A simple method for compiling quantum stabilizer circuits

Brendan Reid
*Entropica Labs, Singapore**

Stabilizer circuits play an important role in quantum error correction protocols, and will be vital for ensuring fault tolerance in future quantum hardware. While stabilizer circuits are defined on the Clifford generating set, $\{H,\ S,\ CX\}$, not all of these gates are native to quantum hardware. As such they must be compiled into the native gateset, with the key difference across hardware archetypes being the native two-qubit gate.

Here we introduce an intuitive and accessible method for Clifford gate compilation. While multiple open source solutions exist for quantum circuit compilation, these operate on arbitrary quantum gates. By restricting ourselves to Clifford gates, the compilation process becomes almost trivial and even large circuits can be compiled manually.

The core idea is well known: if two Clifford circuits conjugate Paulis identically, they are equivalent. Compilation is then reduced to ensuring that the instantaneous Pauli conjugation is correct for each qubit at every timestep. This is Tableaux Manipulation, so called as we directly interrogate stabilizer tableaux to ensure correct Pauli conjugation. We provide a brief explanation of the process along with a worked example to build intuition; we finally show some comparisons for compiling large circuits to open source software, and highlight that this method ensures a minimal number of quantum gates are employed.

I. INTRODUCTION

In the future running a quantum algorithm on a fault tolerant machine will not be unlike running a circuit through a quantum API today. Quantum software will ensure that the complexities of fault tolerance — codes, decoders and logical primitives — are abstracted away from the end user. To achieve this there are a number of stages that must be automated.

At the highest level, converting an algorithm of N logical qubits into a network of $M \gg N$ physical qubits will require insight into the algorithm being executed. Logical qubit placement will depend on (logical) two-qubit interactions, and minimising the (logical) ancillae overhead [1]. Once this task is complete the physical circuit must be constructed. Qubit preparation and logical gates, as well as the fundamental operations of the target error correction scheme (e.g. in the case of the surface code, lattice surgery operations such as growth, merge, split etc. [2]) all require a number of rounds of syndrome extraction to ensure errors are caught and accounted for. The result is an extremely deep circuit across an array of physical qubits, with the bulk of operations likely defined in terms of the Clifford generating set $\{H,\ S,\ CX\}$. The final step will be conversion from this gateset into the hardware instruction set. At this final stage the scope for minimising error events is limited, as the noisest processes in the experiment (two-qubit gates and measurements) have already been determined by the previous stages of the pipeline. Nevertheless, we will show that even modest circuit optimisations can have a positive impact on experiment performance.

Quantum hardware has advanced significantly in recent years, and continues to do so. The race to quantum advantage has resulted in a number of public and private enterprises employing varying quantum hardware modalities. The platforms that have achieved the most experimentally are superconducting qubits [3], trapped ion systems [4] and more recently neutral atoms [5]. Even within a qubit modality, differences between experimental groups abound. Beyond the disparities of noise models and connectivity, a key difference is in the entangling operations.

Trapped ion models employ a Mølmer-Sørensen interaction, which manifests as a $XX(\theta) = e^{-\frac{i\theta}{2}X\otimes X}$ rotation, where setting $\theta = \pi/2$ makes this interaction maximally entangling [6]. In this work, when speaking of a Mølmer-Sørensen gate, we mean $XX(\theta = \pi/2)$ which we denote $\sqrt{XX}$. Superconducting systems developed by IBM use the echoed cross-resonance interaction (ECR) [7]. This gate implements $\frac{1}{\sqrt{2}}(I\otimes X - X\otimes Y)$ onto the qubits, and is equivalent to a CX gate up to single qubit rotations. Conversely, Google's superconducting qubit processors have either an $iSWAP = e^{\frac{i\pi}{4}(X\otimes X + Y\otimes Y)}$ or CZ gate as the native two-qubit gate [8].

This brief work will present a method for rewriting quantum stabilizer circuits into alternative sets of Clifford gates, with a focus on circuits with practical applications. We are referring to this process as "compilation", although it could equivalently be called gate decomposition. We are not setting ourselves the task of amending a circuit to fit a specific qubit connectivity, i.e. we will not be adding or removing qubits. These restrictions create a narrow problem statement, but one that is key to using quantum computers effectively.

