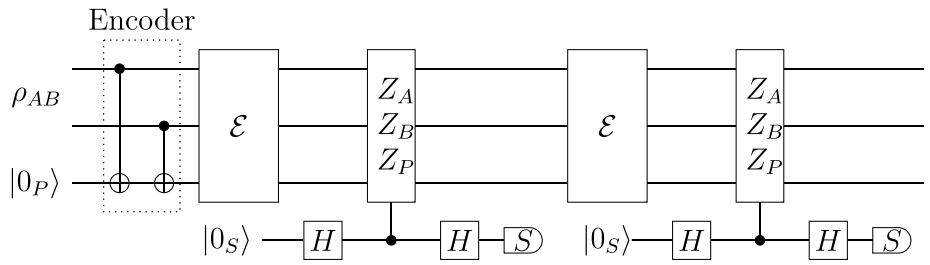


Figure 5. The CPC gadget with continuous error protection using stabilizer measurements.

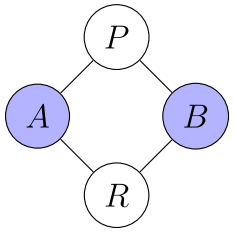


Figure 6. Combining bit and phase checks for two data qubits, A and B, using bit-parity checking qubit P and phase-parity checking qubit R.

We will make use of this later as the general method for computing stabilizers for CPC codes, starting from the known stabilizers of the parity-check qubits.

As presented so far, the CPC gadget has been described in terms of an *encode-error-decode* structure. Whilst this approach is good for demonstrating the fundamental operation of the CPC framework, the disadvantage is that there are gaps in protection during the encode and decode stages of the cycle. We can use the understanding of the CPC as constructing stabilizers to switch instead to a situation standard in QEC: qubits A, B and P remain continuously encoded, and a separate syndrome qubit S is brought in to measure the stabilizer $Z_AZ_BZ_P$, figure 5. An auxiliary qubit S is introduced to extract the stabilizer value before being measured out to yield a syndrome. This auxiliary qubit could be recycled after each cycle allowing the stabilizer to be measured repeatedly with constant overhead. Formulating CPC codes in this way allows for continuous protection at all points in the circuit following the initial encode stage.

It is worth noting, though, that encode-error-decode codings should not be ruled out of consideration when determining the correct way to implement codes on small- or medium- scale machines. On some devices the error rate may be low enough, and the gate speed high enough, that encoding, decoding, and then re-encoding could be good enough to gain an appreciable degree of error mitigation. For small codes and devices, the reduction in the number of qubits required may well be worth it in some situations.

2.3. Combining bit and phase checks: the [[4,2,2]] code

We have seen so far how a single CPC between two data qubits and a parity qubit works to detect one type of error (either a bit or a phase error). A fully quantum code needs to be able to deal with both types of error, and so we need now a method of combining both types of parity check into a single code. By doing this we see that we need another type of check in order to form full quantum codes. The addition of this 'cross-check' between parity-check qubits then leads us to a structural definition of a CPC quantum code that we will use for the remainder of the paper.

The most obvious first method of creating a combined bit and phase check on a pair of data qubits is to simply double the number of parity-check qubits, and use one to check for bit errors and one to check for phase errors (a Pauli Y error would be considered as one X and one Z occurring simultaneously). Figure 6 shows this configuration. The circuit for this initial attempt is given in figure 7.

We can now analyse whether this is, in fact, a working quantum code. The stabilizers for this set-up are:

$$\begin{aligned} &Z_AZ_BZ_P \\ &X_AX_BX_R. \end{aligned} \tag{34}$$

We see that they commute; the minimal requirement. Let us now look at their error detection properties. Consider the ZX-diagram for the set-up of figure 8

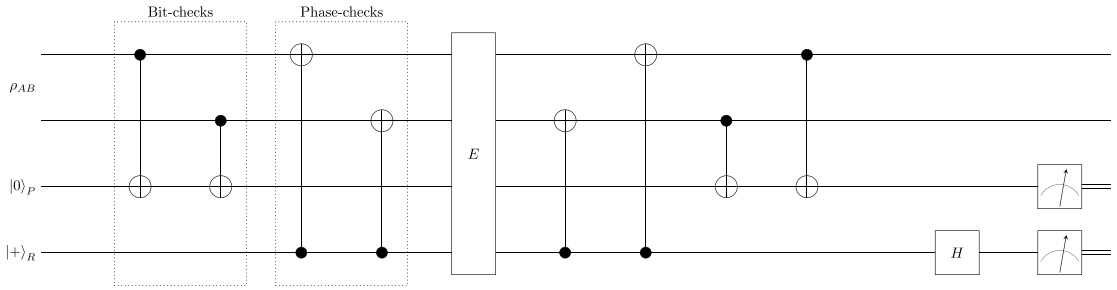


Figure 7. Circuit for a single bit and single phase check on two data qubits as in figure 6.

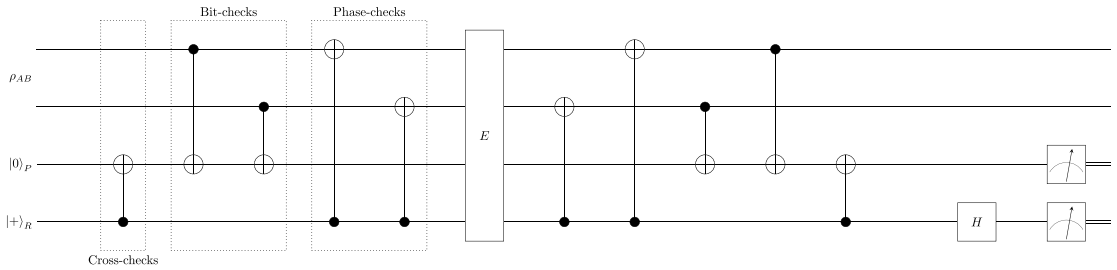
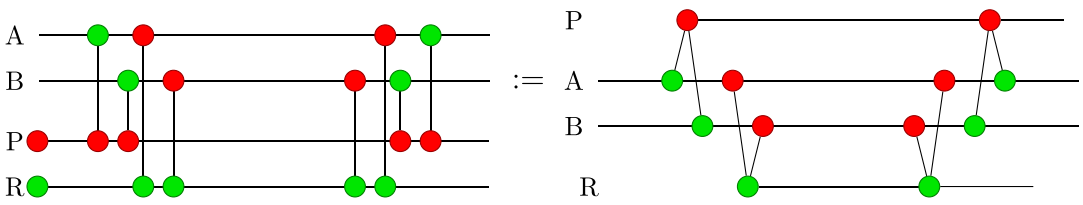
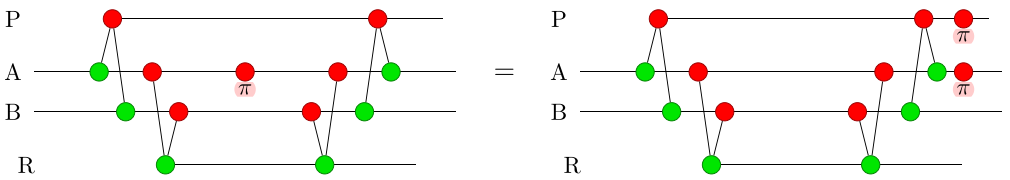


Figure 8. Circuit for the [[4,2,2]] error detection code (41); as figure 8, with the addition of cross-checks between bit- and phase-parity checking qubits.

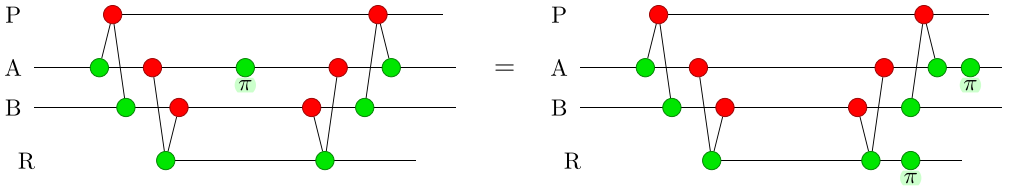


We can see how a bit error on one of the data qubits will be detected (the error propagation is the same on both data qubits):



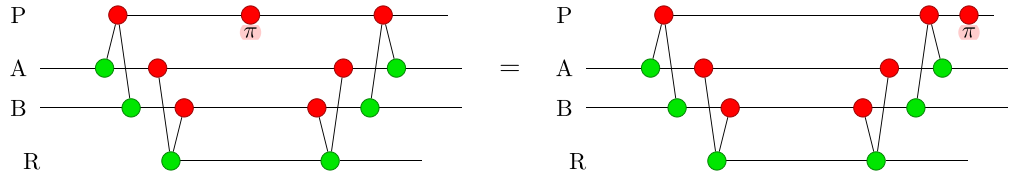
(35)

As before, if P is now measured it will be found in the -1 eigenstate. Similarly, a phase error on a data qubit will be detected at R:



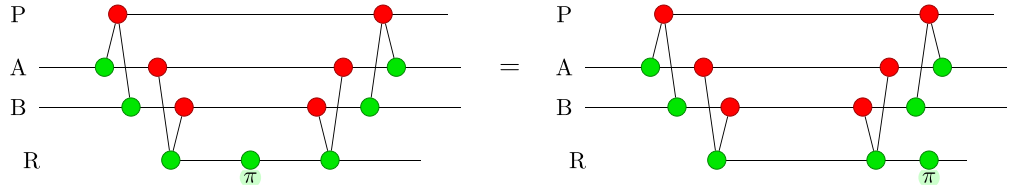
(36)

A bit error on P itself is straightforward, as it will not propagate and is detectable by any subsequent measurement of P:



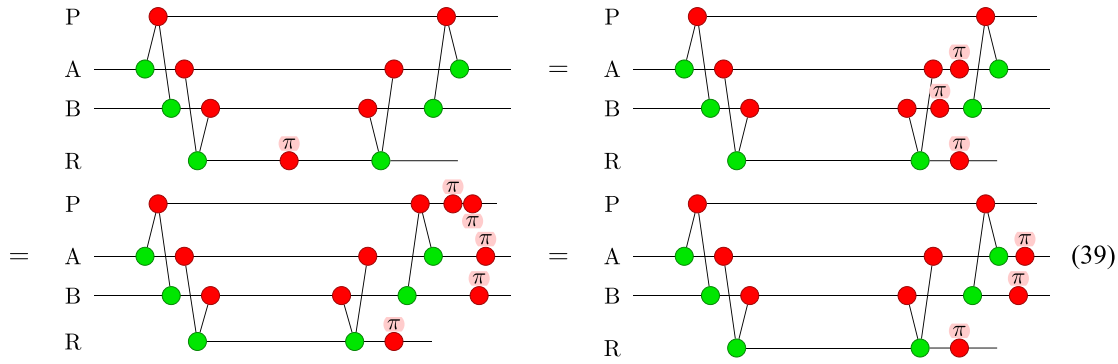
(37)

Similarly, a phase error on R does not propagate and is fully detectable:



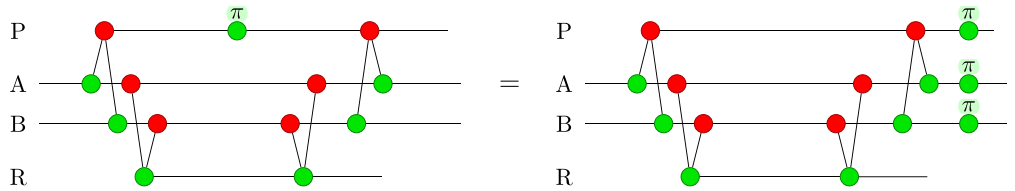
(38)

Most errors will be caught by this pair of parity checks, then—but the final two will not. Firstly, an X error on R is not picked up on P as the multiple checks cancel out:



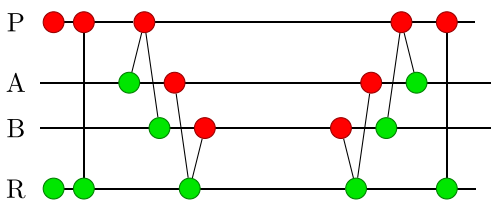
(39)

Secondly, a Z error on P will not propagate to R because of the timings of the gates:



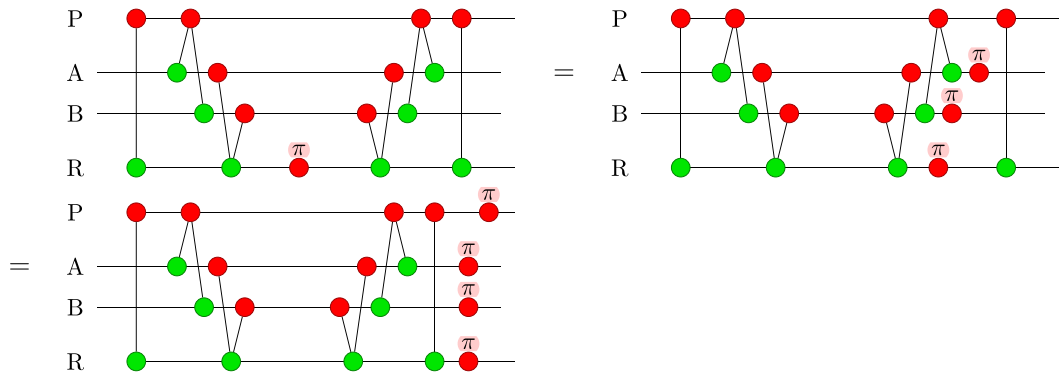
(40)

It is clear that reversing the order of the parity checks will only swap, not eliminate, which errors are undetected. Instead, we add a **cross-check** between the parity-check qubits themselves that is specifically designed to catch these errors:



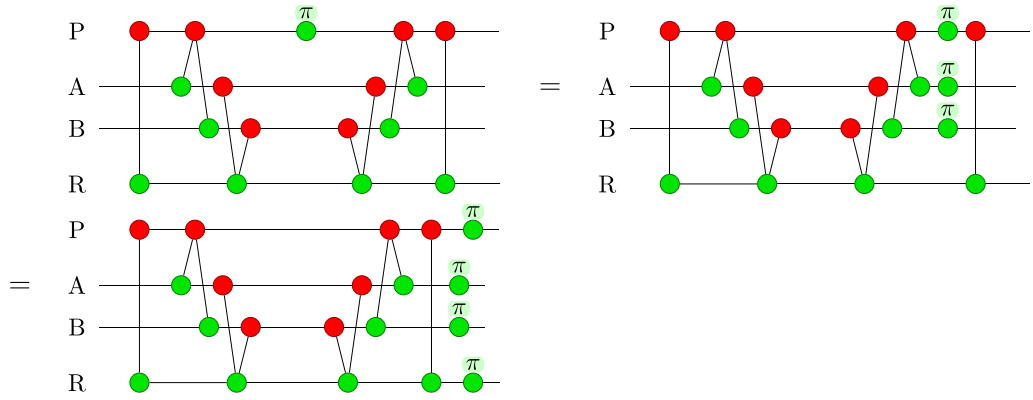
(41)

We can see straight away that this will not affect any of the detections given by (35)–(38), as we would want. The cross-check also enables the previously undetectable errors to propagate so they are detected. Firstly, the error of (39) is now detectable on P:



(42)

Similarly, the error of (40) is now detectable on R:



(43)

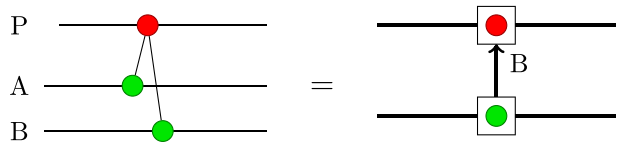
This is now what we want in a code: the ability to detect (although not yet to pin-point and so correct) either type of error on any of the constituent qubits. The corresponding circuit for this code is given in figure 8, and the full set of stabilizers is now

$$\begin{aligned} &Z_A Z_B Z_P Z_R \\ &X_A X_B X_P X_R. \end{aligned} \tag{44}$$

We have therefore found something interesting: what we have constructed here using the CPC procedure is the [[4,2,2]] error detection code [48, 49]. We have used the basic process of bit-parity, phase-parity, and cross- checking to produce a verifiably correct stabilizer code. We will see as we go on that the space of such 'CPC codes' includes many already-known stabilizer codes, as here, as well as enabling us to find many more that are not. However, even this first small example contains all the important structural elements of codes constructed from CPCs, which we define in the next section.

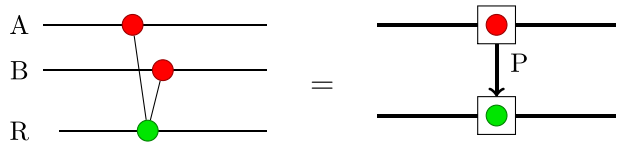
2.4. Defining CPC codes

Within the example of the [[4,2,2]] detection code of the previous section, we can identify the main elements of a CPC-constructed code. Let us now write the encoder (41) using the scalable ZX notation (recalling that the decoder is the time-reverse of the encoder). The bit-parity check is written



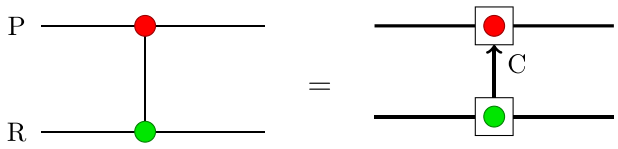
where $B = (1, 1)$ (45)

where this is the special case on the top rail of $n = 1$ (number of rails). Similarly, the phase-parity check becomes



where $P = (1, 1)$ (46)

We can also represent the cross-checks in the scalable notation:

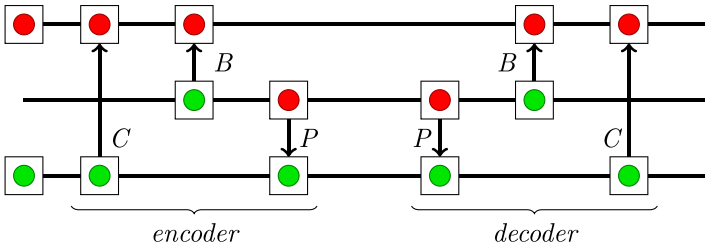


where $C = 1$ (47)

(where the adjacency matrix in this particular example is 1×1, i.e. a scalar).

We can see from this representation that the information about the bit-parity checks is contained in the adjacency matrix B. The phase-parity checks are given by P, and C determines the cross-checks. For different matrices, we have different CPC codes (of course not all matrices will give good or valid codes). This motivates our the definition of the codes we characterise in this paper:

Definition 1. A **CPC code** comprises a set of qubits $\mathcal{Q}$ divided into three subsets $\mathcal{D}, \mathcal{B}, \mathcal{P}$, where the $\mathcal{D}$ are known as *data qubits*, the $\mathcal{B}$ as *bit-parity checking qubits*, and the $\mathcal{P}$ as *phase-parity checking qubits*. The interactions between qubits are given by a triple of binary adjacency matrices B, P, C that determine the bit-parity check, phase-parity check, and cross-checks respectively. The relevant quantum operations, namely the encoder and decoder circuits, are defined as follows:



(48)

In general, B and P will individually be valid classical codes; the 'quantum' addition is the cross-checks given by C, which enable us to combine many different such sets of classical codes without requiring them to satisfy e.g. the duality condition of CSS codes [15]. In subsequent sections we will give conditions for constructing C in general, as well as in specific instances, and investigate the scope and nature of the CPC codes we can construct with this framework.

3. Building a single-error correcting CPC code

We now turn to the problem of producing larger codes based on the CPC. We saw in the previous section how a single bit-parity check can be combined with a single phase-parity check, with cross checks. We now look at constructing larger codes out of multiple bit- and phase- parity checks. Again we use the simplified error model where gates are error-free, and errors occur on all qubits with equal probability in the time between operations.

