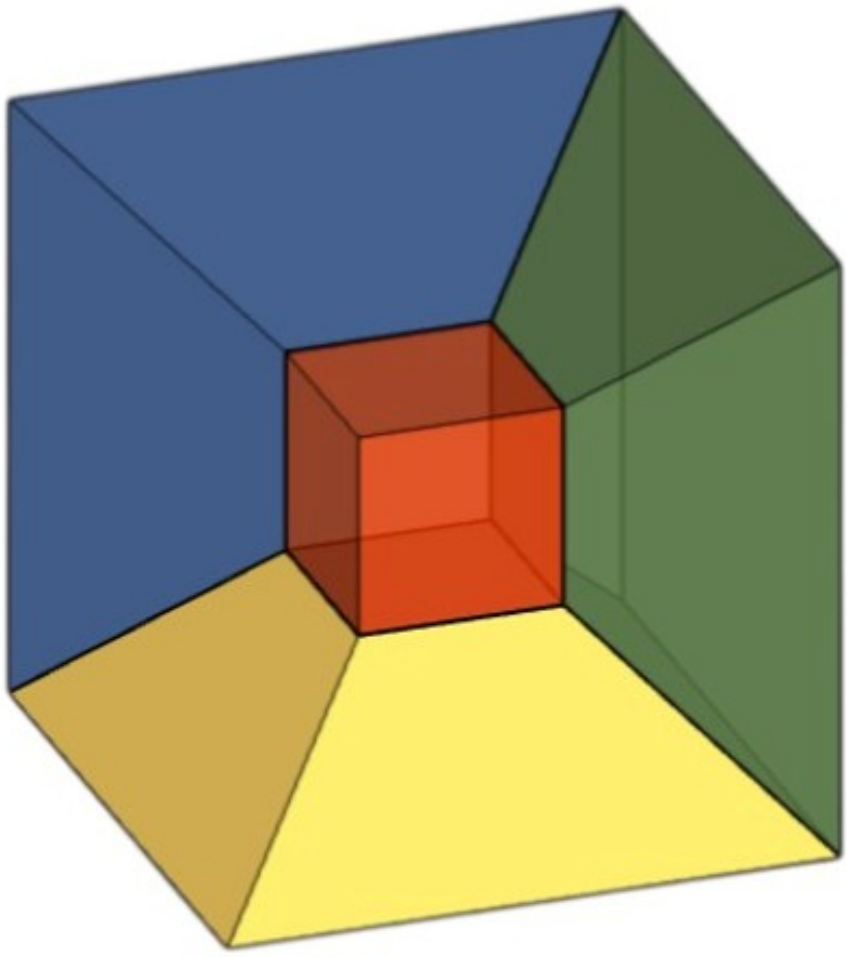
Intermediate Flag Verification for Logical Zero State Preparation

In the $[[15, 1, 3]]$ Code and Other PQRM Codes

S.C. Birkenhäger

Flag-Based Mid-Circuit Verification for Logical State Preparation



TU Delft

Adapted from C. Gidney, https://x.com/CraigGidney/status/1755363647537443070

Intermediate Flag Verification for Logical Zero State Preparation

In the $[[15, 1, 3]]$ Code and Other PQRM Codes

by

S.C. Birkenhäger

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on September 16, 2025 at 9:00 AM.

Student number:	4844882	
Project duration:	November 11, 2024 – July 28, 2025	
Thesis committee:	Prof. dr. ir. B. M. Terhal	TU Delft, supervisor
	Dr. ir. S. Heußen	neQxt GmbH, External Advisor
	Dr. ir. C. K. Andersen	TU Delft, Committee Member
	Dr. ir. B. Janssens	TU Delft, Committee Member

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This thesis concludes my Master of Science in Applied Mathematics at Delft University of Technology. The project was carried out between November 2024 and July 2025 in collaboration with neQxt GmbH and the Quantum Computing Division of the EEMCS faculty at TU Delft.

I would like to express my sincere gratitude to Prof. Dr. Barbara Terhal for her supervision, clear ideas, and detailed feedback throughout the project. I am also very grateful to Dr. Sascha Heußen for his role as my internship supervisor and for the many discussions that led to the development of the intermediate verification approach. His insights and critical questions helped shape the direction of this research.

I would like to thank Dr. Christian Andersen and Dr. Bas Janssens for taking the time to evaluate this work as committee members.

Further thanks go to neQxt GmbH for their warm welcome and the lab tours, and to Quantinuum for providing access to their H1-1 hardware, which made part of this research possible.

I am grateful to my friends and family for their support throughout this project, especially Saqar Khaleefah for many helpful conversations and steady encouragement.

After many early bus rides and more cans of Coke Zero than I care to count, I'm glad to see it finished.

S.C. Birkenhäger
Delft, July 2025

Summary

Logical state preparation is a fundamental step in fault-tolerant quantum computing, and logical $|0\rangle_L$ states are frequently needed throughout fault-tolerant circuits, for example at the start of computation, during error correction, or as part of magic state distillation. It is therefore important that they can be implemented efficiently. This thesis investigates *intermediate verification* (IV) as an alternative strategy, where the verification step is placed partway through the preparation circuit rather than at the end. We use the $|0\rangle_L$ state of the $[[15, 1, 3]]$ code as a case study, and construct IV circuits based on the recursive structure of its encoding circuit. We prove that such circuits remain fault-tolerant for punctured quantum Reed-Muller (PQRM) codes of the form $\mathrm{PQRM}(1, m-2, m)$ with $m \geq 4$ and $d = 3$, including the $[[15, 1, 3]]$ code as the special case $m = 4$.

We compare IV and final verification circuits under realistic noise models, evaluating their logical error rate, acceptance probability, and expected runtime. Our results show that IV does not always improve expected runtime, since the acceptance probability is already high in the relevant noise regime. However, IV could lead to reduced circuit depth after native gate compilation and scheduling. Hardware experiments on the Quantinuum H1-1 trapped-ion quantum chip confirm that the logical error rate falls below the physical error rate of this device, indicating that encoding already offers a practical benefit.

These findings highlight IV as a viable design element for logical state preparation, especially as a design feature within circuit synthesis or reinforcement learning methods that aim to minimise depth under hardware constraints.

Contents

1 Introduction

Quantum error correction (QEC) is essential for executing reliable quantum algorithms on physical devices. Physical qubits are inherently noisy, and without active error correction, quantum computations become unreliable within tens or hundreds of operations. For example, the typical two-qubit gate error rate on the Quantinuum H1-1 quantum processor, used in this thesis, is approximately 10^{-3} [1]. This implies that after roughly 10^3 operations, unencoded quantum results are no longer trustworthy. Fault-tolerant quantum computing (FTQC) aims to suppress logical error rates below such physical thresholds by encoding quantum information into stabilizer codes and executing fault-tolerant circuits. A typical encoded computation begins by preparing a logical state, such as $|0\rangle_L$ or $|+\rangle_L$, followed by a sequence of fault-tolerant gates, stabilizer measurements, and logical readout. The preparation of logical states is thus the first critical step in any encoded computation.

Several strategies for fault-tolerant logical state preparation have been developed. Deterministic methods implement preparation circuits that perform correction rather than postselection [2]. Non-deterministic approaches include postselection and repeat-until-success (RUS), which rely on verification steps to detect uncorrectable errors. If such an error is detected, the state is discarded and the procedure is repeated. These approaches enable fault-tolerant preparation using minimal ancilla overhead and moderate circuit depth, making them suitable for near-term platforms [3, 4]. However, their efficiency depends strongly on the acceptance probability of the verification step. This probability can decrease significantly with increasing code size, leading to large expected runtimes.

This thesis explores an alternative approach called *intermediate verification* (IV), in which the state is verified partway through the preparation circuit rather than at the end. In contrast to postselection or final verification, IV enables partial repetition: only the part of the circuit before the verification point needs to be repeated upon failure. Although our results show that the acceptance probability is so low that the partial repetition only marginally affects the effective runtime, the method introduces an additional design degree of freedom. By placing the verification step earlier, one can explore circuit structures that are not possible with final-only verification, potentially resulting in lower-depth implementations.

We investigate this idea through a case study: the preparation of the logical zero state $|0\rangle_L$ in the $[[15, 1, 3]]$ code. This code serves as a concrete example for analysing IV. First, its recursive structure allows its state preparation to be broken down into smaller verifiable subcircuits [5–7]. Second, it supports universal fault-tolerant computation via transversal T gates and gauge fixing or H teleportation [8, 9], and its logical zero state preparation is a part of 15 to 1 magic state distillation [10]. Third, its moderate size and distance make it a manageable candidate for formal analysis and hardware implementation. Finally, it provides a concrete setting to demonstrate IV as a general concept.

The main research question addressed in this thesis is:

> **Can intermediate verification reduce the effective runtime of repeat until success logical state preparation?**

To answer this, we present a detailed analysis of $|0\rangle_L$ state preparation in the $[[15, 1, 3]]$ code using circuits with intermediate and final verification. We compare several versions of these circuits under a realistic noise model and evaluate their performance using three metrics: logical error probability, acceptance probability, and expected runtime in circuit depth. In addition, we provide a formal proof

of the fault tolerance of the IV circuits for punctured quantum Reed-Muller (PQRM) codes, specifically PQRM$(1, m-2, m)$ with $d = 3$, and discuss the challenges of extending such proofs to larger sizes. In particular, fault tolerance proofs for $d > 3$ can become difficult due to the number of multi-fault events that must be shown to be correctable, which grows exponentially with d.

A key insight from this work is that IV does not necessarily reduce the runtime overhead of RUS circuits, but can yield lower depth implementations due to the flexibility of placing the verification step earlier. We also find that circuit optimization is only meaningful once native gate sets and hardware constraints are taken into account (see Section 5.4). These findings have implications for the design of scalable, hardware-aware fault-tolerant circuits.

This thesis is organized as follows. Chapter 2 introduces the notation used throughout the report. Chapter 3 provides background on quantum error correction, stabilizer codes, and existing preparation methods. Chapter 4 describes the construction and analysis of IV circuits, including a formal fault tolerance argument. Chapter 5 presents simulation, emulation, and hardware results comparing various preparation strategies. Chapter 6 discusses the broader implications, limitations, and potential extensions of this work. Chapter 7 concludes.

2

Notation and Basic Definitions

Let $\mathbb{F}_2$ denote the finite field with two elements, $\{0,1\}$, and let $f \in \mathbb{F}_2^n$ be a binary vector of length n. The *support* of f is defined as

$$\text{supp}(f) := \{j \in \{1,\dots,n\} : f_j = 1\},$$

and the *Hamming weight* of f is given by $|f| := |\text{supp}(f)|$, i.e., the number of ones in the vector.
We denote the all-zero vector in $\mathbb{F}_2^n$ by $\mathbf{0}$, and the all-one vector by $\mathbf{1}$.
Let

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

be the single-qubit Pauli operators. For $P \in \{X, Y, Z\}$, define the n-qubit Pauli string

$$P(f) := \bigotimes_{j=1}^{n} P_j, \quad \text{where } P_j = \begin{cases} P & \text{if } f_j = 1, \\ I & \text{if } f_j = 0. \end{cases}$$

That is, $P(f)$ acts as P on the qubits indexed by $\text{supp}(f)$, and as the identity elsewhere. This notation is used for uniform Pauli strings such as $X(f)$, $Z(f)$, or $Y(f)$. General Pauli strings with mixed types, such as $XZZX$, are written out explicitly where needed.
The *weight* of a Pauli operator refers to the number of qubits on which it acts non-trivially. For operators of the form $P(f)$, this equals the Hamming weight $|f|$.

3

Theory

3.1. Error correction codes

3.1.1. Classical linear codes

Classical codes are used to construct Calderbank-Shor-Steane (CSS) codes, which is why it is important to understand their theory. An $[n, k]$ classical linear binary code C encodes k bits of information into a codeword $\mathbf{c} = c_1 c_2 \dots c_n \in \mathbb{F}_2^n$, the n-dimensional binary vector space in which vectors are represented by n-bit strings and addition corresponds to bitwise addition modulo 2. A k-bit message $x = x_1 x_2 \dots x_k \in \mathbb{F}_2^k$ is mapped to its corresponding codeword via $\mathbf{c} = Gx$, where $G \in \mathbb{F}_2^{n \times k}$ is the generator matrix of the code. The code is thus the image of G, i.e. $C = \mathrm{im}(G)$, which forms a k-dimensional linear subspace of $\mathbb{F}_2^n$. Since $G(x + x') = Gx + Gx'$, the code is closed under addition and hence called a *linear* code [7].

A classical linear code is often described in terms of its parity check matrix H, an alternative but equivalent formulation. The $[n, k]$ code C is then defined as

$$C = \{c \in \mathbb{F}_2^n : Hc = 0\},$$

where $H \in \mathbb{F}_2^{(n-k) \times n}$. Since $\dim(C) = \dim(\mathrm{span}(Ge_1, Ge_2, \dots, Ge_k)) = k$, it follows that $\dim(\ker H) = k$, so $\mathrm{rank}(H) = n - k$. Hence, the rows of the parity check matrix are linearly independent.
The syndrome of a vector $x \in \mathbb{F}_2^n$ is given by $s = Hx$. If $x \in C$, then $Hx = \mathbf{0}$. If the syndrome is non-zero, this indicates that an error has occurred and changed a valid codeword.
The Hamming weight $|x|$ of a vector x is the number of non-zero bits, i.e. the number of entries equal to one. The Hamming distance $d(c_1, c_2)$ between two codewords $c_1, c_2 \in C$ is the weight of their difference:

$$d(c_1, c_2) = |c_1 + c_2|.$$

The distance of a code C is the minimum distance between any two distinct codewords:

$$d(C) = \min_{\substack{c_1, c_2 \in C \\ c_1 \neq c_2}} d(c_1, c_2). \tag{3.1}$$

Since C is linear, $c_1 + c_2 \in C$, so this simplifies to

$$d(C) = \min_{\substack{c \in C \\ c \neq 0}} |c|. \tag{3.2}$$

A code that encodes k bits into n bits with distance d is denoted as an $[n, k, d]$ code. The distance determines the code's error-correcting capability. Suppose $d = 2t + 1$ for some $t \in \mathbb{N}$, and a codeword $c \in C$ is corrupted by an error e of weight $|e| \leq t$, resulting in a received vector $y = c + e$. Given the syndrome $s = Hy = H(c + e) = He$, the decoder finds a minimum-weight error e' such that $He' = s$. Then $|e'| \leq |e| \leq t$, so the residual error $e + e'$ has weight at most $2t$. Since $c + e + e' \in C$ and $d(c, c + e + e') \leq 2t < d(C)$, it must follow that $c + e + e' = c$. The error is therefore corrected. The same argument applies to quantum stabilizer codes [11].
Many quantum codes are examples of stabilizer codes. We now introduce the stabilizer formalism.

3.1.2. Stabilizer codes

Stabilizer codes are defined by their *stabilizer group* $\mathcal{S}$, which is an abelian subgroup of the n-qubit Pauli group P_n that does not contain the scalar $-I$. The group $\mathcal{S}$ is generated by $n-k$ linearly independent Pauli operators:

$$\mathcal{S} = \langle s_1, \dots, s_{n-k} \rangle \subseteq P_n, \tag{3.3}$$

where $\langle s_1, \dots, s_{n-k} \rangle$ denotes the group generated by these elements, i.e., all finite products of the s_i and their scalar multiples.
The Pauli group P_n is the n-fold tensor product of the single-qubit Pauli group P_1, which is defined as

$$P_1 \equiv \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}. \tag{3.4}$$

The code space is then defined as:

$$C = \{|\psi\rangle : s\,|\psi\rangle = |\psi\rangle, \forall s \in \mathcal{S}.\} \tag{3.5}$$

If $\dim(C) = 2^k$, then C has the basis $|(0,\dots,0)_L\rangle, |(0,\dots,1)_L\rangle, \dots, |(1,\dots,1)_L\rangle$, which are called the logical states. Then the centralizer of the stabilizer group $\mathcal{S}$ in P_n is defined as:

$$C_{P_n}(\mathcal{S}) = \{P \in P_n : Ps = sP, \forall s \in \mathcal{S}\} = \langle s_1, \dots, s_{n-k}, \overline{X_1}, \dots, \overline{X_k}, \overline{Z_1}, \dots, \overline{Z_k} \rangle. \tag{3.6}$$

The operators $\overline{X_1}, \dots, \overline{X_k}, \overline{Z_1}, \dots, \overline{Z_k}$ are the logical operators of the code. For any $i, j \in [k]$, where $i \neq j$:

$$\overline{X_i}\,|(x_1,\dots,x_{i-1},0,\dots,x_k)_L\rangle = |(x_1,\dots,x_{i-1},1,\dots,x_k)_L\rangle \tag{3.7}$$

$$\overline{X_i}\,|(x_1,\dots,x_{i-1},1,\dots,x_k)_L\rangle = |(x_1,\dots,x_{i-1},0,\dots,x_k)_L\rangle \tag{3.8}$$

$$\overline{Z_i}\,|(x_1,\dots,x_{i-1},+,\dots,x_k)_L\rangle = |(x_1,\dots,x_{i-1},-,\dots,x_k)_L\rangle \tag{3.9}$$

$$\overline{Z_i}\,|(x_1,\dots,x_{i-1},-,\dots,x_k)_L\rangle = -\,|(x_1,\dots,x_{i-1},+,\dots,x_k)_L\rangle \tag{3.10}$$

$$\overline{X_iX_i} = I_n \tag{3.11}$$

$$\overline{X_iX_j} = \overline{X_jX_i} \tag{3.12}$$

$$\overline{Z_iZ_i} = I_n \tag{3.13}$$

$$\overline{Z_iZ_j} = \overline{Z_jZ_i} \tag{3.14}$$

$$\overline{X_iZ_i} = -\overline{Z_iX_i} \tag{3.15}$$

$$\overline{X_iZ_j} = \overline{Z_jX_i} \tag{3.16}$$

Binary check matrix Each stabilizer generator $s \in \mathcal{S}$ can be written as a Pauli operator of the form

$$s = X(\mathbf{f})Z(\mathbf{g}),$$

where $\mathbf{f}, \mathbf{g} \in \mathbb{F}_2^n$. This notation means that the operator acts with an X on qubit j if $f_j = 1$, with a Z if $g_j = 1$, and with a Y if both $f_j = g_j = 1$. This binary symplectic representation captures only the tensor structure of the Pauli operator and omits global phases such as ± 1 or $\pm i$. As a result, it cannot distinguish between s and $-s$, even though these define different stabilizer groups (e.g., $\{+ZZ, +XX\}$ and $\{-ZZ, -XX\}$ stabilize the states $(|00\rangle + |11\rangle)/\sqrt{2}$ and $(|01\rangle - |10\rangle)/\sqrt{2}$, respectively). Throughout this thesis, we consistently treat stabilizers *up to global phase*, following the standard convention in the literature [12, Section III.B].
Then, the stabilizer code can be compactly represented by a binary check matrix $H \in \mathbb{F}_2^{(n-k)\times 2n}$, whose rows are the concatenated vectors $(\mathbf{f} \mid \mathbf{g})$ for each generator [11].

Logical states and state preparation. Let $C \subseteq (\mathbb{C}^2)^{\otimes n}$ denote the code space stabilized by the group $\mathcal{S}$. If the code encodes k logical qubits, then $\dim C = 2^k$, and there exists a logical basis

$$\{|(x_1,\dots,x_k)_L\rangle : x_i \in \{0,1\}\}$$

consisting of simultaneous eigenstates of all stabilizers and of the logical operators $\overline{Z_i}$, for $i = 1, \dots, k$.

For example, in the case $k = 1$, the state $|0_L\rangle \in C$ is a $+1$ eigenstate of $\overline{Z}$ and all $s \in \mathcal{S}$, while $|1_L\rangle$ is a -1 eigenstate of $\overline{Z}$. Similarly, $|+_L\rangle$ and $|-_L\rangle$ are eigenstates of $\overline{X}$. For $k = 3$, the state $|(0,0,0)_L\rangle$ is stabilized by $\mathcal{S}$ and is a $+1$ eigenstate of each of the operators $\overline{Z_1}$, $\overline{Z_2}$, and $\overline{Z_3}$.
The first k qubits of the input product state encode the logical state $|\psi\rangle$. The encoding circuit C then maps the state

$$|\psi_1 \dots \psi_k\rangle \otimes |0\rangle^{\otimes(n-k)} \mapsto |\overline{\psi}\rangle,$$

where the last $n-k$ qubits are initialized in the $|0\rangle$ state. This convention, for example found in [13], corresponds to preparing the last qubits as +1 eigenstates of the Z-type stabilizers.
In the encoding circuits studied in this thesis, however, the last $n-k$ qubits are often initialized in either $|0\rangle$ or $|+\rangle$ states, depending on the number of Z- and X-type stabilizers present. This choice reflects the CSS code structure, where qubits associated with Z-stabilizers are initialized in $|0\rangle$, and those associated with X-stabilizers in $|+\rangle$. This ensures that each ancillary qubit starts as a +1 eigenstate of the corresponding stabilizer type. These single-qubit stabilizers then get mapped, via the encoding circuit, to the full stabilizer generators of the code.
Thus, the input to the encoding circuit is generally of the form

$$|\psi_1 \dots \psi_k\rangle \otimes \bigotimes_{j=1}^{n-k} |\phi_j\rangle,$$

where each $|\phi_j\rangle$ is either $|0\rangle$ or $|+\rangle$, selected according to the corresponding stabilizer type.
Note that such product states are not necessarily simultaneous eigenstates of all stabilizer generators, but they serve as convenient starting points for the encoding circuits and logical state preparations presented throughout this work.
These input product states are then transformed into logical states using encoding circuits.
Such circuits implement *logical state preparation*. One straightforward method to construct an encoding circuit for a stabilizer code is the *Latin rectangle method*, which uses the binary check matrix H of the stabilizer group to determine a sequence of CNOT gates that maps a product state to a state stabilized by $\mathcal{S}$ [14]. The required initialization of qubits in $|0\rangle$ or $|+\rangle$ is also determined by the structure of H, though we will not go into detail here. In general, these circuits are not fault-tolerant. Fault-tolerant constructions are discussed in Section 3.2.4.

Logical gates A physical unitary U induces a logical gate $\overline{U}$ on the code space if it maps the code space to itself:

$$UC = C,$$

where $C = \{|\psi\rangle \in (\mathbb{C}^2)^{\otimes n} : s\,|\psi\rangle = |\psi\rangle,\ \forall s \in \mathcal{S}\}$ is the codespace stabilized by $\mathcal{S}$. This ensures that U maps valid encoded states to valid encoded states. The induced map on logical states is then called $\overline{U}$.
A sufficient (but not necessary) condition for U to induce a logical gate is that it preserves the stabilizer group under conjugation:

$$U\mathcal{S}U^\dagger = \mathcal{S}.$$

However, this condition is too strict in general. For example, the transversal T gate on the $[[15,1,3]]$ code maps the stabilizer group $\mathcal{S}$ to a different group $\mathcal{S}'$, but the codespace remains unchanged. That is, the codespace stabilized by $\mathcal{S}$ is identical to that stabilized by $\mathcal{S}'$, so U still defines a valid logical operator.
In addition, we require that the logical operators transform correctly under conjugation. For example, if $UXU^\dagger = V$, then

$$\overline{U}\,\overline{X}\,\overline{U}^\dagger = \overline{V},$$

and similarly for Z-type operators:

$$UZU^\dagger = W \quad \Rightarrow \quad \overline{U}\,\overline{Z}\,\overline{U}^\dagger = \overline{W}.$$

This definition ensures that the action of $\overline{U}$ on logical information mirrors the action of U on physical qubits. For further details, see [15]. .

Error correction

Suppose there is some logical state $|\overline{\psi}\rangle \in C$. Some error $E \in P_n$ occurs and now the state is $E|\overline{\psi}\rangle$. If $E \in \mathcal{S}$, there is no problem: $E|\overline{\psi}\rangle = |\overline{\psi}\rangle$. If $E \in C(\mathcal{S}) \setminus \mathcal{S}$, $E|\overline{\psi}\rangle = |\overline{\psi'}\rangle$, where $\psi' \neq \psi$. As a result, the encoded logical state differs from the original logical state. Since $|\overline{\psi'}\rangle \in C$, $s|\overline{\psi'}\rangle = |\overline{\psi'}\rangle, \forall s \in \mathcal{S}$, which means that by measuring the eigenvalues of the stabilizers, it is not possible to know whether $E \in \mathcal{S}$ or $E \in C(\mathcal{S})$.

Suppose some error $E \in P_n \setminus C(\mathcal{S})$ occurs and the state is now $E|\overline{\psi}\rangle \notin C$. Since the corrupted state is no longer in the code space, its eigenvalues with respect to the stabilizer generators of $\mathcal{S}$ are not all equal to 1. This set of eigenvalues is called the *syndrome* of the state, $\vec{s} \in \{-1, 1\}^{n-k}$. Measuring the stabilizers yields a syndrome that indicates which stabilizers have been violated by an error. In other words, the syndrome reveals the subset of stabilizers that no longer have eigenvalue $+1$ under the current (possibly corrupted) state. This provides indirect information about the error without collapsing the encoded quantum information. Suppose that error $E' \neq E$ has the same syndrome, and instead of E, E' is used to correct the error, and the result is $E'E|\overline{\psi}\rangle$.

The process of interpreting a measured syndrome and choosing a correction operator is performed by a *decoder*, which typically selects an error of minimum weight consistent with the syndrome. For small codes, this can be done using a lookup table that maps each syndrome to a corresponding correction [16].