The process is straightforward and surprisingly powerful when compared to existing available open-source compilers. Importantly this method can be implemented with little effort; it is simple to compile a circuit consisting of 10-20 qubits manually, if a little tedious. There is minimal information to keep track of, as this method only relies upon the instantaneous Pauli conjugation of an individual qubit at each layer of the circuit. We employ an example driven approach to build intuition for the method. We also draw comparison to the Qiskit open-source compilation functionality and show that, for a key class of circuits, we

* brendan@entropicalabs.com

can always reduce the number of quantum operations executed.

Much work has been done within the Clifford set, from generating random circuits [9, 10] to optimizing the number of two-qubit unitaries required within a circuit [11, 12]. The techniques used in this work are not new; ultimately we are simply tracking and amending the instantaneous Pauli conjugation of individual qubits. The author could not find any work in the literature with significant overlap to the the results contained within, but welcomes indication toward existing work.

II. PRELIMINARIES

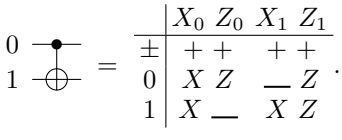
We assume familiarity with quantum computing and quickly summarise the key concepts relevant to this work. The Pauli group consists of four operators $\{I,\ X,\ Y,\ Z\}$ and is defined as the lowest level of the Clifford hierarchy [13]. A Pauli product is a collection of Pauli operators applied to individual qubits: $X_iY_jZ_k$ applies Pauli-X to qubit i, Y to qubit j and Z to qubit k. Any qubit index not involved in the Pauli product has the identity applied.

The next level of the hierarchy, the Clifford group, constitutes a key component of fault tolerant universal quantum computation [14]. The Clifford group contains operators that conjugate Paulis into Paulis, and its generating set of operators consists of $\{H,\ S,\ CX\}$. For example, the conjugation of Paulis by the Hadamard gate H is:

$$H^\dagger X H = Z,\quad H^\dagger Y H = -Y,\quad H^\dagger Z H = X.$$

A stabilizer circuit consists solely of elements from the Clifford group and destructive operations in the Pauli-Z basis. A *unitary* stabilizer circuit does not contain measurements or reset operations. This unitary circuit can be defined by how it conjugates Pauli products, and this information can be summarised via *tableaux*.

Consider the following tableau, representing a CX gate where qubit 0 is the control and qubit 1 is the target:



(1)

The columns X_j, Z_j detail the conjugation of these Paulis by the circuit. For example, X_0 is conjugated by a CX gate into $+X_0X_1$; Z_1 is conjugated into $+Z_0Z_1$. An underscore indicates that the input Pauli does not propagate onto that output wire: a Z (X) term commutes through a CX gate if it is input on the control (target) qubit. Tableaux such as this (or in the alternative Aaronson-Gottesman format) completely define a Clifford interaction, as input Pauli products can be conjugated by first decomposing to their generators and using the distributive property:

$$\begin{aligned} CX^\dagger(Y_0Z_1)\,CX &= CX^\dagger(X_0Z_0Z_1)\,CX \\ &= (X_0X_1)\,(Z_0)\,(Z_0Z_1) \\ &= X_0Y_1. \end{aligned} \tag{2}$$

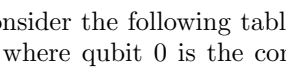
Where we ignore global phase. Importantly unitary stabilizer circuits are also *uniquely* defined via their tableaux. In Ref. [15], Lemma 1 states:

> Let $\mathcal{C}_1$, $\mathcal{C}_2$ be unitary stabilizer circuits, and let $\mathcal{T}_1$, $\mathcal{T}_2$ be their respective final tableaux when we run them on the standard initial tableau. Then $\mathcal{C}_1 \equiv \mathcal{C}_2$ iff $\mathcal{T}_1 \equiv \mathcal{T}_2$.

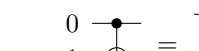
Where here the "standard initial tableau" defines all qubits prepared in the Pauli-Z basis, which we assume implicitly throughout. This Lemma tells us that as long as our tableaux are equivalent, the unitary action will be the same. By observing the differences in two tableaux, we are able to directly modify the circuit in a minimal way to achieve the correct Pauli product conjugation.