In this section we demonstrate the principles of how to produce larger CPC codes that can not just detect errors, but also correct them. We construct a 'ring' code from three copies each of bit- and phase- CPC gadgets operating on three data qubits. As in the case of the [[4,2,2]] code, we see that undetectable errors can occur in the absence of cross-checks between the bit- and phase- parity check qubits. We give the construction of the cross-check matrix that solves these issues, and prove its validity using methods that will be extended to more general codes in subsequent sections. The resultant ring code requires two extra parity check qubits, giving a [[11,3,3]] code whose performance we test numerically under a specific error model.

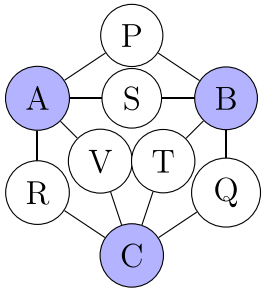
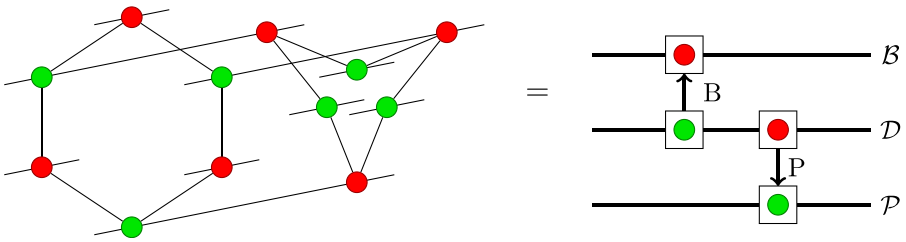


Figure 9. Layout of the ring code (without cross-checks): data qubits A, B, C are checked by three pairs of coherent parity checking qubits, P, Q, R and S, T, V.

3.1. The ring code

Straightforward counting arguments show that the interactions of the [[4,2,2]] code of the previous section cannot be simply changed to give a code that enables errors to be corrected as well as detected: two parity-check qubits give $2^2 = 4$ distinct syndromes, which is not enough to locate bit- and phase-errors on four qubits. We therefore consider a simple extension: three pairs of bit- and phase- parity checks acting on three data qubits, as in figure 9. Six parity-check qubits give $2^6 = 64$ distinct syndromes, which should be more than enough to pinpoint errors on the nine physical qubits.

The encoder made up just of bit- and phase- parity checking, without cross-checks yet, is given in ZX terms as



(49)

where

$$B = P = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}. \tag{50}$$

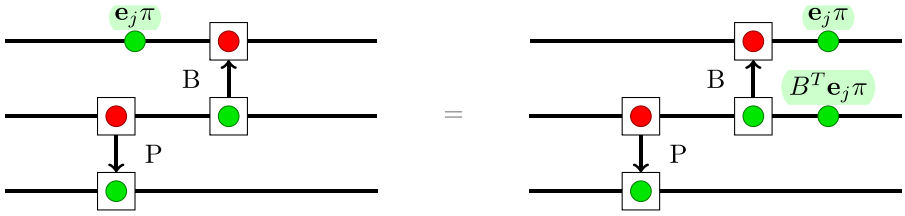
We can also give a full adjacency matrix M for the complete code, acting on data, bit, and phase qubits $(\mathcal{D}|\mathcal{B}|\mathcal{P}) = (A, B, C|P, Q, R|S, T, V)$ respectively. This can be thought of as defining a graph with qubits as vertices and gates as edges. A 0 in the adjacency matrix denotes no edge (so no gate between corresponding qubits), and a 1 denotes an edge and hence a gate (the exact type of gate depending on whether the qubits are in $\mathcal{D}, \mathcal{B}, or \mathcal{P}$):

$$M = \left(\begin{array}{c|c|c} 0 & B^T & P^T \\ \hline B & 0 & 0 \\ \hline P & 0 & 0 \end{array}\right). \tag{51}$$

3.2. Undetected errors in the ring code without cross-checking

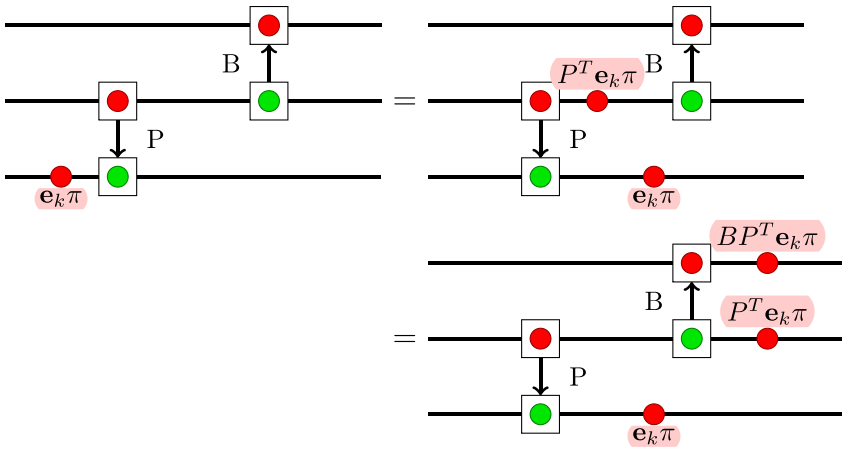
We can completely characterise the error propagation through this proposed code. In doing so we will see again two scenarios where the errors cause the code to fail, this time in the scalable formalism. Errors are represented by a unit vector. We represent data qubits with the subscript i, bit-parity check qubits with j, and phase-parity check qubits with k. The errors will propagate through the scalable representation of the decoder using the rules given in section 1.4, equation (19).

The first problematic case is that of a phase error on a bit-parity check qubit as it comes into the decoder. The error does not propagate at all to the phase-parity check, and is therefore undetectable:



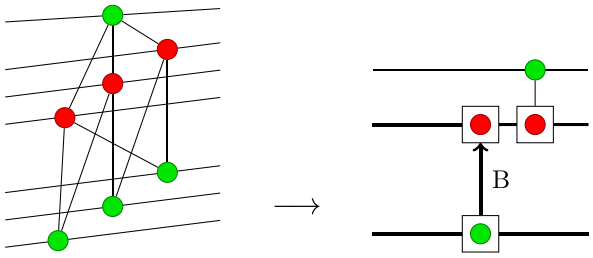
(52)

The second problem is that of a bit error on a phase parity-check qubit, where the error propagates to more than one data qubit, causing the code to fail to identify the error properly as B (a known classical code) can only deal with a single error on the data qubits:



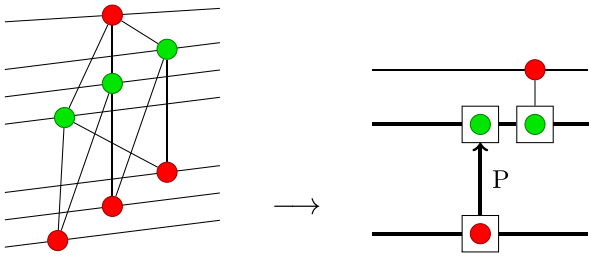
(53)

We now start to add the cross-checks that will make these errors both detectable and correctable. Unlike in the case of the [[4,2,2]] code, we split out the cross-checking into two elements here, to make it clearer what is happening. Firstly, we add overall-parity checking qubits for each of the bit- and phase- checks. This will us to tell whether an error originates from the parity check qubits or not. We have



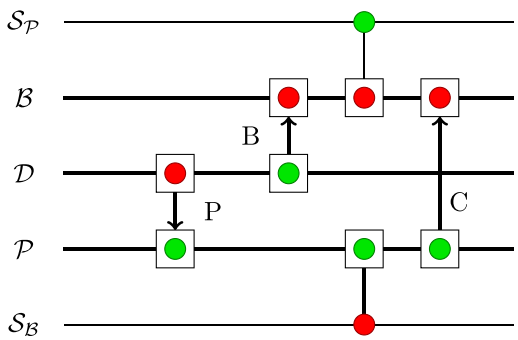
$$B = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \tag{54}$$

and



$$P = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \tag{55}$$

We now add a direct cross-check between the bit- and phase- parity qubits—that is, an addition to the adjacency matrix without additional qubits. There is no guarantee at this point that such a cross-check exists that will make the code work; we investigate this below. Putting both elements together, we have for the decoder (the encoder will be the time-reverse):



(56)

Note that this still has the same form as in definition 1; we have simply chosen to draw the bit-parity and phase-parity check qubits in two pieces ($\mathcal{B}\cup\{\mathcal{S}_\mathcal{B}\}$ and $\mathcal{P}\cup\{\mathcal{S}_\mathcal{P}\}$, respectively).

3.3. Finding the cross-check matrix C

We now show how to construct the cross check matrix for the ring code. In the next section we show that this argument in fact generalises for a large set of codes of distance 3. Throughout this section, we restrict to the case where the number of phase check qubits = the number of bit check qubits. Furthermore, the cross-checks are taken as being performed after the other operations of the code.

Let the set of data qubits be $\mathcal{D}=\{\mathcal{D}_i\}$, that of phase-parity check qubits $\mathcal{P}$, and bit-parity check qubits $\mathcal{B}$. Furthermore let the overall phase check qubit, (54), be $\mathcal{S}_P$ and the overall bit check qubit, (55), $\mathcal{S}_B$.

The full adjacency matrix for the code is

$$\begin{array}{c} \\ \mathcal{D} \\ \mathcal{B} \\ \mathcal{P} \\ \mathcal{S}_B \\ \mathcal{S}_P \end{array} \begin{array}{c} \begin{array}{ccccc} \mathcal{D} & \mathcal{B} & \mathcal{P} & \mathcal{S}_B & \mathcal{S}_P \end{array} \\ \begin{pmatrix} 0 & B^T & P^T & 0 & 0 \\ B & 0 & C^T & 1 & 0 \\ P & C & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}. \tag{57}$$

In the ring, $\mathcal{D}=\{A,B,C\}$, $\mathcal{B}=\{P,Q,R\}$, and $\mathcal{P}=\{S,T,V\}$. We now prove the following:

Theorem 2. *For the full ring given by* (56) *and* (57)*, with* $P=B$ *as in* (54) *and* (55)*, then the addition of cross checks given by the matrix C gives an error correction code of distance* $d=3$*, where C is the permutation matrix with no fixed point*

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}. \tag{58}$$

Proof. To prove this we look at the function of the cross-check matrix C. It will enable the $\mathcal{B}_i$ to check the $\mathcal{P}_k$ for bit errors, and vice versa. The action must be two-fold: firstly it must pick up errors directly on the check qubits, as in (53), and secondly it must pick up any errors that have propagated from parity qubits to bit qubits and then back to parity qubits, as in (52).

We take each set of qubits in turn, and show that single errors in each group give a signature of measurements that differs from those of the previous groups.

Data qubits $\mathcal{D}$. A bit error on a $\mathcal{D}_j$ is detected on the $\mathcal{B}_i$, as B is a valid classical code by construction. Similarly, a phase error on a $\mathcal{D}_j$ is located by the P_k as P is a valid classical code by construction.

Overall parity check qubits $\mathcal{S}_B,\mathcal{S}_P$. A bit error on $\mathcal{S}_B$ will cause a measurement of the -1 eigenstate on $\mathcal{S}_B$ itself. All errors on data qubits cause pairs of -1 measurements, therefore this signature is unique. By symmetry, a phase error on $\mathcal{S}_P$ will give a unique -1 measurements signature on $\mathcal{S}_P$.

A phase error on $\mathcal{S}_B$ will propagate to all the $\mathcal{P}_k$, where it will cause them all to give the -1 eigenstate measurement. As there are more than two $\mathcal{P}_k$, this will be a different signature from other errors considered previously, which give signatures of either single or pairs of -1 measurement outcomes. By symmetry, a bit error on $\mathcal{S}_P$ will give a unique signature of -1 measurement eigenstateoutcomes on all the $\mathcal{B}_i$.

Parity check qubits $\mathcal{B}$ and $\mathcal{P}$. A bit error on a B_i will give a -1 eigenstate outcome for measurements of that qubit. The only signatures previously considered that have a single -1 outcome are measured on $\mathcal{S}_B$ and $\mathcal{S}_P$,

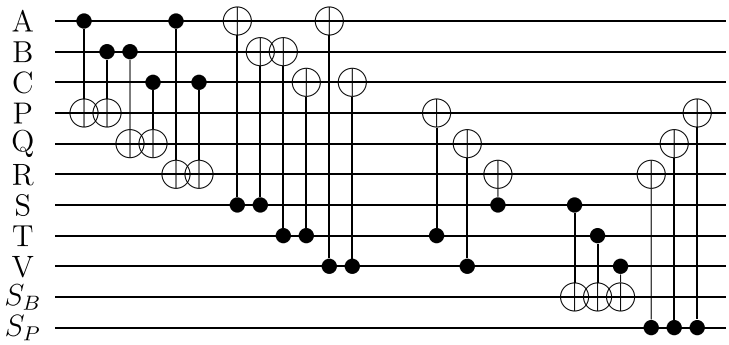


Figure 10. Circuit representation of encoder for the [[11,3,3]] ring code given by (57) and (56). The three groups of circuits represent (B, P), the cross-checks C, and the use of the overall parity check qubits $\mathcal{S}_B$, $\mathcal{S}_P$.

neither of which are in $\mathcal{B}$. Therefore this is a unique signature. By symmetry, a phase error on a $\mathcal{P}_k$ will also give a unique signature of a single -1 eigenstate measurement of itself.

The final cases to consider are those that the original ring code failed under, (52) and (53).

Taking the case of (52) first, a phase error on the jth bit-parity check qubit will now propagate to $\mathcal{S}_P$, and also to the $\mathcal{P}$ as $C^T\mathbf{e}_j$. With C as given, this will then give a signature of a single -1 outcome on $\mathcal{S}_P$, and a single -1 outcome on a $\mathcal{P}_k$ that is unique for each j. No previously-considered error gives this signature; it is unique.

For the case of (53), a bit error on the kth phase-parity check qubit will both propagate to $\mathcal{S}_B$, and also transform as $BP^T \oplus C$ onto the phase-parity check qubits, where '$\oplus$' stands for addition modulo 2 (two errors on the same qubit cancel out). In the case of the ring,

$$BP^T = BB^T = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = 1 \oplus \mathbb{1} \tag{59}$$

where $\mathbf{1}$ is the matrix of all 1 s. With C as given by (58), we therefore have

$$BP^T \oplus C = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{60}$$

That is, a bit error on the kth phase-parity check qubit gives a single -1 outcome on a bit-parity check qubit that is unique for each k, and a -1 outcome on $\mathcal{S}_B$. No other type of error previously considered gives this type of signature. It is therefore a unique signature.

Remark. Note that the situation of (53) by itself only needs the addition of $\mathcal{S}_B$ to produce unique signatures. The addition of C is required to solve the situation of (52). While the matrix $C = \mathbb{1}$ is sufficient for the situation of (52), when then added into the case of (53) this matrix transforms the errors as $BP^T \oplus C = 1 \oplus \mathbb{1} \oplus \mathbb{1} = 1$, which produces non-unique syndromes for error on different qubits. Hence the requirement for $C = M_p$ to satisfy both scenarios.

There are no other cases to consider so this concludes the proof as all single errors of both types are detectable and give rise to unique measurement signatures. □

For completeness, we give an example of a full circuit corresponding to this set of cross-checks in figure 10.

3.4. Numerical test of the [[11,3,3]] ring code

We finish this section by demonstrating the [[11,3,3]] ring code in use in a numerical simulation, with a naïve error model. To do this we choose bit-flip and phase error rates for an existing ion trap system (see [50] and related work), $\epsilon_{\text{bit}} = 0.007\,\text{s}^{-1}$ and $\epsilon_{\text{phase}} = 0.0007\,\text{s}^{-1}$. We assume that errors only occur in the encoded region, and in particular, that no errors are introduced by encoding and decoding.

We consider the protection of a random three qubit state, drawn from a distribution which obeys the Haar measure. We model the code as performing encoding and decoding with a rate r such that the circuit depicted in figure 10 is applied $\frac{1}{r}$ times a second. We assume that all gates are fast and therefore errors can only occur within the window E, and we assume that all gates are perfect. Since the effective error rate which each instance of the code sees in this setup is inversely proportional to r, a code which is able to correct single

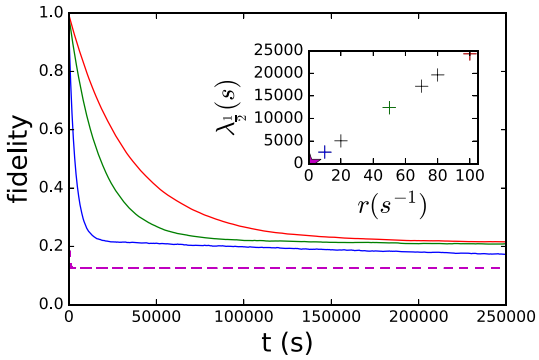


Figure 11. Numerical results for the $[[11,3,3]]$ CPC code using a bit-flip error rate of $\epsilon_{\text{bit}} = 0.007\,\text{s}^{-1}$ and a phase error rate $\epsilon_{\text{phase}} = 0.0007\,\text{s}^{-1}$, sampled over random states drawn from the Haar measure. Shown are the same fidelities for $r = 10\,\text{s}^{-1}$ (blue), $r = 50\,\text{s}^{-1}$ (green), and $r = 100\,\text{s}^{-1}$ (red). The unprotected fidelity appears as a dashed magenta line. Inset: numerical fit of the half life of the fidelity, $\lambda_{\frac{1}{2}}$ versus cycle rate r for the three lines in the main figure plus others. The unprotected ($r = 0$) value of $\lambda_{\frac{1}{2}}$ is represented as a magenta star. Error-bars due to statistical fluctuations are smaller than the depicted lines.

errors will lead to an error rate per cycle of $\frac{1}{r^2}$. The expected lifetime of a state should then be this error rate divided by the cycle rate r, implying that in this simple model a code which corrects single errors should yield state lifetimes which are proportional to r. We measure state lifetimes by extracting a half-life of the fidelity $\lambda_{\frac{1}{2}}$ by numerically fitting fidelity data with an exponential decay model.

Figure 11 presents numerical results for the $[[11,3,3]]$ code. The lifetimes are able to be extended well beyond the limitation of the unprotected lifetime of a single qubit due to bit-flip errors ($\frac{1}{\epsilon_{\text{bit}}} \approx 142\,\text{s}$) and even well beyond those due to the less probable phase errors ($\frac{1}{\epsilon_{\text{phase}}} \approx 1420\,\text{s}$). Moreover, the lifetime scales linearly with r, confirming that the codes are able to correct arbitrary single qubit errors.

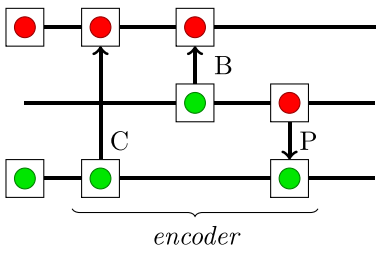
4. Tripartite CPC codes

While we considered in the previous section distance 3 codes in detail, the structure of a CPC given in (56) is general for greater distances: the qubits are divided into data, bit-parity check, phase-parity check, and overall parity checks. This structure enables us to use the associated *ZX* toolkit to build, verify, and analyse new codes. We will also see in later section how this structure enables us to automate a search for codes that returns large numbers of codes that can then be subject to further optimisation based on required characteristics. We also note that the CPC formalism can be easily generalised to use whatever entangling gate a device implements natively and gain a significant improvement in efficiency [43].