Note that the syndrome does not depend on the original state $|\overline{\psi}\rangle$, as $sE|\overline{\psi}\rangle = \pm Es|\overline{\psi}\rangle = \pm|\overline{\psi}\rangle$. So the syndrome depends solely on whether the error itself commutes or anticommutes with the stabilizer generators. If E' and E have the same syndrome, $E'E$ has syndrome $\vec{s} \circ \vec{s} = \mathbf{1}$, the all-ones vector. Here $(\vec{s} \circ \vec{s})_i := s_i\, s_i, \quad i = 1, \ldots, n-k$. This means that either:

1. $E'E \in \mathcal{S}$, so $E'E|\overline{\psi}\rangle = |\overline{\psi}\rangle$, which means the correction was successful, The correction is considered successful if the combined operator $E'E$ lies in the stabilizer group, i.e., $E'E \in \mathcal{S}$, in which case the state is returned to the original code space without a logical error.
2. $E'E \in C(\mathcal{S}) \setminus \mathcal{S}$, so $E'E|\overline{\psi}\rangle \neq |\overline{\psi}\rangle$, which means the correction failed.

A single cycle of quantum error correction consists of measuring the stabilizer syndromes and applying a correction based on the outcome. In general, syndrome extraction is repeated multiple times, since the measurements themselves may be faulty. This repetition helps detect measurement errors before a correction is applied. The required number of rounds typically increases with the code distance [17].

Distance The distance of the code is defined as

$$d = \min_{\substack{P \in C(\mathcal{S}) \setminus \mathcal{S} \\ P \not\propto I}} |P|,$$

excluding P being proportional to the identity operators in $C(\mathcal{S})$. Here, $|P|$ denotes the number of qubits on which P acts non-trivially.

Suppose that $|E| \leq t$, and $d \geq 2t + 1$, then if one finds the minimum weight $E' \in P_n \setminus C(\mathcal{S})$, such that its syndrome is equal to that of E, this means that $|E'| \leq |E| \leq t$ and $|E'E| \leq 2t$. But the syndrome of $E'E$ is the all-ones vector, so $E'E \in C(\mathcal{S})$. Since $|E'E| < d$, this means $E'E|\overline{\psi}\rangle = |\overline{\psi}\rangle$ and the correction is successful.

Quantum error correction codes are described by $[[n, k, d]]$, with the number of qubits n, the encoded number of bits k and the distance d of the code [18].

Example: check matrix of the $[[5, 1, 3]]$ code. The $[[5, 1, 3]]$ code is the smallest quantum error-correcting code that can correct an arbitrary single-qubit error. It has four stabilizer generators:

$$S_1 = XZZXI,$$
$$S_2 = IXZZX,$$
$$S_3 = XIXZZ,$$
$$S_4 = ZXIXZ,$$

which correspond to the rows of the following 4×10 binary check matrix:

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Each row encodes a stabilizer $s = X(f)Z(g)$ via the concatenated binary vector $(f \mid g)$.
The code distance is $d = 3$, which is the minimum weight of any operator in $C(\mathcal{S}) \setminus \mathcal{S}$, i.e., a non-trivial logical operator. Although the transversal operators $\overline{X} = X^{\otimes 5}$ and $\overline{Z} = Z^{\otimes 5}$ have weight 5, there exist equivalent representatives of weight 3. For example:

$$\overline{X} = XXIZI, \qquad \overline{Z} = ZXZII.$$

Both operators commute with all stabilizer generators and anticommute with each other. This confirms that they act as logical Pauli operators on the encoded qubit [19].

Pauli propagation through CNOT gates
Errors may propagate through gates and increase in weight. In particular, Pauli errors propagate through the CNOT gate according to the following rules.
Let X_c and Z_c denote Pauli operators acting on the control qubit, and X_t, Z_t those acting on the target. Then:

$$\begin{aligned} \mathrm{CNOT}^\dagger (X_c \otimes I_t)\,\mathrm{CNOT} &= X_c \otimes X_t, \\ \mathrm{CNOT}^\dagger (I_c \otimes X_t)\,\mathrm{CNOT} &= I_c \otimes X_t, \\ \mathrm{CNOT}^\dagger (Z_c \otimes I_t)\,\mathrm{CNOT} &= Z_c \otimes I_t, \\ \mathrm{CNOT}^\dagger (I_c \otimes Z_t)\,\mathrm{CNOT} &= Z_c \otimes Z_t. \end{aligned} \tag{3.17}$$

These identities show how X errors propagate forward (from control to target) and Z errors backward (from target to control).

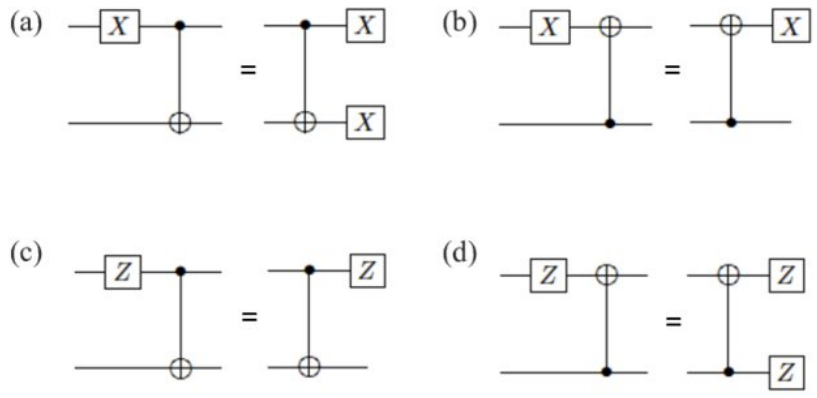


Figure 3.1: Propagation of Pauli X and Z errors through a CNOT gate. (a) and (b): X error propagation. (c) and (d): Z error propagation. Figure from [20].

These propagation rules are essential when analyzing encoding circuits built from layers of CNOT gates, such as in QRM and PQRM codes. In the following sections, we use them to derive the structure of stabilizers and logical operators.

CSS codes
A large class of stabilizer codes is the class of CSS codes. Given a classical binary code $C \subseteq \mathbb{F}_2^n$, its dual code is defined as

$$C^\perp := \{x \in \mathbb{F}_2^n : \langle x, c \rangle = 0 \text{ for all } c \in C\},$$

where $\langle \cdot, \cdot \rangle$ denotes the standard inner product modulo 2. To construct a CSS code, one uses two classical binary codes C_X and C_Z satisfying a dual inclusion condition: an $[n, k_X, d_X]$ code C_X and an $[n, k_Z, d_Z]$ code C_Z. A necessary condition for the CSS construction is

$$C_Z^\perp \subseteq C_X \quad \text{(equivalently } C_X^\perp \subseteq C_Z\text{)}.$$

In this case, the quantum parity-check matrix H takes a simple block form:

$$H = \begin{pmatrix} H_X & 0 \\ 0 & H_Z \end{pmatrix},$$

where H_X is the parity-check matrix of C_X and H_Z of C_Z.
The orthogonality condition $H_X H_Z^T = 0$ implies that all stabilizer generators commute, since each X-type generator overlaps with each Z-type generator in an even number of positions. This condition is equivalent to the dual inclusion $C_Z^\perp \subseteq C_X$, since each row of H_Z lies in $C_Z^\perp$ and each row of H_X in $C_X^\perp$; orthogonality of these rows thus implies $H_X H_Z^T = 0$ [21].
The *dual distances* of a CSS code quantify the weight of the smallest undetectable X- or Z-type logical operator. They are respectively defined as

$$d_Z^+ := \min_{f \in C_Z \setminus C_X^\perp} \mathrm{wt}(f), \quad d_X^+ := \min_{f \in C_X \setminus C_Z^\perp} \mathrm{wt}(f).$$

These parameters satisfy $d_X^+ \geq d_X$ and $d_Z^+ \geq d_Z$, where $d_X := d(C_X)$ and $d_Z := d(C_Z)$ denote the classical distances of the constituent codes. The overall quantum code distance is then given by [21]

$$d := \min(d_X^+, d_Z^+),$$

which guarantees that all Pauli errors of weight at most

$$t := \left\lfloor \frac{d-1}{2} \right\rfloor$$

are correctable.
The naming convention for d_Z^+ and d_X^+ can be confusing at first sight, as d_Z^+ refers to X-type errors, and d_X^+ to Z-type errors. This becomes clear by inspecting the stabilizer construction of CSS codes. The X-type stabilizers have support vectors in the dual code $C_X^\perp$, and the Z-type stabilizers have support vectors in $C_Z^\perp$.
An X-error is undetectable if it commutes with all Z-type stabilizers, which is the case when its support vector lies in C_Z. However, such an error only acts non trivially on the code space if its support vector does not lie in $C_X^\perp$. The undetectable X-errors that are not stabilizers therefore have support vectors in the set $C_Z \setminus C_X^\perp$, and their minimum weight is d_Z^+. The reasoning for Z-type errors is fully analogous: non-trivial undetectable Z-errors have support vectors in $C_X \setminus C_Z^\perp$, and their minimum weight is d_X^+.
To reason explicitly about the correctability of X- and Z-type errors, we define the parameters

$$t_X := \left\lfloor \frac{d_Z^+ - 1}{2} \right\rfloor, \quad t_Z := \left\lfloor \frac{d_X^+ - 1}{2} \right\rfloor. \tag{3.18}$$

Here, the subscript refers to the type of Pauli error (i.e., X or Z), not to the stabilizer type as in d_X^+ and d_Z^+. These parameters are not standard in the literature, but will be used throughout this thesis to indicate that all X- or Z-type errors of weight at most t_X or t_Z are guaranteed to be correctable by the code that is being discussed.
We now turn to a concrete instance of the CSS construction: classical Reed-Muller (RM) codes, and the CSS codes that can be constructed with RM codes.

3.1.3. Reed-Muller codes (classical)

RM codes form a well-studied family of classical linear codes [22]. This section introduces their definition, parameters, and duality properties, with a focus on the identity $\mathrm{RM}(r,m)^\perp = \mathrm{RM}(m-r-1,m)$, which underlies the orthogonality conditions used in the CSS construction. We also define punctured

RM codes and their even-weight subcodes, which will later be used to define stabilizer groups in punctured quantum Reed–Muller codes. Finally, we describe the recursive encoding structure of RM codes based on hypercube geometry, which will be used to construct encoding circuits in the quantum setting.

A classical RM code $\mathrm{RM}(r,m)$ is a binary linear code of length $n = 2^m$, dimension $k = \sum_{j=0}^{r} \binom{m}{j}$, and distance $d = 2^{m-r}$, so it is a $\left[2^m,\ \sum_{j=0}^{r} \binom{m}{j},\ 2^{m-r}\right]$ code.

A classical punctured Reed–Muller code $\mathrm{RM}^*(r,m)$ is defined only for $r < m$, and is obtained by removing the bit labelled by $00\ldots0$, corresponding to the hypercube coordinate $00\ldots0 \in \mathbb{F}_2^m$, from each codeword of $\mathrm{RM}(r,m)$. This bit corresponds to the last entry of the codeword under lexicographic ordering of the hypercube coordinates from $(1,\ldots,1)$ to $(0,\ldots,0)$. This yields a code of length $2^m - 1$ with parameters

$$\left[2^m - 1,\ \sum_{j=0}^{r} \binom{m}{j},\ 2^{m-r} - 1\right].$$

Its even-weight subcode, denoted $\overline{\mathrm{RM}}(r,m)$, consists of all codewords in $\mathrm{RM}^*(r,m)$ with even Hamming weight. It is defined as:

$$\overline{\mathrm{RM}}(r,m) := \left\{c \in \mathrm{RM}^*(r,m) : |c| \equiv 0 \mod 2\right\}. \tag{3.19}$$

This even-weight subcode has parameters

$$\left[2^m - 1,\ \sum_{j=0}^{r} \binom{m}{j} - 1,\ 2^{m-r}\right].$$

The dimension decreases by one because the all-ones vector is the only generator with odd weight, and it is excluded from the even-weight subcode. The distance remains 2^{m-r}, since this vector is also the only one with a 1 at the punctured coordinate $00\ldots0$, so all remaining codewords retain their weight.

Encoding of classical Reed-Muller codes.

Classical $\mathrm{RM}(r,m)$ codes can be defined via recursive encoding circuits, based on a hypercube geometry, or equivalently via an explicit generator matrix $G = F^{\otimes m}$ as we explain below. Here,

$$F := \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{3.20}$$

is the generator matrix of $\mathrm{RM}(1,1)$. The full matrix $G = F^{\otimes m}$ generates the space $\mathrm{RM}(m,m)$.

Figure 3.2 shows a hypercube encoding circuit for $n = 2^m$ bits with $m = 3$. The hypercube encoding circuit prepares RM codewords using a recursive structure. To obtain the circuit for 2^m qubits, one builds on the circuit for 2^{m-1} by applying transversal CNOT gates between corresponding qubits in two copies. The qubits are labelled by binary strings of length m, and CNOT gates connect qubits whose labels differ in one bit. This structure mirrors the geometry of the m-dimensional hypercube and directly leads to the encoding matrix $F^{\otimes m}$.

The inputs b_1 through b_4 represent the information bits of the $\mathrm{RM}(1,3)$ code, which encodes four bits, since its dimension k equals $\binom{3}{0} + \binom{3}{1} = 4$. The remaining bits are initialized to zero to ensure that the output satisfies the parity constraints and lies in the code. This initialization structure also appears in the quantum setting, where a similar labelling determines which qubits are fixed and which carry logical information.

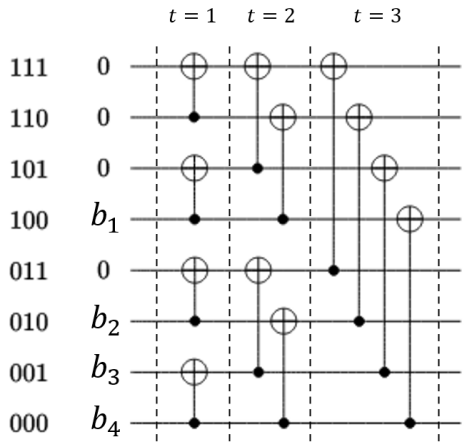


Figure 3.2: Classical hypercube encoding circuit for $\mathrm{RM}(1,3)$ with $m = 3$. Each column t corresponds to a layer of CNOT gates between bit pairs whose binary labels differ at bit t. The four information bits $b_1, \ldots, b_4$ correspond to labels of Hamming weight ≤ 1. The remaining bits are frozen to 0 to ensure the output lies in the code. The information bit labels are the 3-bit strings of Hamming weight at most 1, corresponding to monomials of degree at most $r = 1$ [7].

The action of each CNOT layer in the encoding circuit can be described by a simple 2×2 matrix:

$$F = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix},$$

which implements the classical transformation $(a, b) \mapsto (a, a \oplus b)$ on two bits $(a, b) \in \mathbb{F}_2^2$.
The full encoding circuit acts on 2^m bits and corresponds to the m-fold tensor product

$$F^{\otimes m} := F \otimes F \otimes \cdots \otimes F,$$

which results in a $2^m \times 2^m$ binary matrix. This matrix acts on the input bit vector $x \in \mathbb{F}_2^{2^m}$ by classical matrix multiplication, producing the codeword Gx. The output is the bit string that results from propagating the input values through the recursive CNOT layers of the circuit. Each row of this matrix describes how a particular monomial (labelled by a binary string of length m) is evaluated on all 2^m inputs in $\mathbb{F}_2^m$.
For example, for $m = 2$, we have

$$F^{\otimes 2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

Here, the rows and columns are indexed by binary strings 00, 01, 10, and 11 in lexicographic order. Row x shows the evaluation of the monomial labelled by x on all possible inputs y.
Each row of $G_{\mathrm{RM}}(m, m)$ corresponds to a Boolean monomial in m variables. In each layer, the circuit connects bits located at hypercube vertices that differ in one coordinate direction, reflecting the geometry of $\mathbb{F}_2^m$ and the recursive structure of $F^{\otimes m}$. Figure 3.3 shows $G_{\mathrm{RM}}(3, 3)$, the generator matrix of $\mathrm{RM}(3, 3)$ [6].
The code $\mathrm{RM}(r, m)$ is defined by selecting from $G_{\mathrm{RM}}(m, m)$ the rows labelled by Boolean monomials of degree at most r. Each such row corresponds to the evaluation vector of a monomial: it lists the values of that monomial on all 2^m input vectors in $\mathbb{F}_2^m$, ordered from $(1, \ldots, 1)$ to $(0, \ldots, 0)$. A monomial of degree $s \leq r$ evaluates to 1 on exactly 2^{m-s} of these inputs, so its evaluation vector has Hamming weight $2^{m-s} \geq 2^{m-r}$.
For instance, $\mathrm{RM}(1, 3)$ consists of the rows with labels 000, 001, 010, and 100, which are the bottom three and the fifth-from-bottom row of $G_{\mathrm{RM}}(3, 3)$, corresponding to the monomials 1, x_1, x_2, and x_3. These form a $[8, 4, 4]$ linear code.

$$
\begin{array}{cc|cccccccc}
 & & x_3 = & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
 & & x_2 = & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
 & & x_1 = & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
\hline
111 & x_3x_2x_1 & & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
110 & x_3x_2 & & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
101 & x_3x_1 & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
100 & x_3 & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
011 & x_2x_1 & & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
010 & x_2 & & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
001 & x_1 & & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
000 & 1 & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
$$

Figure 3.3: Generator matrix $G_{\mathrm{RM}}(3,3) = F^{\otimes 3}$ with $F = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$, where each row corresponds to a Boolean monomial over three variables. Subcodes $\mathrm{RM}(r,3)$ are formed by selecting rows with Hamming weight $\geq 2^{3-r}$ [7].

Note that $G_{\mathrm{RM}}(m,m)$ is lower triangular with non-zero diagonal entries, so its rows are linearly independent and form a valid generator set. Together, Figures 3.2 and 3.3 illustrate the recursive structure of $\mathrm{RM}(1,3)$.

The $\mathrm{RM}(r,m)$ codes, the punctured $\mathrm{RM}^*(r,m)$ codes, and the even-weight subcodes $\overline{\mathrm{RM}}(r,m)$ of the punctured codes, will be used to construct CSS codes in Section 3.1.4.1. The punctured $\mathrm{RM}^*(r,m)$ codes are constructed by starting with a $\mathrm{RM}(r,m)$ with $r < m$, and then deleting the bottom bit of Figure 3.2, or equivalently the rightmost column of Figure 3.3. All rows of $G_{\mathrm{RM}}(r,m)$ have even weight when $r < m$; for $r = m$, the top row is odd weight. When the rightmost column is deleted, only the weight of the bottom row changes. This means that the even-weight subcode $\overline{\mathrm{RM}}(r,m)$ of the punctured code $\mathrm{RM}^*(r,m)$ has the same codeword generators except for the last row [7].

The recursive structure of these classical codes forms the foundation for the encoding circuits of QRM and PQRM codes, as we discuss next in the quantum setting.

3.1.4. Quantum Reed-Muller codes (QRM)

QRM codes are CSS codes constructed from classical RM codes via a pair of mutually orthogonal RM codes. This section introduces the definition and parameters of QRM codes, and explains how the classical duality $\mathrm{RM}(r,m)^\perp = \mathrm{RM}(m-r-1,m)$ ensures commutation of the X- and Z-type stabilizers. The encoding circuits for punctured QRM codes are derived from those of QRM codes by omitting one qubit and its associated gates.

QRM codes are defined as CSS codes based on classical RM codes. We define

$$\mathrm{QRM}(r_X, r_Z, m) := \mathrm{CSS}\big(\mathrm{RM}(m-r_X-1,m),\ \mathrm{RM}(m-r_Z-1,m)\big),$$

subject to $0 \leq r_X < m - r_Z$, which enforces the orthogonality conditions

$$\mathrm{RM}(r_Z,m) \subseteq \mathrm{RM}(r_X,m)^\perp \quad \text{and} \quad \mathrm{RM}(r_X,m) \subseteq \mathrm{RM}(r_Z,m)^\perp,$$

or equivalently $r_X + r_Z \leq m-1$, ensuring that all X– and Z–stabilizers commute [5, §5.2].

These orthogonality conditions are satisfied due to the identity

$$\mathrm{RM}(r,m)^\perp = \mathrm{RM}(m-r-1,m),$$

which follows from the fact that codewords in $\mathrm{RM}(r,m)$ are evaluation vectors of Boolean monomials (see Section 3.1.3). The product of two such monomials, of degrees r and $m-r-1$, has degree at most $m-1$ and thus evaluates to 1 on an even number of inputs. Therefore, their inner product is zero, and the corresponding codewords are orthogonal.

The code acts on $n = 2^m$ qubits. The number of logical qubits equals the number of Boolean monomials of degree i with $r_X < i < m - r_Z$, namely:

$$k = \sum_{i=r_X+1}^{m-r_Z-1} \binom{m}{i}.$$

Each such monomial defines a function whose evaluation vector lies in the code space but is orthogonal to all stabilizer generators.
The code has parameters

$$\left[\left[2^m,\ \sum_{i=r_X+1}^{m-r_Z-1}\binom{m}{i},\ \min(2^{r_X+1},\ 2^{r_Z+1})\right]\right].$$

By removing one qubit, typically the one labelled 000, a punctured quantum Reed-Muller (PQRM) code is obtained. This yields a code acting on $2^m - 1$ qubits. However, puncturing breaks the commutation relations between X- and Z-type stabilizers: the constant monomial corresponds to the all-ones vector, which becomes odd-weight after puncturing. It defines Pauli operators X_{all} and Z_{all} that anticommute and cannot both be part of the stabilizer group.
To restore commutativity, we restrict the stabilizers to the even-weight subcodes of the punctured codes. These subcodes are defined by removing the all-ones vector, which is the *only* odd-weight codeword in $\mathrm{RM}^*(r,m)$. This restriction ensures that all X- and Z-type stabilizers commute after puncturing, as required in any CSS code construction.
The resulting PQRM code is defined as

$$\mathrm{PQRM}(r_X, r_Z, m) = \mathrm{CSS}\left(\mathrm{RM}(m-r_X-1,m)^*,\ \mathrm{RM}(m-r_Z-1,m)^*\right),$$

with stabilizer groups given by the duals:

$$\mathcal{S}_X = (\mathrm{RM}(m-r_X-1,m)^*)^\perp = \overline{\mathrm{RM}}(r_X,m), \quad \mathcal{S}_Z = (\mathrm{RM}(m-r_Z-1,m)^*)^\perp = \overline{\mathrm{RM}}(r_Z,m),$$

where $\overline{\mathrm{RM}}(r,m)$ denotes the even-weight subcode of $\mathrm{RM}^*(r,m)$, as defined in Equation (3.19).
The resulting code has parameters

$$\left[\left[2^m-1,\ 1+\sum_{i=r_X+1}^{m-r_Z-1}\binom{m}{i},\ \min(2^{r_X+1},\ 2^{r_Z+1})-1\right]\right], \tag{3.21}$$

as shown in [7].
For QRM codes, the minimum weight of undetectable Z- and X-errors are respectively

$$d_X^+ = 2^{r_X+1}, \qquad d_Z^+ = 2^{r_Z+1}.$$

After puncturing, these quantities decrease by one:

$$d_X^+ = 2^{r_X+1} - 1, \qquad d_Z^+ = 2^{r_Z+1} - 1. \tag{3.22}$$

These quantities will be used in Sections 4.4, 4.4.1, 4.7.1, and 4.8 to reason about error propagation and correctability.
The properties of QRM and PQRM codes are further explained in Sections 3.1.4.1 and 3.1.5, respectively.

Encoding and stabilizers of QRM codes
Quantum Reed–Muller codes are CSS codes as defined in Section 3.1.4. In this section, we describe their circuit structure and stabilizer propagation in more detail. A QRM code is denoted $\mathrm{QRM}(r_X, r_Z, m)$, where $r_X, r_Z \in \mathbb{N}_0$ and $m \in \mathbb{N}_+$. The X- and Z-type stabilizers are defined using classical Reed–Muller codes $\mathrm{RM}(r_X, m)$ and $\mathrm{RM}(r_Z, m)$, respectively. This yields a quantum CSS code with parameters $[[2^m, k, d]]$, where k and d are determined by the dimensions and distances of the classical codes [5]. For further details on the CSS construction and the derivation of these parameters, see Section 3.1.4.

Encoding circuit and stabilizer structure
QRM codes admit a recursive encoding circuit derived from the structure of classical RM codes. Qubits whose binary labels have Hamming weight $\leq r_X$ are initialized in $|+\rangle$, and those with weight $\geq m - r_Z$ in $|0\rangle$. The remaining qubits, with Hamming weights between $r_X + 1$ and $m - r_Z - 1$, carry the logical input

states $|\varphi_i\rangle$ for $i = 1, \dots, k$. The stabilizers of the resulting quantum state are derived by propagating the single-qubit stabilizers through the circuit [7].
The set of X-stabilizers generated by the circuit corresponds to the classical code $\mathrm{RM}(r_X, m)$, while the Z-stabilizers correspond to $\mathrm{RM}(r_Z, m)$. These codes satisfy the CSS orthogonality condition

$$\mathrm{RM}(r_Z, m) \subseteq \mathrm{RM}(r_X, m)^{\perp}$$

whenever $r_X + r_Z \leq m - 1$, which is equivalent to $r_Z \leq m - r_X - 1$, using the identity $\mathrm{RM}(r_X, m)^{\perp} = \mathrm{RM}(m - r_X - 1, m)$. This inclusion implies that all X- and Z-type stabilizers commute, which should always hold for a (in this case CSS) stabilizer code.
A more formal and geometric perspective on this stabilizer propagation can be found in [5], where the code structure is described in terms of subcubes of the Boolean hypercube.