In addition to the conjugation of Pauli terms, we will also speak of the *propagation* of Paulis. Propagation simply refers to the qubit indices present in the Pauli conjugation, i.e. $X_0Z_1Y_2$ has the same propagation as $Y_0X_1Z_2$. This is a useful abstraction when thinking about the action of Paulis in circuits, specifically their commutation relations with two-qubit gates. If qubits j and k only interact once and k does not appear in the conjugation of P_j, then the instantaneous conjugation[1] of P_j must commute through the two-qubit gate acting on j and k.

We will employ a number of single qubit Clifford gates to aid intuition when converting between Pauli bases. The nomenclature for these gates is borrowed directly from the `stim` package [16]. We list them here along with their decomposition into S, $\sqrt{X}$ and Pauli gates:

- H: standard Hadamard gate, swaps X and Z bases.
  - 

- H_{XY}: a Hadamard-like gate that swaps X and Y bases.
  - 

- H_{YZ}: a Hadamard-like gate that swaps Y and Z bases.
  - 

- C_{XYZ}: a basis-cycle gate, moves $X \to Y \to Z$.
  -

[1] 'Instantaneous conjugation' here referring to the conjugation of a Pauli at a certain point in the circuit.

- C_{ZYX}: a reverse basis-cycle gate, moves $Z \to Y \to X$.

$$\circ\text{—}\boxed{C_{ZYX}}\text{—} \equiv \text{—}\boxed{S}\text{—}\boxed{\sqrt{X}}\text{—}\boxed{Z}$$

The role of Pauli gates in these circuits is to ensure that the sign changes are consistent on both sides of each equivalency. For example, H_{XY} conjugates Pauli-Z like $H_{XY}^\dagger Z H_{XY} = -Z$, whereas an S gate conjugates: $S^\dagger Z S = +Z$. As such, H_{XY} and S are equivalent up to the application of a Pauli gate. In practice this equivalency can be tracked in software and we refer to it as the Pauli frame.

The CX and $iSWAP$ gates are typically used to classify the different kinds of entangling Cliffords. Gates such as CZ, $\sqrt{XX}$ and ECR are classed as CX-like, as they are within single qubit rotations to CX. Gates such as $CXSWAP$ and $CZSWAP$ (again borrowing terminology from the `stim` package) are $iSWAP$-like, as they are within single qubit rotations to $iSWAP$. Throughout we will focus on compiling CX-like gates into CX-like gates, although the process outlined below is equally valid for compiling $iSWAP$-like into $iSWAP$-like. In Appendix A we provide discussion around exchanging entangling Clifford type: compiling a CX-like gate into an $iSWAP$-like gate and vice versa. We omit entirely discussion around the two-qubit, non-entangling Clifford gate $SWAP$.

III. TABLEAUX MANIPULATION

A common approach to circuit compilation is to identify repeated components of the circuit and compile these individually, before replacing them back into the circuit in the relevant locations. From here, gate optimisations and cancellations can take place to reduce the number of operations and the circuit depth. When our circuits are composed solely of Clifford operations, we can instead tackle the compilation of the entire circuit at once.

Consider a circuit $\mathcal{C}$ that we wish to compile into a gateset G. This circuit will provide us with a desired (or 'target') Pauli conjugation on each qubit: $\bar{P}_j = \mathcal{C}^{-1} P_j \mathcal{C}$ where $P \in \{X, Z\}$. Rather than working on $\mathcal{C}$ directly, we can instead create a new circuit $\mathcal{C}'$ whose only entries are gates in G. This initial condition provides us with conjugations $P'_j = (\mathcal{C}')^{-1} P_j \mathcal{C}'$. From here the task becomes: what gates do I need to add to $\mathcal{C}'$ to make $P'_j \equiv \bar{P}_j$ for all j?

Motivated by the disparities of native entangling Cliffords in different quantum hardware, we employ the following initial condition for our compiled circuit $\mathcal{C}'$: for each entangling operation in $\mathcal{C}$, introduce an entangling operation native to G into $\mathcal{C}'$. A circuit consisting solely of entangling operations is the shortest possible circuit after compilation. From this starting point, we can work through qubits individually and ensure that their Pauli conjugation is correct.