In this section, we demonstrate that this framework and graphical toolkit works not only for codes that are encoded and decoded at every cycle, but is also capable of constructing codes in the standard model of QEC. We look first at the dual roles of logical operators and stabilizers, and of error propagation. We then demonstrate how CPC codes using an encode-decode framework correspond to the standard code method of measuring stabilizers. We end the section by showing how CSS codes thereby are shown to be part of the set of CPC codes.

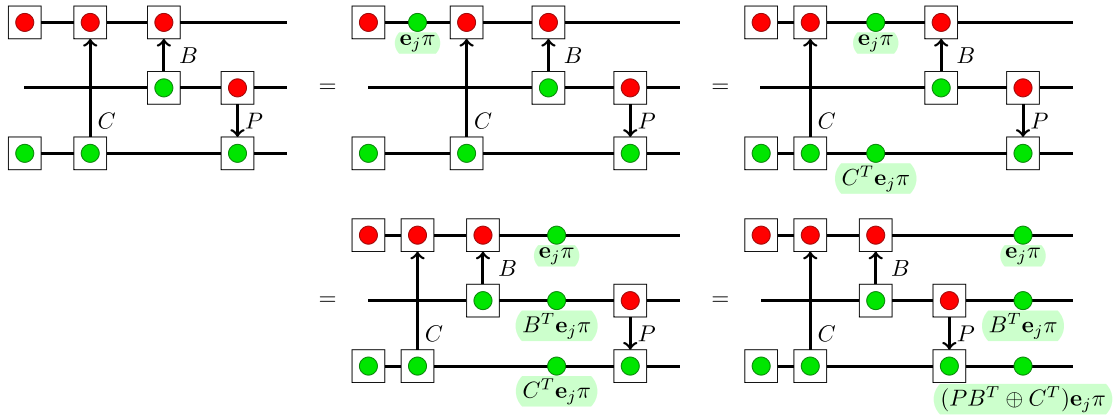
4.1. Stabilizers and logical operators

CPC codes are, in particular, stabilizer codes. We can compute the associated stabilizers by looking at the first part of the map in definition 1, consisting of the initialisation of the parity check qubits and the encoder unitary:



(61)

For $\mathcal{D} = \{\mathcal{D}_i\}_{1\leqslant i\leqslant k}$ data qubits and $\mathcal{B} = \{\mathcal{B}_i\}_{1\leqslant i\leqslant b}$ bit parity qubits and $\mathcal{P} = \{\mathcal{P}_i\}_{1\leqslant i\leqslant p}$ phase parity qubits, the map above is an isometry which embeds k-qubit space as a stabilizer subspace of n-qubit space, where $n := k+b+p$. We can compute the $b+p$ independent stabilizers for this subspace by pushing stabilizers from the (unencoded) parity qubits forward across the encoder. More concretely, since $Z|0\rangle = |0\rangle$, we have:

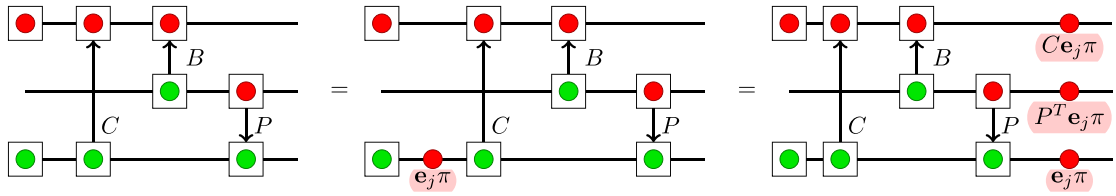


Hence, it follows that anything in the image of the embedding (61) is a '+1' eigenstate of:

$$\mathcal{Z}_j := Z_{\mathcal{B}_j} \cdot \prod_{(B^T)_{ij}=1} Z_{\mathcal{D}_i} \cdot \prod_{(PB^T\oplus C^T)_{ij}=1} Z_{\mathcal{P}_i}. \tag{62}$$

For E the encoder, we have shown that $EZ_{\mathcal{B}_j} = \mathcal{Z}_j E$. Since the maps $Z_{\mathcal{B}_j}$ are independent and E is a unitary, the maps $\mathcal{Z}_j$ give us the first b generators for the stabilizer group.

Similarly, $X|+\rangle = |+\rangle$, so:



Hence any state in the image of (61) is also a '+1' eigenstate of:

$$\mathcal{X}_j := \prod_{C_{ij}=1} X_{\mathcal{B}_i} \cdot \prod_{(P^T)_{ij}=1} X_{\mathcal{D}_i} \cdot X_{\mathcal{P}_j}. \tag{63}$$

Again these are independent because $EX_{\mathcal{P}_j} = \mathcal{X}_j E$, so this gives us the remaining p stabilizers of the code.

The $2k$ logical operators for the code are computed similarly, by placing a Pauli X or Z on the jth data qubit $\mathcal{D}_j$ and pushing it through the encoder. They are given by:

$$\widehat{Z}_j = Z_{\mathcal{D}_j} \cdot \prod_{P_{ij}=1} Z_{\mathcal{P}_i}$$

$$\widehat{X}_j = \prod_{B_{ij}=1} X_{\mathcal{B}_i} \cdot X_{\mathcal{D}_j}$$

where $EZ_{\mathcal{D}_j} = \widehat{Z}_j E$ and $EX_{\mathcal{D}_j} = \widehat{X}_j E$. Taking the adjoint yields $Z_{\mathcal{D}_j}E^\dagger = E^\dagger \widehat{Z}_j$ and $X_{\mathcal{D}_j}E^\dagger = E^\dagger \widehat{X}_j$, and noting that $E^\dagger$ is the decoder, we can conclude that, as expected, measuring the jth logical qubit after decoding is equivalent to measuring the associated logical operator before decoding.

We can prove a similar result connecting the CPC measurements as described in section 2.1 to syndrome measurements.

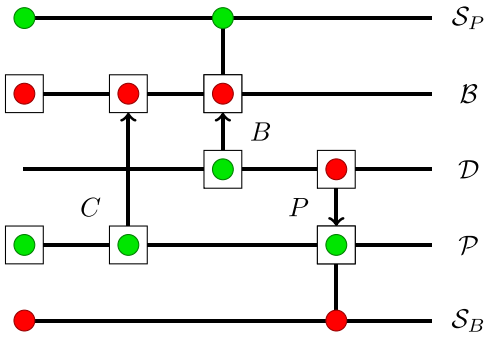
Theorem 3. *For a tripartite CPC code with associated stabilizers $\{\mathcal{Z}_j\} \cup \{\mathcal{X}_j\}$, an element $\mathcal{E}$ of the Pauli group which occurs before decoding will yield a '−1' outcome on the jth bit-parity (respectively phase-parity) qubit if and only if $\mathcal{E}$ anti-commutes with $\mathcal{Z}_j$ (respectively $\mathcal{X}_j$).*

Proof. Let $E^\dagger$ be the decoder and let $\mathcal{E}'$ be the decoded error coming from $\mathcal{E}$. That is, $\mathcal{E}' := E^\dagger \mathcal{E} E$. Then, following the calculations in section 2.1, we will observe a '-1' outcome of the jth bit parity qubit $\mathcal{B}_j$ if and only if $\mathcal{E}'$ has X-support at $\mathcal{B}_j$. In that case, $\mathcal{E}' Z_{\mathcal{B}_j} = -Z_{\mathcal{B}_j}\mathcal{E}'$. Applying $(E\ldots E^\dagger)$ to both sides, we see that: $\mathcal{E}\mathcal{Z}_j = -\mathcal{Z}_j\mathcal{E}$. Conversely, if $\mathcal{E}$ and $\mathcal{Z}_j$ anti-commute, then so do $\mathcal{E}'$ and $Z_{\mathcal{B}_j}$. Since $\mathcal{E}'$ is an element of the Pauli group, it must have X-support at $\mathcal{B}_j$.

The argument is identical for $\mathcal{X}_j$ stabilizers, interchanging the role of X and Z. □

Hence syndrome measurements give us the exact same information as decoding, measuring parity qubits, and re-encoding, while the data qubits remain continuously protected.

Note that for the ring code from the previous section, and the more general constructions of distance 3 codes in section 4.3, we consider tripartite CPC codes where one of bit- and phase-parity check qubits plays a special role. Namely, one serves as a check $\mathcal{S}_B$ for bit errors on all the phase-parity check qubits and the other as a check $\mathcal{S}_P$ for phase errors on all of the bit-parity check qubits. Its encoder is therefore:



As this is a special case of a tripartite CPC code, stabilizers are computed just as before. For the sake of completeness, we list them explicitly. There are now $b+1$ Z-stabilizers, given by each of the bit-check qubits in $\mathcal{B}$ and one for the global bit check $\mathcal{S}_B$. Similarly, there are $p+1$ X-stabilizers, given by each of the phase-check qubits in $\mathcal{P}$ plus $\mathcal{S}_P$. So, the full list of stabilizers is:

$$\mathcal{Z}_j := Z_{\mathcal{S}_P} \cdot Z_{\mathcal{B}_j} \cdot \prod_{(B^T)_{ij}=1} Z_{\mathcal{D}_i} \cdot \prod_{(PB^T \oplus C^T)_{ij}=1} Z_{\mathcal{P}_i} \qquad 1 \leqslant j \leqslant b$$

$$\mathcal{Z}_{b+1} := \prod_{1 \leqslant i \leqslant b} Z_{\mathcal{P}_i} \cdot Z_{\mathcal{S}_B}$$

$$\mathcal{X}_j := \prod_{C_{ij}=1} X_{\mathcal{B}_i} \cdot \prod_{(P^T)_{ij}=1} X_{\mathcal{D}_i} \cdot X_{\mathcal{P}_j} \qquad 1 \leqslant j \leqslant p$$

$$\mathcal{X}_{p+1} := X_{\mathcal{S}_P} \cdot \prod_{1 \leqslant i \leqslant p} X_{\mathcal{B}_i}.$$

4.2. CPC construction for CSS codes

In this section, we will show that the family of tripartite CPC codes is equivalent to the family of CSS codes. Hence, another way to see tripartite CPC codes is as a CSS code with some extra underlying structure, namely the three parity check matrices. Equivalently, they provide a new way to search for CSS codes based on (not necessarily self-dual) pairs of classical codes.

In this section, we will adopt symplectic notation for stabilizer codes (see e.g. [51]). Namely, a set of generators for a stabilizer subgroup (up to signs) can be written as a matrix of the form:

$$G = (G_Z | G_X) \tag{64}$$

where G_X and G_Z are binary matrices. The associated stabilizers are then given, up to a global phase, as:

$$S_i := \prod_{(G_Z)_{ij}=1} Z_j \cdot \prod_{(G_X)_{ij}=1} X_j. \tag{65}$$

In this form, standard CSS codes are written

$$G_{\text{CSS}} = \left(\begin{array}{c|c} G_Z & 0 \\ 0 & G_X \end{array}\right). \tag{66}$$

For example, for the Steane [[7,1,3]] code, $G_{X,Z}$ are both the parity check matrix of the classical [7,4,3] Hamming code.

Comparing equation (65) with equations (62) and (63) from the previous section, we see that the stabilizers for a tripartite CPC code can be given by the following block matrix:

$$G_{tri} = \left(\begin{array}{ccc|ccc} \mathbb{1} & B & (BP^T \oplus C) & 0 & 0 & 0 \\ 0 & 0 & 0 & C^T & P & \mathbb{1} \end{array}\right) \tag{67}$$

where $\mathbb{1}$ is the identity matrix.

We can immediately conclude that tripartite codes are indeed CSS codes. For the converse, we start with a generic CSS code in the form of (64). We can always replace a generator of the stabilizer group by the product of itself and another generator, which has the effect of adding one row to another. Using this fact, we can transform (64) into the following standard form, essentially by doing Gaussian elimination [52]:

$$G'_{\mathrm{CSS}} = \left(\begin{array}{ccc|ccc} \mathbb{1} & J & K & 0 & 0 & 0 \\ 0 & 0 & 0 & L & M & \mathbb{1} \end{array}\right). \tag{68}$$

In order for the Z-stabilizers to commute with the X-stabilizers, it must be the case that $G_Z G_X^T = 0$. Hence:

$$\begin{pmatrix} \mathbb{1} & J & K \end{pmatrix} \begin{pmatrix} L^T \\ M^T \\ \mathbb{1} \end{pmatrix} = 0 \qquad \Longrightarrow \qquad L^T \oplus JM^T \oplus K = 0. \tag{69}$$

Hence $K = JM^T \oplus L^T$. Comparing with 67, we define a tripartite CPC code by letting $B := J$, $P := M$, $C := L^T$. It then follows from (69) that $K = BP^T \oplus C$. Hence, any CSS code can be realised as a tripartite CPC code.

4.3. Finding cross check matrices for tripartite codes

The tripartite CPC construction uses pairs of classical codes as B and P bit and phase checks. The 'quantum' part of the construction comes from the cross-checking, given by C. Finding this matrix is then the important core of the construction. We now show how to construct cross-checks for distance 3 codes.

4.3.1. Cross checks for distance 3 codes from Hamming codes

With a little work, we can extend the proof for the cross-check of the ring code, theorem 2, to a general construction for distance 3 quantum codes. Note that this is not the only way to produce $d = 3$ CPC codes (and in subsequent sections we will consider numerical search techniques); however, this construction is guaranteed to produce a valid quantum code, and gives a bound on the resources required. The result is encapsulated in the following theorem:

Theorem 4. *Let L be a classical Hamming code with parameters $[n, k, 3]$ and adjacency matrix A_L, where the adjacency matrix relates to the generator matrix as*

$$G = [\mathbb{1}|A'] = \left[\begin{array}{cc|c} \mathbb{1} & & A_L \\ & \mathbb{1} & 1 \end{array}\right]$$

*where **1** is a row of all 1 s. For all such L, with $k > 2$, a valid quantum code may be constructed using the structure of (56) and (57), with $B = P = A_L^T$ and $C = M_P$, where M_P is the $(n-k) \times (n-k)$ permutation matrix with no fixed point that permutes elements with their successor (where $(n-k)$ is the number of parity check bits in the classical code L). This quantum code has parameters $[[2n-k+1, k-1, 3]]$.*

Proof. We parallel the proof for the ring code, and take each set of qubits in turn, showing that the construction gives unique signatures for each single-qubit error.

We note that the form of the Hamming code given in the definition of A_L above means that $G_S = [\mathbb{1}|A_L]$ is the generator of the 'shortened Hamming code' L_s [53, sections 34–93]. This code is obtained from L by removing the data (qu)bit whose row in the original adjacency matrix A' is all 1 s. Equivalently, this data (qu)bit may be considered as set to a fixed 'zero' state. By construction, the shortened Hamming code is a valid classical code; that is, A_L generates a valid classical code L_s with parameters $[n-1, k-1, 3]$. Note further that if the generator matrix of a code L is $H = [\mathbb{1}|A]$ then the generator of the dual code $L^\perp$ is $G = [A^T|\mathbb{1}]$. The dual code of a Hamming code is a simplex code, which is a valid classical code [53]. $B = P = A_L^T$ therefore denotes B and P as the adjacency matrices of simplex codes $L_s^\perp$. Note further that as A_L by given construction has no row of all 1 s, A_L^T has no column of all 1 s.

Data qubits $\mathcal{D}$. A bit error on a $\mathbf{D}_i$ is detected on the $\mathcal{B}_j$, as $L_s^\perp$ is a valid classical code. Similarly, a phase error on a $\mathbf{D}_i$ is located by the P_k as $L_s^\perp$ is a valid classical code.

Overall parity check qubits $\mathcal{S}_B, \mathcal{S}_P$. A bit error on $\mathcal{S}_B$ will result in the measurement of the '−1' eigenstate on $\mathcal{S}_B$ itself. All errors on data qubits cause pairs of '−1' measurements, therefore this signature is unique. By symmetry, a phase error on $\mathcal{S}_P$ will give a unique '−1' measurement signature on $\mathcal{S}_P$.

A phase error on $\mathcal{S}_B$ will propagate to all the $\mathcal{P}_k$, where it will cause them all to give the measurement outcome corresponding to the '−1' eigenstate. As the adjacency matrix $P = A_L^T$ contains no column of all 1 s, then there is no single error on a data qubit that can give rise to these all '−1' outcomes. This will therefore be a unique signature in this case. By symmetry, a bit error on $\mathcal{S}_P$ will give a unique signature of '−1' eigenstate measurements on all the $\mathcal{B}_j$.

Parity check qubits $\mathcal{B}$ and $\mathcal{P}$. As before, a bit error on a B_j will give a '−1' outcome for measurements of that qubit. The only signatures previously considered that have a single '−1' outcome are measured on $\mathcal{S}_B$ and $\mathcal{S}_P$, neither of which are in $\mathcal{B}$. Therefore this is a unique signature. By symmetry, a phase error on a $\mathcal{P}_k$ will also give a unique signature of a single '−1' measurement of itself.

We now consider the propagation of a phase error on the jth bit-parity check qubit. As in the ring case, in the general case this will propagate both to $\mathcal{S}_P$ directly, and to the $\mathcal{P}$ as $C^T\mathbf{e}_j$. With C as given, this will then give a signature of a single '−1' outcome on $\mathcal{S}_P$, and a single '−1' outcome on a $\mathcal{P}_k$ that is unique for each j. No previously-considered error gives this signature; it is unique.

A bit error on the kth phase-parity check qubit propagates to give a single '−1' measurement on the $\mathcal{S}_B$, and also propagates to the bit-parity check qubits as $BP^T \oplus C$.

We now use the following Lemma whose proof is shown in appendix B:

Lemma 5. *For classical simplex codes with generator* $H = [A_S|1|\mathbb{1}] = [A_L^T|1|\mathbb{1}]$, $A_S A_S^T = 1 \oplus \mathbb{1}$, *where* A_L *gives the generator of the shortened Hamming code, as above.*

Using this Lemma, for a code with $B = P = A_S$, when a bit error on a $\mathcal{P}_k$ propagates to the $\mathcal{B}_j$ under $BP^T \oplus C$, with C as given this is therefore $A_S A_S^T \oplus M_P = 1 \oplus \mathbb{1} \oplus M_P = M$. Now, in order for this to fulfil the correct role in the code we want that a bit error on the kth phase-parity check qubit gives a single '−1' outcome on a bit-parity check qubit that is unique for each i, and a '−1' outcome on $\mathcal{S}_B$. In order to do this, this resultant matrix must have no two rows the same. The matrix M will have two '0' entries in each row (all the rest are '1's). For the ith row, one '0' is always in the ith column (i.e. there are '0's down the diagonal, from $1 \oplus \mathbb{1}$). The second '0' will come from the ith row of M_P, which will not be in the ith column as M_P has no fixed point. We denote its column position as j. Therefore, as long as the pair i, j is not repeated in different rows, the matrix M will have unique rows. This is accomplished (not uniquely) by taking M_P as the matrix that permutes element n with element $n+1$.

No other type of error previously considered gives this type of signature. It is therefore a unique signature.

There are no other cases to consider. Therefore in all cases the construction given takes a classical $[n, k, 3]$ Hamming code and generates a valid distance 3 quantum code from two copies of the dual to the shortened form s.t. $[n-1, k-1, 3]$. The number of data qubits is $k-1$ and the number of parity check qubits is $n-k+2$. Therefore the valid quantum code has parameters $[[2n-k+1, k-1, 3]]$. This concludes the proof. □

We can now see that the [[11,3,3]] ring is a special case of this more general Hamming code construction. The classical code used, with adjacency matrix (50), is the shortened form of the Hamming [7,4,3] code, with the row of all 1 removed from the adjacency matrix. Explicitly, the generator of the dual simplex code (that is the parity check matrix of the [7,4,3] Hamming code) is the list of all 3-digit binary numbers excluding zero:

$$H_3 = \left[\begin{array}{ccccccc} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}\right] = [\mathbb{1}|A_3|1]\,. \tag{70}$$

The adjacency matrix for the CPC code is therefore

$$A_3 = \left[\begin{array}{ccc} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{array}\right] \tag{71}$$

which is the adjacency matrices for the ring code, (50).