Deriving the stabilizers.
For an X-type stabilizer, the input $|+\rangle$ state is stabilized by X, which propagates as a row of $F^{\otimes m}$. Therefore, the X-stabilizers at the output correspond to the rows of $F^{\otimes m}$ indexed by binary strings of weight $\leq r_X$, generating $\mathrm{RM}(r_X, m)$.
The Z-type stabilizers propagate backwards through the encoding circuit and again form the code $\mathrm{RM}(r_Z, m)$ at the output, consistent with the CSS construction.
This stabilizer propagation is illustrated in Figure 3.4, which shows a QRM encoding circuit for $m = 3$, $r_X = 0$, and $r_Z = 1$. Qubits with Hamming weight $\leq r_X$ are initialized in $|+\rangle$; those with weight $\geq m - r_Z$ in $|0\rangle$; and the remaining qubits carry the logical input state.
The figure shows how a Z-type stabilizer initially supported on one qubit (here: label 101) propagates through three layers of CNOT gates to multiple output qubits. This demonstrates how single-qubit input stabilizers become multiqubit stabilizers consistent with the generator matrix of $\mathrm{RM}(r_Z, m)$. A similar mechanism applies for X-type stabilizers propagating from $|+\rangle$ inputs.

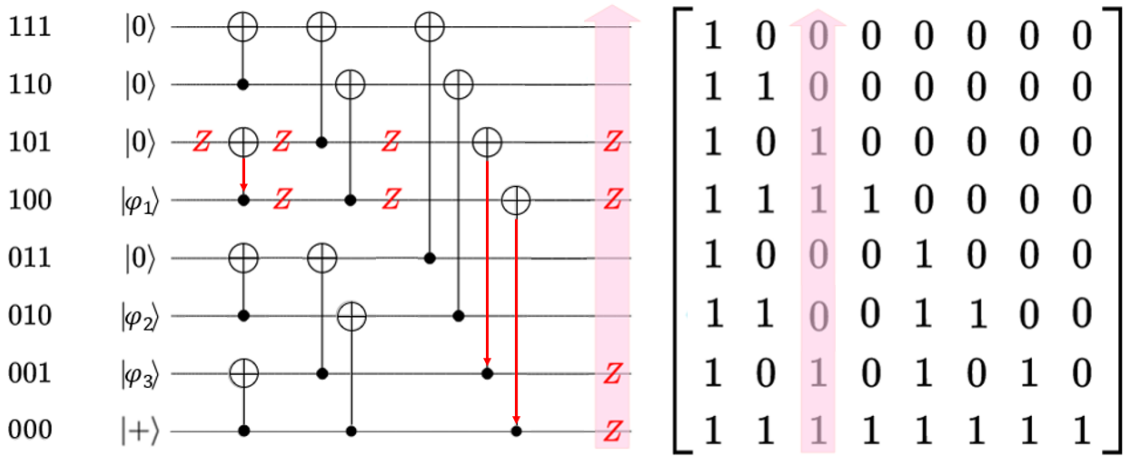


Figure 3.4: Encoding circuit for the $[[8, 3, 2]]$ code with $m = 3$, $r_X = 0$, and $r_Z = 1$. Qubits of Hamming weight $\leq r_X$ (here only 000) are initialized in $|+\rangle$; those with weight $\geq m - r_Z$ in $|0\rangle$. The three qubits labelled 100, 010, and 001 carry the logical input states $|\varphi_1\rangle, |\varphi_2\rangle, |\varphi_3\rangle$. The red Z operators illustrate the propagation of an initial Z_{101} stabilizer through the three layers of the circuit: from 101 to 100 in layer 1, and then further to 001 and 000 in layer 3. On the right, the corresponding propagated stabilizer is highlighted in the generator matrix of $\mathrm{RM}(3, 3)$. Figure adapted from [7].

3.1.5. Punctured QRM codes (PQRM)

Punctured quantum Reed–Muller (PQRM) codes are defined by puncturing a QRM code. This changes the stabilizer structure and increases the number of logical qubits. To preserve commutation between X- and Z-type stabilizers, the stabilizer generators are restricted to even-weight subcodes of the punctured RM codes. The resulting codes retain the recursive encoding structure of QRM codes and will play a central role in this thesis, in particular for encoding the $[[15, 1, 3]]$ code and its generalizations to $\mathrm{PQRM}(1, m-2, m)$.
We now describe how puncturing affects the stabilizer group and the number of logical qubits, and illustrate the resulting encoding circuit with a concrete example.

Stabilizer group. Before puncturing, the X and Z type stabilizers of a QRM code satisfy the CSS orthogonality condition $C_Z \subseteq C_X^{\perp}$, which ensures that all stabilizer generators commute. After puncturing, this property may be violated: in particular, the all-ones vector, corresponding to the constant monomial, becomes odd weight and no longer orthogonal to itself. The Pauli operators associated with this vector, $X^{\otimes(n-1)}$ and $Z^{\otimes(n-1)}$, then fail to commute.
To restore commutativity, we restrict the stabilizer generators to the even weight subcodes $\overline{\mathrm{RM}}(r,m)$, defined in Equation (3.19) as the set of codewords in $\mathrm{RM}^*(r,m)$ with even Hamming weight. This excludes the all-ones vector and ensures that the CSS orthogonality condition is satisfied even after puncturing, so that the resulting stabilizer group defines a valid CSS code.

Logical state encoding The error propagation rules derived in Section 3.1.2.2 are used throughout this paragraph to track stabilizers and logical operators.
In the special case where the punctured QRM code encodes a single logical qubit, that is when $r_X + r_Z + 1 = m$, one can prepare the logical $|0_L\rangle$ state by reusing the QRM encoding circuit and omitting the 000 qubit and its gates [7, Lemma 5]. In this thesis, we restrict to the case where $\mathrm{PQRM}(r_X, r_Z, m)$ encodes a single logical qubit. The literature on punctured quantum Reed–Muller codes with $k > 1$ is limited, and such cases are not considered throughout this thesis.
The X-type stabilizers propagate as in the QRM case, except that the all-X stabilizer is no longer one of the X-type stabilizer generators. The Z-type stabilizers are shortened versions of the original QRM stabilizers, since the qubit labelled 000 has been removed. These operators form the punctured code $\mathrm{RM}(r_Z,m)^*$ and define the stabilizers of the prepared logical state $|0_L\rangle$, which include the logical operator $\overline{Z} = Z^{\otimes n}$, where $n = 2^m - 1$.
Figure 3.4 shows how a Z-type stabilizer, initialized as Z_{101}, propagates through the QRM encoding circuit and spreads to qubits 101, 100, 001, and 000. In the PQRM version of the code, however, the qubit labelled 000 is removed. This results in a stabilizer with support on only three qubits: 101, 100, and 001. This operator has odd weight because it is the product of an even-weight stabilizer ($\in \overline{\mathrm{RM}}(r_Z,m)$) and the logical operator $\overline{Z} = Z^{\otimes n}$.

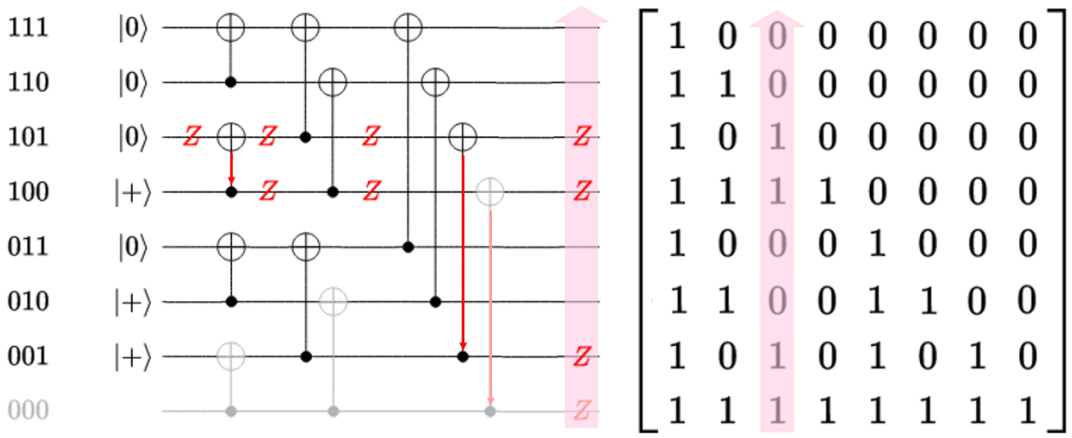


Figure 3.5: Encoding circuit for the logical $|0_L\rangle$ state in $\mathrm{PQRM}(1,1,3)$ (i.e., the Steane code). This figure shows the same stabilizer propagation as in Figure 3.4, but now in the PQRM setting with a fully frozen input and the 000 qubit removed. The red arrows show how the initial stabilizer Z_{101} propagates through the circuit to qubits 100, 001, and 000. Since the 000 qubit is absent in the punctured code, the resulting operator acts only on three qubits. It has odd weight because it equals the product of an even-weight Z-type stabilizer and the logical operator $\overline{Z} = Z^{\otimes n}$. On the right, the resulting stabilizer is highlighted as a column in the generator matrix. Figure adapted from [7].

3.2. Fault-tolerant quantum circuits

A *logical error* occurs when faults accumulate that are not corrected by the code. To analyse under which conditions this happens, quantum circuits are partitioned into *extended rectangles* (exRecs). An exRec consists of a logical operation (such as a gate), together with the error correction steps that precede and follow it. For two-qubit gates, the exRec includes both involved code blocks. An example is shown in Figure 3.6.

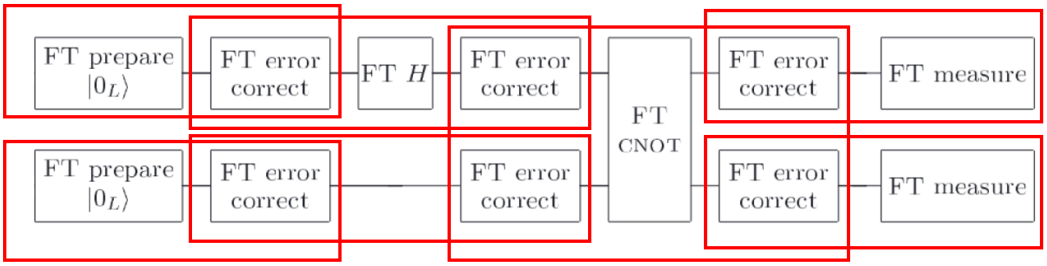


Figure 3.6: Structure of a fault-tolerant circuit with two code blocks. Each red box indicates an extended rectangle (exRec), which includes a fault-tolerant logical operation (such as a state preparation, gate, or measurement) together with surrounding error correction. Each unit must obey certain fault-tolerance properties to ensure that the overall circuit suppresses logical errors; such properties are formalized in [23]. Figure based on [11].

The *code distance* d determines the number of faults that can be corrected:

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor.$$

A fault-tolerant implementation guarantees that no logical error occurs as long as each exRec contains at most t faults per code block [23].
In the low-error regime, the logical error probability per exRec scales as

$$P_L \sim A \cdot p^{\lceil d/2 \rceil}, \tag{3.23}$$

where p is the physical error probability and A reflects properties of the code and the circuit. This scaling reflects that at least $\lceil d/2 \rceil$ faults are required to cause an uncorrectable error [10].
Fault-tolerant quantum computation requires [24]:

- *Fault-tolerant state preparation*, to initialize encoded states without uncorrectable errors (Section 3.2.4)
- *Fault-tolerant logical gates*, such as transversal gates (Section 3.2.2)
- *Fault-tolerant error correction*, especially fault-tolerant syndrome measurement
- *Fault-tolerant measurement* of logical observables

In this report, we focus on the first two: state preparation and logical gate implementation.

3.2.1. Fault model

To analyse the performance and fault tolerance of quantum circuits, it is necessary to specify a noise model. In this thesis, we use the *circuit-level noise model*, which captures errors at the gate level and reflects realistic physical imperfections.
Under this model, the following fault processes are assumed to occur with probability p:

1. **Single-qubit gate errors:** After each single-qubit gate or during idling (i.e., when a qubit is not involved in any operation while others are), any of X, Y, or Z occurs with probability $p/3$.
2. **Two-qubit gate errors:** After each two-qubit gate, any error in the set $I, X, Y, Z^{\otimes 2} \setminus I \otimes I$ occurs with equal probability $p/15$.
3. **Reset errors:** After reset, an ancilla qubit flips from $|+\rangle$ to $|-\rangle$ or from $|0\rangle$ to $|1\rangle$ with probability p.
4. **Measurement errors:** The sign of the measurement outcome is flipped with probability p.

This model follows the assumptions used in [25], with a generalization to all two-qubit gates.
In the remainder of this thesis, we focus on specific components such as fault-tolerant state preparation. The relevant fault-tolerance conditions for these circuits are discussed in Section 3.2.4.

3.2.2. Transversal logical gates

Transversal gates form an important class of fault-tolerant quantum operations. They are designed to prevent the spread of errors within a code block. A common example is the transversal CNOT between two code blocks of n qubits, which applies CNOT_i between qubit i in code block A and qubit i in code block B for all $i \in \{1, \dots, n\}$. In general, transversal operations avoid coupling multiple qubits within a single code block, so that a single fault cannot propagate to multiple locations in that code block [11].

To define this class more precisely, one first introduces a *transversal partition* of the physical subsystems. This is a partition of the qubits such that each part contains exactly one subsystem from each code block. For instance, in a code block code of n qubits, part i might consist of the ith qubit from each encoded block.

A *transversal operator* is then an operator that acts only within each part of such a partition. That is, it applies a unitary to each part independently, without coupling qubits across parts. This ensures that no fault in one part can spread to multiple subsystems within a code block, and the number of faults required to cause a logical failure remains unchanged.

This notion of transversality is already more general than the simplest form, in which each subsystem has size one and a transversal gate on two code blocks is a tensor product of two-qubit unitaries, each acting on qubit i of one code block and qubit i of the other, for all $i \in [n]$. Nevertheless, even under this relaxed definition, the Eastin-Knill theorem [26] shows that no nontrivial quantum code admits a universal set of transversal logical gates.

Even in classical coding theory, not all logical operations can be implemented transversally. For example, the bitwise Toffoli operation maps codewords (c_1, c_2, c_3) to $(c_1, c_2, c_3 + c_1 \cdot c_2)$, where $c_1 \cdot c_2$ denotes the bitwise product. For certain linear codes such as $\mathrm{RM}(1,2)$, the result may lie outside the code space. For instance, with $c_1 = (1100)$, $c_2 = (1010)$, and $c_3 = (0000)$, one obtains

$$c_3 + c_1 \cdot c_2 = (0000) + (1000) = (1000),$$

which is not a codeword in $\mathrm{RM}(1,2)$.

This means that although many stabilizer codes admit transversal implementations of the Clifford group, which is generated by the gates H, S, and CNOT, they cannot support transversal non-Clifford gates such as the T gate, or conversely, support the T gate transversally but not all Clifford gates. Together with the non-Clifford T gate, the Clifford group generates a universal gate set for quantum computation [11].

In QRM and PQRM codes, the set of transversal gates is determined by the algebraic structure of the underlying Reed–Muller codes. All three codes used in this work are QRM or PQRM codes: the Steane code is the $[[7,1,3]]$ code, corresponding to $\mathrm{PQRM}(1,1,3)$; the $[[8,3,2]]$ code is the $\mathrm{QRM}(0,1,3)$ code; and the so-called tetrahedral code is the $[[15,1,3]]$ code, corresponding to $\mathrm{PQRM}(1,2,4)$.

These structural properties determine which gates can be implemented transversally. The Steane code admits transversal implementation of the full Clifford group [5, 7, 9]. The $[[8,3,2]]$ code allows transversal controlled-Z operations between any pair of logical qubits [5, 27]. The $[[15,1,3]]$ code admits all Clifford gates transversally except the Hadamard, and also supports a transversal implementation of the T gate [5, 7, 9]. With the Hadamard gate added via gauge fixing, this code can implement the universal gate set $\{T, \mathrm{CNOT}, H\}$ transversally, noting that $S = T^2$.

3.2.3. Transversal gates in PQRM codes

The set of transversal logical gates in PQRM codes is determined by the code parameters (r_X, r_Z, m). Since these are CSS codes, the logical CNOT gate is always transversal. Other Clifford gates can be implemented transversally only under specific additional conditions.

Clifford gates. The family of codes $\mathrm{PQRM}(\frac{m-1}{2}, \frac{m-1}{2}, m)$, defined for odd $m > 1$, admits transversal implementation of the full Clifford group, including the Hadamard and S gates. These codes all encode a single logical qubit ($K = 1$). The required condition is that the code has $c_X = c_Z$ and that all stabilizer generators have weight divisible by four (the doubly-even property). The Steane code $[[7,1,3]]$ is an example of this type, corresponding to $\mathrm{PQRM}(1,1,3)$.

Rotations around the Z axis. For codes encoding a single logical qubit ($K = 1$), namely the $\mathrm{PQRM}(r_X, m - r_X - 1, m)$ codes, transversal application of the gate

$$R_\nu = \mathrm{diag}(1, e^{2\pi i/2^\nu}),$$

where $\nu \in \mathbb{Z}_{>0}$, on each physical qubit results in a diagonal logical operation equal to $R_\nu^\dagger$, provided that $r_X \leq \frac{m-1}{\nu}$; see [28] for details.
In particular, $R_3 = \mathrm{diag}(1, e^{i\pi/4})$ corresponds to the T gate, and transversal application of R_3 yields a logical $T^\dagger$ gate. The code $[[15, 1, 3]]$, corresponding to $\mathrm{PQRM}(1, 2, 4)$, satisfies $r_X = 1 \leq \frac{4-1}{3}$, and thus admits a transversal T gate.

These conditions explain the transversal gate sets observed in the codes discussed in Section 3.2.2. In particular, they characterize when a PQRM code supports Clifford gates, non-Clifford gates such as T, or both.
While no single code admits a universal set of transversal gates, different codes support different subsets. By transitioning between such codes using code switching, code conversion, or gauge fixing, it becomes possible to combine their gate sets into a fault-tolerant universal gate set. These approaches have been explored in various forms, including code switching [29, 30], code conversion [31, 32], and gauge fixing [9, 32]. Alternative strategies for implementing universal gates fault-tolerantly include magic state distillation and lattice surgery [32].

3.2.4. Fault-Tolerant State Preparation

The definitions discussed in this section apply to individual components in a fault-tolerant quantum circuit, such as state preparation, logical gates, measurement, and error correction. In this thesis, we focus specifically on logical state preparation.
There are multiple definitions of fault tolerance for state preparation circuits. A commonly used *strict definition* requires that small-weight errors do not propagate uncontrollably through the circuit. For example:

> A state preparation circuit is strictly fault-tolerant if for all $t \leq \lfloor d/2 \rfloor$, any error of probability order t propagates to an error of weight at most t.
> [4]

Such definitions are related to the broader notion of t-fault tolerance, which appears in many foundational works on fault-tolerant quantum computing. For instance, a fault-tolerant error correction protocol is typically required to satisfy:

> 1. For an input codeword with error of weight s_1, if s_2 faults occur during the protocol with $s_1 + s_2 \leq t$, ideally decoding the output state gives the same codeword as ideally decoding the input state.
> 2. For $s \leq t$ faults during the protocol, regardless of the input state, the output differs from a codeword by at most weight-s error
> [33].

Such conditions form the basis for many threshold theorems and full circuit-level fault tolerance; see for example [34].
In contrast, a *weaker and more practical definition* considers a circuit fault-tolerant if the *final error* lies within the correctable set of the code. For example:

> All error events with probability p^α, where $\alpha < (d + 1)/2$, should be tolerable in the sense that they are corrected after QEC cycles
> [3].

In practice, this weaker definition is often more relevant, especially for circuits that include verification or postselection.
In this thesis, we adopt the *weaker definition*: a state preparation is considered fault-tolerant if the resulting logical state contains only correctable errors.

3.2.5. Fault-tolerant State Preparation Circuits

Fault-tolerant logical state preparation is the first step of any fault-tolerant quantum computation. It consists of constructing a logical state from physical qubits, while ensuring that the resulting state remains correctable whenever $\alpha < \frac{d+1}{2}$ faults occur. Three main strategies can be distinguished: *repeat-until-success*, *postselection*, and *deterministic preparation with correction*. In these strategies, verification plays a central role.

Verification is implemented using auxiliary qubits, called *flags* or *ancillas*, next to the data qubits. CNOT gates are applied between the data qubits and the flag qubits. If a fault occurs in the preparation circuit, it may propagate to the flags and flip their measurement outcomes. Based on those outcomes, the prepared state can be accepted or discarded, or a correction can be applied. Verification is thus part of repeat-until-success, postselection and deterministic schemes, using a minimal number of ancilla qubits and tailored to detect only the relevant uncorrectable errors.

An alternative to flagged state preparation is to repeatedly measure all stabilizers of the code until the all-$+1$ syndrome is obtained. For instance, when preparing the logical $|0_L\rangle$ state of a $[[n, k, d]]$ CSS code for which $|0\rangle^{\otimes n}$ is an eigenstate of $\overline{Z}$, one may simply measure the stabilizers and apply corrections, and repeat this process until the measured syndrome is $\mathbf{1}$. This strategy is also fault-tolerant, but can be unfavorable on hardware platforms where mid-circuit measurements are expensive or limited. See [35] for a recent implementation using up to 41 repeated measurements on neutral atom hardware.

Several recent works propose automated methods to synthesize fault-tolerant preparation circuits. These include reinforcement learning (RL) methods [3], methods based on satisfiability solving (SAT) combined with heuristic methods [4], and (also using SAT techniques and heuristic methods) deterministic circuits with explicit correction operations [2]. These approaches support synthesis of preparation circuits for both $|0_L\rangle$ and $|+_L\rangle$ states. The RL method defines reward functions such as fidelity, energy of the state in a Hamiltonian constructed using the logical operators of the target state, or distance between the target state and the output state.

Two examples of such synthesized circuits are shown in Figures 3.7 and 3.8. Both circuits were generated using the same synthesis framework combining SAT solvers and heuristic methods. The circuit in Figure 3.7 appears in [4], while the circuit in Figure 3.8 was generated independently using the MQT Python package provided by the same authors. These circuits will be used again in the simulations in Chapter 5.

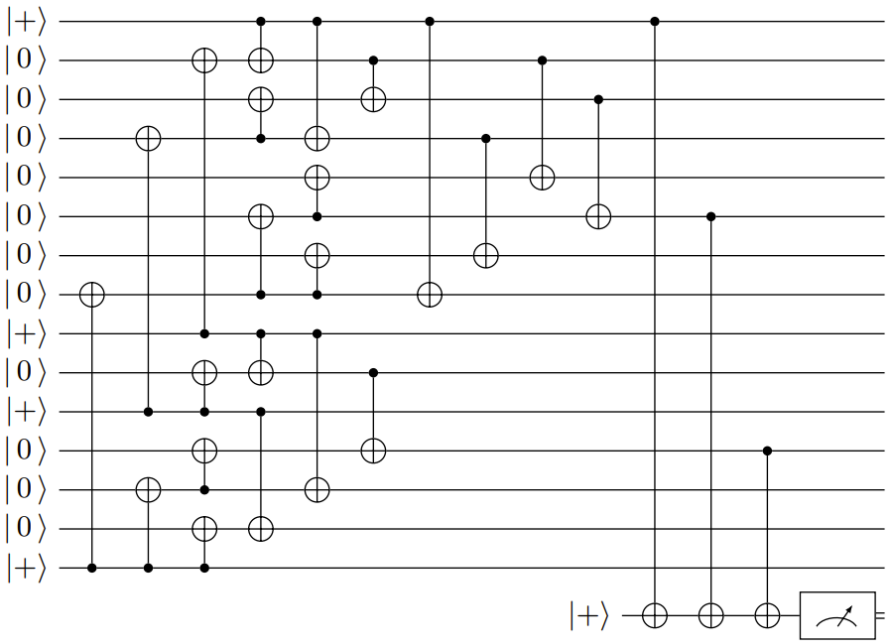


Figure 3.7: Fault-tolerant state preparation circuit for the $|0\rangle_L$ state in the $[[15, 1, 3]]$ code, as shown in [4].

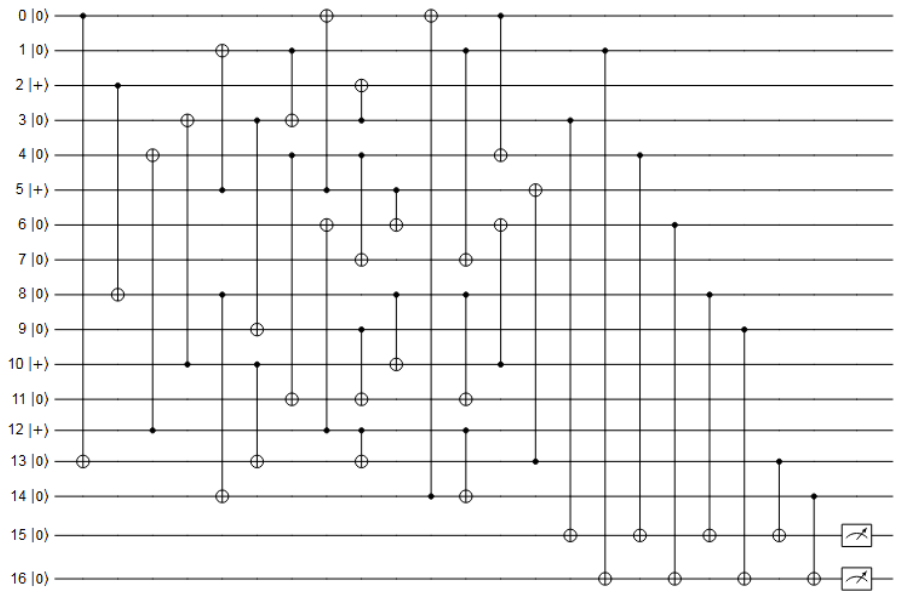


Figure 3.8: Fault-tolerant state preparation circuit for the $|0\rangle_L$ state in the $[[15, 1, 3]]$ code, generated using the MQT Python package (same synthesis method as [4]).