Tableaux Manipulation has two key steps: first fix the *propagation* of Paulis in the circuit, and then fix the *conjugation* of Paulis after the circuit. To fix the propagation, the process is as follows:

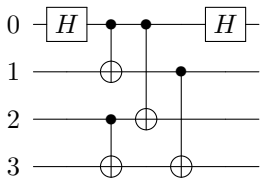
1. For each qubit index j, isolate its subcircuit (the timeline of gates it is involved in)
2. For each entangling operation between qubits j and k, determine if k is in the propagation of the target $\bar{P}_j$:
   - 2.1. If it is, add a gate to the circuit (prior to the entangling operation) such that the instantaneous Pauli conjugation of P_j *anti-commutes* with the two-qubit gate.
   - 2.2. If it is not, add a gate to the circuit (prior to the entangling operation) such that the instantaneous Pauli conjugation of P_j *commutes* with the two-qubit gate.

With the caveat that, if the instantaneous conjugation already anticommutes with the two-qubit interaction (or commutes, as the case may be) then no gate need be added.

Once the propagation of all P' matches that of $\bar{P}$, we must fix the conjugation. For each qubit j, we compare the output on qubit j for both $\bar{P}_j$ and P'^2_j. Correcting the conjugation is then choosing the appropriate basis-change Pauli from a lookup table, Table I.

A. Worked Example

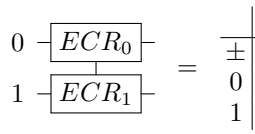
Here we work through an example of using the Tableau Manipulation method to compile a 4-qubit system, ostensibly measuring a XX and ZZ stabiliser with two auxiliary qubits:



(3)

	X_0	Z_0	X_1	Z_1	X_2	Z_2	X_3	Z_3
$\pm$	+	+	+	+	+	+	+	+
0	X	Z	—	X	—	X	—	X
1	—	X	X	Z	—	—	—	Z
2	—	X	—	—	X	Z	—	Z
3	—	X	X	—	X	—	X	Z

For this exercise we'll use an ECR gate, common to some superconducting systems. We represent it with the following circuit diagram and display it's tableau:



	X_0	Z_0	X_1	Z_1
$\pm$	−	−	+	+
0	Y	Z	—	Z
1	X	—	X	Y

(4)

As the ECR gate is not symmetric we must keep track of which qubit is the control and target during each

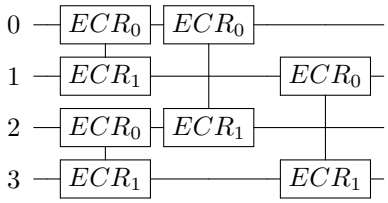
[2] This is equivalent to inspecting the diagonal entries of the tableaux we are comparing.

P' \ $\bar{P}$		$\bar{X}=X$ $\bar{Z}=Y$	$\bar{X}=X$ $\bar{Z}=Z$	$\bar{X}=Y$ $\bar{Z}=X$	$\bar{X}=Y$ $\bar{Z}=Z$	$\bar{X}=Z$ $\bar{Z}=X$	$\bar{X}=Z$ $\bar{Z}=Y$
$X'=X$	$Z'=Y$	I	H_{YZ}	H_{XY}	C_{XYZ}	C_{ZYX}	H
	$Z'=Z$	H_{YZ}	I	C_{XYZ}	H_{XY}	H	C_{ZYX}
$X'=Y$	$Z'=X$	H_{XY}	C_{ZYX}	I	H	H_{YZ}	C_{XYZ}
	$Z'=Z$	C_{ZYX}	H_{XY}	H	I	C_{XYZ}	H_{YZ}
$X'=Z$	$Z'=X$	C_{XYZ}	H	H_{YZ}	C_{ZYX}	I	H_{XY}
	$Z'=Y$	H	C_{XYZ}	C_{ZYX}	H_{YZ}	H_{XY}	I

TABLE I: Lookup table for changing Pauli basis depending on the desired conjugation $\bar{P}$ and the current conjugation P'. For example if our desired conjugation is $(\bar{X},\ \bar{Z}) = (Y,\ X)$ and our current conjugation is $(X',\ Z') = (Z,\ Y)$, we apply a C_{ZYX} gate to that qubit.

interaction. Similar to a CX gate, Pauli-Z commutes through on the 'control' wire (0) and Pauli-X commutes through on the 'target' wire (1).