4.3.2. Cross check matrices for general distance 3 codes

We now generalise this result further, for arbitrary distance 3 codes. Following the notation from the previous section, we say a classical code has adjacency matrix A if its generators in standard form are $G = [\mathbb{1}|A]$.

We do not give a construction for the cross-check matrix, but prove the following theorem that tells us such a cross-check matrix can always be found, provided the classical codes do not contain a 'global' parity check.

Theorem 6. *For any pair of $d=3$ classical codes L_1 and L_2, with adjacency matrices A_1 and A_2 respectively, there exists a matrix C such that a valid quantum code may be constructed using the structure of* (56) *and* (57)*, with $B=A_1$, $P=A_2$, and C, provided only that:*

(i) *L_1 and L_2 have the same number of parity-check (qu)bits, and*
(ii) *A_1 and A_2 do not contain a row of all 1 s*

Proof. The proof is the same as for theorem 4, except that we are replacing the pair of Hamming codes with the arbitrary distance 3 codes L_1 and L_2. Most elements of the proof go through exactly as before. The requirement that A_1 and A_2 do not contain a row of all 1 s eliminates repeated syndromes in the case of a phase error on $\mathcal{S}_B(\mathcal{S}_P)$, which will propagate to all the $\mathcal{P}_k(\mathcal{B}_j)$. We are left then with the two types of error propagation that depend most clearly on C: a bit error on the kth phase-parity check qubit, and a phase error on the jth bit-parity check qubit.

As before, a phase error on the jth bit-parity check qubit will propagate both to $\mathcal{S}_P$ directly, and to the $\mathcal{P}$ as $C^T\mathbf{e}_j$. We therefore require that C^T is a valid code matrix: that is, that it has unique columns. C must therefore have unique rows.

Also as before, a bit error on the kth phase-parity check qubit propagates to give a single '+1' measurement outcome on the $\mathcal{S}_B$, and also propagates to the bit-parity check qubits as $BP^T \oplus C$. We therefore require $BP^T \oplus C$ to be a valid code; that is, it must have unique columns.

We now use the theorem of linear algebra, that any square matrix M may be written $M=N_1+N_2$ where N_1 and N_2 are invertible [54]. If L_1 and L_2 share the same number of parity (qu)bits, then BP^T will be a square matrix. Therefore restricting to codes L_1 and L_2 with the same number of parity (qu)bits, we have

$$\begin{aligned} BP^T &= N_1 \oplus N_2 \\ BP^T \oplus N_1 &= N_2 \end{aligned} \tag{72}$$

(recalling addition modulo 2). Let $N_1 = C$. We may therefore conclude the following:

(i) $C=N_1$ is invertible. It will therefore have unique rows, as required.
(ii) $BP^T \oplus C = N_2$ is invertible. It will therefore have unique columns, as required.

As N_1 and N_2 are known always to exist, C will always exist with the relevant properties for a full quantum code. This concludes the proof. □

We now prove our final theorem for these general distance 3 codes:

Theorem 7. *For any classical $[n,k,d\geqslant 3]$ codes L and M with adjacency matrices A_L, A_M, a valid quantum code may be constructed using the structure of* (56) *and* (57)*, with $B=A_L$, $P=A_M$, and some C, provided that A_L and A_M do not contain rows of all 1 s. Such a quantum code will have parameters $[[2n-k+2,k,3]]$.*

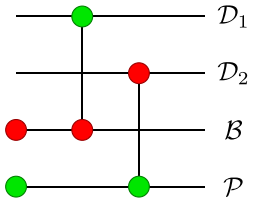
Proof. We use theorem 6, which guarantees the existence of C. Note that any $d\geqslant 3$ code is also a valid code for correcting a single error, so may be used in this construction. To use theorem 6 here we have $A_1=A_L$ and $A_2=A_L$. The structure of (56) and (57) means that the two copies of L give a total of k data qubits, and $2(n-k)+2$ parity check qubits. The total number of qubits is therefore $2(n-k)+2+k=2n-k+2$. The overall distance of the quantum code is 3. This concludes the proof. □

5. Generalised CPC codes and automated search

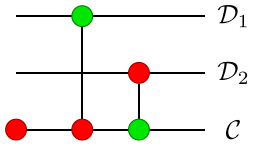
We have considered up to now codes where X and Z errors are detected on separate sets of parity check qubits. This tripartite construction gives rise to codes with separation between stabilizers that are all X or all Z. We now extend the formalism by adding the ability to express general codes, by introducing a combined parity check gadget capable of determining combinations of both types of error.

5.1. Combined parity checking

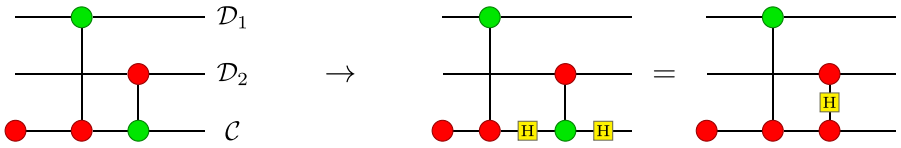
Previously, bit and phase parity checks needed to be done using separate check qubits. For instance, if we want to check for a bit error on qubit $\mathcal{D}_1$ and a phase error on qubit $\mathcal{D}_2$, we could define an encoder which looks like this:



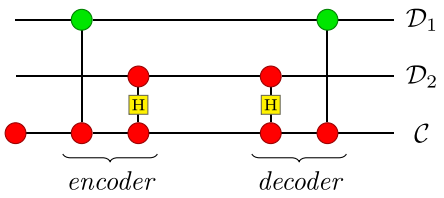
Suppose however, that we wish to store the *parity* of the bit error on $\mathcal{D}_1$ with the phase error on $\mathcal{D}_2$ into a *combined* parity check bit $\mathcal{C}$. Our first guess might be something like this:



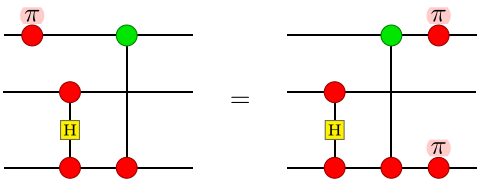
This clearly will not work. A Z error on $\mathcal{D}_2$ will indeed propagate to $\mathcal{C}$, but since this check bit always remains in a Z-eigenstate, we will never detect it. However, we can fix this problem by wrapping the control qubit of the second CNOT gate in Hadamards. Namely:



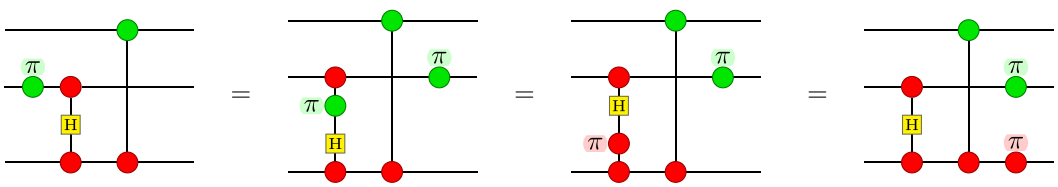
So, the full combined parity check circuit looks like this:



Now, if a bit error occurs on $\mathcal{D}_1$ or a phase error occurs on $\mathcal{D}_2$, the decoder propagates it to a bit error on $\mathcal{C}$. The bit error propagates as before:

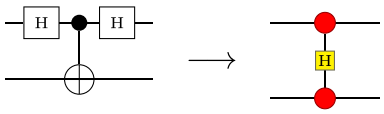


whereas the phase error changes to a bit error when it propagates:



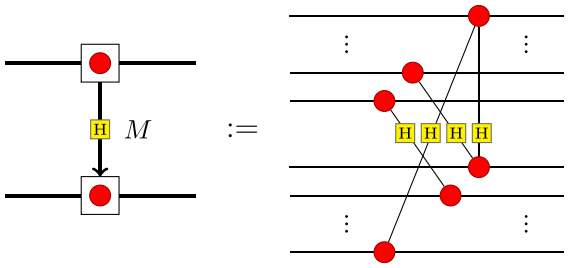
If a bit error occurs on $\mathcal{D}_1$ and a phase error on $\mathcal{D}_2$, the two X operators on $\mathcal{C}$ cancel out. Hence, $\mathcal{C}$ indeed detects the parity of these two errors.

A CNOT with Hadamards on its control qubit is the same as a controlled-Z gate, but written with respect to the X-basis rather than the $\mathcal{Z}$ basis. Since there does not seem to be a standard name for this gate in the literature, we will refer to it as the *conjugate propagator*:



(73)

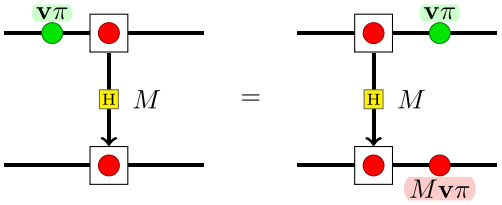
It will be useful to extend the scalable ZX notation introduced in section 1.4 to represent conjugate propagators as well. We do this in the obvious way:



(74)

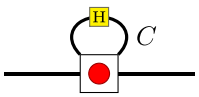
That is, we label a directed edge with an adjacency matrix such that $M_{ij} = 1$ indicates that the jth input qubit is connected by a conjugate propagator to the ith output qubit.

X errors commute with the conjugate propagators, whereas Z errors propagate much like in the CNOT case, except that the Hadamards change the colour of the propagated errors:



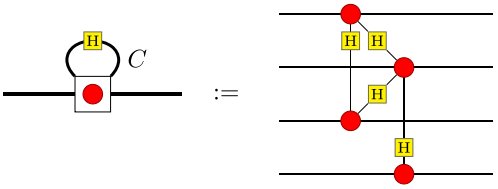
(75)

In order to implement generalised cross-checks, we also allow self-loops labelled by a full adjacency matrix:



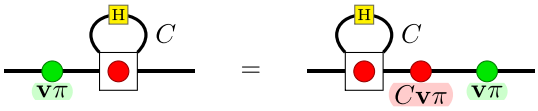
Here C is a symmetric matrix, where $C_{ij} = C_{ji} = 1$ indicates the presence of a conjugate propagator between the ith and jth qubit in the block. For example:

$$C = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad \Longrightarrow$$



(76)

X errors commute with self-loops, whereas Z errors propagate to X errors on adjacent qubits. That is:

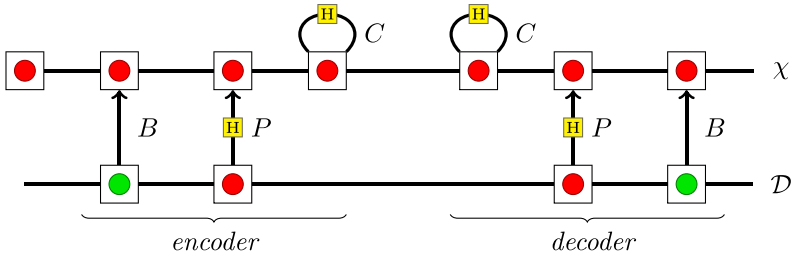


(77)

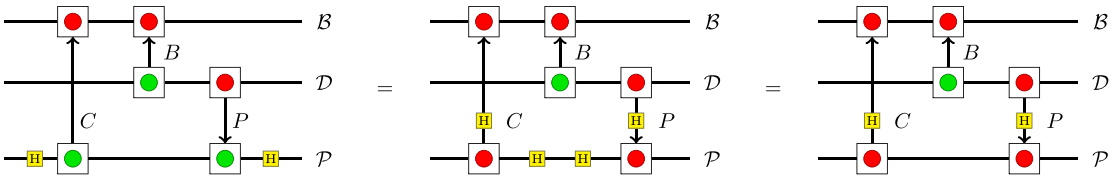
Remark 8. Note that in the graph theory literature, matrices such as M in equation (74) (and in all of the examples prior to this section) are often referred to as *bi-adjacency* matrices, as they give the connectivity between one set of nodes and a separate, disjoint set. On the other hand, symmetric matrices such as C in (76) are the usual notion of adjacency matrix. As it is always clear from context which kind of matrix we mean, we refer to either simply as an adjacency matrix.

Using this notation, we can make the following definition:

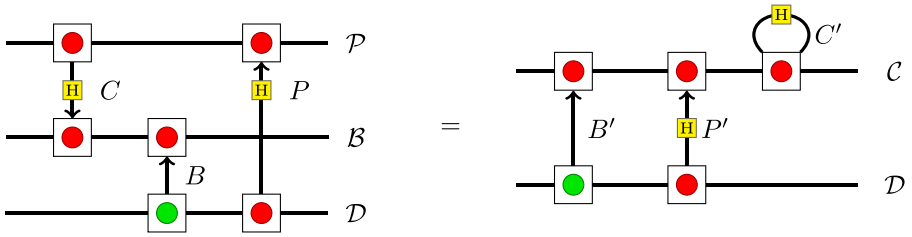
Definition 9. A *generalised CPC code* consists of two adjacency matrices B, P and a symmetric adjacency matrix C whose associated encoder and decoder circuits are defined as:



We can also see that, for any tripartite CPC code, we can construct an equivalent generalised code, up to local Clifford operations. Starting with the encoder for a tripartite code, pre- and post-composing the phase check qubits with Hadamards gives us the following:



Combining the bit-check and phase-check qubits into a one larger set $\mathcal{C} := \mathcal{B} \cup \mathcal{P}$ gives:



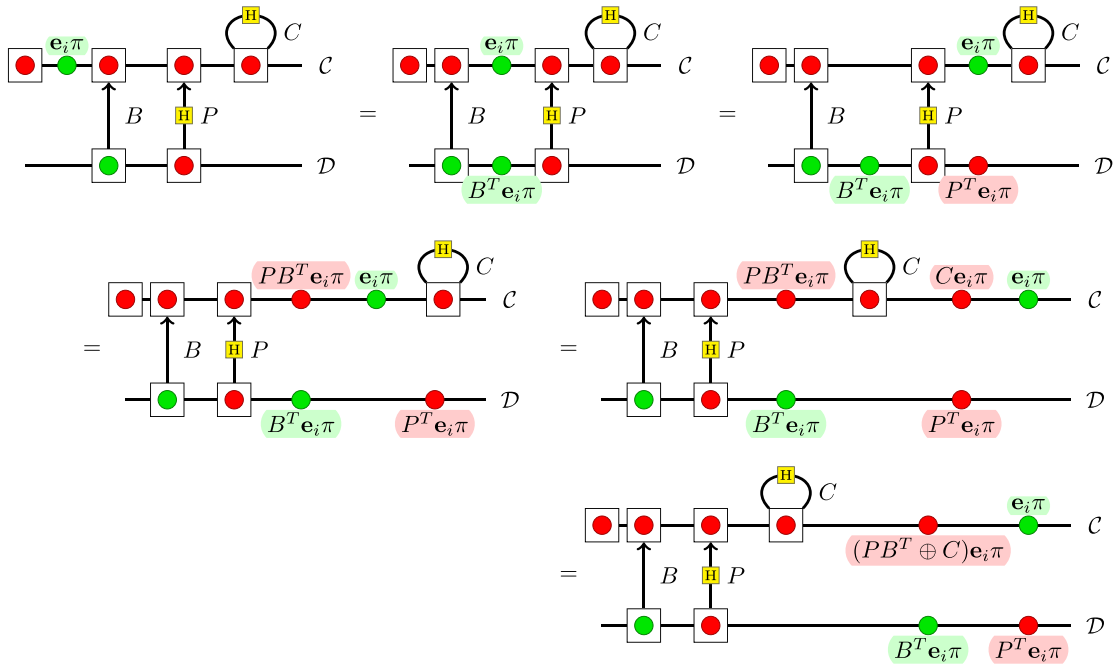
where B', P' and C' can be written as block matrices as follows:

$$B' := \begin{pmatrix} B \\ 0 \end{pmatrix} \qquad P' := \begin{pmatrix} 0 \\ P \end{pmatrix} \qquad C' := \begin{pmatrix} 0 & C \\ C^T & 0 \end{pmatrix}.$$

However, there are generalised CPC codes which are not equivalent to tripartite CPC codes. In particular, we will see in the next section that the stabilizers for a generalised code typically contain both Z and X operators, and hence are not in CSS form.

5.2. Stabilizers and error propagation for generalised codes

Much like in the tripartite case, we can compute error propagations and stabilisers by pushing Paulis through the decoder or encoder. We begin by computing stabilisers. For this, we introduce a Z operator on a single generalised check qubit and pushing it through the encoder:



Hence, we obtain one independent stabilizer for each generalised check qubit, given by:

$$\mathcal{S}_j := Z_{\mathcal{C}_j} \cdot \prod_{(PB^T \oplus C)_{ij}=1} X_{\mathcal{C}_i} \cdot \prod_{P^T_{ij}=1} X_{\mathcal{D}_i} \cdot \prod_{B^T_{ij}=1} Z_{\mathcal{D}_i}.$$

Note that, in this expression, Z and X operators may be applied to the same qubit, yielding a Y.

For example, the $[[9,3,3]]$ code (82) given in the next section has stabilizers:

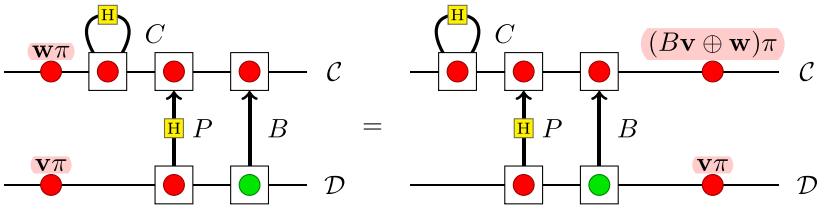
$$\begin{array}{ccccccccc} X & 1 & Y & Y & X & 1 & X & X & 1 \\ Z & 1 & X & X & Z & X & X & 1 & 1 \\ 1 & Z & Z & X & 1 & Z & 1 & 1 & 1 \\ X & Z & X & 1 & 1 & X & Z & X & X \\ Z & X & 1 & 1 & 1 & X & X & Z & X \\ Y & X & 1 & X & X & X & X & 1 & Y \end{array}. \tag{78}$$

The stabilizer tables for all codes presented here can be found in appendix D. We also provide a short Python program for converting matrices to stabilizers automatically in appendix C. This code as well as an Octave version is available through [55].

Using suitably-chosen matrices B, P and C, it is possible to generate a wide variety of stabilizer codes. In fact, we conjecture an analogue of the result of section 4.2, for general stabilizer codes:

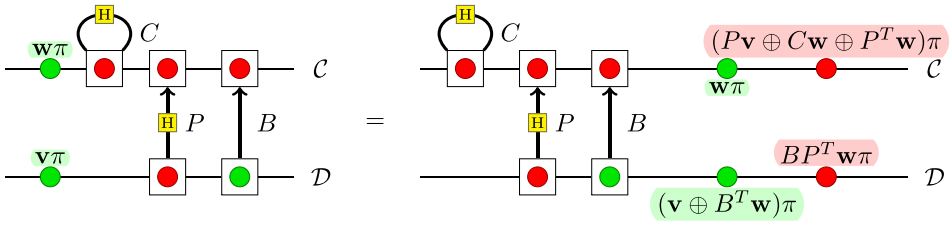
Conjecture 10. Up to local Clifford unitaries, all stabilizer codes are CPC codes.