Goto et al. [36]. Goto et al. propose an efficient method for preparing the logical $|0_L\rangle$ state of the Steane code, using only one flag qubit. This circuit is constructed manually. In our simulations in Chapter 5, we use this circuit as a component of the full preparation routine that is compared to the method described in Section 4.2. This circuit is illustrated in Figure 3.9.

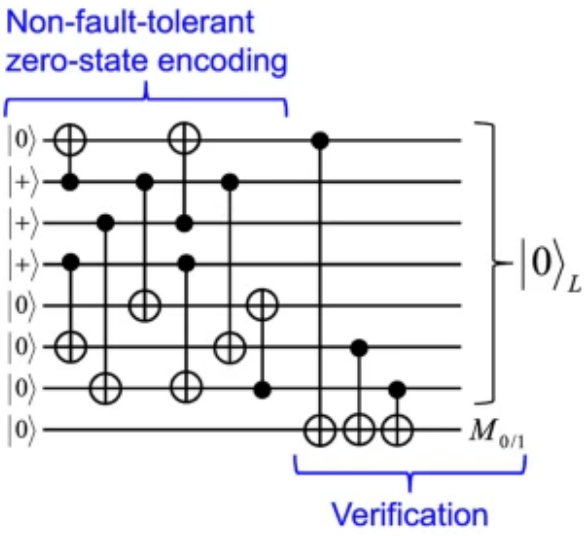


Figure 3.9: Fault-tolerant state preparation circuit for the $|0_L\rangle$ state in the Steane code, from [36].

Heußen et al. [37]. Heußen et al. construct a fault-tolerant preparation circuit for the Steane code using two flag qubits: one connected to all qubits in the support of a logical operator, and one to its complement. This arrangement enables detection of specific high-weight errors without full syndrome extraction. Their circuit includes corrections conditioned on the flag qubits. Their circuit, shown in Figure 3.10, includes corrections conditioned on the flag qubits.

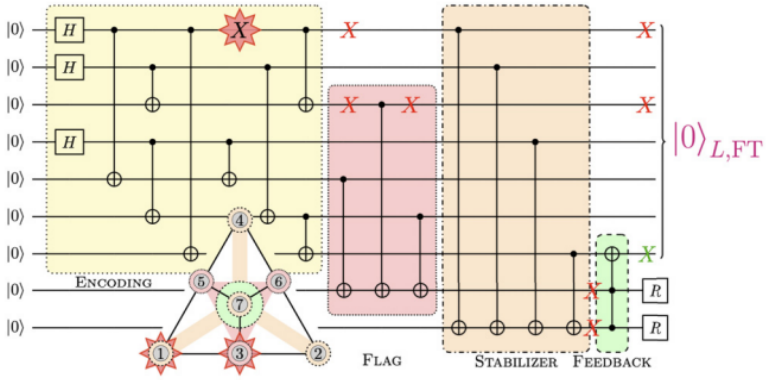


Figure 3.10: Deterministic fault-tolerant state preparation circuit for the $|0_L\rangle$ state in the Steane code using two flag qubits [37].

We adopt this mechanism partially in our own construction. Specifically, we use a modified version of this verification subroutine for the Steane preparation that is embedded in the $|0_L\rangle$ preparation for the $[[15, 1, 3]]$ code (see Section 4.2). The corresponding verification circuit is shown in Figure 4.11 in Section 4.5.1. The flag-based scheme aims to detect only the two relevant uncorrectable errors, X_9X_{11} and X_9X_{13}. These are detected using two flag qubits: one coupled to qubits $10, 11, 13$ and one to $9, 12, 14, 15$. Both flags trigger if and only if one of these specific weight-2 errors occurs, although other errors of weight ≥ 2 may also trigger them (but those occur with lower probability). Unlike Heußen's construction, our preparation follows a repeat-until-success strategy: the circuit is repeated until the verification accepts.

4

Methods

4.1. Motivation for using the $[[15,1,3]]$ code and logical $|0_L\rangle$ state

In this work, we focus on the preparation of the logical $|0_L\rangle$ state in the $[[15,1,3]]$ code, which serves as a primary test case for evaluating IV. This choice is motivated by its role in universal fault-tolerant quantum computing, as well as by its suitability as a minimal and practically relevant benchmark.

The $[[15,1,3]]$ code is a QRM code with distance 3. As shown by Kubica [9], the family of PQRM codes of the form PQRM$(1, m-2, m)$ enables universal quantum computation by combining transversal implementation of the non-Clifford T gate with logical Hadamard gates implemented via gauge fixing. This framework does not prescribe a specific initialization method, but for computation, initialisation of logical states (often in $|0_L\rangle$ or $|+_L\rangle$) is needed. Moreover, if Steane-style quantum error correction is used, both types of ancilla states are required to extract Z- and X-syndromes fault-tolerantly.

Although gauge fixing offers a way to implement the logical Hadamard gate, it is not the only possibility. Reliable preparation of both logical $|0_L\rangle$ and $|+_L\rangle$ states is important, and depending on the implementation, either can serve as the starting point. Various methods exist to prepare one from the other, such as gauge fixing or gate teleportation, providing flexibility in fault-tolerant protocols.

In addition to its role in fault-tolerant computation, the $|0_L\rangle$ state in the $[[15,1,3]]$ code appears in magic state distillation protocols. In particular, the 15-to-1 scheme by Fowler et al. [10] uses logical $|0_L\rangle$ states as inputs. This is a variant of a 'standard' distillation circuit based on the $[[15,1,3]]$ code; see for example Kubica's PhD thesis [8], where different variants are discussed (see Figures 3.5 and 3.6). Notably, some variants prepare the logical $|+_L\rangle$ state instead of the logical $|0_L\rangle$. It is worth noting that when creating a Bell pair $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$ between a physical qubit and an encoded $[[15,1,3]]$ qubit, one might alternatively prepare the physical qubit in the $|0\rangle$ state and the encoded qubit in the $|+_L\rangle$ state, then apply a CNOT with the encoded qubit as control. Moreover, in magic state distillation protocols the 'qubits' involved are often themselves encoded in an underlying base code, such as the surface code, adding an additional layer of encoding.

Finally, the PQRM$(1, m-2, m)$ codes allow transversal implementation of rotation gates

$$R_{m-1} = \mathrm{diag}(1, e^{2\pi i/2^{m-1}}),$$

enabling access to increasingly fine phase rotations as m grows. This property is of particular interest in quantum algorithms that require small-angle gates.

While recent work has produced highly optimized circuits for preparing logical states in small-distance codes [2–4], these typically apply verification at the end of the circuit. The goal of this work is to demonstrate that verification can also occur mid-circuit, by repeating only a smaller section of the preparation circuit, thereby reducing overhead. If automated synthesis and RL methods could incorporate IV as an option, they might discover circuits with shorter effective runtimes when including the cost of repetitions.

In the following section, we detail the construction of the logical $|0_L\rangle$ state in the $[[15,1,3]]$ code by combining smaller code states via quasitransversal CNOT gates.

4.2. Overview of the construction

The goal of this method is to fault-tolerantly prepare a logical $|0_L\rangle$ state in the $[[15,1,3]]$ code. The approach is based on combining encoded states in smaller quantum codes via a quasitransversal CNOT. Specifically, we combine:

- a $|(0,0,0)_L\rangle$ state encoded in the $[[8,3,2]]$ code,
- a $|0_L\rangle$ state encoded in the $[[7,1,3]]$ Steane code,

to produce a state that is equivalent to the $|0_L\rangle$ state of the $[[15,1,3]]$ code. This construction exploits the recursive structure of QRM codes and allows for fault-tolerant IV.

Definition: Quasitransversal CNOT. In this context, we define a *quasitransversal* CNOT as a layer of parallel CNOT gates applied between two code blocks, where the gate structure closely resembles a transversal operation but is not fully transversal. Specifically, a fully transversal CNOT between two code blocks would apply a CNOT between every corresponding pair of qubits. In our construction, however, the $[[7,1,3]]$ (Steane) code acts as the control block, and only seven of the eight qubits in the $[[8,3,2]]$ code (the target block) participate in the operation. One qubit in the $[[8,3,2]]$ block remains untouched. This deviation from full transversality motivates the term “quasitransversal.”

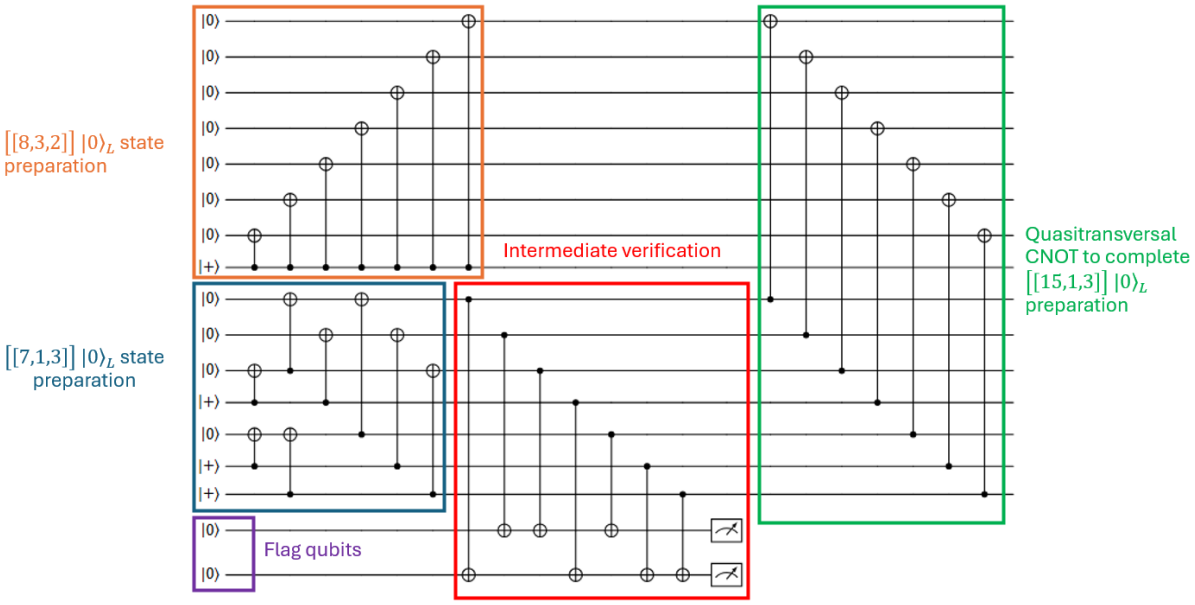


Figure 4.1: Circuit for fault-tolerant preparation of the logical $|0_L\rangle$ state in the $[[15,1,3]]$ code. The lower-left block (blue) prepares a $|0_L\rangle$ state in the $[[7,1,3]]$ (Steane) code, and the upper-left block (orange) prepares a $|(0,0,0)_L\rangle$ state in the $[[8,3,2]]$ code. The IV block (red) uses flag qubits (purple) to detect potentially uncorrectable X-errors. The final CNOT layer (green) completes preparation of the logical $|0_L\rangle$ state in the $[[15,1,3]]$ code.

This chapter first describes the construction of the logical $|0_L\rangle$ state in the $[[15,1,3]]$ code from two smaller QRM codes, and then analyses the fault tolerance of each component under a single-fault assumption. Throughout, fault tolerance is understood relative to the $[[15,1,3]]$ code: a preparation step is considered fault-tolerant if any single fault occurring during its execution leads to an error that is correctable by the $[[15,1,3]]$ code.

4.2.1. Fault model and error types

Throughout this chapter, we assume a single fault model: at most one fault may occur during the entire state preparation procedure. A *fault* is defined as a single physical error, either a single-qubit Pauli error (e.g., from a faulty single-qubit gate or idle qubit), or a two-qubit Pauli error acting on both qubits involved in a faulty two-qubit gate (typically a CNOT).

At the moment of occurrence, such faults have support on at most two qubits, but they may propagate through the circuit and result in higher-weight errors.

We restrict our analysis to X-type errors. Any Z-type error that is incorrectly corrected can at most result in a logical Z operator being applied. Since the intended output state is $|0_L\rangle$, which is an eigenstate of the logical Z_L operator, such errors leave the state invariant and do not affect correctness.

These modelling choices, both the single fault assumption and the focus on X-type errors, follow from the weaker and more practical definition of fault tolerance discussed in Section 3.2.4, and explicitly stated in [3]:

> "All error events with probability p^α, where $\alpha < (d+1)/2$, should be tolerable in the sense that they are corrected after QEC cycles."

In our case, the code has distance $d = 3$, so this condition becomes $\alpha < 2$. This means that any single fault event (which has probability scaling with p) must be correctable, motivating our use of the single fault assumption.

- Section 4.2 introduces the $[[8,3,2]]$, $[[7,1,3]]$ (Steane), and $[[15,1,3]]$ codes, and describes how a quasitransversal CNOT between the two smaller codes produces a valid logical $|0_L\rangle$ state in the $[[15,1,3]]$ code.

- Lemma 4.1 (Section 4.3) proves that the $|(0,0,0)_L\rangle$ state in the $[[8,3,2]]$ code can be prepared fault-tolerantly, in the sense that any resulting error from a single fault is correctable by the $[[15,1,3]]$ code.

- Lemma 4.2 shows that the Steane $|0_L\rangle$ state can be prepared fault-tolerantly with verification, again under the single fault assumption and relative to the $[[15,1,3]]$ code.

4.3. State preparation of the $[[15,1,3]]$ code

The full preparation is summarized in Figure 4.1. It combines a $|0_L\rangle$ state in the $[[7,1,3]]$ (Steane) code and a $|(0,0,0)_L\rangle$ state in the $[[8,3,2]]$ code via a quasitransversal CNOT, resulting in the logical $|0_L\rangle$ state of the $[[15,1,3]]$ code.

To verify that this construction produces the correct logical state, we analyse the stabilizers before and after the CNOT. Specifically, we begin with the stabilizers of the $|(0,0,0)_L\rangle$ state in the $[[8,3,2]]$ code.

4.3.1. Stabilizers and logical operators of the $[[8,3,2]]$ code

The $[[8,3,2]]$ code can be geometrically described as a cube, as shown in Figure 4.2.

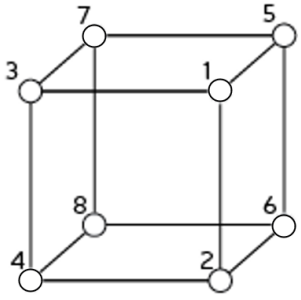
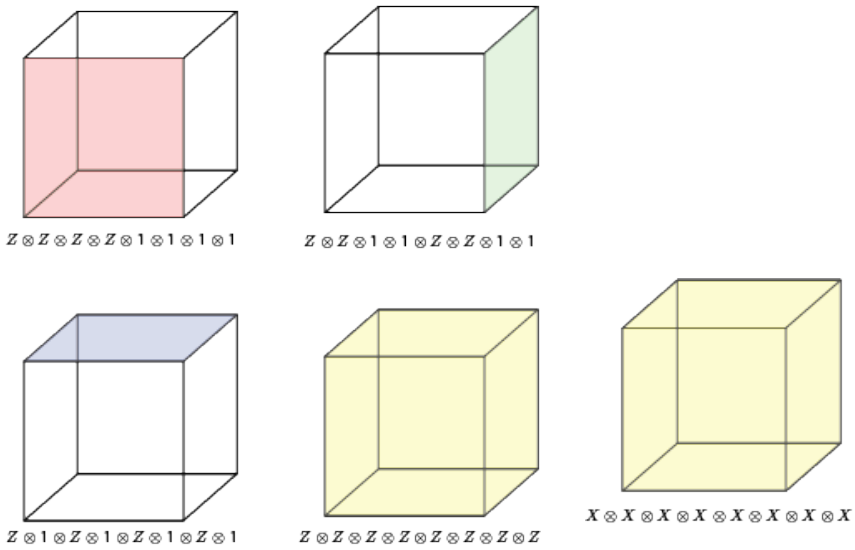


Figure 4.2: The 8 qubits of the $[[8,3,2]]$ code and the cubic lattice used to define the stabilizers and logical operators. Figure adapted from [27].

The generators of the Z-stabilizer group and the only X-stabilizer of the $[[8,3,2]]$ code are shown in Figure 4.3.



(a) The generators of the Z-stabilizers of the $[[8,3,2]]$ code.

(b) The only X-stabilizer of the $[[8,3,2]]$ code.

Figure 4.3: The generators of the stabilizers of the $[[8,3,2]]$ code. Figure adapted from [27].

The logical operators of the $[[8,3,2]]$ code are shown in Figure 4.4.

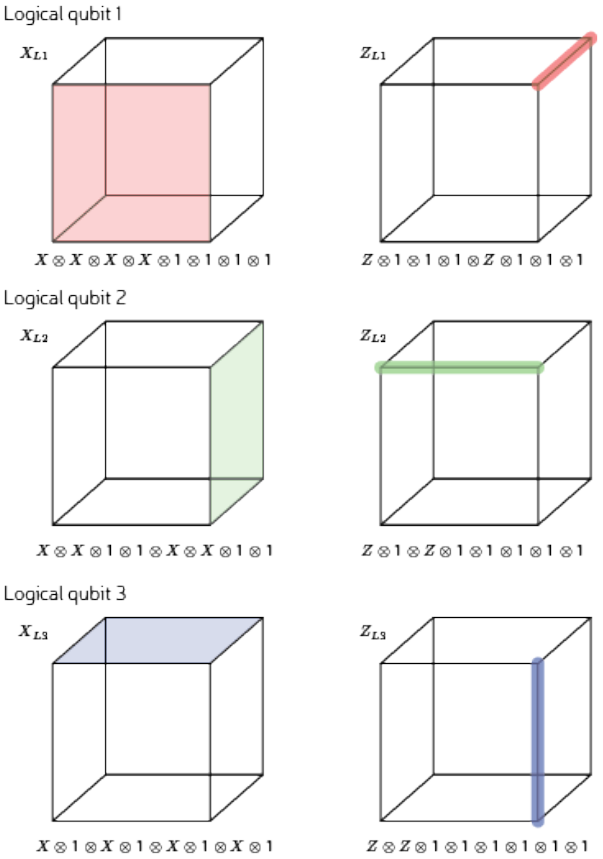


Figure 4.4: The logical operators of the $[[8,3,2]]$ code. Figure from [27].

So, the logical X operators are $X_{L1} = X_1X_2X_3X_4$, $X_{L2} = X_1X_2X_5X_6$, $X_{L3} = X_1X_3X_5X_7$ and the logical Z operators are $Z_{L1} = Z_1Z_5$, $Z_{L2} = Z_1Z_3$, $Z_{L3} = Z_1Z_2$.
The logical $|(0,0,0)_L\rangle$ state is an eigenstate of the logical Z operators, which is why the quasitransversal

CNOT maps the logical operators to either the Z_L of the $[[15,1,3]]$ code or one of its Z-type stabilizers.

4.3.2. Stabilizers and logical operators of the Steane code

The stabilizers of the Steane code can be represented on the cubic lattice if the eighth qubit is left out, see Figure 4.5. The support of the generators of both the X-stabilizers and the Z-stabilizers is shown in Figure 4.5. Usually, a Steane code is represented by a triangle, see Figure 4.6, which is what will also be done in the rest of this paper. The representation on the cube is shown to explain a non-conventional numbering of qubits, which will simplify the explanation of the $[[15,1,3]]$ state preparation.

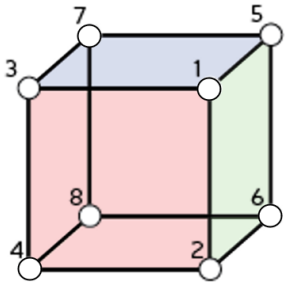


Figure 4.5: The generators of both the X-stabilizers and the Z-stabilizers act on the coloured faces of the cube. Figure adapted from [27].

Using this numbering on the conventional representation of the Steane code, the result is shown in Figure 4.6.

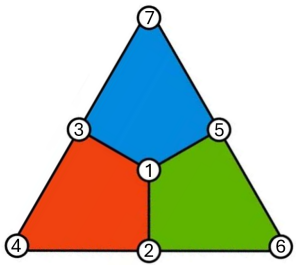


Figure 4.6: Both the X-type and Z-type generators of the stabilizer group are defined on the coloured faces that tile the triangle. Figure adapted from [38].

Thus, the stabilizer group of the Steane code is $\mathcal{S} = \langle X_1X_2X_3X_4, X_1X_2X_5X_6, X_1X_3X_5X_7, Z_1Z_2Z_3Z_4, Z_1Z_2Z_5Z_6, Z_1Z_3Z_5Z_7\rangle$, and the logical operators are $X_L = X_2X_4X_6$ and $Z_L = Z_2Z_4Z_6$. Since the logical $|0_L\rangle$ is an eigenstate of Z_L, the quasitransversal CNOT maps Z_L to either the Z_L of the $[[15,1,3]]$ code or one of its Z-type stabilizers.

4.3.3. Stabilizers and logical operators of the $[[15,1,3]]$ code

The Z-type stabilizers of the $[[15,1,3]]$ code are generated by the faces of the pyramid shown in Figure 4.7, and the X-type stabilizers are generated by its weight-8 cells. The logical operators can be written as $X_L = X_{\text{all}}$ and $Z_L = Z_{\text{all}}$, and their lowest-weight representatives have weight 7 and weight 3 respectively.

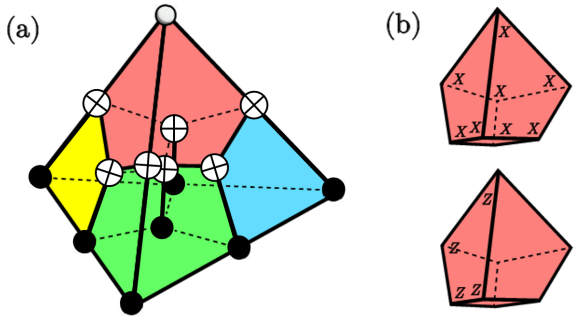


Figure 4.7: The so-called "tetrahedral" code, i.e. the $[[15, 1, 3]]$ code, visualized as a tetrahedron of 15 physical qubits. **(a)** The faces (Z-type stabilizers) and cells (X-type stabilizers) of the tetrahedron define the code. The $\oplus$ symbols mark the *targets* of the quasitransversal CNOTs, and the connected black dots mark their *controls*. Each control-target pair is connected by a thick black line. **(b)** The coloured faces represent the Z-type stabilizers; the coloured cells represent the X-type stabilizers. Any one of these cells (the one shown is the top one) corresponds exactly to the stabilizer structure of the $[[8, 3, 2]]$ code. Figure adapted from [39].

Hence, the stabilizer group of the tetrahedral code is $\mathcal{S} = \langle X_1X_2X_3X_4X_5X_6X_7X_8, X_1X_2X_3X_4X_9X_{10}X_{11}X_{12}, X_1X_2X_5X_6X_9X_{10}X_{13}X_{14}, X_1X_3X_5X_7X_9X_{11}X_{13}X_{15}, Z_1Z_2Z_3Z_4, Z_1Z_2Z_5Z_6, Z_1Z_3Z_5Z_7, Z_2Z_4Z_6Z_8, Z_1Z_2Z_9Z_{10}, Z_1Z_3Z_9Z_{11}, Z_1Z_5Z_9Z_{13}, Z_9Z_{10}Z_{11}Z_{12}, Z_9Z_{10}Z_{13}Z_{14}, Z_9Z_{11}Z_{13}Z_{15}\rangle$ and the logical $|0_L\rangle$ state is also stabilized by the logical Z operator, which is equivalent to $Z_L = Z_{10}Z_{12}Z_{14}$.

4.3.4. Stabilizers before and after quasitransversal CNOT

To understand the effect of applying a quasitransversal CNOT on encoded states, we analyse the stabilizers before and after the operation. Specifically, we start with qubits 1 through 8 initialized in a $|(0, 0, 0)_L\rangle$ state encoded in the $[[8, 3, 2]]$ code, and qubits 9 through 15 initialized in a $|0_L\rangle$ state encoded in the Steane code ($[[7, 1, 3]]$).

The goal is to prepare a logical $|0_L\rangle$ state in the $[[15, 1, 3]]$ code, by applying a set of seven CNOT gates between the Steane qubits and a subset of the $[[8, 3, 2]]$ qubits. This operation is quasitransversal in the sense that it consists of parallel CNOTs between matching qubit positions in two separate QRM codes.

To verify that this indeed results in a logical $|0_L\rangle$ state in the $[[15, 1, 3]]$ code, we explicitly track the stabilizers using the CNOT propagation rules from Equation (3.17).

Before applying the CNOTs, the joint state is stabilized by the direct product of the stabilizer groups of the $[[8, 3, 2]]$ and $[[7, 1, 3]]$ codes. Table 4.1 lists these stabilizers and the logical Z operators.

Qubits	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Generators of stabilizer group $[[8, 3, 2]]$	X	X	X	X	X	X	X	X	I	I	I	I	I	I	I
	Z	Z	Z	Z	I	I	I	I	I	I	I	I	I	I	I
	Z	Z	I	I	Z	Z	I	I	I	I	I	I	I	I	I
	Z	I	Z	I	Z	I	Z	I	I	I	I	I	I	I	I
	I	Z	I	Z	I	Z	I	Z	I	I	I	I	I	I	I
Z_{L1}	Z	Z	I	I	I	I	I	I	I	I	I	I	I	I	I
Z_{L2}	Z	I	Z	I	I	I	I	I	I	I	I	I	I	I	I
Z_{L3}	Z	I	I	I	Z	I	I	I	I	I	I	I	I	I	I
Generators of stabilizer group $[[7, 1, 3]]$	I	I	I	I	I	I	I	I	Z	Z	Z	Z	I	I	I
	I	I	I	I	I	I	I	I	Z	Z	I	I	Z	Z	I
	I	I	I	I	I	I	I	I	Z	I	Z	I	Z	I	Z
	I	I	I	I	I	I	I	I	X	X	X	X	I	I	I
	I	I	I	I	I	I	I	I	X	X	I	I	X	X	I
	I	I	I	I	I	I	I	I	X	I	X	I	X	I	X
Z_L	I	I	I	I	I	I	I	I	I	X	I	X	I	X	I

Table 4.1: Operators stabilizing $|(0, 0, 0)_L\rangle_{[[8,3,2]]} \otimes |0_L\rangle_{[[7,1,3]]}$

Recall from Equation (3.17) the CNOT propagation rules for Pauli operators.

When the quasitransversal CNOT,

$$\overline{CNOT} = CNOT_{9,1} \cdot CNOT_{10,2} \cdot CNOT_{11,3} \cdot CNOT_{12,4} \cdot CNOT_{13,5} \cdot CNOT_{14,6} \cdot CNOT_{15,7}$$

is applied, where each $CNOT_{i,j}$ denotes a CNOT gate with control on qubit i and target on qubit j, the stabilizers of the resulting state are shown in Table 4.2.

Qubits	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Generators of stabilizer group $[[15,1,3]]$	X	X	X	X	X	X	X	X	I	I	I	I	I	I	I
	Z	Z	Z	Z	I	I	I	I	Z	Z	Z	Z	I	I	I
	Z	Z	I	I	Z	Z	I	I	Z	Z	I	I	Z	Z	I
	Z	I	Z	I	Z	I	Z	I	Z	I	Z	I	Z	I	Z
sZ_L	I	Z	I	Z	I	Z	I	Z	I	Z	I	Z	I	Z	I
Generators of stabilizer group $[[15,1,3]]$	Z	Z	I	I	I	I	I	I	Z	Z	I	I	I	I	I
	Z	I	Z	I	I	I	I	I	Z	I	Z	I	I	I	I
	Z	I	I	I	Z	I	I	I	Z	I	I	I	Z	I	I
	I	I	I	I	I	I	I	I	Z	Z	Z	Z	I	I	I
	I	I	I	I	I	I	I	I	Z	Z	I	I	Z	Z	I
	I	I	I	I	I	I	I	I	Z	I	Z	I	Z	I	Z
	X	X	X	X	I	I	I	I	X	X	X	X	I	I	I
	X	X	I	I	X	X	I	I	X	X	I	I	X	X	I
	X	I	X	I	X	I	X	I	X	I	X	I	X	I	X
Z_L	I	I	I	I	I	I	I	I	I	Z	I	Z	I	Z	I

Table 4.2: Operators stabilizing $|0_L\rangle_{[[15,1,3]]}$. The stabilizer s that is multiplied by the logical Z_L is $s = Z_2Z_4Z_6Z_8$.

Gaussian elimination has not been applied to the parity check matrices that correspond to Table 4.1 and Table 4.2, because in their current form, it is easier to compare them to Figures 4.3, 4.4, 4.6 and 4.7.

4.4. State preparation of the $[[8,3,2]]$ code

Figure 3.4 illustrates the stabilizer propagation mechanism in a QRM encoding circuit for the $[[8,3,2]]$ code, based on the hypercube construction. However, our implementation uses a different circuit, derived directly from the stabilizers and logical operators of the $|(0,0,0)_L\rangle$ state.

The state preparation of the $|(0,0,0)_L\rangle$ state of the $[[8,3,2]]$ code follows directly from its stabilizers and the logical operators that stabilize the state. Recall that the stabilizer group of the cubic code is

$$\mathcal{S} = \langle X_1X_2X_3X_4X_5X_6X_7X_8,\ Z_1Z_2Z_3Z_4,\ Z_1Z_2Z_5Z_6,\ Z_1Z_3Z_5Z_7,\ Z_1Z_2Z_3Z_4Z_5Z_6Z_7Z_8\rangle,$$

and the logical Z operators are $Z_{L1} = Z_1Z_5$, $Z_{L2} = Z_1Z_3$, and $Z_{L3} = Z_1Z_2$. The state $|(0,0,0)_L\rangle$ is the shared eigenstate of the stabilizers and the three logical Z operators.

Note that the group generated by

$$\langle X_1X_2X_3X_4X_5X_6X_7X_8,\ Z_1Z_2Z_3Z_4,\ Z_1Z_2Z_5Z_6,\ Z_1Z_3Z_5Z_7,\ Z_1Z_2Z_3Z_4Z_5Z_6Z_7Z_8,\ Z_1Z_2,\ Z_1Z_3,\ Z_1Z_5\rangle$$

is equivalent to

$$\langle X_1X_2X_3X_4X_5X_6X_7X_8,\ Z_1Z_2,\ Z_2Z_3,\ Z_3Z_4,\ Z_4Z_5,\ Z_5Z_6,\ Z_6Z_7,\ Z_7Z_8\rangle.$$

The shared eigenstate of the latter group is the 8-qubit Greenberger–Horne–Zeilinger (GHZ) state [40]:

$$|00000000\rangle + |11111111\rangle.$$

Such a state can be straightforwardly encoded using the circuit shown in Figure 4.8.

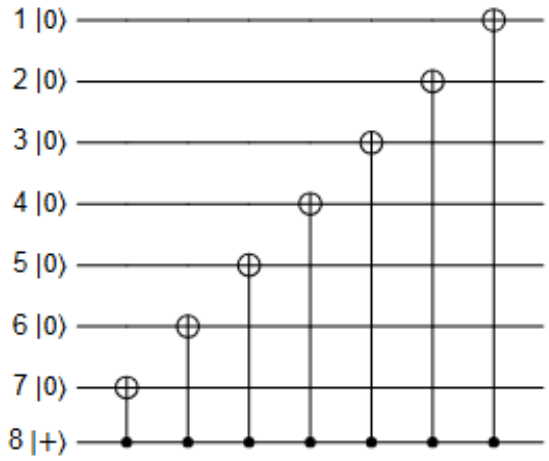


Figure 4.8: State preparation of the $[[8,3,2]]$ code in the $|(0,0,0)_L\rangle$ state.

Recall that our analysis of fault tolerance is restricted to X-type errors and assumes at most one fault during the full state preparation; see Subsection 4.2.1. The final state is a logical state of the $[[15,1,3]]$ code, which is a punctured quantum Reed–Muller code $\mathrm{PQRM}(r_X, r_Z, m)$ with $r_Z = 2$ and $m = 4$. As established in Equation (3.22), the minimum weight of undetectable X-type errors is

$$d_Z^+ = 2^{r_Z+1} - 1 = 7.$$

This means that the logical X operator $X_L = X_{\text{all}}$ has weight 15 but can be reduced to weight 7 by multiplication with X-type stabilizers. Hence, all X-type errors of weight at most 3 are correctable.
Under the single-fault model, any error of weight greater than 4 can always be multiplied with the stabilizer $X_1X_2X_3X_4X_5X_6X_7X_8$ to yield an equivalent error of weight at most 3. The only dangerous weight-4 X-error arises when the fourth CNOT (from qubit 5 to 8) fails and produces X on both control and target. This propagates into the $X_1X_2X_3X_8$ error shown in Figure 4.9.

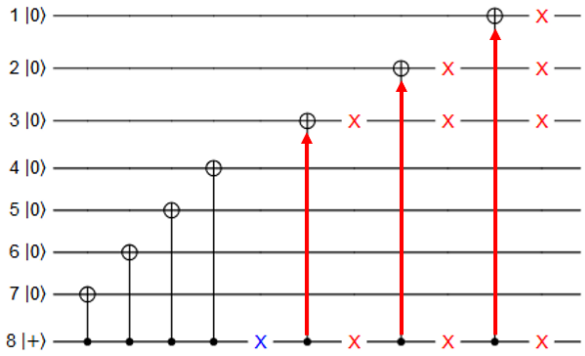


Figure 4.9: Propagation of a single fault in the $[[8,3,2]]$ state preparation circuit. The blue X marks the initial error, while the red X symbols show the propagated errors resulting from a faulty CNOT. Red arrows indicate the propagation paths through the CNOT gates. The resulting X-type error is $X_1X_2X_3X_8$.

4.4.1. Fault tolerance of the state preparation of the $[[8,3,2]]$ code

Figure 4.8 shows the preparation of the $|(0,0,0)_L\rangle$ state in the $[[8,3,2]]$ code. This state is then used as the target in the quasitransversal CNOT of Figure 4.1, which completes the preparation of the logical $|0_L\rangle$ state in the $[[15,1,3]]$ code.

Lemma 1. *The circuit in Figure 4.8 (see also Figure 4.1) prepares the* $|(0,0,0)_L\rangle$ *state in the* $[[8,3,2]]$ *code fault-tolerantly, in the sense that any single fault occurring during the preparation results in an* X*-error that is correctable by the* $[[15,1,3]]$ *code.*

We emphasize that no quantum error correction is applied between this preparation circuit and the subsequent quasi-transversal CNOTs used to construct the final $|0_L\rangle$ state in the $[[15,1,3]]$ code. A

single fault in the $[[8,3,2]]$ preparation may propagate to a higher-weight X-error in the final state, but as we prove below, it always remains correctable by the $[[15,1,3]]$ code.
Alternative approaches exist that prepare a $|0_L\rangle$ state in the $[[15,1,3]]$ code directly using fault-tolerant flag-based circuits [2–4]. These methods typically perform verification at the end of the circuit, while our approach places verification earlier in the preparation sequence, reducing the number of wasted full-circuit attempts. See Section 3.2.5 for a detailed comparison.

Fault model and assumptions As discussed in Section 4.2.1, we assume a single-fault model in which any error introduced by a fault has weight at most two at the moment of occurrence and may propagate to a higher-weight X-error.
Although the total distance of the $[[15,1,3]]$ code is $d = 3$, its distance against X-type errors is $d_Z^+ = 7$. This means that all X errors of weight up to 3 are correctable.

Identification of the critical error. A fault-path analysis shows that a single fault during the $[[8,3,2]]$ preparation circuit may propagate to at most one dangerous X-error:

$$E = X_1X_2X_3X_8.$$

This error occurs on the qubits that encode the $|(0,0,0)_L\rangle$ state in the $[[8,3,2]]$ code. Since this code block serves as the **target** in the quasi-transversal CNOT gates used to prepare the $|0_L\rangle$ state in the $[[15,1,3]]$ code, X-errors on these qubits **do not propagate** through the CNOTs. Therefore, we can analyse E directly as the effective X-error acting on the final encoded state in the $[[15,1,3]]$ code.
All other propagated errors resulting from a single fault have weight at most 3 and are guaranteed to be correctable by the $[[15,1,3]]$ code. Figure 4.9 illustrates a fault and its propagation path that may lead to the error E.

Proof of correctability We assume a minimum-weight decoder (e.g., a lookup table), which assigns to each syndrome a correction operator E' of minimum weight. We want to show that for the specific error

$$E = X(f), \quad \text{with } f = (1,1,1,0,0,0,0,1,0,0,0,0,0,0,0),$$

no correction $E' = X(f')$ of weight ≤ 4 can result in a logical error, that is, we cannot have $E'E \in C(\mathcal{S})\backslash\mathcal{S}$.
Note that $\overline{X} = X_{\text{all}}$ has weight 15, and every stabilizer $s \in \mathcal{S}$ has even weight. Therefore, $\overline{X}s$ has weight either 15 (if $s = I$) or 7 (if $s \neq I$), and we have $|E'E| = |\overline{X}s| = 7$.
Now observe that

$$|E'E| \leq |E| + |E'| = 4 + |E'|.$$

Since $|E'E| = 7$ and $|E| = 4$, it follows that $|E'| \geq 3$. Suppose $|E'| = 4$. Then we compute:

$$|E'E| = |f + f'| = |f| + |f'| - 2|\text{supp}(f) \cap \text{supp}(f')| = 8 - 2|\text{supp}(f) \cap \text{supp}(f')|.$$

This expression is even, but $|E'E| = 7$ is odd, a contradiction. Therefore, $|E'| = 3$, and f and f' must have disjoint support.
Let $g = f + f'$, so that $E'E = X(g) = \overline{X}s$. The remainder of the proof proceeds by showing that this leads to a contradiction with the structure of logical operators.
Recall that the X-type generators of the stabilizer group $\mathcal{S}$ are:

$$\begin{aligned}
s_{x,1} &= X_1X_2X_3X_4X_5X_6X_7X_8 \\
s_{x,2} &= X_1X_2X_3X_4X_9X_{10}X_{11}X_{12} \\
s_{x,3} &= X_1X_2X_5X_6X_9X_{10}X_{13}X_{14} \\
s_{x,4} &= X_1X_3X_5X_7X_9X_{11}X_{13}X_{15}
\end{aligned}$$

Let $h^1, h^2, h^3, h^4 \in \mathbb{F}_2^{15}$ be the corresponding binary support vectors, where $h_j^i = 1$ if qubit j is in the support of $s_{x,i}$, and 0 otherwise. Explicitly:

$$\begin{aligned}
h^1 &= (1,1,1,1,1,1,1,1,0,0,0,0,0,0,0) \\
h^2 &= (1,1,1,1,0,0,0,0,1,1,1,1,0,0,0) \\
h^3 &= (1,1,0,0,1,1,0,0,1,1,0,0,1,1,0) \\
h^4 &= (1,0,1,0,1,0,1,0,1,0,1,0,1,0,1)
\end{aligned}$$

Any X-type logical operator in $C(\mathcal{S}) \setminus \mathcal{S}$ can be written as:

$$g = \mathbf{1} + \sum_{i=1}^{4} c_i h^i,$$

where $\mathbf{c} \in \mathbb{F}_2^4 \setminus \{\mathbf{0}\}$. The condition $\mathrm{supp}(f) \subseteq \mathrm{supp}(g)$ then imposes the following constraints:

$$\begin{aligned} g_1 = f_1 = 1 &\Rightarrow c_1 + c_2 + c_3 + c_4 = 0 \\ g_2 = f_2 = 1 &\Rightarrow c_1 + c_2 + c_3 = 0 \\ g_3 = f_3 = 1 &\Rightarrow c_1 + c_2 + c_4 = 0 \\ g_8 = f_8 = 1 &\Rightarrow c_1 = 0 \end{aligned}$$

Solving this system gives $\mathbf{c} = 0$, contradicting the assumption that $E'E$ is a logical operator. Hence, E is correctly handled by the decoder. Thus, The circuit in Figure 4.8 prepares the $|(0,0,0)_L\rangle$ state in the $[[8,3,2]]$ code fault-tolerantly, in the sense that any single fault occurring during the preparation results in an X-error that is correctable by the $[[15,1,3]]$ code. □

Geometric intuition. The formal argument above can be complemented with an intuitive reasoning based on the geometry of the $[[15,1,3]]$ code.

Suppose a fault leads to the weight-4 error $E = X_1X_2X_3X_8$, and the decoder mistakenly applies a different correction E' of weight up to 4 with the same syndrome. This is only problematic if $E \cdot E'$ is logically non-trivial, i.e., a logical X operator.

Figure 4.10 shows the stabilizer cell structure of the $[[15,1,3]]$ code. One representative of the logical X operator is $\overline{X} = X_{\text{all}}$, which acts on all 15 qubits. Since E and E' both have weight at most 4, their product $E \cdot E'$ has weight at most 8. Therefore, if $E \cdot E' \sim \overline{X}$, the decoder must have selected a representative of $\overline{X}$ with weight ≤ 8. But X_{all} has weight 15, so one must multiply it by at least one X-type stabilizer (i.e., a cell) to reduce its weight.

We now ask whether there exists any stabilizer-equivalent logical X operator whose support contains all of qubits $\{1,2,3,8\}$. To construct such an operator, one may attempt to reduce the support of $\overline{X}$ by multiplying it with cells:

- Multiplying by the top cell (red) removes qubit 8, which is not allowed.
- Multiplying by any single bottom cell removes qubit 2 or 3.
- Multiplying by two bottom cells restores one but removes another.
- Multiplying by all three bottom cells removes qubit 1.

Hence, there is no combination of cells one can multiply with $\overline{X}$ to obtain a logical X operator whose support includes all of qubits 1, 2, 3, and 8. This implies that the error $X_1X_2X_3X_8$ cannot be logically equivalent to any lower-weight error, and is therefore correctly handled by a minimum-weight decoder.

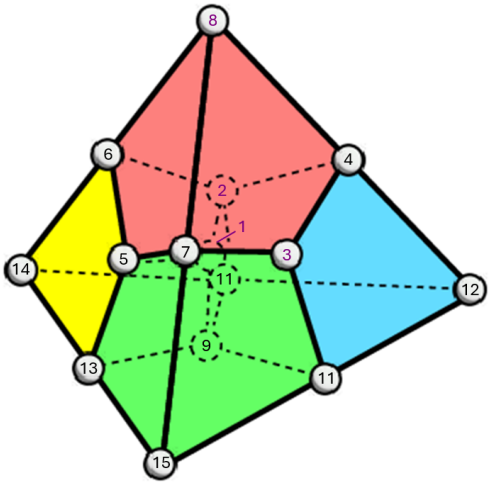


Figure 4.10: Visualization of the stabilizer cell structure of the $[[15,1,3]]$ code. The four qubits highlighted in purple (1, 2, 3, 8) represent the support of the error $X_1X_2X_3X_8$. The top cell (red) and three bottom cells (yellow, green, blue) correspond to the X-type stabilizer generators. Any combination of these fails to yield a logical X operator that includes all four marked qubits. Figure adapted from [39].

Generalization This reasoning generalizes: a weight-4 X-error $E = X(f)$ results in a logical error under minimum-weight decoding if and only if there exists $\mathbf{c} \in \mathbb{F}_2^4 \setminus \{\mathbf{0}\}$ such that

$$\forall i \in \mathrm{supp}(f): \quad c_1 h_i^1 + c_2 h_i^2 + c_3 h_i^3 + c_4 h_i^4 = 0.$$

In the case of $f = (1,1,1,0,0,0,0,1,...)$, this does not hold. Therefore, the preparation is fault-tolerant with respect to the $[[15,1,3]]$ code.

4.5. State preparation of the $[[7,1,3]]$ code

4.5.1. Fault tolerance of the state preparation of the $[[7,1,3]]$ code

Lemma 2. *The $|0_L\rangle$ state in the $[[7,1,3]]$ (Steane) code can be prepared fault-tolerantly, in the sense that any single fault during preparation or verification results in an error that is correctable by the $[[15,1,3]]$ code.*

Fault model and assumptions As discussed in Section 4.2.1, we assume a single-fault model in which any error introduced by a fault has weight at most two at the moment of occurrence and may propagate to a higher-weight X-error. Only X-type errors are considered in this analysis, as logical Z errors have no effect on the logical $|0_L\rangle$ state.

Preparation and verification strategy. The Steane $|0_L\rangle$ state is prepared via a standard encoding circuit, followed by a verification step that uses two flag qubits. If both flag measurements yield -1, the prepared state is discarded and the procedure is restarted. The full circuit is shown in Figure 4.11.

This circuit design is based on the two-flag verification scheme used in deterministic fault-tolerant state preparation by Heußen and Hilder [37], but here we omit the final correction step. Compared to single-flag circuits such as those proposed by Zen et al. [3] and Goto [36], using two flag qubits has the advantage of a higher acceptance rate: states that trigger a single flag due to a harmless weight-1 error on the flag qubit are still accepted, rather than unnecessarily discarded.

The unitary part of the Steane preparation circuit, including verification, has depth 7, which is the same as the depth of the $[[8,3,2]]$ preparation circuit. Although the final measurement of the flag qubits requires additional time, this allows the two components to be prepared largely in parallel.

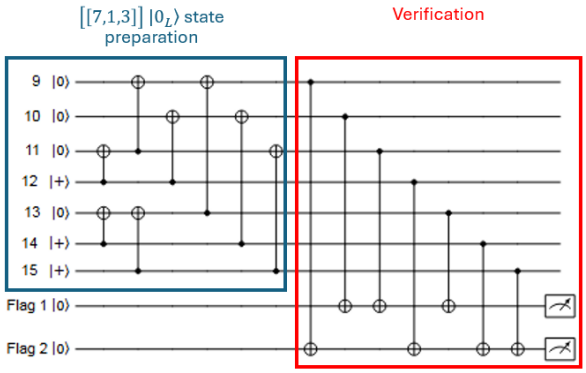


Figure 4.11: Circuit for preparing the $|0_L\rangle$ state in the $[[7,1,3]]$ (Steane) code with verification using two flag qubits.

This verification step is motivated by the structure of the full logical state preparation. Specifically, the Steane block acts as the control in a subsequent quasitransversal CNOT, applied between the Steane block and a $|(0,0,0)_L\rangle$ state encoded in the $[[8,3,2]]$ code. Under this operation, any X-error on the control propagates to the target: more precisely, $X \mapsto X \otimes X$ under conjugation by CNOT. Thus, an X-error of weight w on the Steane block becomes an X-error of weight $2w$ on the combined system. Since the $[[15,1,3]]$ code has $d_Z^+ = 7$, all X-errors of weight up to 3 are correctable. Therefore, weight-1 X-errors are acceptable, as they propagate to weight 2. However, weight-2 X-errors become weight-4 errors and may be uncorrectable. It is therefore essential that the verification step detect and reject all weight-2 X-errors.

Detection of weight-2 X-errors.

Proof. Under the single-fault model and circuit structure assumed here, there are exactly two weight-2 X-errors on the Steane block that may arise from a single fault and are not detected by the stabilizers of the Steane code. After propagation through the quasitransversal CNOT, both of these errors become uncorrectable by the $[[15,1,3]]$ code under minimum-weight decoding.
The first such error is X_9X_{11}, which propagates to

$$X_9X_{11} \mapsto X_1X_3X_9X_{11}.$$

Let $\overline{X} = X_{\text{all}}$ denote the logical X operator of the $[[15,1,3]]$ code, and recall the stabilizers

$$s_{x,2} = X_1X_2X_3X_4X_9X_{10}X_{11}X_{12}, \quad s_{x,4} = X_1X_3X_5X_7X_9X_{11}X_{13}X_{15}.$$

Multiplying, we obtain

$$X_1X_3X_9X_{11} \cdot \overline{X} \cdot s_{x,2} \cdot s_{x,4} = X_6X_8X_{14},$$

which has lower weight and the same syndrome. Since the product of the two errors is a logical operator, minimum-weight decoding may apply $X_6X_8X_{14}$, resulting in a logical error.
The second possible weight-2 error is X_9X_{13}, which propagates to $X_1X_5X_9X_{13}$. Let

$$s_{x,3} = X_1X_2X_5X_6X_9X_{10}X_{13}X_{14}.$$

Then

$$X_1X_5X_9X_{13} \cdot \overline{X} \cdot s_{x,3} \cdot s_{x,4} = X_4X_8X_{12}.$$

Again, the decoder may apply this lower-weight correction, introducing a logical error.
These are the only two weight-2 X-errors that can result from a single fault in the circuit under the adopted assumptions, and both are uncorrectable. The verification circuit is designed to reject precisely these cases. In both Figure 4.12 and Figure 4.13, the fault paths leading to these errors activate both flags, and the state is discarded.

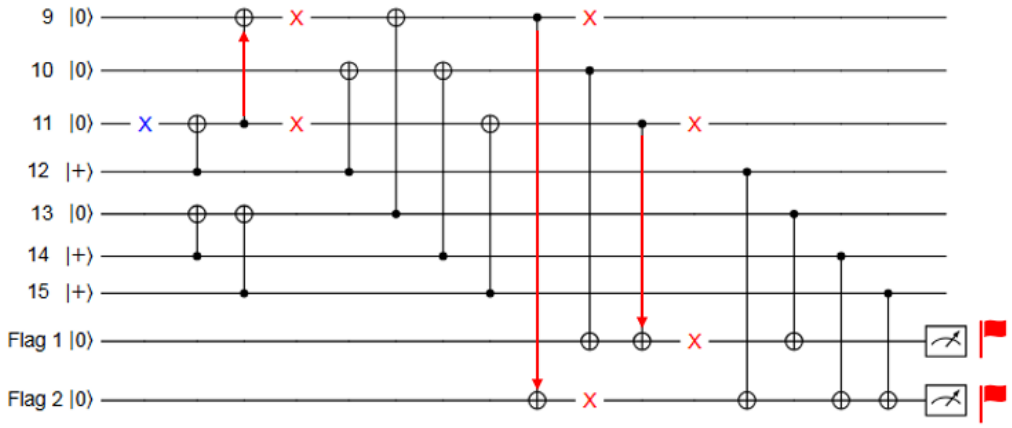


Figure 4.12: Single fault during Steane $|0_L\rangle$ preparation circuit. The blue X marks the initial fault, which occurs on qubit 11. This fault propagates through subsequent CNOT gates, resulting in two red X errors on qubits 9 and 13. The red arrows indicate the paths of propagation. The final error has weight 2 and is detected by both flag qubits, causing the state to be discarded.

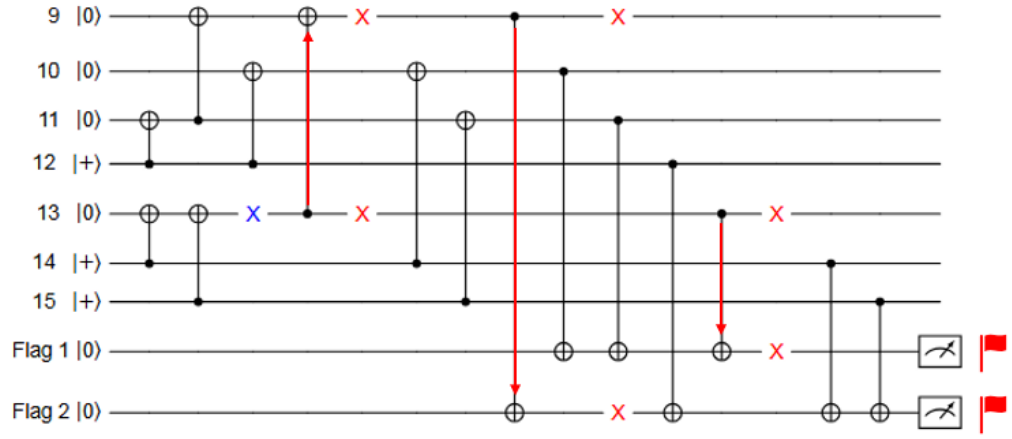


Figure 4.13: Another example of a single fault during Steane $|0_L\rangle$ preparation. The blue X marks the initial fault on qubit 13. This error propagates through the circuit, leading to red X errors on qubits 9 and 14. The red arrows indicate the propagation paths via CNOT gates. The resulting weight-2 X-error is detected by both flag qubits, and the faulty state is discarded.