Error propagation is computed similarly. Bit errors propagate as:



(79)

and phase errors propagate as:



(80)

As check bits are always measured in the Z basis, only bit-flip errors on the parity qubits will be detected by error correction. We can show using a similar calculation to the one in section 4.1, that a bit-flip error propagates to the kth parity qubit if and only if it anti-commutes with the kth stabiliser. Hence, if we first compute the stabilisers for a code, these error propagations can be computed simply as the error syndromes in the usual sense.

5.3. Automated design and search

This generalised CPC formalism gives a framework in which a large number of codes can be constructed. This is ripe for search by automated techniques. This will enable us to find, for instance, codes that give the greatest number of logical qubits for a given availability of physical qubits. The automated techniques outlined in this section are also capable of being combined with more sophisticated search and optimisation strategies, in order to produce codes to order based on hardware and desired optimality conditions.

Assuming an $[[n,k,d]]$ code (where d is initially not known), we start with k data qubits and $n-k$ generalised check qubits. The code itself is given by two matrices B and P of size $(n-k)\times k$, and a third, symmetric matrix C of size $(n-k)\times(n-k)$. It will be convenient to represent the latter as $C = C_u \oplus C_u^T$ where C_u is a strictly upper triangular matrix.

For generalised codes, check qubits are all measured in the computational basis. Hence error syndromes correspond to occurrences of bit errors on the check qubits after decoding. For the following initial errors:

$$\begin{aligned}\mathbf{v}_b &:= \text{bit errors on data qubits}\\ \mathbf{v}_p &:= \text{phase errors on data qubits}\\ \mathbf{w}_b &:= \text{bit errors on check qubits}\\ \mathbf{w}_p &:= \text{phase errors on check qubits}\end{aligned}$$

the error syndrome consists of all of the bit-errors which propagate to check qubits, according to equations (79) and (80) from section 5.2. Namely:

$$\mathbf{s} = B\mathbf{v}_b \oplus P\mathbf{v}_p \oplus \mathbf{w}_b \oplus \left(C \oplus P^T \cdot B\right)\mathbf{w}_p. \tag{81}$$

Equipped with the ability to calculate syndromes, the task of determining how many errors a code can tolerate is a matter of testing how many errors can be included while still preserving unique syndromes. Because $\mathbf{s}$ can be calculated efficiently, it is straightforward to test the code distance by exhaustive search, as

long as this distance is relatively low. In order for this to be scalable to large-distance codes, more sophisticated search techniques will be required, possibly exploiting sparsity properties in the adjacency matrices B, P and C. We comment on this further in section 7.

Remark 11. As with general stabiliser codes, the requirement that each error produce a unique syndrome is in fact too strong, and always yields non-degenerate codes. We can include degenerate codes by simply loosening the requirement as follows. If two errors $(\mathbf{v}_b, \mathbf{v}_p, \mathbf{w}_b, \mathbf{w}_p)$ and $(\mathbf{v}'_b, \mathbf{v}'_p, \mathbf{w}'_b, \mathbf{w}'_p)$ produce the same syndrome $\mathbf{s}$, then they should also yield the same error on the data qubits after decoding. Again applying equations (79) and (80), this amounts to checking the following equations are satisfied:

$$\mathbf{v}_b \oplus BP^T\mathbf{w}_p = \mathbf{v}'_b \oplus BP^T\mathbf{w}'_p$$
$$\mathbf{v}_p \oplus B^T\mathbf{w}_p = \mathbf{v}'_p \oplus B^T\mathbf{w}'_p.$$

Whenever these equations are satisfied, pushing the product of these two errors through the decoder yields nothing but phase errors on the check qubits. From this, we can conclude that the errors themselves differ only by a stabiliser.

5.3.1. Small codes from random search

The simplest method for automated code design is to randomly search the CPC code space. This is done by populating the matrices B, P and C_u randomly, and then checking all the syndromes for each code. We keep codes for which the desired code distance is achieved.

Using these methods, we found the following [[9, 3, 3]] code:

$$B = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad P = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \quad C_u = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{82}$$

This technique in fact finds many such codes. The search technique is straightforward. Therefore, rather than listing the codes, we instead provide a simple package to find these codes, available publicly at [55]. As an example a Python program to find codes can be found in appendix C. On a single core of a standard desktop computer, the given program generates around 141 $[[9,3,3]]$ codes in 10 minutes runtime on a single core. Approximately 0.18% of all randomly generated $[[9,3,d]]$ codes are able to correct single errors (d of at least 3). An Octave version of this code is also available through [55].

This simple search technique for CPC codes is not confined to distance 3. In appendix D we give the matrices for a $[[18,3,5]]$ code and $[[20,3,5]]$ code thus discovered.

We would expect the methods given here to also interface well with more sophisticated search methods. These include simulated annealing [56] or some of its more advanced variants, parallel tempering [57, 58] and population annealing [59–61], or by genetic algorithms [62]. To try to increase code distance, the cost function used in these techniques could be chosen to be proportional to the number of error patterns with one more error than the code is currently able to correct (which yield non-unique syndromes). Penalties related to hardware constraints could also be added—for example penalties on gates based on the separation between the qubits on which they are performed.

6. Encoded computation for CPC codes

So far we have concentrated on generating the CPC framework for codes for quantum memory. We now look at the addition of computation to the formalism. Performing full quantum computation is a complex procedure; we concentrate here on outlining the formalism for Clifford operations only.

We have seen that the CPC framework covers (amongst others) CSS codes. It is therefore possible to perform encoded computation in the standard way. However, one major advantage of the CPC framework is that we have a great deal of control over structuring codes to deal with specific situations that we want to use them for. We can put this to use in finding efficient ways of performing encoded gates that will be particularly of use in small-scale scenarios, where resource optimisation is the overriding concern. Specifically, we show that an operation, like a CNOT, in the encoded space can be performed by changing the encoder, and then performing the operation as if the qubits were unencoded. That is, the combination of the modified encoder plus the original operation performs the action of the encoded operation. Contrasting this with standard practise where the encoder is the same no matter what operation is then performed, it is the use of this extra information (what operation is to be performed) that enables the qubit and operation resources to be used much more efficiently.

6.1. Modifying the encoder

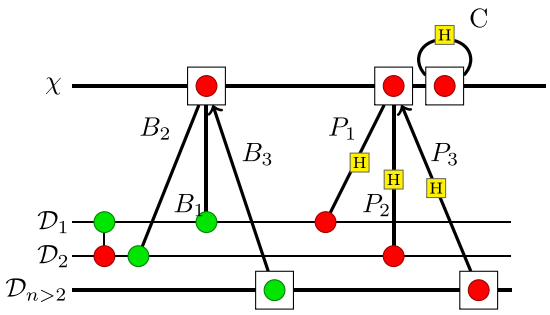
The key element to our new procedure for encoded computation is the modification of the encoding circuit to prepare the qubits for the Clifford operation. This then permits the procedure to be used in either the encode-wait-decode scenario, or else the standard encode-stabilizer measurement one. By changing the encoder, the procedure simultaneously sets up the code space and prepares it for the operation. The subsequent decoder (which is not changed) determines errors in the same way as when the code is used for memory.

In appendix E we perform such a modification of the encoder explicitly in Quantomatic for the ring code. This shows how the CNOT passes through the encoder, modifies it, and emerges intact as a two-qubit CNOT out the other side. In order to generalise this to a procedure for Clifford unitaries on any CPC code, we prove the following theorem:

Theorem 12. *Any unitary operation belonging to the Clifford group can be performed in the logical space between two data qubits in a CPC code by modifying the preparation state of the parity-check qubits and the B, P, C matrices that define the encoder only. Specifically, the decoder is not modified.*

Proof. The generators of the Clifford group are the Hadamard, CNOT and phase gate $S = \mathrm{diag}(1, i)$ [63]. It is sufficient then to show that each generator passes through a CPC encoder modifying only the B, P, and C matrices, and retaining its own structure.

CNOT gate. The operation of a logical CNOT in the encoded space must be equivalent to the action of a single CNOT between the raw, unencoded qubits. We can therefore find the encoded procedure by 'pushing' the CNOT operation on data qubits D_i and D_j through the encoder as re-writes. Using (13), we split out the data qubits that the CNOT operates on (here, without loss of generality we take them to be the first two in the spider box) from the rest of the data qubits, giving

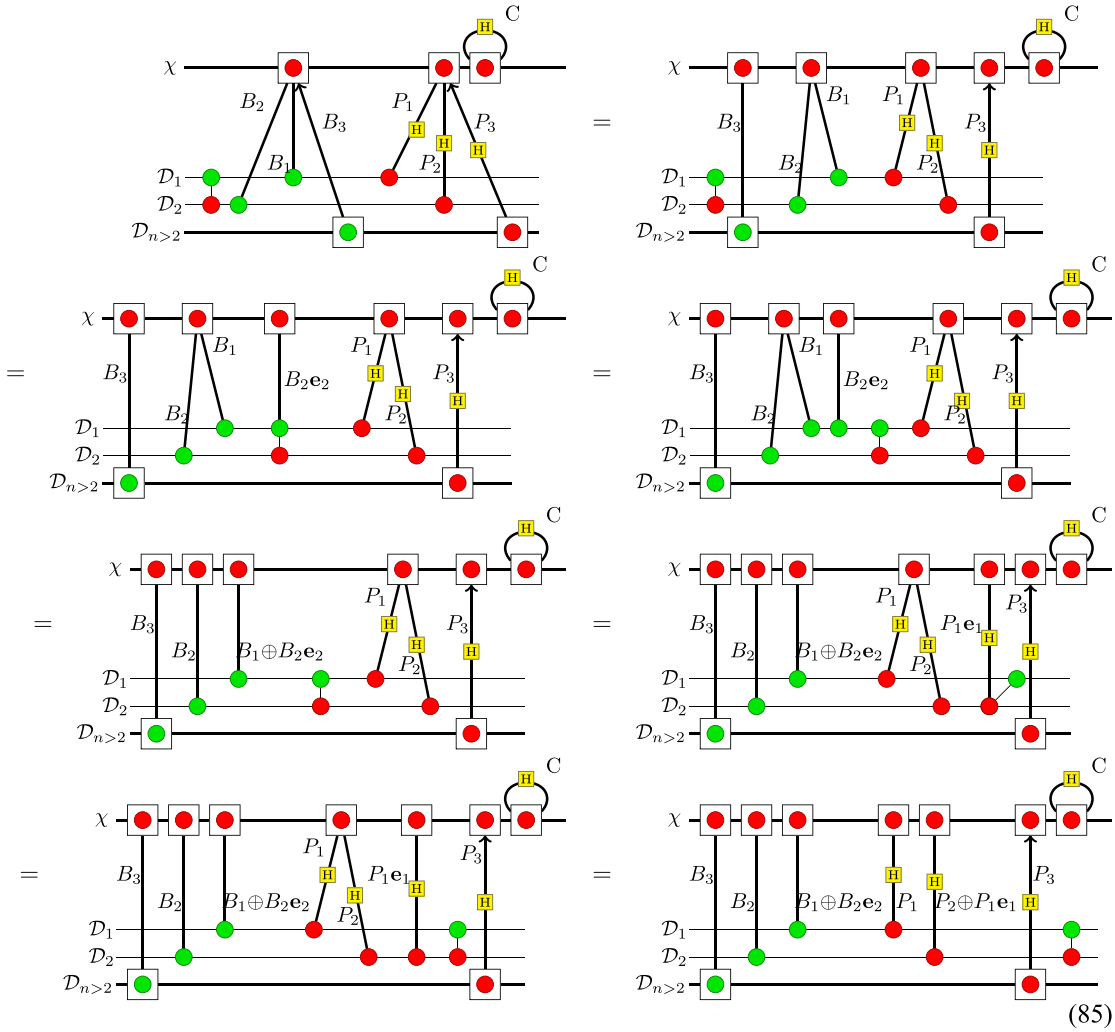


(83)

where

$$\left(\frac{B_1}{B_2}{B_3}\right) = B \quad , \quad \left(\frac{P_1}{P_2}{P_3}\right) = P \tag{84}$$

are the standard bit- and phase- parity check matrices. The CNOT through the encoder thus written then re-writes to



(85)

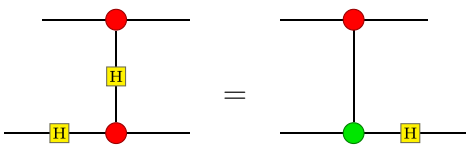
The second step above uses the strong complementarity rule of the *ZX* calculus (6), and the fifth step uses a variation of this by combining strong complementarity and the colour-change rule (3). The remaining steps are spider-fusion.

The modified encoder that prepares for the CNOT is therefore given by transforming the bit and phase parity check matrices as

$$B_{k1} \to B_{k1} \oplus B_{k2}\ ,\ \forall k \quad ; \quad P_{l2} \to P_{l2} \oplus P_{l1}\ ,\ \forall l \tag{86}$$

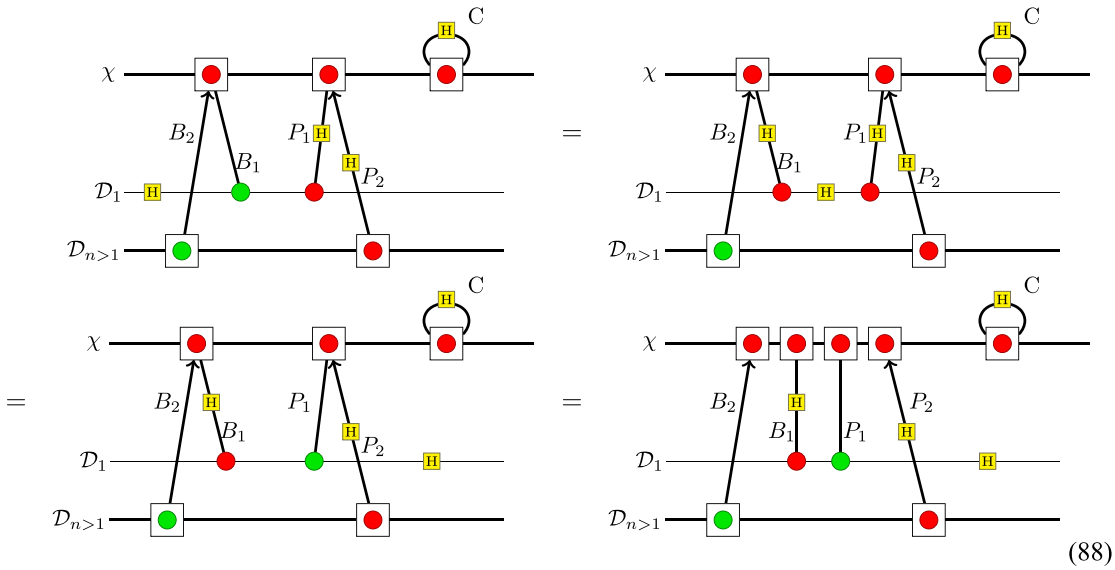
where k, l run over all the parity-check qubits χ. Again $\oplus$ denotes addition modulo 2 in the components of the P and B matrices.

Hadamard gate. Pushing a Hadamard gate through the encoder changes a conjugate propagator to a CNOT and vice versa:



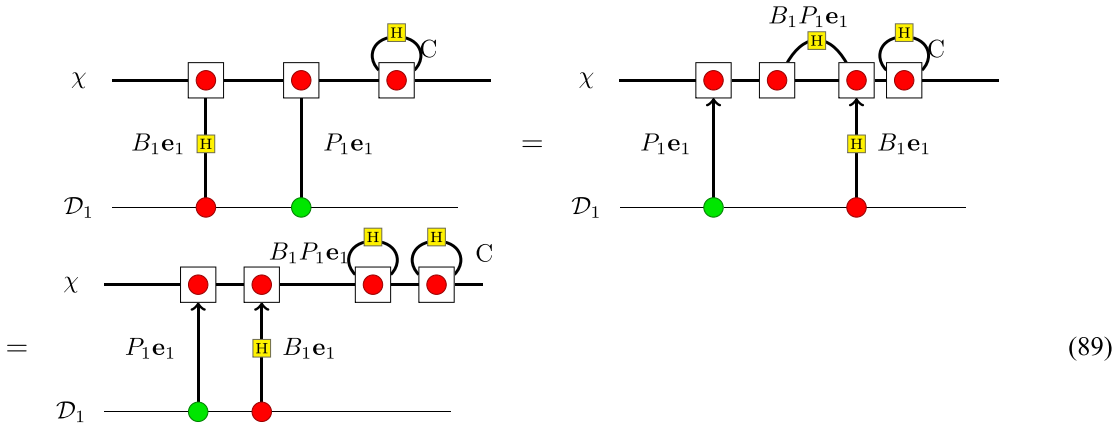
(87)

The Hadamard therefore modifies the encoder as (extracting out a single data qubit, w.l.o.g. the first, this time):



(88)

Let us re-write the inner two (sets of) gates in detail separately:



(89)

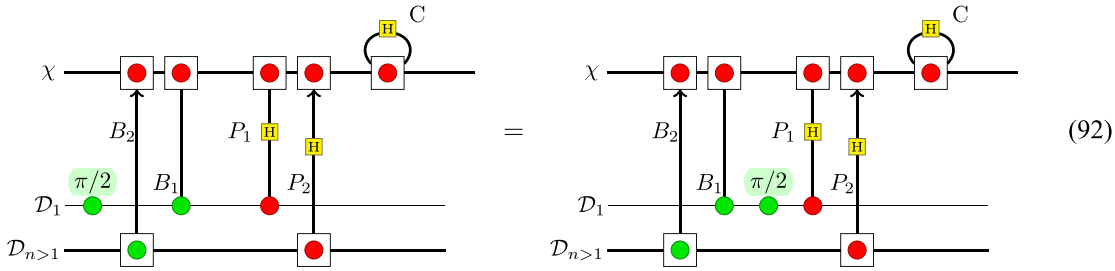
The modified encoder for a Hadamard on the first data qubit is therefore given by transforming the cross check matrix as

$$C_{kl} \to C_{kl} \oplus B_{k1}P_{l1} \quad \forall k,l \tag{90}$$

where again k,l indices run over all parity-check qubits χ. The bit-check and phase-check matrices also have their components that include the first data qubit interchanged:

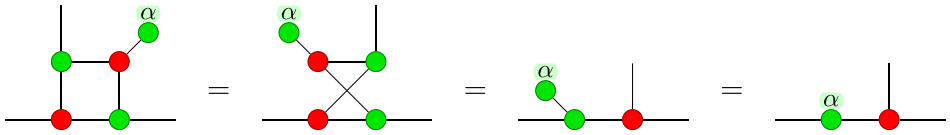
$$B_{i1} \leftrightarrow P_{i1} \quad \forall i. \tag{91}$$

Phase gate. Passing a phase gate (again, w.l.o.g. specified on the first data qubit) through the encoder rewrites as



(92)

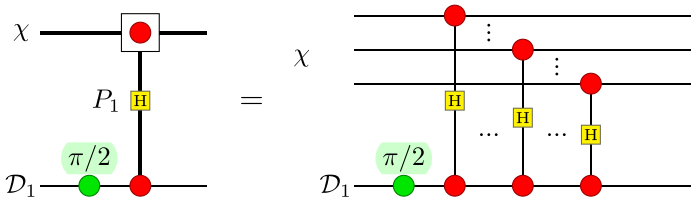
To pass the $\pi/2$ phase through the red node we use the following identity (for any α):



(93)

where the first re-write is simply re-arranging the upper nodes, the second uses the bialgebra rule [30, 9.3], and the third an application of the spider rule.