Other faults. If a single fault occurs during the verification step itself, it can affect at most one qubit in the data block or corrupt one of the flag measurements. In such cases, either the final error remains correctable by the $[[15,1,3]]$ code, or the state is falsely rejected. Neither scenario violates the adopted definition of fault tolerance.

Conclusion. The verification circuit filters out all weight-2 X-errors that could arise from a single fault and become uncorrectable after the quasitransversal CNOT. All remaining errors are of weight at most 1 and propagate to correctable errors. Therefore, the preparation of the Steane $|0_L\rangle$ state is fault-tolerant relative to the $[[15,1,3]]$ code. □

4.6. The fault-tolerant preparation of the logical $|0_L\rangle$ state in the $[[15,1,3]]$ code

Lemma 3. *If the* $[[8,3,2]]$ *and* $[[7,1,3]]$ *components are prepared fault-tolerantly as described in Lemmas 1 and 2, and a single fault occurs anywhere during these preparations or in the CNOT layer, then the final output of the circuit in Figure 4.1 is a correctable error on the logical* $|0_L\rangle$ *state of the* $[[15,1,3]]$ *code.*

Setup. Let the input be a $|(0,0,0)_L\rangle$ state in the $[[8,3,2]]$ code (prepared as in Lemma 1) and a verified $|0_L\rangle$ state in the $[[7,1,3]]$ (Steane) code (prepared as in Lemma 2). The Steane code acts as the control in a quasitransversal CNOT operation applied across corresponding qubits in the two codes. The combined system is then in the $[[15,1,3]]$ code space.

Correctness. Assume that the input states lie in the code spaces of the $[[8,3,2]]$ and $[[7,1,3]]$ codes, and are stabilized by their respective stabilizer groups and logical Z operators. Then, as shown in Section 4.3.4, the output state is stabilized by the $[[15,1,3]]$ stabilizer group and its logical Z operator. Hence, the construction correctly prepares logical $|0_L\rangle_{[[15,1,3]]}$.

Fault tolerance. Case analysis. We now consider all possible locations where a single fault may occur:

Case 1: Fault in the $[[8,3,2]]$ preparation.
This case is covered by Lemma 1. The worst-case propagated error is a weight-4 X-error that remains correctable by the $[[15,1,3]]$ code.

Case 2: Fault in the $[[7,1,3]]$ preparation.
If the fault occurs during the Steane encoding circuit and the verification has passed, then the resulting error has weight at most 1. As shown in Lemma 2, such an error will propagate to an error of weight at most 2 after the quasitransversal CNOT, which is still within the correctable range of the $[[15,1,3]]$ code.

Case 3: Fault in the flag qubit interaction.
A fault may occur in the flag circuit, either introducing a data error or corrupting one of the flags. Lemma 2 guarantees that all such cases either trigger rejection or result in a correctable error.

Case 4: Fault during measurement of the flag qubits.
Measurement faults may lead to false positives (rejection of a valid state) or false negatives (acceptance with a weight-1 error). Since the accepted state still only has weight at most 1 on the Steane block, this propagates to at most weight 2, which is still correctable by the $[[15,1,3]]$ code.

Case 5: Fault during the quasitransversal CNOT.
Each CNOT gate acts between a single pair of qubits. A fault in such a gate can lead to an X-error on either qubit, or both. This results in an error of weight at most 2 on the combined system. Since $d_Z^+ = 7$, all X-errors of weight up to 3 are correctable by the $[[15,1,3]]$ code.

Conclusion. In all single-fault scenarios, the resulting error is guaranteed to be correctable by the $[[15,1,3]]$ code. Therefore, the full preparation procedure, including the quasitransversal CNOT, is fault-tolerant under the adopted model. □

Visual summary. The full construction is shown in Figure 4.1, with the $[[7,1,3]]$ preparation (blue), verification step (red), and final CNOT layer (green).

4.7. General fault tolerance for PQRM with $d = 3$ and $n = 1$

The family of punctured quantum Reed-Muller codes with parameters $[[2^m - 1, 1, 3]]$ can be identified with a subclass of $(m-1)$-dimensional colour codes, as shown in Appendix B of [9]. In particular, the $[[7,1,3]]$ and $[[15,1,3]]$ codes correspond to the cases $m = 3$ and $m = 4$, respectively. This connection motivates our focus on these codes for recursive constructions with IV.

Moreover, increasing m in $\mathrm{PQRM}(1, m-2, m)$ allows transversal implementation of diagonal gates $R_{m-1} = \mathrm{diag}(1, e^{2\pi i/2^{m-1}})$, which enables smaller-angle phase rotations in fault-tolerant quantum circuits (see Section 3.2.3).

The fault-tolerant preparation scheme studied in this section is depicted in Figure 4.14. Throughout this section, we assume $m \geq 4$. This restriction arises from the structure of our construction: we consider circuits in which a distance-3 code with $r_x = 1$ is prepared first, and then used as part of a larger state via a quasitransversal CNOT layer. Since the smallest such code is the $[[7,1,3]]$ Steane code (corresponding to $m = 3$), the resulting output code necessarily has $m \geq 4$.

Since all codes under consideration have distance $d = 3$, they must be able to correct any single fault. We therefore analyse whether any single fault in the preparation circuit can lead to a logical error.

In the following subsection, we treat all possible locations of a single fault and show that none of them lead to an uncorrectable error.

4.7.1. Fault tolerance of intermediate verification for PQRM code with $r_x = 1$

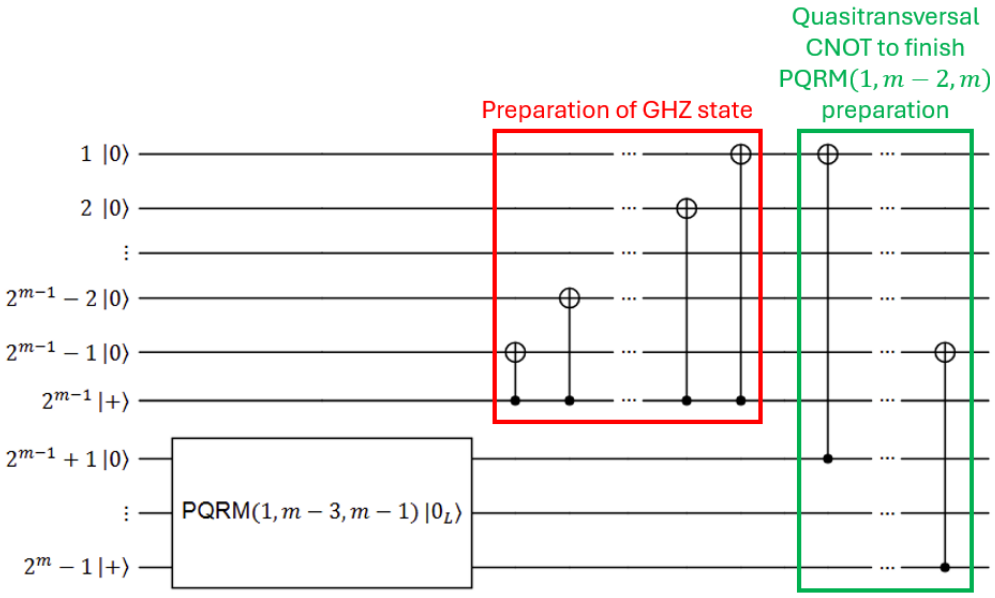


Figure 4.14: Circuit representation of state preparation for $\mathrm{PQRM}(1, m-2, m)$, for general $m \geq 4$. The upper red box prepares a GHZ state on the first 2^{m-1} qubits. The lower-left block prepares a $|0_L\rangle$ state in $\mathrm{PQRM}(1, m-3, m-1)$, e.g. the $[[15, 1, 3]]$ code for $m = 4$. The final layer (green) applies a quasitransversal CNOT to complete the preparation. This construction generalizes the scheme from Figure 4.1, which used the Steane code ($\mathrm{PQRM}(1, 1, 3)$) and the $[[15, 1, 3]]$ code ($\mathrm{PQRM}(1, 2, 4)$) as base cases.

We now analyse the fault tolerance of the circuit in Figure 4.14.

Depending on the location of the single fault, we distinguish the following cases:

- **Case 1:** The fault occurs during the preparation of the GHZ state in the top half.
- **Case 2:** The fault occurs during the preparation of the $\mathrm{PQRM}(1, m-3, m-1)$ block.
- **Case 3:** The fault occurs during the final quasitransversal CNOT layer.

We now analyse each case in turn.

Case 1: Fault in GHZ preparation (top half)

Lemma 4. *Let $m \geq 4$, and let $f = e_1 + e_2 + \cdots + e_{2^{m-2}-1} + e_{2^{m-1}} \in \mathbb{F}_2^n$ be the support vector of an X-type error arising from a single fault in the GHZ preparation of the top half of the $\mathrm{PQRM}(1, m-2, m)$ code. The propagation of such an error is shown in Figure 4.15. Let $E = X(f)$. Then there does not exist any X-type Pauli error $E' = X(f')$ with $|f'| \leq |f|$ such that $E'E$ is equivalent to the logical operator X_{all} up to multiplication by X-type stabilizers.*

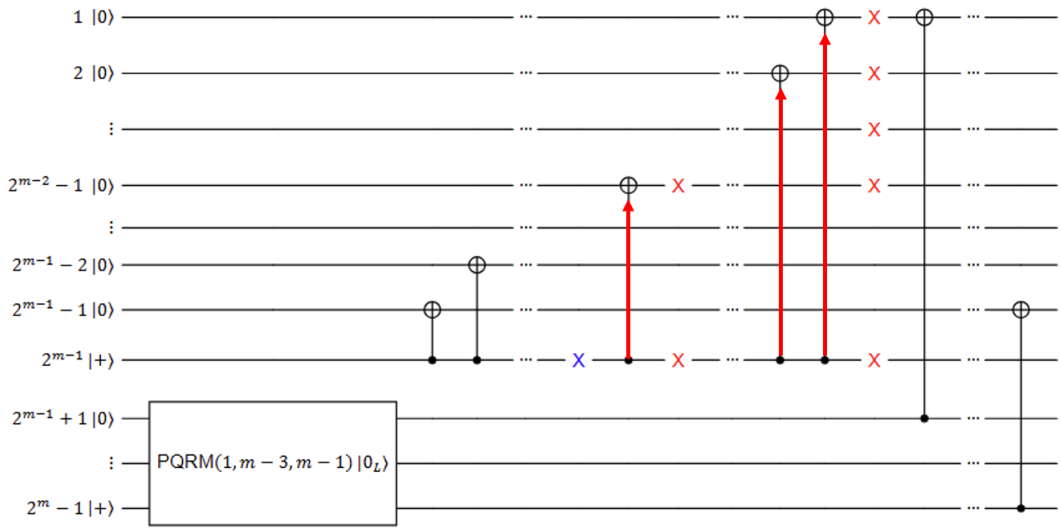


Figure 4.15: Visualization of how an X-type error on qubit 2^{m-1} (blue) propagates through the GHZ state in the first stage of the circuit. The red arrows indicate propagation through CNOTs from control to target, resulting in an error $E = X_1 \cdots X_{2^{m-2}-1} X_{2^{m-1}}$ of weight 2^{m-2}. This error is shown to be correctable under minimum-weight decoding in Lemma 4.

Proof. We first describe the structure of the X-type stabilizers. The $n = 2^m - 1$ qubits are labelled by the binary vectors $v_i = \mathrm{bin}(2^m - i)$ for $i = 1, \ldots, n$, with the most significant bit first. The X-type stabilizer generators are $h_1, \ldots, h_m \in \mathbb{F}_2^n$, defined by:

$$h_k(i) = v_i^{(k)},$$

i.e., h_k is the k-th coordinate function. The logical operator X_{all} corresponds to the all-ones vector $\mathbf{1} \in \mathbb{F}_2^n$.
Suppose for contradiction that there exists $c = (c_1, \ldots, c_m) \in \mathbb{F}_2^m$ such that

$$\mathrm{supp}(f) \subseteq \mathrm{supp}\left(\mathbf{1} + \sum_{k=1}^{m} c_k h_k\right).$$

Then for all $i \in \mathrm{supp}(f)$,

$$\sum_{k=1}^{m} c_k h_k(i) = 0.$$

We now use m carefully chosen qubits in $\mathrm{supp}(f)$ to show that $c = 0$:
(1) Qubit 2^{m-1}: its binary label is $(1, 0, \ldots, 0)$, so $h_1(i) = 1$ and $h_k(i) = 0$ for $k > 1 \Rightarrow c_1 = 0$.
(2) Qubits $2^k + 1$ for $k = 0, \ldots, m-3$: each has a single zero at position $m - k$, so $h_{m-k}(i) = 0$, and $h_j(i) = 1$ for $j \neq m - k$. This gives:

$$\sum_{j \neq m-k} c_j = 0.$$

(3) Qubit 1: binary label is $(1, 1, \ldots, 1) \Rightarrow \sum_{k=1}^{m} c_k = 0$.
Together, these m linearly independent equations imply $c = 0$.
Thus, $\sum_k c_k h_k = 0$, and

$$X_{\mathrm{all}} \cdot \prod_k X(h_k)^{c_k} = X_{\mathrm{all}}.$$

Now suppose, for contradiction, that a minimum weight decoder interprets $E = X(f)$ as leading to a logical fault X_{all}. Then there exists an X-type Pauli error $E' = X(f')$ with $|f'| \leq |f|$ such that:

$$E'E = X(f')X(f) = X(f + f') \sim X_{\mathrm{all}}.$$

This implies $f' = \mathbf{1} + f$, so:

$$|f'| = |f + \mathbf{1}| \geq |\mathbf{1}| - |f| = (2^m - 1) - 2^{m-2} = 3 \cdot 2^{m-2} - 1.$$

But $|f| = 2^{m-2}$, so $|f'| > |f|$, contradicting the assumption of minimum-weight decoding.
Therefore, such E' cannot exist, and E will be correctly decoded. □

Case 2: Fault in intermediate $\mathrm{PQRM}(1, m-3, m-1)$ block
So far, we have considered faults arising in the top half of the circuit. For completeness, we now examine the case where a fault occurs during the preparation of the $\mathrm{PQRM}(1, m-3, m-1)$ block. We show that if such an error is correctable at that stage, it remains correctable after propagation through the quasitransversal CNOTs.

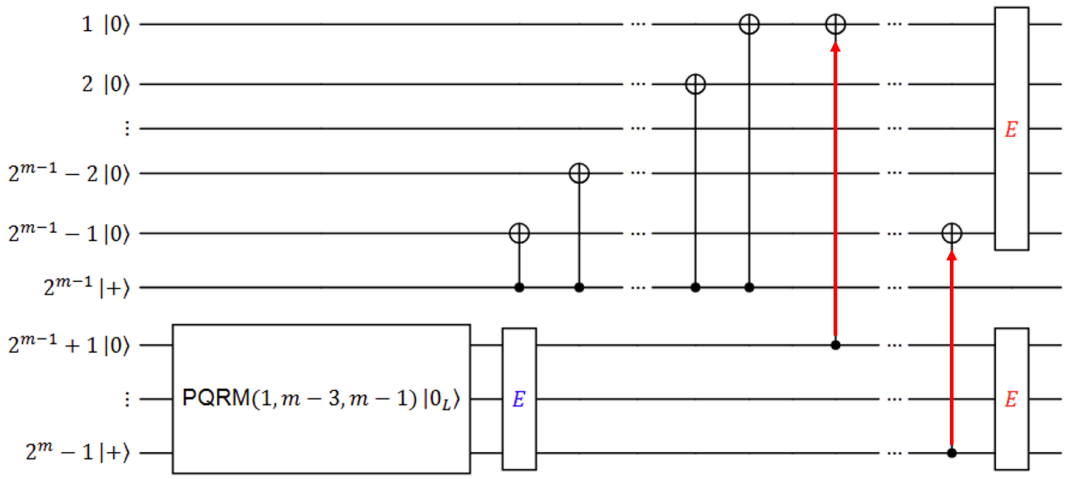


Figure 4.16: Visualization of an X-type error E (blue) occurring during the preparation of a $\mathrm{PQRM}(1, m-3, m-1)$ state. The error propagates through the quasitransversal CNOTs into a symmetric error $E \otimes I \otimes E$, shown in red. Lemma 5 proves that such errors remain correctable in the larger $\mathrm{PQRM}(1, m-2, m)$ code.

This is illustrated in Figure 4.16.
We now analyse whether a single fault during the preparation of the logical zero state $|0_L\rangle$ of the $\mathrm{PQRM}(1, m-2, m)$ code can result in an uncorrectable X-type error. To this end, we assume that the preparation of the lower-level code $\mathrm{PQRM}(1, m-3, m-1)$ involved at most one fault. We then examine which error E this could have caused on the lower block and whether the resulting propagated error remains correctable in $\mathrm{PQRM}(1, m-2, m)$.
There are three relevant types of single-fault-induced errors to consider, each leading to a different propagated error in $\mathrm{PQRM}(1, m-2, m)$:

1. A residual error from the preparation of the Steane state ($\mathrm{QRM}(1, 1, 3)$), which has weight exactly 1 after verification. This error propagates through $m-3$ transversal CNOT layers, resulting in an error of weight exactly 2^{m-3}.

2. A single-qubit fault occurring during one of the transversal CNOT layers used to prepare $\mathrm{PQRM}(1, m-3, m-1)$. This can propagate through up to $m-3$ layers, resulting in an error of weight at most 2^{m-3} on the final code.

3. A fault during the GHZ preparation in the upper part of a subcode $\mathrm{PQRM}(1, m'-2, m')$, for some $m' \leq m-1$. Such a fault may have weight up to $2^{m'-2}$ at that level. After propagation through the remaining $m - m'$ transversal CNOT layers, it results in an error of weight up to 2^{m-2}.

Since the code $\mathrm{PQRM}(1, m-2, m)$ has $d_Z^+ = 2^{m-1} - 1$ (see Equation (3.22)), it can correct all X-type errors of weight at most $2^{m-2} - 1$. This implies that the third case may exceed the correctable weight and must therefore be analysed explicitly. This is the purpose of the following lemma.

Lemma 5. *Let E be a correctable X-type error in $\mathrm{PQRM}(1, m-3, m-1)$. Then the propagated error*

$$E' = E \otimes I \otimes E$$

is correctable in $\mathrm{PQRM}(1, m-2, m)$.

Proof. Let H_X be the parity-check matrix of size $(m-1) \times n$, where $n = 2^{m-1} - 1$, defining the X-type stabilizers of $\mathrm{PQRM}(1, m-3, m-1)$, and let

$$H'_X = \begin{pmatrix} H_X & 0 & H_X \\ \mathbf{1} & 1 & \mathbf{0} \end{pmatrix}$$

be the $m \times n'$ check matrix for $\mathrm{PQRM}(1, m-2, m)$, where $n' = 2^m - 1$. This construction reflects the recursive structure of the underlying classical Reed–Muller codes, as described in Section 3.1.3.1. The X-type stabilizers of the larger code are generated by

$$s'_{x,i} = s_{x,i} \otimes I \otimes s_{x,i} \quad \text{for } i = 1, \ldots, m-1, \quad s'_{x,m} = X_{1,\ldots,n} \otimes X_{n+1} \otimes I.$$

Assume for contradiction that $E' = E \otimes I \otimes E$ is not correctable. Then there exists a logical operator $X_{\log}$ and a stabilizer S such that

$$\mathrm{supp}(E') \subseteq \mathrm{supp}(X_{\log} \cdot S).$$

Any such stabilizer S has the form

$$S = \prod_{i=1}^{m-1} (s_{x,i} \otimes I \otimes s_{x,i})^{c_i} \cdot \left(X_{1,\ldots,n} \otimes X_{n+1} \otimes I\right)^{c_m},$$

for some $c \in \mathbb{F}_2^m$. Since E' is symmetric under exchange of the first and third block, and the operator $\left(X_{1,\ldots,n} \otimes X_{n+1} \otimes I\right)$ breaks this symmetry, we must have $c_m = 0$.
It follows that

$$\mathrm{supp}(E) \subseteq \mathrm{supp}\left(X_{1,\ldots,n} \cdot \prod_{i=1}^{m-1} s_{x,i}^{c_i}\right).$$

We distinguish two cases:

Case 1: $\mathrm{wt}(E) \leq 2^{m-3} - 1$**.** Then $\mathrm{wt}(E') = 2 \cdot \mathrm{wt}(E) \leq 2^{m-2} - 2$, which is strictly below the correction threshold $t_X = 2^{m-2} - 1$. Hence E' is correctable.

Case 2: $\mathrm{wt}(E) > 2^{m-3} - 1$**.** The decoder would apply the correction

$$E'' = X_{1,\ldots,n} \cdot \prod_{i=1}^{m-1} s_{x,i}^{c_i} \cdot E,$$

which has weight strictly less than $\mathrm{wt}(E)$. This results in a residual error equal to the logical operator $X_{1,\ldots,n}$, so the correction introduces a logical error. This contradicts the assumption that E is correctable.

Conclusion. In both cases, we reach a contradiction with the assumption that E' is not correctable. Therefore, E' must be correctable in $\mathrm{PQRM}(1, m-2, m)$. □

Case 3: Fault during final quasitransversal CNOT layer
Finally, consider a fault that occurs during the application of the quasitransversal CNOT layer. Such a fault affects at most two qubits: one in the control block and one in the target block. Since we consider preparation circuits based on $\mathrm{PQRM}(1, m-2, m)$ codes with $m \geq 4$, these codes have $d_Z^+ = 2^{r_z+1} - 1 \geq 7$, and are therefore capable of correcting all X-type errors of weight up to 3. It follows that any such fault during the final CNOT layer is correctable.

Conclusion. All three types of single faults lead to X-type errors that remain correctable under minimum-weight decoding, due to the symmetric structure and the distance $d_Z^+ = 2^{m-1} - 1$ of the final code. This confirms the fault tolerance of the preparation circuit for $\mathrm{PQRM}(1, m-2, m)$.

4.8. Fault propagation under quasitransversal CNOTs

QRM codes have recursive encoding circuits: larger codes are built from smaller codes by applying quasitransversal CNOT gates between encoded blocks. For example, to prepare the logical $(0,0,0)_L$ state in the $[[8,3,2]]$ code, one can first prepare two blocks encoded in the $\mathrm{PQRM}(0,0,2)$ code (i.e., $[[4,2,2]]$) in the logical $(0,0)_L$ state, followed by a layer of CNOTs applied transversally between corresponding qubits of the two blocks.
Figure 4.17 shows such a construction: the first $[[4,2,2]]$ state is prepared and verified (with a flag), and is then coupled to a second freshly initialized block via quasitransversal CNOTs. This completes preparation of the $[[8,3,2]]$-encoded $|000\rangle_L$ state.

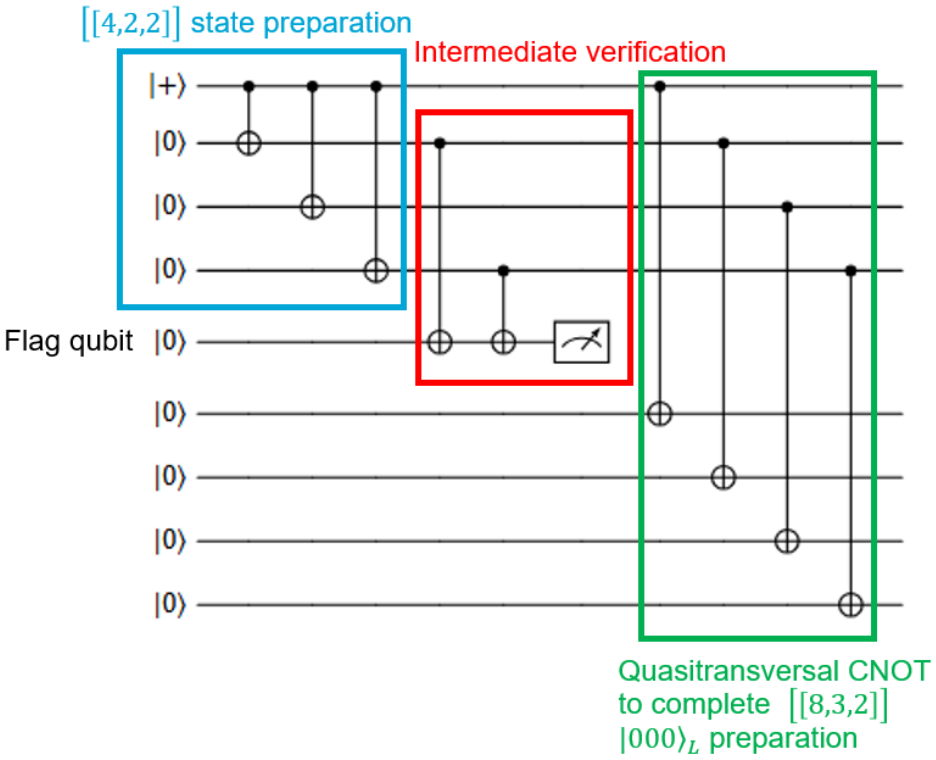


Figure 4.17: State preparation via IV and quasitransversal CNOT.

Although this example involves the preparation of a multiqubit logical state ($k = 3$), the same underlying structure, combining smaller encoded blocks via quasitransversal CNOTs, can also be used in recursive constructions of codes that encode a single logical qubit. This is particularly relevant for PQRM codes, which are defined to have $k = 1$ by choosing parameters $r_x, r_z \geq 1$ such that $r_x + r_z + 1 = m$, see Eq. 3.21.
Figure 4.18 shows such a construction step. A $\mathrm{PQRM}(r_x, r_z, m)$-encoded $|0_L\rangle$ state is prepared from:

- a block of $k = \binom{m-1}{r_x}$ logical qubits in the state $|0\rangle_L^{\otimes k}$, encoded using $\mathrm{QRM}(r_x-1, r_z-1, m-1)$,
- and a single $|0\rangle_L$ state encoded in $\mathrm{PQRM}(r_x, r_z-1, m-1)$.

To ensure that undetectable X-type errors have higher minimum weight than undetectable Z-type errors, we impose the condition $r_x \leq r_z - 1$, which implies $r_z \geq 2$. Under this assumption, and using Eq. 3.22, the code $\mathrm{PQRM}(r_x, r_z, m)$ satisfies $d_Z^+ > d_X^+$.

This separation is crucial for fault-tolerant state preparation, since X-type faults may propagate and grow across blocks under quasitransversal CNOTs. A higher minimum weight for undetectable X-type errors ensures that such faults remain correctable in the output code.
This recursive structure increases the d_Z^+ at each step: the lower left code has $d_Z^+ = 2^{r_z} - 1$, and the output code achieves $d_Z^{+'} = 2^{r_z+1} - 1$. However, as we discuss in the next subsection, the CNOT layer may spread errors across blocks, which imposes constraints on the fault tolerance of the recursive scheme in Figure 4.18.

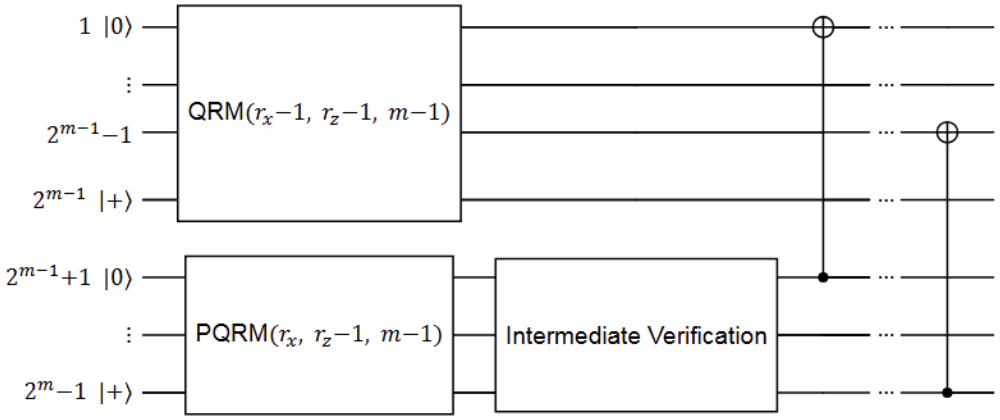


Figure 4.18: Recursive construction of a $\mathrm{PQRM}(r_x, r_z, m)$-encoded $|0_L\rangle$ state, under the assumption $r_x, r_z \geq 1$ and $r_x + r_z + 1 = m$. The top block prepares $k = \binom{m-1}{r_x}$ logical qubits in the state $|0\rangle_L^{\otimes k}$ using a $\mathrm{QRM}(r_x-1, r_z-1, m-1)$ code. The bottom block prepares a single logical $|0\rangle_L$ in $\mathrm{PQRM}(r_x, r_z-1, m-1)$ with $d_Z^+ = 2^{r_z} - 1$, and applies an IV. A quasitransversal CNOT layer then combines the two into a $\mathrm{PQRM}(r_x, r_z, m)$-encoded state with $d_{Z,final}^+ = 2^{r_z+1} - 1$.

Fault growth and correctability after coupling

Suppose each input block is a code with minimum weight d_Z^+ for undetectable X-type errors. Then each block can correct up to

$$t_X = \left\lfloor \frac{d_Z^+ - 1}{2} \right\rfloor$$

faults of this type. Any X-error of weight up to t_X will go undetected by the IV step, since it is within the correctable range of the block.
Such a fault, however, may grow during the subsequent coupling: if the error occurs before the quasitransversal CNOT layer, it may spread across multiple blocks and become an X-error of weight up to $2t_X$ in the output code.
To ensure fault tolerance, the output code must be able to correct such propagated errors. If its minimum weight for undetectable X-type errors is $d_{Z,\mathrm{final}}^+$, then this requires

$$\left\lfloor \frac{d_{Z,\mathrm{final}}^+ - 1}{2} \right\rfloor \geq 2t_X.$$

This condition implies that the output code must have strictly higher d_Z^+ than the input blocks:

$$d_{Z,\mathrm{final}}^+ > d_Z^+.$$

This inequality is necessary to compensate for potential fault growth after IV. In practice, some higher-weight errors may still be correctable depending on their stabilizer syndrome, but the bound above ensures fault tolerance under worst-case propagation and greatly simplifies the analysis by avoiding case-by-case considerations.

The challenge for higher d

We consider the construction shown in Figure 4.18, where a $|0_L\rangle$ state is prepared from smaller QRM and PQRM blocks via IV and a quasitransversal CNOT layer. For simplicity, we assume $k = 1$, so that a single logical qubit is encoded.

In this setup:

- The control block is a QRM code with parameters $[2^{m-1}, k, 2^{r_X}]$.
- The target block is a PQRM code with parameters $[2^{m-1}-1, 1, 2^{r_X+1}-1]$.
- The output code is a PQRM code with parameters $[2^{m}-1, 1, 2^{r_X+1}-1]$.

We assume throughout that $r_X \geq 1$ and $r_Z - 1 \geq r_X$, which implies $d_Z^+ \geq d_X^+$. Let

$$d_X^+ = 2^{r_X+1}-1, \quad d_Z^+ = 2^{r_Z}-1, \quad \text{and} \quad d_{Z,final}^+ = 2^{r_Z+1}-1.$$

Then the distance of the output code is

$$d = \min(d_Z^{+'}, d_X^+) = d_X^+,$$

and the corresponding number of faults that should be correctable for fault tolerance is

$$\alpha \leq \left\lfloor \frac{d-1}{2} \right\rfloor.$$

Since we want to analyse the worst case, we assume equality in the above equation. We define the number of correctable X errors before and after the quasitransversal layer as

$$t_X = \left\lfloor \frac{d_Z^+ - 1}{2} \right\rfloor, \qquad t_X' = \left\lfloor \frac{d_{Z,final}^+ - 1}{2} \right\rfloor = 2t_X + 1.$$

As discussed in Section 3.2.4, a state preparation circuit is fault-tolerant if every configuration of α faults leads to a correctable final error. We now analyse whether this still holds when only IV is used.

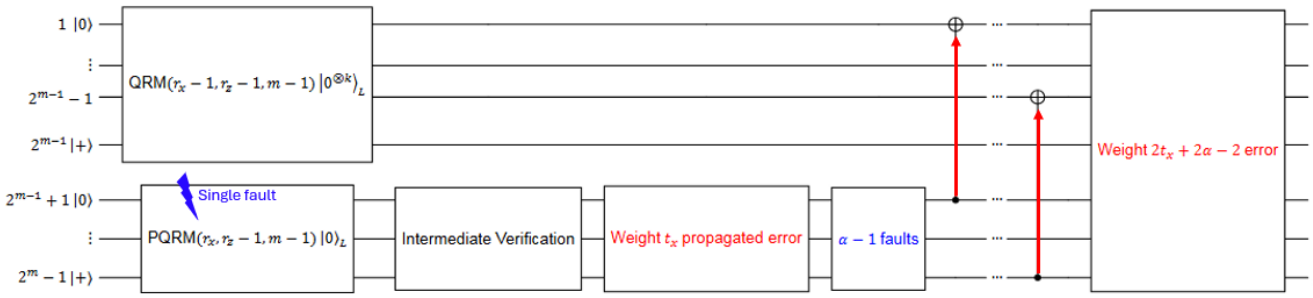


Figure 4.19: Fault propagation scenario in the preparation of a $|0_L\rangle$ state in a $\mathrm{PQRM}(r_X, r_Z, m)$ code. A single fault occurs in the control block (a $\mathrm{PQRM}(r_X, r_Z-1, m-1)$ code) and propagates during the preparation of the larger code via quasitransversal CNOTs. After IV, $\alpha-1$ additional faults occur in the control block. Each of these propagates an X error into the target block, resulting in a total X error of weight $2t_X + 2\alpha - 2$ at the output.

We illustrate this with the fault-propagation scenario shown in Figure 4.19, which serves as the basis for the analysis below. We analyse whether fault tolerance still holds when only IV is used.
We consider the following critical scenario:

- One fault occurs early in the control block, before the quasitransversal CNOTs. In principle, this may propagate to an X-error in the target block with weight greater than t_X.
- However, such high-weight errors are rejected by IV. We thus consider the case where the propagated error has weight at most t_X and is accepted.
- After verification, $\alpha-1$ additional faults occur in the control block.
- Each such fault introduces an X error in control and a propagated X error in the target block.

The resulting X-error at the end of the preparation consists of:

$$t_X \quad \text{(control)} + \quad t_X \quad \text{(propagated)} + \quad (\alpha - 1) \quad \text{(control)} + \quad (\alpha - 1) \quad \text{(propagated)},$$

yielding a total of

$$2t_X + 2(\alpha - 1) = 2t_X + 2\alpha - 2.$$

This must be compared to the X-error threshold $t_X' = 2t_X + 1$. Comparing both expressions shows that

$$2t_X + 2\alpha - 2 > 2t_X + 1 \quad \Leftrightarrow \quad \alpha > 1.$$

Thus, for all $\alpha > 1$, the resulting error weight may exceed the correctability guarantee, regardless of the value of t_X.

Therefore, for $\alpha > 1$, the output error weight may exceed the correctability guarantee. Although such final errors may still be correctable, this is not ensured by the code distances alone. Analysing correctability in all such cases becomes computationally expensive due to the large number of possible fault combinations.

This analysis generalizes the setting of Section 4.7.1 to $r_X \geq 1$ and $r_Z - 1 \geq r_X$. While it does not establish that IV is insufficient for $\alpha > 1$ or equivalently $d > 3$, it shows that further investigation is needed to determine whether additional final verification is required. Such a classification would rely on explicit enumeration or simulation of final errors and lies beyond the scope of this thesis.

In addition, it would be worthwhile to explore how the balance between intermediate and final verification affects practical performance metrics. For example, different strategies may vary in their impact on the logical error probability, the time until IV is performed, or the total number of required measurements. Understanding these trade-offs is essential for the design of efficient verification schemes in realistic fault-tolerant quantum architectures.

5 Results

We compare four fault-tolerant state preparation circuits for the $[[15, 1, 3]]$ code. All prepare the logical zero state $|0\rangle_L$, and each includes a single verification step: either intermediate or final. The circuits are:

- **IV with 2 ancillas:** The proposed method from Section 4.2, show in Figure 4.1, using the Steane subcircuit from Section 4.5.1.
- **IV with 1 ancilla:** A variant of the above using the Steane subcircuit from [36], shown in Figure 3.9. The Goto-based circuit requires one fewer measurement and has a total depth of 5, compared to depth 6 for the author's own version, even when one of the flags and its associated CNOT gates are omitted from the author's own circuit.
- **Final verification (Peham):** A 1-ancilla circuit from [4], shown in Figure 3.7.
- **Final verification (Munich):** A 2-ancilla circuit generated with the same tool as Peham, but not published in their paper. Shown in Figure 3.8.

The Python code used to simulate the circuits, compile them for the hardware, execute them on the device, and generate the figures and results in this chapter is available at:

`https://github.com/sbirkenhager/master_thesis`

The 1-ancilla IV and Peham circuits were selected for hardware implementation as they are the shortest compiled circuits for intermediate and final verification, respectively (see Section 5.4).
We evaluate the circuits using three metrics: the logical error probability p_{log}, the acceptance probability p_{acc}, and the expected runtime in circuit depth. All simulations use a circuit-level noise model and error proability p specified in Section 3.2.1, and are run using Stim. Emulator and hardware runs were performed on Quantinuum's H1-1 system. The emulator uses the same native gate set and constraints as the hardware, with realistic noise including leakage, crosstalk, and dephasing due to transport and idling. Quantum error correction is only meaningful when $p_{\text{log}} < p$, so we restrict attention to that regime in Sections 5.2 and 5.3.

5.1. Logical error probability

Figure 5.1 shows p_{log} as a function of p on a log-log scale. Each simulation point is based on 10^6 runs. A dashed line proportional to p^2 is included to reflect the quadratic relation predicted by Eq. (3.23) in Section 3.2.
A run is counted as a logical error when the minimum-weight Pauli correction inferred from the measured syndrome predicts a different outcome of the $\overline{Z}$ measurement than the actual outcome.
Among the four circuits tested, the final verification circuits show better logical performance than the IV circuits for $p \gtrsim 0.0006$. In that range, the error bars remain small enough that the differences are

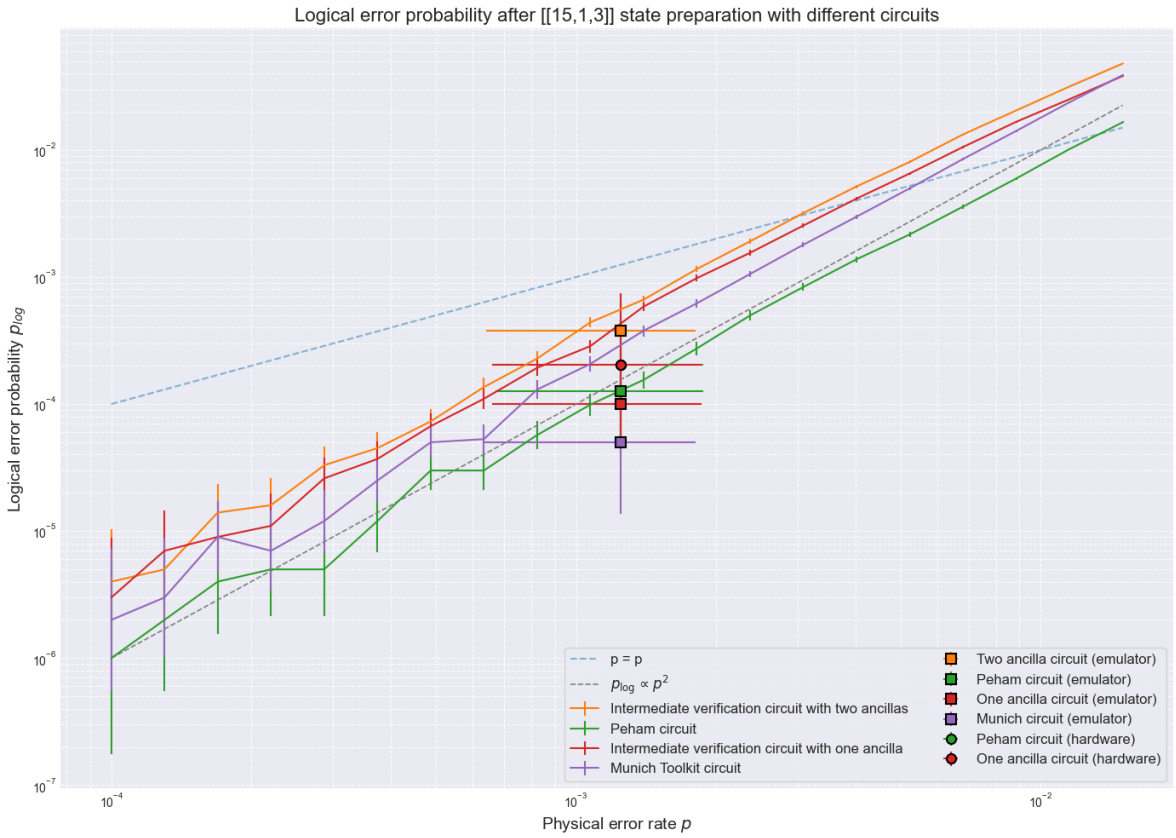


Figure 5.1: Logical error probability p_{log} after $|0\rangle_L$ state preparation in the $[[15, 1, 3]]$ code, plotted against the physical error rate p on a log-log scale. Each curve represents simulation data under circuit-level noise in Stim. Circular markers show hardware results from Quantinuum's H1-1 system; square markers indicate emulator values using typical operation-dependent error rates. These markers are plotted at $p \approx 1.25 \cdot 10^{-3}$, corresponding to the typical 2-qubit gate error reported in the hardware data sheet. The dashed black line shows the expected scaling $p_{\text{log}} \propto p^2$ based on Equation (3.23), while the dashed blue line denotes the threshold $p_{\text{log}} = p$. Final verification circuits achieve lower p_{log} than IV circuits, due to improved flagging of multi-fault events that potentially cause uncorrectable errors.

statistically significant. This behaviour can be attributed to the placement of the verification step: both intermediate and final verification circuits are designed to detect all problematic single faults, but not all problematic multi-fault events are guaranteed to be flagged. It is plausible that IV, by checking earlier in the circuit, is less likely to detect certain multi-fault configurations than final verification. As a result, a larger fraction of such events may be accepted, which leads to a higher logical error rate.

Emulator and hardware data points are shown at $p = 1.25 \cdot 10^{-3}$, corresponding to the typical two-qubit gate error in the Quantinuum H1-1 data sheet [1]. While state preparation and measurement errors are even higher in the data sheet (e.g. $2 \cdot 10^{-3}$), they do not contribute equally to propagation. State preparation errors can affect subsequent gates, but measurement errors occur last and cannot propagate. The choice $p = 1.25 \cdot 10^{-3}$ thus reflects a physically relevant estimate for logic-affecting noise.

The horizontal error bars are computed as a root-mean-square weighted average:

$$p_{\text{circuit,eff}} = \sqrt{\frac{N_{1q}\, p_{1q}^2 + N_{2q}\, p_{2q}^2 + N_{\text{idle}}\, p_{\text{idle}}^2 + (N_{\text{meas}} + N_{\text{prep}})\, p_{\text{spm}}^2}{N_{1q} + N_{2q} + N_{\text{idle}} + N_{\text{meas}} + N_{\text{prep}}}}$$

Here p_{1q}, p_{2q}, p_{idle}, and p_{spm} are the typical or maximal physical error rates. We chose the RMS form rather than a linear average because p_{log} scales quadratically in the physical error rate in the low-noise regime. Larger physical error probabilities contribute more strongly to logical failure, and the RMS average better reflects this behaviour.

Emulator results are based on $4 \cdot 10^4$ shots per circuit, hardware on 10^4. The hardware results are broadly consistent with the simulation data. The emulator data points, however, lie systematically

below both the simulation and hardware values. This difference may be due to the emulator's use of typical per-gate error values and detailed noise modelling. In particular, the emulator employs distinct fault probabilities per gate type, many of which are lower than the typical or maximal values reported in the Quantinuum H1-1 hardware data sheet. For example, the emulator sets the two-qubit gate error to $p_2 = 8.8{\cdot}10^{-4}$ and the single-qubit gate error to $p_1 = 2.1{\cdot}10^{-5}$, while the corresponding typical hardware values are $1{\cdot}10^{-3}$ and $3{\cdot}10^{-5}$, respectively, with even higher maximal values. This discrepancy in error modelling likely contributes to the lower p_{log} and higher p_{acc} values observed in emulator runs. The difference between hardware and emulator may also reflect the fact that the hardware was operating closer to its maximal error rates at the time of execution.

Our values for the Peham final verification circuit can also be compared to the simulation results in Figure 9 of [4]. Their curve also shows $p_{\text{log}} \sim p^2$, but the absolute values are lower: $p_{\text{log}} < 10^{-8}$ at $p = 10^{-4}$ and $p_{\text{log}} \approx 4 \cdot 10^{-3}$ at $p = 10^{-2}$. Our corresponding values are approximately 10^{-6} and $8 \cdot 10^{-3}$. In addition, the Peham curve reports $p_{\text{log}} \approx 5 \cdot 10^{-5}$ at $p \approx 1.25 \cdot 10^{-3}$, which is consistent with the error bars of our emulator data points, but not with those of our circuit-level simulations or hardware runs. The discrepancy likely results from their assumption $p_{\text{idle}} = p/100$, while we assume $p_{\text{idle}} = p$. The Quantinuum data sheet suggests that $p_{\text{idle}} \approx p_{2q}/6$ [1], so the effective idle error rate likely lies between p and $p/100$.

5.2. Acceptance probability

Figure 5.2 shows the acceptance probability p_{acc} for each circuit. A run is accepted if no flag is triggered; for 2-ancilla circuits, rejection occurs only when both flags are triggered.

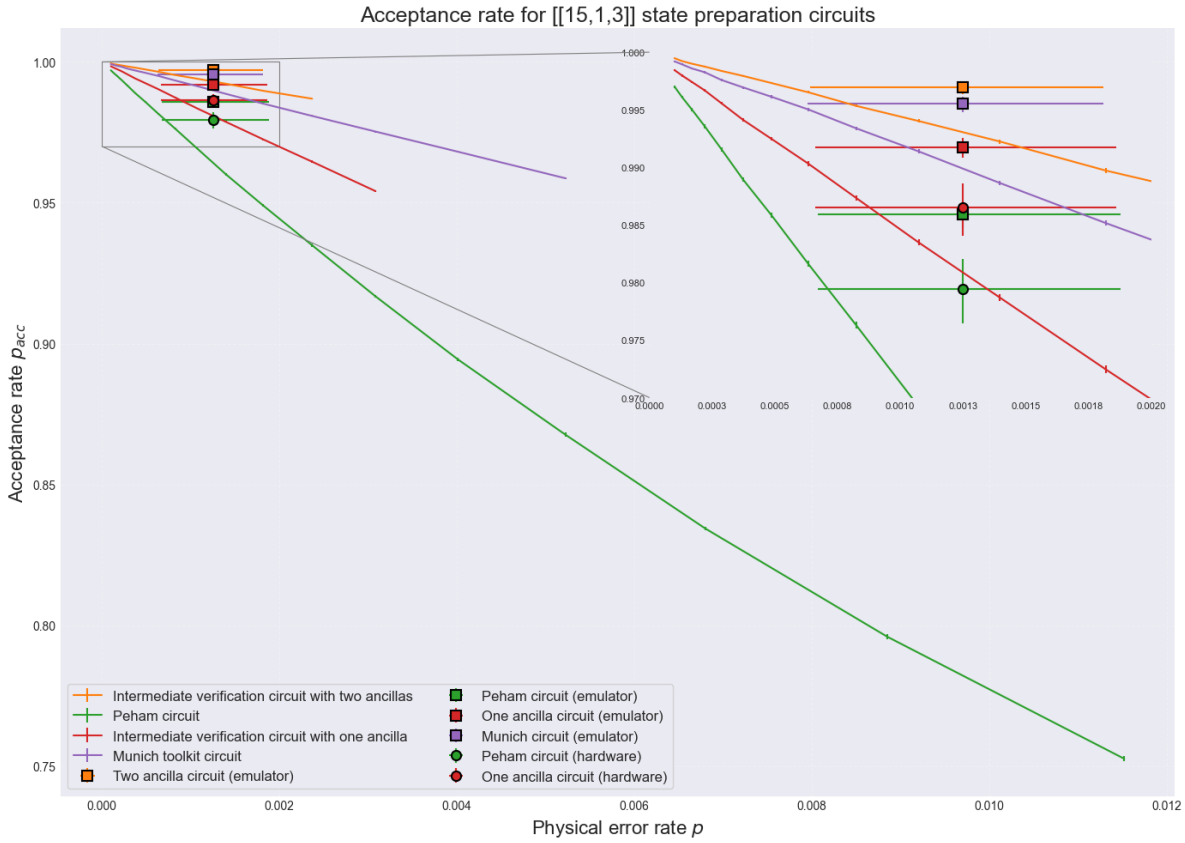


Figure 5.2: Acceptance probability p_{acc} as a function of the physical error rate p for different $[[15, 1, 3]]$ state preparation circuits. In circuits with one ancilla qubit, a run is accepted if the flag does not trigger. In circuits with two ancillas, a run is accepted unless both flags are triggered. IV circuits show higher p_{acc} than final verification circuits, as errors occurring after the verification point do not affect acceptance. Two-ancilla circuits further improve acceptance by reducing false rejections. Circular markers denote hardware results; square markers show emulator values using typical operation-dependent error rates. All markers are plotted at $p \approx 1.25 \cdot 10^{-3}$, reflecting the typical 2-qubit gate error reported in the Quantinuum H1-1 data sheet.

Circuits with 2 ancillas achieve consistently higher acceptance probabilities than their 1-ancilla counterparts. With two flags, a rejection only occurs when both flags flip, which reduces false rejections due to non-problematic faults or multi-fault events that do not trigger both flags. Within each ancilla configuration, circuits with IV have higher p_{acc} than those with final verification. In intermediate circuits, errors occurring after the verification step do not lead to rejection.
As in Section 5.1, emulator and hardware data points are shown at $p = 10^{-3}$. Hardware results fall within the error bars of the simulator. Emulator values lie systematically above both the simulated and hardware values. This difference may be due to the use of typical error values and gate-specific noise in the emulator. The difference between hardware and emulator could mean the hardware was operating at physical error rates closer to the maximal values in the data sheet at the time of execution.
These acceptance probabilities can also be compared to those reported in Figure 9 of [4]. In both cases, $p_{\text{acc}} \to 1$ as $p \to 0$. At $p = 10^{-2}$, [4] report $p_{\text{acc}} \approx 0.8$, whereas we find $p_{\text{acc}} \approx 0.78$. At the intermediate value $p \approx 1.25 \cdot 10^{-3}$, the resolution of Figure 9 in [4] makes it difficult to extract precise values for p_{acc}. The reported value appears to agree well with our data across simulation, emulator, and hardware. As before, differences at higher noise levels may be explained by their use of $p_{\text{idle}} = p/100$, while we use $p_{\text{idle}} = p$.
In addition, the emulator's acceptance probabilities are systematically higher than those from circuit-level simulations and hardware. This may again be attributed to the emulator's use of per-gate error rates, which are generally lower than the typical or maximal values reported in the hardware data sheet. For instance, the emulator sets $p_2 = 8.8 \cdot 10^{-4}$ and $p_1 = 2.1 \cdot 10^{-5}$, whereas the data sheet lists typical values of $1 \cdot 10^{-3}$ and $3 \cdot 10^{-5}$, respectively. Such differences in error assumptions likely reduce the total number of faults during circuit execution, increasing p_{acc}.