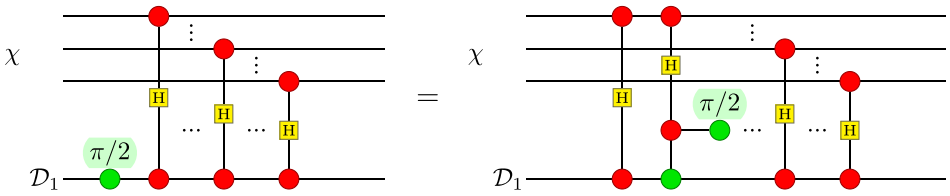
Using this set of equations in reverse we can now push the green phase through the set of conjugate propagators denoted by P_1 iteratively. To make this operation clear, we expand out the spider box notation:



(94)

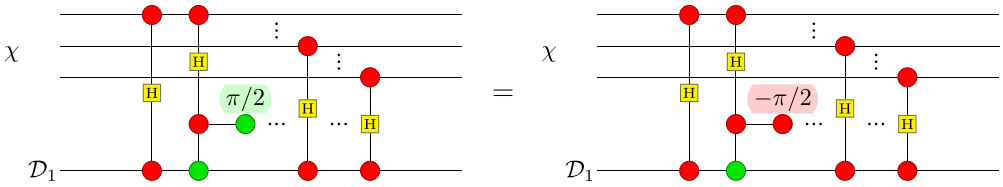
where the number of conjugate propagator gates and their target qubits in χ are determined by P_1.

Consider now how the phase gate re-writes through the first of these conjugate propagators, using (93):



(95)

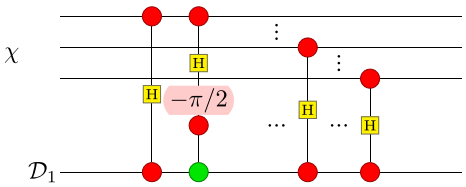
We can re-write the green $\pi/2$ node in this second diagram as a red $-\pi/2$ node, as it is now a state; the green $\pi/2$ state is the $+Y$ eigenstate, and the red $\pi/2$ is the $-Y$ eigenstate [30, 9.4.2]. That is,



(96)

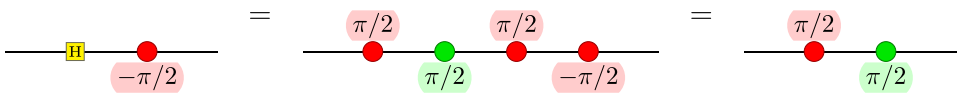
=

(97)



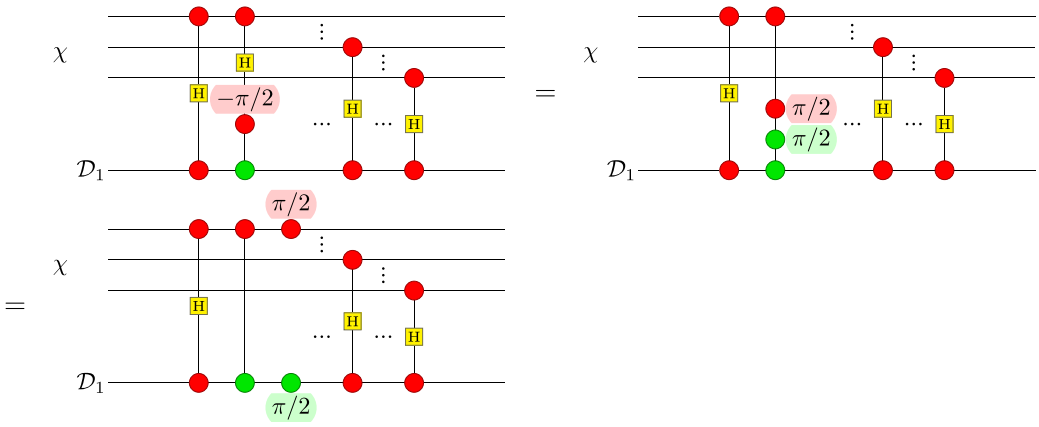
(98)

We can now re-write the central rail of the new gate using an Euler decomposition of the Hadamard gate (4) in the following way:



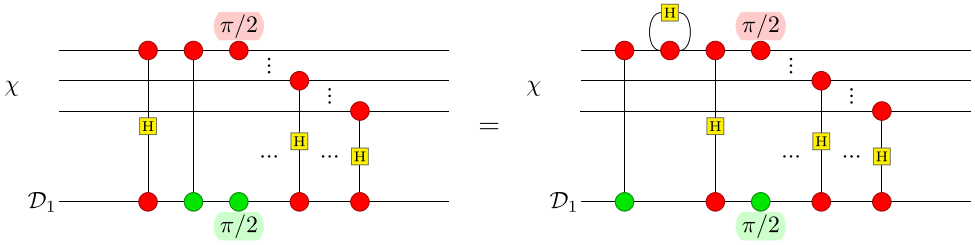
(99)

We therefore have



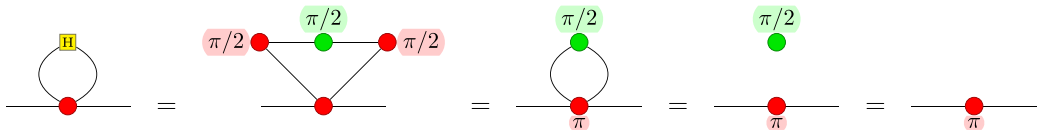
(100)

We now swap the order of the new CNOT gate(s) and the conjugate propagator, using the commutation relation found previously (89):



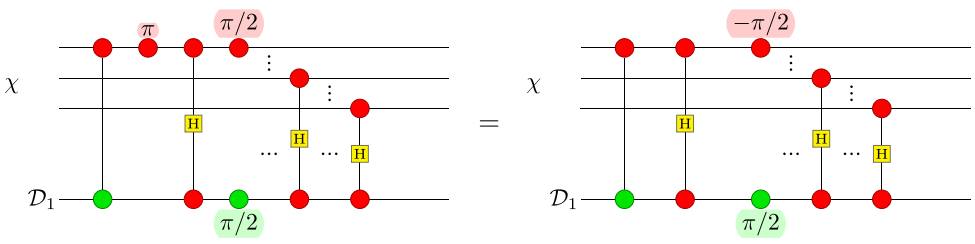
(101)

We can reduce the new conjugate propagator with control and target on the same qubit as follows



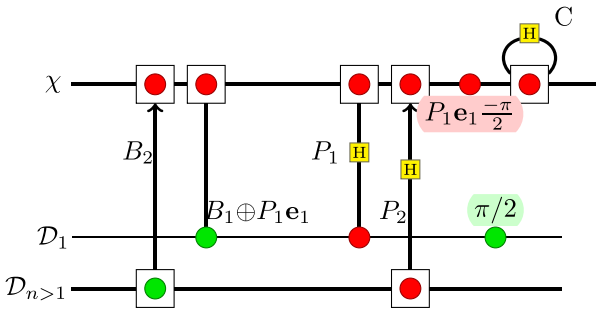
(102)

The final state after passing the phase gate through the first of the conjugate propagators given by P_1 is therefore



(103)

It is now clear how we can iterate this procedure, and push the phase through the remaining conjugate propagators. For every conjugate propagator given by P_1 between $\mathcal{D}_1$ and χ_i the action of pushing the $\pi/2$ green node introduces a new CNOT between $\mathcal{D}_1$ and χ_i, and a $\frac{-\pi}{2}\mathbf{e}_i$ phase. The final state of the full encoder after passing the phase gate through it is therefore



(104)

The modified encoder for the phase gate S on the ith data qubit is therefore given by transforming the components of the bit-parity check matrix as (recalling that the CNOT is self-inverse)

$$B_{i1} \to B_{i1} \oplus P_{i1} \quad \forall i \tag{105}$$

where again i index runs over all parity-check qubits χ. The addition of the set of red $-\pi/2$ phases on the parity-check qubits can be seen as a modification of any of the operations, a separate set of single-qubit operations, or as a modification of the states in which the parity-check qubits are prepared.

Therefore the set of gates comprising the phase gate, the CNOT gate, and Hadamard pass through the encoder modifying only the matrices B, P, C, or the state preparation of the parity-check qubits. This concludes the proof. □

6.2. Error propagation through the decoder or stabilizer measurement

Theorem 12 states that the decode circuit is unmodified when a Clifford unitary is encoded. Error information will therefore be detected in exactly the same way as if there had been no computation. This means that the only effect on how the code is decoded will come from the behaviour of that unitary, U_{Cliff}, itself.

The first of these additional complications is that the syndromes may be different. If U_{Cliff} contains any Pauli X or Z operations, these operations will be detected on the parity check qubits as if they were errors. Fortunately, this will happen in a completely predictable fashion, and the syndromes can be redefined appropriately, leading to no loss in code performance.

In addition to this, we also have the possibility that errors may be transformed. An error passing through U_{Cliff} may be transformed to a different type of error. For instance a Pauli X error may be transformed into a Y. This is not a problem for codes that correct X and Z errors independently, but may decrease the code distance otherwise. This decrease in code distance may be avoided by guaranteeing that error patterns including Y errors also produce unique syndromes.

A final issue is that two qubit gates may propagate single qubit errors to multiple qubits. In principle error correction can still be performed (albeit at reduced performance) if a single error cannot be propagated into a larger number of errors than the code can handle. For instance a single CNOT can only propagate one error into two errors, so a distance 5 code would still perform some error correction if U_{Cliff} were a single CNOT gate. However, it would act as an effective distance 3 code. A better way to cope with this issue is to take advantage of the fact that we know how errors will propagate in U_{Cliff}, so we can make sure that these specific error patterns produce unique syndromes. We refer to this process as *local hardening* of a code.

To finish, we give an example of a numerically discovered $[[11, 3, 3]]$ code which is locally hardened against the errors propagated by a CNOT gate, where the first data qubit acts as the control and the second as the target:

$$B = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

$$C_u = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{106}$$

7. Outlook and conclusions

We have seen how CPC codes enable the use of the *ZX*-calculus as a high-level language for designing and verifying a (potentially large) class of stabilizer codes. We have seen how the construction gives structure to quantum codes and how that is reflected in the graphical representation. By using the graphical methods as a reasoning tool, we have taken classical codes and created quantum ones from them in ways that have not been done before. We have shown how pairs of classical codes can be combined to form quantum codes, without onerous restrictions on which codes we can use. In the case of classical distance 3 codes, we have an explicit construction for turning any pair of codes into a quantum code by doubling the parity (qu)bits and adding appropriate cross-checks. *ZX* tools can be used to characterise any discovered code in terms of its stabilizers in a straightforward way.

A major result of the CPC construction is that large numbers of valid codes can be generated very quickly from the basic structural template, and generally have a reasonably large ratio of logical qubits to physical qubits. We have given a number of new small ($\leqslant$20 qubits) codes, and shown the simple search program that was used to find them. It is worth noting that while we have chosen to focus on small codes for presentation reasons, there is no reason that our search program would not work for much larger codes. As well as increasing the efficiency of the codes that are available to the first generations of quantum devices, CPC codes can also be performed in two different modes: either as single-shot codes that are encoded and decoded at each round, or else as standard stabilizer codes where syndromes are gathered through non-disturbing stabilizer measurements. In both cases, the graphical tools and structures are the same.

Finally, we have shown how in principle computation can be performed efficiently in the encoded space between logical qubits in the same code block. Rather than explicit operations, gates are performed by changing the encoder and/or stabilizer measurements. This has the potential to reduce significantly the overhead of physical operations for performing logical gates. It is also another tool to add to the performance of computation in error corrected systems, along with standard methods (braiding, transversal gates, and lattice surgery).

The CPC construction opens up the use of the *ZX*-calculus for rigorous and intuitive reasoning about stabilizer codes, with significant practical and theoretical benefits. It gives both a new structural understanding of quantum codes and their relation to classical, and a new tool for the design and analysis of quantum error correcting codes which are capable of being uniquely tailored to the resources and error profile of target hardware.

There are a number of avenues for future exploration for this work, and we finish by outlining some of them. We have shown (in particular in theorem 3) how the CPC framework can be considered in two equivalent ways, each with their separate use. Firstly, it is a new framework in which to understand how stabilizer codes operate, and their relation to classical codes. It enables us to construct, search, and analyse stabilizer codes for use with standard stabilizer measurement and syndrome extraction techniques. As such, they are amenable to the usual methods of performing fault-tolerant syndrome extraction, and can be viewed purely in this light.

However, they also perform a second function that is potentially important for use with small-scale quantum devices that are being produced in the immediate term. The CPC framework can there be considered as a method of constructing small codes, with the encode and decode operations representing

physical operations performed on the system. A 'round' of correction entails moving to and from the code space. These codes tolerate certain faults when decoded directly. Although single errors can duplicate, the codes are constructed so that the patterns of multiple errors will be uniquely recognised and corrected. By tracking the individual error propagation routes, using the tools the *ZX*-calculus gives us, we can produce low-overhead codes that are *de facto* fault tolerant.

Recent findings have demonstrated that protocols using a CPC-like encode-decode structure can be used to construct single-shot error mitigation circuits, further showing just how useful CPC codes can be outside the fault-tolerant regime. Debroy and Brown advanced a technique in which, similar to section 6 above, unitary encode-decode operations *sandwich* non-trivial Clifford circuits, thus enabling an increase in circuit fidelity within a single circuit sample [64]. Subsequently, van den Berg *et al* simulated CPC procedures on small computations, demonstrating that the logical error of CPC-protected circuits approaches a linear scaling in the number of physical qubits when the number of checks increases [65]. They also validated their findings experimentally. The notion of employing CPC-like circuits for error mitigation has also been explored in studies by Gonzales *et al* [66].

One further possible use-case for CPC codes is in fact as codes for themselves performing fault-tolerant syndrome extraction. It would in this regard (and others) be interesting to explore the relationship between CPC codes and the more recently-developed flag quantum computing protocols [67–70]. If the CPC codes were full error correcting (not just detecting) codes, then this could provide a pathway towards determining fault-tolerant circuits for extraction of syndromes given the stabilizer description of a code. Syndrome extraction circuits are needed in order to run full calculations of a code's threshold, which is a key metric for deciding if a code is any good in practice or not. This is therefore a live problem in evaluating novel codes, most recently the new quantum LDPC codes.

Another future direction is how to represent these quantum codes as classical codes. Given the simplicity of the rules for propagation of errors, these codes should be representable as a restricted class of classical error correcting codes within the factor graph formalism [71, 72]. Such a representation [43], has several advantages. By developing a simple graphical representation of graphical propagation rules, we will be able to create a toolkit for developing QEC codes without understanding the underlying quantum mechanics, allowing classical error correction experts to develop them. Furthermore, graphical models such as factor graphs can be naturally represented as Ising models, allowing decoding by specialized analog Ising model machines, such as quantum annealers [73, 74], optical systems [75], or a special purpose CMOS machine [76].

The examples we have demonstrated in this present paper are for small quantum codes. It is, however, generally understood in classical error correction that the performance of codes dramatically improves for larger codes. In fact, the Shannon limit can only be reached in the limit of code size approaching infinity. While it is not clear whether the structure of the parity codes constructed in the CPC formalism will allow them to approach the Shannon limit, it is likely that larger codes will perform better than smaller codes. This is in direct contrast to many QEC models which derive relatively little benefit from encoding more data qubits.

Low-density parity check (LDPC) codes are known to be state-of-the-art codes. This is both due to their performance close to the Shannon limit when their size becomes large, as well as due to the fact that there exist very fast (approximate) inference algorithms to decode them (see [77]). It will be important to discover how far those properties translate over to the coherent parity code construction. The main inference algorithm for LDPCs is based on belief propagation on graphical models. While such an algorithm is only exact for graphs which are trees, the algorithm is known to perform very well in practice for graphs which have not many small loops. Such properties can usually be achieved in practice. Furthermore, for given static codes one can also create decode tables. Our prime concern is thus the encoding level.

The main reason why LDPCs have such good performance is their large minimum code distance. This is achieved through their random construction. In particular, for LDPCs the minimum code size scales linearly with the size of the code. The behaviour is essential to achieve performance which approaches the Shannon limit. Give the various constraints in the construction of our codes it is likely that for large codes we will not be able to achieve a minimum code which scales linearly. The situation would thus be similar to that of the QEC codes derived in [78]. However, as was shown in [78], as long as such a codes have bounded code distance, they perform well in practice. Linearly growing minimal distance is only required formally when the noise level is taken to zero. For any small but finite noise level, a code with bounded distance will perform well in practice.

The quantum code framework presented here can in principle be applied to many different types of classical codes. To facilitate this line of research, we recently developed a mapping from the techniques presented in this paper to standard classical graphical models [43]. Following this, one interesting future direction to consider is constructing quantum codes based on classical turbo codes [79, 80], which have been

used, for example, in 4G and 5G mobile transmissions. While turbo codes fall into the class of so-called convolutional codes, they are very closely related to LDPCs [81], and it is reasonable to consider that QEC schemes based on turbo codes can be constructed using our CPC formalism.

The CPC construction also paves the way for software design tools for quantum computers that give codes for specific hardware layouts and specifications. We have demonstrated automated design based on random search, but more powerful design tools could be constructed using more sophisticated algorithms such as evolutionary algorithms or those within the Monte Carlo 'family' of techniques (such as simulated annealing or parallel tempering). The CPC formalism also enables high-performance classical codes to be imported for use on quantum devices, closing the gap between the tools that have been developed in classical computer science and the theoretical structures of QEC.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://doi.org/10.15128/r1bn999672k.

Acknowledgments

The authors would like to thank Viv Kendon for many useful discussions and comments on the text. We would also like to thank Samson Abramsky, Niel de Beaudrap, Earl Campbell, Bob Coecke, Ross Duncan, Elham Kashefi, and Tim Proctor for useful discussions of various aspects of these codes, and anonymous referees for constructive comments.

D H and N C were funded by the UK EPSRC Grant EP/L022303/1, N C was also funded by EP/S00114X/1. A K was supported by the ERC under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant No. 320 571. During the early stages of this project S Z was funded by Nokia Technologies, Lockheed Martin and the University of Oxford through the Quantum Optimisation and Machine Learning (QuOpaL) project. J R is funded by BMBF (RealistiQ) and the DFG (CRC 183). J R was also supported by the QCDA Project (EP/R043825/1) which received funding from the QuantERA ERA-NET Cofund in Quantum Technologies implemented within the European Union's Horizon 2020 Programme. In the early phases of this project, J R was additoinally supported by a Durham University Doctoral Studentship.

Appendix A. Fidelity analysis of an elementary three-qubit CPC gadget

The CPC gadget is one of the simplest possible detection codes for the identification of bit-flip errors in a quantum computer. Whilst the ultimate aim is to build full QEC codes capable of identifying and localising errors, detection codes remain of interest as they can be simple enough to implement on current hardware. Such experiments will adopt a *repeat-until-success* style approach with a detection code dictating which runs should be discarded. For example, in the case of the CPC gadget, only runs which return a 0 syndrome would be accepted. We demonstrate in this appendix that, assuming the 0 syndrome is measured, the fidelity of qubits encoded via the CPC gadget is greater than that for unprotected qubits.