5.3. Runtime

Figure 5.3 shows the expected runtime, expressed in circuit depth, based solely on simulation data. For IV, we compute

$$\text{runtime} = \left(1 - \frac{1}{p_{\text{acc}}}\right) \cdot D_{\text{IV}} + D_{\text{total}},$$

where D_{IV} is the depth up to the verification point. For final verification circuits, we use runtime $= D_{\text{total}}/p_{\text{acc}}$.
At small p, where $p_{\text{acc}} \approx 1$, the Peham circuit has the lowest runtime due to its minimal depth. In this regime, repetition has little effect. At larger p, the 2-ancilla IV circuit becomes most efficient because of its higher acceptance rate and partial repetition strategy. The Munich and 1-ancilla IV circuits follow. Peham has the highest runtime due to its low p_{acc} and full-circuit repetition.
However, this performance advantage for IV circuits occurs in the regime where $p_{\text{log}} > p$, indicated by the dashed lines. Since quantum error correction is only meaningful when $p_{\text{log}} < p$, this regime is not practically relevant. In the regime where $p_{\text{log}} < p$, the Peham circuit remains the most runtime-efficient option.
These runtimes are based exclusively on abstract circuit depth, where abstract depth refers to the idealised circuit depth before native gate decomposition or hardware-specific scheduling. Although the emulator incorporates such hardware-specific details, the depths of those recompiled circuits were deliberately not used here. Including them would place all runtimes in a completely different range (e.g. depth 12–14 for optimized circuits versus depth 24–29 for unoptimized ones), and would obscure the comparison between different circuits.

5.4. Native gate depth and scheduling

Table 5.1 shows the compiled native depths for each circuit under the Quantinuum gate set, with a constraint of at most five parallel operations. We report both the total depth D and the depth D_{IV} up to the verification point, where applicable. Two versions are shown:

- **Unoptimized:** compiled with `optimization_level = 0` and compiler options `no-opt`, `no-reduce`, `no-qir-convert`, and `tket-opt-level = False`.
- **Optimized:** compiled with `optimization_level = 2` and default options.

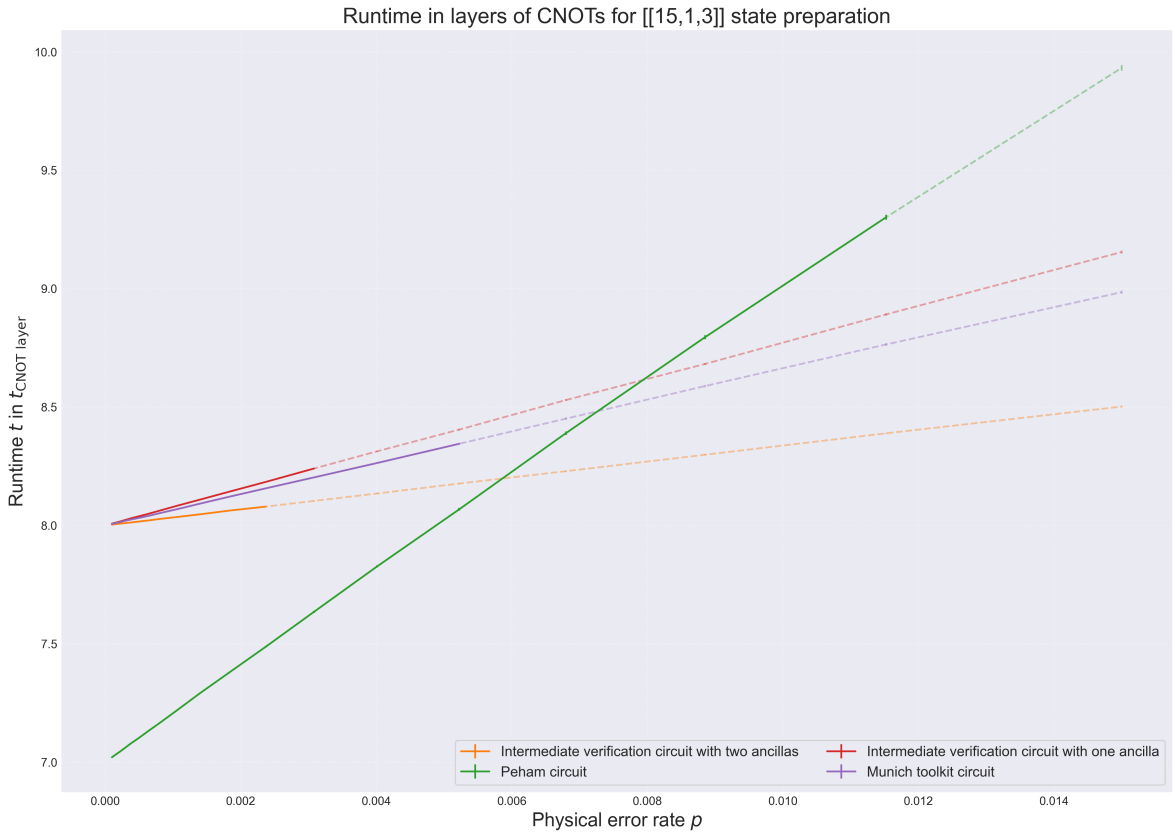


Figure 5.3: Expected runtime in layers of CNOT gates for different $[[15, 1, 3]]$ state preparation circuits, as a function of the physical error rate p. For IV circuits, runtime is computed as $t = \left(1 - \frac{1}{p_{\text{acc}}}\right) D_{\text{IV}} + D_{\text{total}}$, accounting for partial repetition up to the verification point. For final verification circuits, full repetition is required, with $t = D_{\text{total}}/p_{\text{acc}}$. The curves shown are based on simulation data under a circuit level noise model. No hardware specific constraints or native gate decomposition are included.

Circuit	D_{IV} (unopt.)	D (unopt.)	D_{IV} (opt.)	D (opt.)
1 ancilla IV	21	25	9	12
2 ancilla IV	24	28	11	14
Munich (2 anc)	–	29	–	14
Peham (1 anc)	–	24	–	13

Table 5.1: Native gate depth before and after optimization.

These depth values were not used in the runtime estimates of Section 5.3, which are based on depth prior to native gate compilation. Rather, this table illustrates how relative circuit depths can shift once hardware constraints are applied. For instance, the optimized 1-ancilla IV circuit becomes shorter than the optimized Peham circuit, reversing their order from the unoptimized case. These results highlight that runtime comparisons are incomplete without explicit consideration of compilation and hardware scheduling.
The hardware and emulator data points in Sections 5.1 and 5.2 are based on the unoptimized circuits, to allow a consistent comparison with simulation results that do not include gate-level optimizations or platform-specific constraints.

6

Discussion

6.1. Summary of Findings

IV was proposed to avoid full-circuit repeats in repeat-until-success schemes for logical state preparation, by detecting uncorrectable faults early in the circuit before additional qubits and gates are applied. We introduced a logical state preparation strategy based on this principle. While this approach does not necessarily reduce the expected runtime compared to final verification, it can lead to shorter compiled circuit depth after native gate translation and hardware scheduling. Notably, the 1-ancilla IV circuit becomes shorter than the Peham circuit after optimisation (Table 5.1), even though it is initially deeper. This suggests that IV can yield favourable depth characteristics and that compilation to the native gate set should precede circuit optimisation.

Hardware and emulator results at $p \approx 1.25 \cdot 10^{-3}$ confirm that $p_{\text{log}} < p$ for all tested circuits (Figure 5.1), demonstrating that error suppression is achievable even with code distance $d = 3$, and even with only 20 physical qubits (and in our case 16 or 17).

Although the emulator systematically underestimates p_{log} and overestimates the acceptance probability p_{acc} (Figure 5.2), the relative performance of the circuits remains consistent across simulation, emulation, and hardware.

Among all circuits, those using two ancilla qubits achieve the highest p_{acc}, regardless of verification point. Fixing the ancilla count, IV circuits yield slightly higher p_{acc} than final verification circuits. However, as seen in Figure 5.3, the acceptance probability is high enough that the effective depth is close to the full circuit depth. In this regime, IV becomes attractive primarily due to its potentially lower depth.

6.2. Interpretation and Implications

The results highlight several practical implications of using IV for logical state preparation. One of the main benefits is a potential reduction in total circuit depth after native gate compilation and scheduling. Since circuit depth directly impacts resource usage, it is relevant for systems with limited coherence times, hardware-imposed restrictions on parallelism, and other practical constraints. Reducing circuit depth thus contributes to general resource efficiency across a range of architectures.

The results in Table 5.1 show that IV can lead to shorter optimised depth, but not always. In particular, the 2-ancilla IV and Munich circuits have equal depth after optimisation. Moreover, Figure 5.1 shows that IV tends to result in slightly higher p_{log} than final verification. This reflects the fact that some multi-fault configurations occurring after the intermediate check may not be detected and can propagate to the output. These trade-offs imply that IV is not universally advantageous, and should be assessed in context.

For larger codes, the relative difference between partial and full repetition becomes more relevant: the total depth increases with m, while the intermediate verification point remains fixed after the Steane encoding (see Section 4.7.1). This makes full-circuit rejection increasingly costly, and may enhance the runtime benefit of IV.

Figure 5.3 presents expected runtimes based on acceptance probabilities and abstract circuit depth. However, this model does not account for hardware-specific factors such as gate duration, transport

delays, or limited parallelism. Consequently, the compiled and scheduled depth, as reported in Table 5.1, may be a more reliable metric when evaluating circuits for near-term hardware. Circuit designs should therefore be assessed after mapping to the target gate set, not before. This distinction is often missing in existing synthesis methods such as [4], although frameworks like [3] do allow for optimisation after recompiling in the native gate set.
This distinction is critical: as the 1-ancilla IV circuit becomes shallower than Peham only after native compilation, it shows that abstract depth is not predictive of final circuit depth.

6.3. Limitations

This study focuses exclusively on the preparation of the $|0\rangle_L$ state in the $[[15, 1, 3]]$ code. This state can be used for various components of encoded computation in the $[[15, 1, 3]]$ code, such as initialisation and Steane error correction (if Steane-type error correction is used) [9]. It is also used as an input in the 15-to-1 magic state distillation protocol [10]. The method presented here is specific to $|0\rangle_L$, and the preparation of other logical basis states, such as $|+\rangle_L$, was not analysed.
All simulations were performed under a symmetric depolarising noise model with $p_{\text{idle}} = p$. This model does not capture important features of realistic hardware, such as leakage, crosstalk, asymmetric gate noise, or dephasing due to transport. In addition, a single physical error rate p was used for all operations, whereas in practice error rates vary significantly between gate types. The Quantinuum H1-1 datasheet suggests that $p_{\text{idle}} \approx p_{2q}/6$, while [4] uses the more optimistic assumption $p_{\text{idle}} = p/100$.
Furthermore, the analysis is limited to distance-three codes. It remains to be seen whether IV provides similar benefits for larger codes, both in terms of circuit depth and logical error probability.

6.4. Comparison with prior work

IV circuits are non-deterministic: acceptance depends on the absence of triggered flags. In contrast, deterministic schemes such as [2] avoid rejection at the cost of increased circuit size or correction steps. However, as shown in Figure 5.3, p_{acc} is already high for all circuits, so the effective runtime is largely determined by circuit depth. In this regime, determinism offers little advantage.
Figures 5.1 and 5.2 enable direct comparison to [4], whose results are slightly more favourable. At $p = 10^{-2}$, they report $p_{\text{log}} \approx 4 \cdot 10^{-3}$ and $p_{\text{acc}} \approx 0.8$, compared to our values of $8 \cdot 10^{-3}$ and 0.78. This is likely due to their more optimistic assumption $p_{\text{idle}} = p/100$, while we use $p_{\text{idle}} = p$. The Quantinuum datasheet suggests an intermediate value, $p_{\text{idle}} \approx p_{2q}/6$.
Their simulations agree well with our emulator results, but lie below hardware values for p_{log}. Notably, at $p \approx 1.25 \cdot 10^{-3}$, our simulations match hardware results more closely than the emulator or [4], albeit for a single value of p.

6.5. Generalisations and Future Work

The formal analysis in Section 4.7.1 shows that IV can be applied fault-tolerantly to $\mathrm{PQRM}(1, m-2, m)$ codes, where verification occurs early in the recursive structure. These generalisations rely on the fact that the code distance d remains constant throughout the construction, while the Z-distance in the upper half of the circuit, denoted d_Z^+, increases with m. As discussed in Section 4.8, this implies that the number of X faults that remain correctable after verification and final encoding also increases. This observation motivates further exploration of recursive constructions in which one of the input codes already has the same distance as the output code, including constructions such as doubled CSS codes [31, 41]. In such cases, existing verification schemes for the constituent code, designed to detect up to $\alpha = \lfloor (d-1)/2 \rfloor$ faults, may be reused as intermediate checks. Their role then shifts from final to IV, while preserving their original detection properties.
Beyond analysis, future work should investigate whether the advantages of IV persist for codes with $d > 3$, and whether similar depth reductions occur after native gate compilation. As discussed in Section 4.8, there exist fault configurations that propagate to errors with weight well above the weight for guaranteed correction. When multiple faults are allowed, this makes manual fault tolerance proofs increasingly difficult. A numerical approach is therefore recommended.
Another promising direction is the integration of IV into circuit synthesis workflows. Current synthesis tools [3, 4] typically treat verification as a fixed final step. Allowing verification to occur earlier in the circuit could enlarge the space of admissible circuits, at the cost of increased optimisation complexity.

This may be especially useful for synthesising low-depth circuits for near-term hardware implementations.

6.6. User Experience with the Quantinuum H1-1

To complement simulation and emulation, two circuits were executed on Quantinuum's H1-1 trapped-ion processor: the 1-ancilla IV circuit and the Peham final verification circuit. Access was granted via the Nexus portal under a promotional agreement, which allowed for easy circuit submission, visualisation, and result download. Execution runtimes were not reported automatically and had to be requested separately.

Hardware usage was limited to 10,000 Hardware Quantum Credits (HQC) in total, with a cap of 2,000 HQC per job and 10,000 shots per circuit. A representative run of both circuits used 1,866 HQC. These limits did not constrain the experiments but required some planning. Queue times increased with successive submissions, reflecting the platform's dynamic priority model.

All circuits were compiled using `tket`, with barriers inserted to preserve intermediate measurements. Each was run with 10,000 shots. Results were consistent with emulator predictions using the same native gate set and scheduling constraints, though the emulator systematically underestimated p_{log} and overestimated p_{acc}. This is likely due to its use of gate error rates below those reported in the hardware datasheet.

These experiments emphasised the importance of compiling to the native gate set and accounting for limited parallelism, both of which significantly affect total circuit depth. While small-scale $\mathrm{PQRM}(1, m-2, m)$ circuits are feasible on current hardware, scaling is difficult: even on Quantinuum's newer 56-qubit H2 device [42], only $m = 4$ and $m = 5$ are possible. Preparing $m = 6$ would already require 63 qubits.

7

Conclusion

This thesis explored whether intermediate verification (IV) can reduce the effective runtime of repeat-until-success logical state preparation. Using the $[[15, 1, 3]]$ code as a test case, we developed and compared several fault-tolerant preparation circuits, with and without IV. While IV circuits do not consistently outperform final verification circuits in terms of runtime, they introduce an additional degree of freedom in circuit design.

We observed that IV circuits can yield shorter compiled depth after native gate translation and scheduling, even when their abstract depth is higher. In our experiments, the 1-ancilla IV circuit became shallower than the Peham circuit only after hardware-aware compilation. This highlights that circuit efficiency should be evaluated after native gate mapping rather than in terms of uncompiled abstract depth.

We also showed that IV circuits can be made fault-tolerant for $\mathrm{PQRM}(1, m-2, m)$ codes with $m \geq 4$ and distance $d = 3$, and successfully executed such a circuit on Quantinuum hardware. Hardware results confirmed that $p_{\text{log}} < p$ holds at physical error rates around $1.25 \cdot 10^{-3}$, indicating that encoding does offer an advantage for current hardware error rates.

While IV circuits exhibit slightly higher p_{log} than final verification circuits, they also achieve higher acceptance probabilities. In the relevant noise regime, the acceptance probability is sufficiently high that partial repetition offers little runtime advantage. This trade-off may become more relevant for larger codes, where acceptance decreases and the difference in depth between partial and full repetition grows.

In summary, IV is a viable strategy for logical state preparation. It can be included in synthesis or learning-based methods to broaden the design space of fault-tolerant circuits. These results further emphasize that runtime comparisons should be made only after native gate compilation, under hardware-specific constraints.

Bibliography

[1] *Quantinuum H1 Product Data Sheet Version 8.0.1*, June 2025.

[2] L. Schmid, T. Peham, L. Berent, M. Müller, and R. Wille, *Deterministic Fault-Tolerant State Preparation for Near-Term Quantum Error Correction: Automatic Synthesis Using Boolean Satisfiability*, _eprint: 2501.05527, 2025.

[3] R. Zen, J. Olle, L. Colmenarez, M. Puviani, M. Müller, and F. Marquardt, *Quantum Circuit Discovery for Fault-Tolerant Logical State Preparation with Reinforcement Learning*, arXiv:2402.17761 [quant-ph], May 2024.

[4] T. Peham, L. Schmid, L. Berent, M. Müller, and R. Wille, *Automated Synthesis of Fault-Tolerant State Preparation Circuits for Quantum Error Correction Codes*, arXiv:2408.11894 [quant-ph], Sept. 2024.

[5] A. Barg, N. J. Coble, D. Hangleiter, and C. Kang, *Geometric structure and transversal logic of quantum Reed-Muller codes*, arXiv:2410.07595 [quant-ph], Oct. 2024.

[6] E. Arkan, "A performance comparison of polar codes and Reed-Muller codes", en, IEEE Communications Letters **12**, 447–449 (2008).

[7] A. Gong and J. M. Renes, *Computation with quantum Reed-Muller codes and their mapping onto 2D atom arrays*, _eprint: 2410.23263, 2024.

[8] A. M. Kubica, "A study of topological quantum codes as toy models for fault-tolerant quantum computation and quantum phases of matter", en,

[9] A. Kubica and M. E. Beverland, "Universal transversal gates with color codes - a simplified approach", Physical Review A **91**, arXiv:1410.0069 [quant-ph], 032330 (2015).

[10] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation", Physical Review A **86**, arXiv:1208.0928 [quant-ph], 032324 (2012).

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

[12] C. Gerhard and T. A. Brun, *Weakly Fault-Tolerant Computation in a Quantum Error-Detecting Code*, arXiv:2408.14828 [quant-ph], Aug. 2024.

[13] J. Preskill, *Physics 219 Course Lecture Notes Chapter 7. Quantum Error Correction*, 1999.

[14] A. M. Steane, *Fast fault-tolerant filtering of quantum codewords*, arXiv:quant-ph/0202036, Apr. 2004.

[15] D. Gottesman, "A Theory of Fault-Tolerant Quantum Computation", Physical Review A **57**, arXiv:quant-ph/9702029, 127–137 (1998).

[16] J. Roffe, "Quantum Error Correction: An Introductory Guide", Contemporary Physics **60**, arXiv:1907.11157 [quant-ph], 226–245 (2019).

[17] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory", Journal of Mathematical Physics **43**, arXiv:quant-ph/0110143, 4452–4505 (2002).

[18] B. Terhal, *Quantum Error Correction*, Transcript of presentation, Dec. 2022.

[19] D. Gottesman, *Stabilizer Codes and Quantum Error Correction*, arXiv:quant-ph/9705052, May 1997.

[20] A. Rynbach, M. Ahsan, A. Mehta, J. Hussmann, and J. Kim, "A Quantum Performance Simulator based on fidelity and fault-path counting", (2012).

[21] J. Wright, *Lecture 8: CSS Codes*, Published: Lecture notes, UC Berkeley CS294: Quantum Coding Theory, Feb. 2024.

[22] F. MacWilliams and N. Sloane, *The Theory of Error-correcting Codes*, Mathematical Library (North-Holland Publishing Company, 1977).

[23] P. Aliferis, D. Gottesman, and J. Preskill, *Quantum accuracy threshold for concatenated distance-3 codes*, arXiv:quant-ph/0504218, Oct. 2005.

[24] S. Herbert and P. Murali, *Lecture 14: Fault Tolerant Quantum Computing*, Publication Title: Quantum Computing (CST Part II), 2025.

[25] J. Fujisaki, K. Maruyama, H. Oshima, S. Sato, T. Sakashita, Y. Takeuchi, and K. Fujii, "Quantum error correction with an Ising machine under circuit-level noise", en, Physical Review Research **5**, 043261 (2023).

[26] B. Eastin and E. Knill, "Restrictions on Transversal Encoded Quantum Gate Sets", Physical Review Letters **102**, arXiv:0811.4262 [quant-ph], 110502 (2009).

[27] E. T. Campbell, *The smallest interesting colour code*, Sept. 2016.

[28] M. B. Hastings and J. Haah, "Distillation with Sublogarithmic Overhead", Physical Review Letters **120**, Publisher: American Physical Society, 050504 (2018).

[29] S. Heußen and J. Hilder, *Efficient fault-tolerant code switching via one-way transversal CNOT gates*, arXiv:2409.13465, Sept. 2024.

[30] F. Butt, "Fault-Tolerant Code-Switching Protocols for Near-Term Quantum Processors", PRX Quantum **5**, 10.1103/PRXQuantum.5.020345 (2024).

[31] M. Sullivan, "Code conversion with the quantum Golay code for a universal transversal gate set", en, Physical Review A **109**, 042416 (2024).

[32] C. Vuillot, L. Lao, B. Criger, C. Almudéver, K. Bertels, and B. Terhal, "Code Deformation and Lattice Surgery Are Gauge Fixing", New Journal of Physics **21**, 10.1088/1367-2630/ab0199 (2019).

[33] C. Chamberland and M. E. Beverland, "Flag fault-tolerant error correction with arbitrary distance codes", Quantum **2**, arXiv:1708.02246 [quant-ph], 53 (2018).

[34] D. Gottesman, "Surviving as a Quantum Computer in a Classical World", en,

[35] J. A. Muniz, D. Crow, H. Kim, J. M. Kindem, W. B. Cairncross, A. Ryou, T. C. Bohdanowicz, C.-A. Chen, Y. Ji, A. M. W. Jones, E. Megidish, C. Nishiguchi, M. Urbanek, L. Wadleigh, T. Wilkason, D. Aasen, K. Barnes, J. M. Bello-Rivas, I. Bloomfield, G. Booth, A. Brown, M. O. Brown, K. Cassella, G. Cowan, J. Epstein, M. Feldkamp, C. Griger, Y. Hassan, A. Heinz, E. Halperin, T. Hofler, F. Hummel, M. Jaffe, E. Kapit, K. Kotru, J. Lauigan, J. Marjanovic, M. Meredith, M. McDonald, R. Morshead, S. Narayanaswami, K. A. Pawlak, K. L. Pudenz, D. R. Pérez, P. Sabharwal, J. Simon, A. Smull, M. Sorensen, D. T. Stack, M. Stone, L. Taneja, R. J. M. v. d. Veerdonk, Z. Vendeiro, R. T. Weverka, K. White, T.-Y. Wu, X. Xie, E. Zalys-Geller, X. Zhang, J. King, B. J. Bloom, and M. A. Norcia, *Repeated ancilla reuse for logical computation on a neutral atom quantum computer*, arXiv:2506.09936 [quant-ph], June 2025.

[36] H. Goto, "Minimizing resource overheads for fault-tolerant preparation of encoded states of the Steane code", en, Scientific Reports **6**, Publisher: Nature Publishing Group, 19578 (2016).

[37] S. Heußen, D. F. Locher, and M. Müller, *Measurement-free fault-tolerant quantum error correction in near-term devices*, arXiv:2307.13296 [quant-ph], July 2023.

[38] Byron Bay Quantum Computing Workshop, *Dave Hayes – Good and evil in the garden of full-connectivity*, July 2023.

[39] L. Paletta, A. Leverrier, A. Sarlette, M. Mirrahimi, and C. Vuillot, "Robust sparse IQP sampling in constant depth", Quantum **8**, 1337 (2024).

[40] Z. Bao, S. Xu, Z. Song, K. Wang, L. Xiang, Z. Zhu, J. Chen, F. Jin, X. Zhu, Y. Gao, Y. Wu, C. Zhang, N. Wang, Y. Zou, Z. Tan, A. Zhang, Z. Cui, F. Shen, J. Zhong, T. Li, J. Deng, X. Zhang, H. Dong, P. Zhang, Y.-R. Liu, L. Zhao, J. Hao, H. Li, Z. Wang, C. Song, Q. Guo, B. Huang, and H. Wang, "Creating and controlling global Greenberger-Horne-Zeilinger entanglement on quantum processors", en, Nature Communications **15**, Publisher: Nature Publishing Group, 8823 (2024).

[41] S. Bravyi and A. Cross, *Doubled Color Codes*, arXiv:1509.03239, Sept. 2015.

[42] L. Daguerre, R. Blume-Kohout, N. C. Brown, D. Hayes, and I. H. Kim, *Experimental demonstration of high-fidelity logical magic states from code switching*, arXiv:2506.14169 [quant-ph], June 2025.