In our analysis, we will assume that, over the time of an error cycle t_c, a single qubit Q is subject to an error process of the form

$$E_Q = e^{-\mathrm{i}\epsilon X} = \cos(\epsilon) I_Q - \mathrm{i}\sin(\epsilon) X_Q, \tag{A.1}$$

where ϵ is proportional to the error probability in the time-frame t_c. Applying this error model to an unprotected data register of two raw qubits $\left|\psi_{\mathrm{reg}}(0)\right\rangle = |\psi_A\psi_B\rangle$ yields the following state

$$\begin{aligned}\left|\psi_{\mathrm{reg}}(t_c)\right\rangle = E_A E_B |\psi_A\psi_B\rangle &= \cos^4(\epsilon) I_A I_B |\psi_A\psi_B\rangle \\ &- i\sin(\epsilon)\cos(\epsilon)\left(X_A I_B + I_A X_B\right)|\psi_A\psi_B\rangle \\ &- \sin^2(\epsilon) X_A X_B |\psi_A\psi_B\rangle .\end{aligned} \tag{A.2}$$

It is convenient to quantify the overlap of the evolved state, $\left|\psi_{\mathrm{reg}}(t_c)\right\rangle$, with the original state, $\left|\psi_{\mathrm{reg}}(t_c)\right\rangle$, in terms of the fidelity $F_{\mathrm{unprotected}}$. This yields

$$F_{\mathrm{unprotected}} = \left|\left\langle \psi_{reg}(t_c) \mid \psi_{reg}(0)\right|\right\rangle^2 = \cos^4(\epsilon) \approx 1 - 2\epsilon^2, \tag{A.3}$$

where we have made a Taylor expansion under the assumption that ϵ is small. In order to show that the CPC gadget suppresses the error rate, we need to show that when the 0 syndrome is measured the CPC gadget

Table A1. The syndrome table for the CPC gadget.

	Error, $\mathcal{E}$	Probability amplitude, p_A	Syndrome, S
No error	I	$\cos^3(\epsilon)$	0
1-qubit error	X_A	$-i\sin(\epsilon)\cos^2(\epsilon)$	1
	X_B		1
	X_P		1
2-qubit error	X_AX_B	$-\sin^2(\epsilon)\cos(\epsilon)$	0
	X_AX_P		0
	X_BX_P		0
3-qubit error	$X_AX_BX_P$	$-i\sin^3(\epsilon)$	1

outputs a state with higher fidelity than the unprotected case, such that $F_{\text{CPC}|S=0} > F_{\text{unprotected}}$. The error operator across three qubits of the CPC gadget $|\psi_A\psi_b\rangle|0_P\rangle$ is

$$\begin{aligned} E_AE_BE_P = e^{i\epsilon(X_A+X_B+X_P)} = cos^3(\epsilon)\, I_AI_BI_P \\ - i\sin(\epsilon)\cos^2(\epsilon)\,(X_AI_BI_P + I_AX_BI_P + I_AI_BX_P) \\ - \sin^2(\epsilon)\cos(\epsilon)\,(X_AX_BI_P + I_AX_BX_P + X_AI_BX_P) \\ - i\sin^3(\epsilon)\, X_AX_BX_P. \end{aligned} \tag{A.4}$$

Table A1 shows syndromes for the CPC gadget under the above error model. A 0 syndrome measurement will most likely indicate that no error has occurred. However, with lower probability, the two-qubit errors $\{X_AX_B, X_AX_P, X_BX_P\}$ could also result in a 0 syndrome. A 0 syndrome measurement therefore projects the output of the CPC gadget onto the state

$$|\psi_{CPC}(t_c)\rangle_{S=0} = \frac{\cos^3(\epsilon)\, I_AI_BI_P - \sin^2(\epsilon)\cos(\epsilon)\,(X_AX_B + X_BX_P + X_AX_P)}{\sqrt{|\cos^3(\epsilon)|^2 + 3|\sin^2(\epsilon)\cos(\epsilon)|^2}}\,|\psi_A\psi_B\rangle|0_P\rangle\,, \tag{A.5}$$

where the numerator represents the superposition of all the possible errors, weighted by their respective probabilities, that will result in a 0 syndrome. The denominator is the renormalisation factor. The conditional fidelity after a single cycle is now given by

$$F_{\text{CPC}|S=0} = |\langle\psi_{reg}(t_c)\,|\,\psi_{reg}(0)|\rangle|^2 \tag{A.6}$$

$$= \frac{\cos^6(\epsilon)}{\cos^6(\epsilon) + \sin^4(\epsilon)\cos^2(\epsilon)} = \frac{1}{3\tan^4(\epsilon)+1} \approx 1 - 3\epsilon^4, \tag{A.7}$$

where we have again assumed that ϵ is small. We have now demonstrated that $F_{\text{CPC}|S=0} > F_{\text{unprotected}}$. The bit-flip error rate of qubits encoded via a CPC gadget is therefore lower than that for unprotected qubits.

Appendix B. Orthogonality of classical simplex codes

We prove the following lemma, used in the proof of cross-checks for distance 3 codes, section 4.3.1:

Lemma 13. *For classical simplex codes with generator* $H = [A_S|1|\mathbb{1}] = [A_L^T|1|\mathbb{1}]$, $A_SA_S^T = 1 \oplus \mathbb{1}$, *where* A_L *gives the generator of the shortened Hamming code, as in section 4.3.1.*

Proof. The generator of a simplex code is obtained by listing all k-digit binary strings and removing the all-zero string. We first show that simplex codes are self-orthogonal; that is $HH^T = 0$, for $k > 2$.

We first show that the $k = 2$ case is not self-orthogonal. The generator (not in standard form) is the list of 2-digit binary numbers as columns, without zero:

$$H_2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{B.1}$$

it is trivial to show that $H_2H_2^T \neq 0$, either by direct computation or by noting that the elements of the matrix $H_2H_2^T$ are the different dot-products of the code words (the rows). As addition is modulo 2, each code word

producted with itself gives 0 (as there are an even number of 1 s in each word) but the off-diagonal elements will be 1 as the two code words share only a single 1.

We now prove self-orthogonality for simplex codes > 3 by induction. The base case is $k = 3$:

$$H_3 = \left[\begin{array}{ccccccc} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right] = \left[\begin{array}{c|c|c} 0^{(3)} & 1 & 1^{(3)} \\ \hline H_2 & \begin{array}{c}0\\0\end{array} & H_2 \end{array}\right]. \tag{B.2}$$

By direct computation $H_3 H_3^T = 0$. All words have an even number of 1 s, and each pair of words share an even number of 1 s.

For the inductive step, we note that

$$H_k = \left[\begin{array}{c|c|c} 0^{(2^{k-1}-1)} & 1 & 1^{(2^{k-1}-1)} \\ \hline H_{k-1} & \begin{array}{c}0\\0\end{array} & H_{k-1} \end{array}\right]. \tag{B.3}$$

The first row (word) contains an even number of 1 s as $(2^{k-1} - 1)$ is necessarily odd. By the inductive hypothesis, $H_{k-1} H_{k-1}^T = 0$, therefore all other words also contain an even number of 1 s. The diagonal of the matrix $H_k H_k^T$ is therefore all 0 s. The first word (first row) also necessarily shares an even number of 1 s with each other word. All other pairs of words are made up of two copies of the equivalent code words in H_{k-1}; by the inductive hypothesis, they therefore share an even number of 1 s. Therefore if $H_{k-1} H_{k-1}^T = 0$ then $H_k H_k^T = 0$. Combined with the base case, we can therefore conclude that $H_k H_k^T = 0$ for all $k > 3$.

To complete the proof of the lemma, we note that we can write the generator of a simplex code as $H = [A_S|1|\mathbb{1}]$, by splitting off the column of all 1 s. Note further that as a consequence, $[\mathbb{1}|A_S^T] = [\mathbb{1}|A_L]$ is the generator of the shortened Hamming code. We therefore have

$$\begin{aligned} 0 &= HH^T \\ &= [A_S|1|\mathbb{1}] \left[\begin{array}{c} A_S^T \\ \hline 1 \\ \hline \mathbb{1} \end{array}\right] \\ &= A_S A_S^T \oplus 1 \oplus \mathbb{1} \\ &\Rightarrow A_S A_S^T = 1 \oplus \mathbb{1}. \end{aligned} \tag{B.4}$$

□

Appendix C. Source code for converting CPC matrices to stabilizer tables

Python code for converting CPC matricies to stabalizer tables in latex array form Feel free to reuse/modify but please attribute the source and cite this paper in any published work. This code (along with an Octave version) is available at the repository at [55] (module name 'python_CPC_functions'). Code written by Nicholas Chancellor.

```
# import necessary modules
import numpy as np
# converts CPC matrices to latex formatted stabilizer tables, and
# save latex formatted versions if desired
def CPC_mats_2_stabilizers(Mb, Mp, Mc, saveName = None):
        # Mb, Mp, and Mc are bit phase and cross check matrices written
        # in the format given in arXiv:1611.08012 saveName is an optional
        # parameter giving the name of the text file where the stabilizer
        # matrix is saved

        # n.b. convention in paper uses transpose of what we use in this
          ↪ code
        Mb = Mb.T
```

```
        Mp = Mp.T

        k = Mb.shape[0] # number of logical qubits
        n = Mb.shape[0] + Mb.shape[1] # number of total qubits

        strCellLines = [None]*(n-k) # list for storing lines of the latex
            ↪ array
        strCellLinesDisplay = [None]*(n-k) # list for storing lines of the
            ↪ display array

        # indirectly propagated phase information
        indirectProp = np.dot(np.transpose(Mp),Mb)

        for i in range(n-k): # iterate over stabilizers
                # list for storing (X, Z, Y or 1) elements of stabilizer row
                strCellChars = [None]*n

                # number of times Z or X stabilizer elements are found on a
                    ↪ given qubit
                numZmultList = np.zeros(n)
                numXmultList = np.zeros(n)

                # apply matrix formula to create stabilizers

                # Z stabilizers
                # bit information of measured qubit
                numZmultList[k+i] = numZmultList[k+i]+1
                # bit information from measured qubits
                numZmultList[range(k)] = numZmultList[range(k)]+Mb[:,i]

                # X stabilizers
                # phase information from measured qubits
                numXmultList[range(k)] = numXmultList[range(k)]+Mp[:,i]
                # phase information propagated by cross checks
                numXmultList[range(k,len(numXmultList))] = (
                        numXmultList[range(k,len(numXmultList))]+Mc[:,i]+np.
                            ↪ transpose(Mc[i,:]))
                # phase information propagated indirectly
                numXmultList[range(k,len(numXmultList))] = (
                        numXmultList[range(k,len(numXmultList))]
                            ↪ +indirectProp[:,i])
                # write stabilizer table
                for iWrite in range(n):
                        if (numZmultList[iWrite]strCellChars[iWrite] = '1'
                            ↪ # if there are neither X nor Z stabilizers
                        elif numZmultList[iWrite]strCellChars[iWrite] = 'Z'
                            ↪ # if there is only a Z stabilizer
                        elif numZmultList[iWrite]strCellChars[iWrite] = 'X'
                            ↪ # if there is only an X stabilizer
                        elif numZmultList[iWrite]strCellChars[iWrite] = 'Y'
                            ↪ # if there are both X and Z stabilizers
                strCellLines[i] = '␣&␣'.join(strCellChars)
                strCellLinesDisplay[i] = '␣'.join(strCellChars)

        # combine lines to make total latex array
        latex_output = '\\\\\n'.join(strCellLines)
        # combine lines to make display version of table
```

```
        display_output = '\n'.join(strCellLinesDisplay)

        if saveName: # write latex array to file if file name provided
                np.savetxt(saveName,[latex_output],fmt
                   ↪ = 'print(display_output)
```

Appendix D. Additional codes

In this appendix we give examples of codes discovered using the CPC formalism. We characterise the codes by giving both the CPC matrices B, P, and C_u matrices, and the associated code stabilizer table. The Python function which generated these codes is given in the repository at [55].

D.1. Numerically discovered codes

Numerically discovered $[[18, 3, 5]]$ code

Running on a single core of a standard desktop, our program will be able to find approximately one code in two and a half hours. On running, we found that approximately 0.018% (1.8×10^{-4}) of randomly generated matrices yielded a valid code. The following is an example of such a valid codes (note that transposes are shown to save space on the page):

$$B = \begin{pmatrix} 0&0&1&1&1&1&1&1&0&0&0&1&1&1&0 \\ 0&0&1&0&1&0&1&0&1&0&0&0&0&1&1 \\ 1&0&1&1&1&1&0&1&1&1&0&0&1&0&0 \end{pmatrix}^{\mathrm{T}} \tag{D.1}$$

$$P = \begin{pmatrix} 1&1&0&0&1&1&1&0&1&0&0&0&0&1&1 \\ 1&0&1&1&0&0&0&0&0&0&0&1&1&1&1 \\ 1&0&1&1&0&1&0&1&0&0&1&1&0&0&0 \end{pmatrix}^{\mathrm{T}} \tag{D.2}$$

$$C_u = \begin{pmatrix} 0&0&1&0&0&1&0&0&0&1&1&0&1&1&0 \\ 0&0&0&1&0&0&1&0&0&1&0&1&1&1&0 \\ 0&0&0&0&0&0&1&1&0&0&1&1&1&0&0 \\ 0&0&0&0&1&1&1&0&1&1&1&1&0&1&1 \\ 0&0&0&0&0&1&0&0&1&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&0&0&0&1&1 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&1&1 \\ 0&0&0&0&0&0&0&0&0&1&0&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix}. \tag{D.3}$$

The stabilizer table for this code is

$$\begin{array}{cccccccccccccccccc}
X & X & Y & Y & 1 & 1 & X & 1 & 1 & 1 & X & 1 & X & 1 & X & X & X & 1 \\
X & 1 & 1 & 1 & Z & 1 & X & 1 & 1 & X & 1 & 1 & X & 1 & X & X & X & 1 \\
Z & Y & Y & 1 & X & Z & 1 & X & 1 & 1 & 1 & X & 1 & 1 & X & 1 & 1 & 1 \\
Z & X & Y & 1 & 1 & X & Y & 1 & X & 1 & X & 1 & X & 1 & 1 & 1 & 1 & 1 \\
Y & Z & Z & X & X & 1 & X & Y & X & X & X & 1 & 1 & X & X & X & 1 & 1 \\
Y & 1 & Y & X & X & X & 1 & 1 & Z & X & X & 1 & X & X & X & 1 & 1 & 1 \\
Y & Z & 1 & 1 & 1 & 1 & 1 & X & X & Y & X & 1 & 1 & 1 & X & X & X & X \\
Z & 1 & Y & 1 & X & 1 & X & X & 1 & 1 & Y & X & 1 & X & 1 & 1 & 1 & 1 \\
X & Z & Z & 1 & 1 & 1 & X & X & 1 & X & X & Z & X & X & 1 & 1 & X & 1 \\
1 & 1 & Z & 1 & X & X & 1 & 1 & 1 & 1 & X & X & Z & 1 & 1 & X & X & X \\
1 & 1 & X & X & 1 & X & X & 1 & 1 & 1 & 1 & 1 & X & Z & 1 & X & 1 & X \\
Z & X & X & X & 1 & X & X & 1 & X & X & X & X & X & 1 & Z & 1 & 1 & 1 \\
Z & X & Z & X & 1 & 1 & X & X & 1 & X & X & 1 & X & 1 & X & Z & 1 & X \\
Y & Y & 1 & X & 1 & X & 1 & X & 1 & 1 & X & X & X & 1 & 1 & 1 & Z & X \\
X & Y & 1 & X & 1 & X & 1 & 1 & X & X & X & X & X & X & 1 & X & 1 & Y
\end{array} . \tag{D.4}$$

Numerically discovered [[20, 3, 5]] code

Running on a single core of a standard desktop, our program will be able to find approximately 31 working codes in 10 minutes. Approximately 4.2% of randomly generated matrices yielded a valid code. One example is:

$$B = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}^{\mathrm{T}} \tag{D.5}$$

$$P = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0
\end{pmatrix}^{\mathrm{T}} \tag{D.6}$$

$$C_u = \begin{pmatrix}
0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} . \tag{D.7}$$

The stabilizer table for this code is

$$\begin{array}{cccccccccccccccccccc}
Y & Z & X & Y & X & 1 & X & X & X & 1 & 1 & 1 & X & 1 & 1 & X & 1 & 1 & X & X \\
X & Y & Y & 1 & Z & 1 & 1 & X & X & X & X & 1 & X & 1 & X & 1 & 1 & 1 & 1 & X \\
1 & 1 & 1 & 1 & 1 & Z & 1 & X & 1 & 1 & X & X & 1 & 1 & 1 & X & 1 & X & X & X \\
1 & Y & Y & X & 1 & 1 & Z & 1 & 1 & X & 1 & 1 & 1 & X & X & X & 1 & X & X & X \\
1 & Z & 1 & X & 1 & X & X & Z & 1 & X & X & 1 & 1 & X & X & X & 1 & 1 & X & 1 \\
X & Y & X & X & 1 & 1 & X & X & Y & 1 & 1 & 1 & X & 1 & X & 1 & X & 1 & 1 & X \\
X & 1 & Z & 1 & 1 & 1 & 1 & X & X & Z & X & 1 & 1 & 1 & 1 & X & X & 1 & 1 & X \\
Z & X & Z & X & 1 & X & 1 & 1 & X & 1 & Z & 1 & 1 & 1 & 1 & 1 & X & X & 1 & 1 \\
1 & Y & 1 & X & 1 & X & 1 & X & 1 & 1 & X & Y & 1 & 1 & X & X & X & 1 & 1 & 1 \\
X & X & X & X & X & 1 & 1 & X & 1 & X & 1 & X & Z & X & X & 1 & X & X & X & X \\
Z & Z & 1 & X & 1 & 1 & 1 & X & 1 & X & X & X & X & Z & X & X & 1 & X & X & 1 \\
Y & Y & Y & 1 & 1 & 1 & X & 1 & X & 1 & X & X & 1 & X & Y & X & X & X & X & 1 \\
Z & X & 1 & X & 1 & X & 1 & 1 & 1 & 1 & 1 & 1 & X & 1 & X & Z & X & 1 & X & 1 \\
Y & X & Y & 1 & 1 & 1 & X & X & 1 & X & X & 1 & X & 1 & 1 & 1 & Z & 1 & 1 & X \\
Y & 1 & X & 1 & 1 & X & 1 & 1 & X & 1 & X & 1 & 1 & 1 & 1 & X & X & Y & 1 & 1 \\
Z & Z & Z & X & X & X & X & X & X & X & X & X & 1 & X & 1 & 1 & X & 1 & Z & 1 \\
X & Z & Z & X & X & X & X & 1 & X & X & 1 & X & X & X & X & 1 & 1 & 1 & X & Z
\end{array} . \tag{D.8}$$

D.2. Other codes

[[10, 3, 3]] code from combining checks for parity bit flips on the [[11, 3, 3]] code

A relatively straightforward design alteration to the [[11, 3, 3]] code is to have a single qubit check all parity check qubits for phase errors, rather than one for those which check for bit errors on the data qubits, and a separate one which checks for phase errors. The resulting parity check matrices for this code are:

$$B = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}^{\mathrm{T}} \tag{D.9}$$

$$P = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}^{\mathrm{T}} \tag{D.10}$$

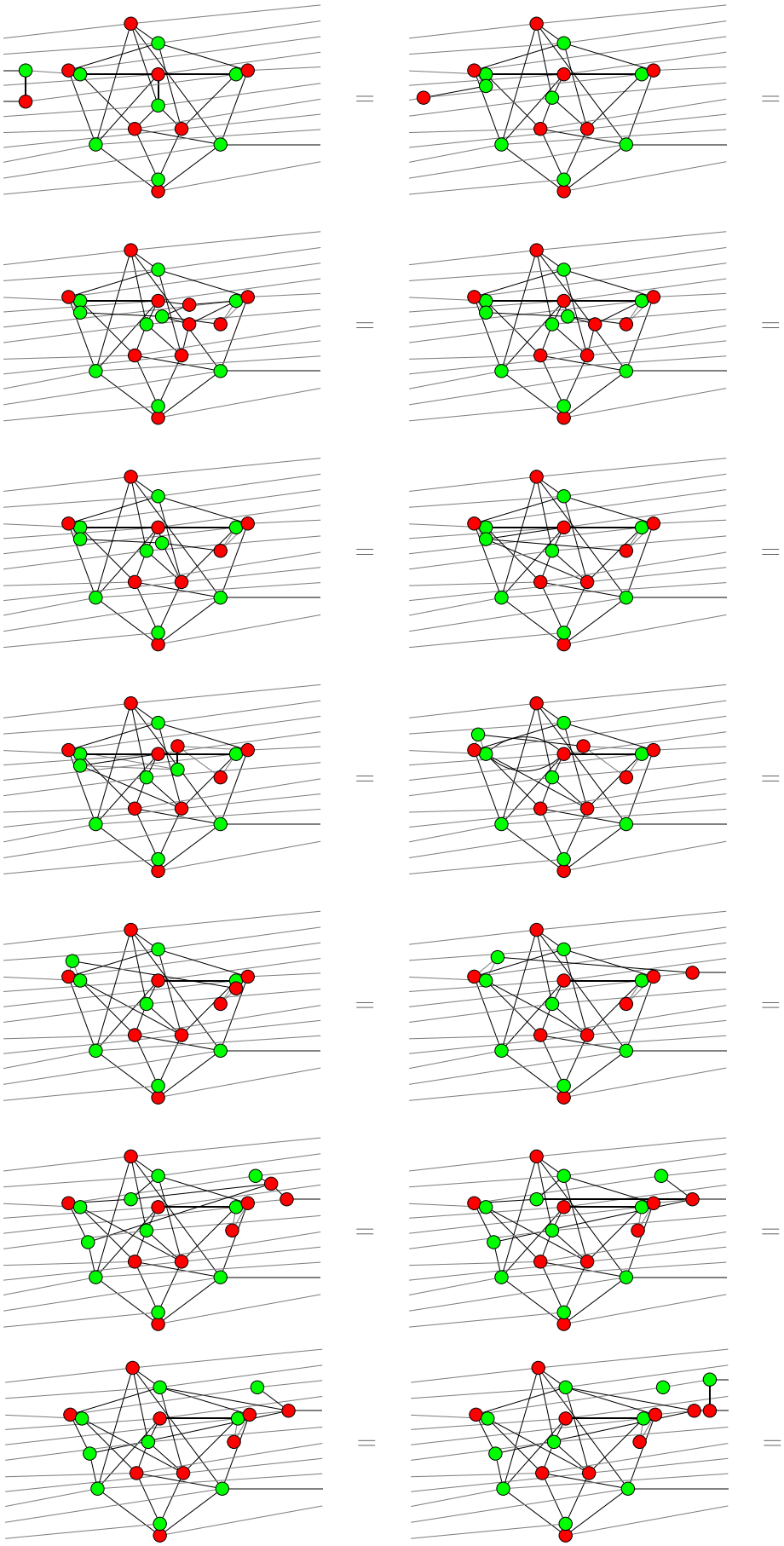
$$C_u = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . \tag{D.11}$$

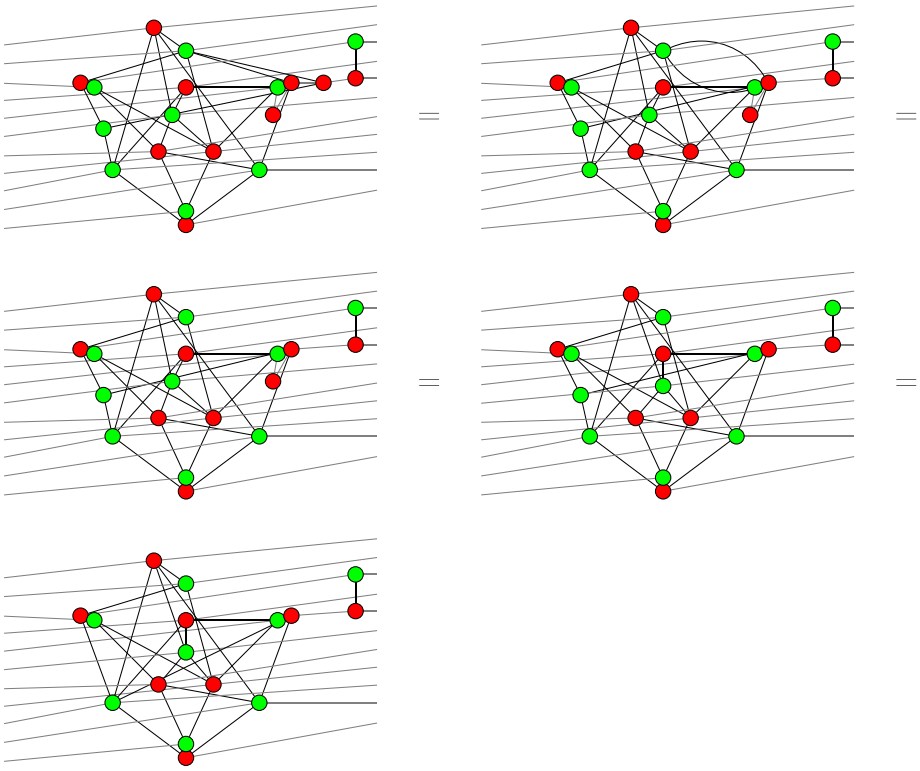
The corresponding stabilizer table is:

$$\begin{array}{cccccccccc}
Z & Z & 1 & Z & 1 & 1 & 1 & X & 1 & X \\
1 & Z & Z & 1 & Z & 1 & 1 & 1 & X & X \\
Z & 1 & Z & 1 & 1 & Z & X & 1 & 1 & X \\
X & X & 1 & 1 & X & 1 & Z & 1 & 1 & X \\
1 & X & X & 1 & 1 & X & 1 & Z & 1 & X \\
X & 1 & X & X & 1 & 1 & 1 & 1 & Z & X \\
1 & 1 & 1 & X & X & X & X & X & X & Z
\end{array} . \tag{D.12}$$

Appendix E. Encoded CNOT with Quantomatic

We give the modification of the encoder for the [[11,3,3]] ring code of section 3 to perform a CNOT operation in the encoded space. The general solution for CPC codes is given in section 6; this specific example is generated in Quantomatic by passing the CNOT operation through the encoder as follows:





This completes the modification.

ORCID iDs

Nicholas Chancellor https://orcid.org/0000-0002-1293-0761
Aleks Kissinger https://orcid.org/0000-0002-6090-9684
Joschka Roffe https://orcid.org/0000-0001-9202-1156
Dominic Horsman https://orcid.org/0000-0003-4965-0584

References

[1] Campbell E T, Terhal B M and Vuillot C 2017 Roads towards fault-tolerant universal quantum computation *Nature* **549** 172–9
[2] Devitt S J, Munro W J and Nemoto K 2013 Quantum error correction for beginners *Rep. Prog. Phys.* **076001** 1–41
[3] Anderson C, Remm A, Lazar S, Krinner S, Lacroix N, Norris G J, Gabureac M, Eichler C and Wallraff A 2020 Repeated quantum error detection in a surface code *Nat. Phys.* **16** 875
[4] Marques J F *et al* 2021 Logical-qubit operations in an error-detecting surface code *Nat. Phys.* **18** 80–86
[5] Krinner S *et al* 2022 Realizing repeated quantum error correction in a distance-three surface code *Nature* **605** 669–74
[6] Zhao Y *et al* 2022 Realization of an error-correcting surface code with superconducting qubits *Phys. Rev. Lett.* **129** 030501
[7] Chen Z *et al* 2021 Exponential suppression of bit or phase errors with cyclic error correction *Nature* **595** 383–7
[8] Erhard A *et al* 2021 Entangling logical qubits with lattice surgery *Nature* **589** 220–4
[9] Egan L *et al* 2021 Fault-tolerant control of an error-corrected qubit *Nature* **598** 281–6
[10] Ryan-Anderson C *et al* 2022 Implementing fault-tolerant entangling gates on the five-qubit code and the color code (arXiv:2208.01863 [quant-ph])
[11] Acharya R *et al* 2023 Suppressing quantum errors by scaling a surface code logical qubit *Nature* **614** 676–81
[12] Fowler A G, Mariantoni M, Martinis J M and Cleland A N 2012 Surface codes: towards practical large-scale quantum computation *Phys. Rev. A* **86** 032324
[13] Stephens A M 2014 Fault-tolerant thresholds for quantum error correction with the surface code *Phys. Rev.* A **89** 022321
[14] Calderbank A R and Shor P W 1996 Good quantum error-correcting codes exist *Phys. Rev.* A **54** 1098–106
[15] Steane A 1996 Error correcting codes in quantum theory *Phys. Rev. Lett.* **77** 793–7
[16] Tillich J-P and Zémor G 2013 Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength *IEEE Trans. Inf. Theory* **60** 1193–202
[17] Hastings M B, Haah J and O'Donnell R 2021 Fiber bundle codes: breaking the n 1/2 polylog (n) barrier for quantum LDPC codes *Proc. 53rd Annual ACM SIGACT Symp. on Theory of Computing* pp 1276–88
[18] Breuckmann N P and Eberhardt J N 2021 Balanced product quantum codes *IEEE Trans. Inf. Theory* **67** 6653–74
[19] Panteleev P and Kalachev G 2022 Quantum ldpc codes with almost linear minimum distance *IEEE Trans. Inf. Theory* **68** 213–29
[20] Panteleev P and Kalachev G 2022 Asymptotically good quantum and locally testable classical LDPC codes *Proc. 54th Annual ACM SIGACT Symp. on Theory of Computing* (https://doi.org/10.1145/3519935.3520017)
[21] Roffe J, White D R, Burton S and Campbell E 2020 Decoding across the quantum low-density parity-check code landscape *Phys. Rev. Res.* **2** 043423

[22] Dinur I, Hsieh M-H, Lin T-C and Vidick T 2023 Good quantum LDPC codes with linear time decoders *Proc. 55th Annual ACM Symp. on Theory of Computing* (ACM)
[23] Stehlik J *et al* 2021 Tunable coupling architecture for fixed-frequency transmons *Phys. Rev. Lett.* **127** 080505
[24] Kosen S *et al* 2022 Building blocks of a flip-chip integrated superconducting quantum processor *Quantum Sci. Technol.* **7** 035018
[25] Bravyi S B and Kitaev A Y 2001 Quantum codes on a lattice with boundary (arXiv:quant-ph/9811052) Translation of *Quantum Comput. Comput.* **2** 43–48 1998
[26] Dennis E, Kitaev A, Landahl A and Preskill J 2002 Topological quantum memory *J. Math. Phys.* **43** 4452
[27] Fowler A G, Stephens A M and Groszkowski P 2009 High threshold universal quantum computation on the surface code *Phys. Rev.* A **80** 052312
[28] Horsman D, Fowler A, Devitt S and Van Meter R 2012 Surface code quantum computing by lattice surgery *New J. Phys.* **14** 123011
[29] Coecke B and Duncan R 2011 Interacting quantum observables: categorical algebra and diagrammatics *New J. Phys.* **13** 043016
[30] Coecke B and Kissinger A 2017 *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning* (Cambridge University Press)
[31] Backens M 2014 The ZX-calculus is complete for stabilizer quantum mechanics *New J. Phys.* **16** 093021
[32] Jeandel E, Perdrix S and Vilmart R 2018 A complete axiomatisation of the ZX-calculus for clifford+t quantum mechanics *Proc. 33rd Annual ACM/IEEE Symp. on Logic in Computer Science* (ACM)
[33] Ng K F and Wang Q 2017 A universal completion of the ZX-calculus (arXiv:1706.09877 [qunat-ph])
[34] Wang Q 2022 Completeness of the ZX-calculus (arXiv:2209.14894)
[35] Horsman D 2011 Quantum picturalism for topological cluster-state computing *New J. Phys.* **13** 095011
[36] Duncan R and Lucas M 2013 Verifying the steane code with quantomatic *Electron. Proc. Theor. Comput. Sci.* **171** 33–49
[37] Garvie L and Duncan R 2018 Verifying the smallest interesting colour code with quantomatic *Electron. Proc. Theor. Comput. Sci.* **266** 147–63
[38] de Beaudrap N and Horsman D 2020 The ZX calculus is a language for surface code lattice surgery *Quantum* **4** 218
[39] Sivarajah S, Dilkes S, Cowtan A, Simmons W, Edgington A and Duncan R 2020 t|ket⟩: a retargetable compiler for NISQ devices *Quantum Sci. Technol.* **6** 014003
[40] Bombin H, Litinski D, Nickerson N, Pastawski F and Roberts S 2023 Unifying flavors of fault tolerance with the zx calculus (arXiv:2303.08829 [quant-ph])
[41] Gidney C and Fowler A G 2019 Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation *Quantum* **3** 135
[42] Kissinger A and Zamdzhiev V 2015 Quantomatic: a proof assistant for diagrammatic reasoning *Int. Conf. on Automated Deduction* (Springer) pp 326–36
[43] Roffe J, Zohren S, Horsman D and Chancellor N 2020 Quantum codes from classical graphical models *IEEE Trans. Inf. Theory* **66** 130–46
[44] Roffe J, Zohren S, Horsman D and Chancellor N 2019 Decoding quantum error correction with ising model hardware (arXiv:1903.10254)
[45] Penrose R 1971 Applications of negative dimensional tensors *Combinatorial Mathematics and its Applications* (Academic) pp 221–44
[46] Jeandel E, Perdrix S and Vilmart R 2018 Diagrammatic reasoning beyond clifford+t quantum mechanics *Proc. 33rd Annual ACM/IEEE Symp. on Logic in Computer Science* (ACM)
[47] Carette T, Horsman D and Perdrix S 2019 SZX-calculus: scalable graphical quantum reasoning *LIPIcs* **138** 55:1–55:15
[48] Vaidman L, Goldenberg L and Wiesner S 1996 Error prevention scheme with four particles *Phys. Rev.* A **54** R1745
[49] Grassl M, Beth T and Pellizzari T 1997 Codes for the quantum erasure channel *Phys. Rev. A* **56** 33
[50] Harty T P, Allcock D T C, Ballance C J, Guidoni L, Janacek H A, Linke N M, Stacey D N and Lucas D M 2014 High-fidelity preparation, gates, memory and readout of a trapped-ion quantum bit *Phys. Rev. Lett.* **113** 220501
[51] Preskill J 2015 Quantum information and computation (available at: www.theory.caltech.edu/people/preskill/ph229/)
[52] Gottesman D 1997 Stabilizer codes and quantum error correction *PhD Thesis* California Institute of Technology (https://doi.org/10.7907/rzr7-dt72)
[53] Oklobdzija V G and Dorf R C 2001 *The Computer Engineering Handbook: Electrical Engineering Handbook* (CRC Press Inc)
[54] Lord N J 1987 Matrices as sums of invertible matrices *Math. Mag.* **60** 33–35
[55] Chancellor N 2017 *CPC Code Creation Software as Presented in Graphical Structures for Design and Verification of Quantum Error Correction* (Durham University)
[56] Kirkpatrick S, Gelatt C D and Vecchi M P 1983 Optimization by simulated annealing *Science* **220** 671–80
[57] Swendsen R H and Wang J-S 1968 Replica Monte Carlo simulation of spin-glasses *Phys. Rev. Lett.* **57** 2607
[58] Earl D J and Deem M W 2005 Parallel tempering: theory, applications and new perspectives *Phys. Chem. Chem. Phys.* **7** 3910–6
[59] Hukushima K Iba Y and Gubernatis J E 2003 *The Monte Carlo Method in the Physical Sciences: celebrating the 50th Anniversary of the Metropolis Algorithm* vol 690 (American Institute of Physics)
[60] Matcha J 2010 Population annealing with weighted averages: a Monte Carlo method for rough free energy landscapes *Phys. Rev.* E **82** 026704
[61] Wang W, Machta J and Katzgraber H G 2015 Population annealing: theory and application in spin glasses *Phys. Rev.* E **92** 063307
[62] Fogel D B 1994 An introduction to simulated evolutionary optimization *IEEE Trans. Neural Netw.* **5** 3–14
[63] Gottesman D 1999 The Heisenberg representation of quantum computers *Proc. XXII Int. Coll. on Group Theoretical Methods in Physics* ed S P Corney, R Delbourgo and P D Jarvis pp 32–43
[64] Debroy D M and Brown K R 2020 Extended flag gadgets for low-overhead circuit verification *Phys. Rev.* A **102** 052409
[65] van den Berg E, Bravyi S, Gambetta J M, Jurcevic P, Maslov D and Temme K 2022 Single-shot error mitigation by coherent pauli checks (arXiv:2212.03937)
[66] Gonzales A, Shaydulin R, Saleem Z H and Suchara M 2023 Quantum error mitigation by pauli check sandwiching *Sci. Rep.* **13** 2122
[67] Chao R and Reichardt B W 2018 Quantum error correction with only two extra qubits *Phys. Rev. Lett.* **121** 050502
[68] Chao R and Reichardt B W 2018 Fault-tolerant quantum computation with few qubits *npj Quantum Inf.* **4** 42
[69] Chamberland C and M E Beverland 2018 Flag fault-tolerant error correction with arbitrary distance codes *Quantum* **2** 53
[70] Reichardt B W 2020 Fault-tolerant quantum error correction for steane's seven-qubit color code with few or no extra qubits *Quantum Sci. Technol.* **6** 015007
[71] Forney G D Jr. 2001 Codes on graphs: normal realizations *IEEE Trans. Inf. Theory* **47** 520–48
[72] Loeliger H A 2004 An introduction to factor graphs *IEEE Signal Process. Mag.* **21** 28–41
[73] D-Wave Quantum Inc.2023 *Company webpage* (available at: www.dwavesys.com/) (Accessed 6 January 2023)

[74] Chancellor N, Szoke S, Vinci W, Aeppli G and Warburton P A 2016 Maximum-entropy inference with a programmable annealer *Sci. Rep.* **6** 22318
[75] Inagaki T, Inaba K, Hamerly R, Inoue K, Yamamoto Y and Takesue H 2016 Large-scale ising spin network based on degenerate optical parametric oscillators *Nat. Photon.* **10** 415–9
[76] Yamaoka M, Yoshimura C, Hayashi M, Okuyama T, Aoki H and Mizuno H 2016 Advanced research into AI, Ising computer *Hitachi Rev.* **65** 6 (available at: www.hitachi.com/rev/archive/2016/r2016_06/110/index.html)
[77] MacKay D J C 2003 *Information Theory, Inference and Learning Algorithms* (Cambridge University Press)
[78] MacKay D J C, Mitchison G and McFadden P L 2004 Sparse graph codes for quantum error-correction *IEEE Trans. Inf. Theory* **50** 2315
[79] Berrou C, Glavieux A and Thitimajshima P 1993 Near Shannon limit error-correcting coding and decoding: turbo-codes *Proc. 1993 IEEE Int. Conf. on Communications* (*Geneva, Switzerland*) p 1064
[80] Berrou C and Glavieux A 1996 Near optimum error correcting coding and decoding: turbo-codes *IEEE Trans. Commun.* **44** 1261
[81] MacKay D J C 2002 Turbo codes are low density parity check codes (available at: www.inference.eng.cam.ac.uk/mackay/turbo-ldpc.pdf)