We employ our initial condition from Tableaux Manipulation, and create a new circuit whose only entries are ECR:



(5)

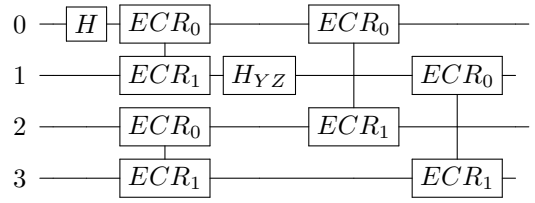
	X_0	Z_0	X_1	Z_1	X_2	Z_2	X_3	Z_3
$\pm$	$-$	$+$	$-$	$+$	$+$	$-$	$+$	$-$
0	X	Z	$\_\_$	Z	Z	Z	$\_\_$	Z
1	Y	$\_\_$	Y	X	$\_\_$	$\_\_$	$\_\_$	Z
2	X	$\_\_$	$\_\_$	$\_\_$	Z	Y	$\_\_$	Y
3	X	$\_\_$	X	X	X	$\_\_$	X	Z

First we must fix the propagation for each X_j, Z_j, starting with $j = 0$. Note that our desired conjugation $\bar{Z}_0 = Z_0X_1X_2X_3$ has the same propagation as our current $X_0' = X_0Y_1X_2X_3$, and vice versa. In this case we can simply prepend a Hadamard gate to qubit 0 to swap the action of X and Z Paulis.

Skipping ahead to qubit 1, we see that the propagation of $\bar{X}_1$ and X_1' are equal, and the only erroneous term is X_3 in the conjugation Z_1'. Qubit 1 is initially acting as a target qubit in an ECR gate with qubit 0, then as a control qubit in an ECR gate with qubit 3. We need the instantaneous Pauli conjugation of Z_1 to commute through this last ECR gate. Below, we track the conjugation where the Paulis on the right hand side represent the conjugation after the operation on the left.

$$\begin{aligned} \text{Initial} &: Z_1 \\ ECR(0,1) &: Z_0Y_1 \\ ECR(1,3) &: Z_0X_1X_3 \end{aligned}$$

As the instantaneous conjugation of Z_1 prior to the $ECR(1,3)$ gate is a Pauli-Y, we have to convert this to a Pauli-Z in order to commute through. Therefore, we add a H_{YZ} after $ECR(0,1)$. The circuit now looks like:



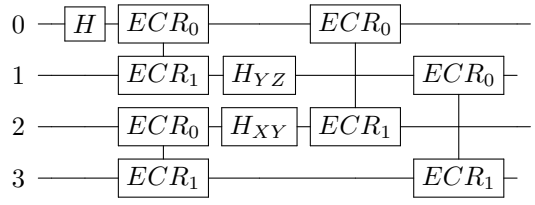
(6)

	X_0	Z_0	X_1	Z_1	X_2	Z_2	X_3	Z_3
$\pm$	$+$	$+$	$+$	$+$	$+$	$-$	$+$	$-$
0	Z	X	$\_\_$	Z	Z	Z	$\_\_$	Z
1	$\_\_$	Y	Y	Z	$\_\_$	$\_\_$	$\_\_$	Z
2	$\_\_$	X	$\_\_$	$\_\_$	Z	Y	$\_\_$	Y
3	$\_\_$	X	X	$\_\_$	X	$\_\_$	X	Z

For qubit 2, the propagations of $\bar{Z}_2$ and Z_2' are equal, and the only erroneous term is the Z_0 term in X_2'. Again, let's look at the instantaneous Pauli conjugation of qubit 2:

$$\begin{aligned} \text{Initial} &: X_2 \\ ECR(2,3) &: Y_2X_3 \\ ECR(0,2) &: Z_0Z_2X_3 \end{aligned}$$

To avoid picking up the Z_0 term in the $ECR(0,2)$ gate, we need Y_2 to commute through. As qubit 2 is a target in this gate we must convert Y_2 into X_2:

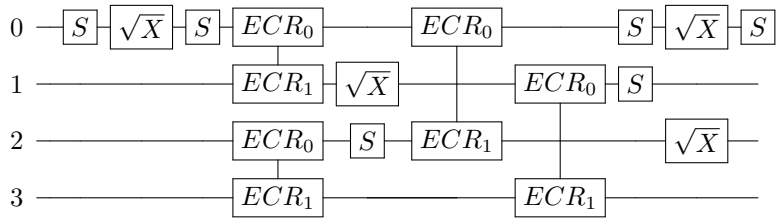


(7)

	X_0	Z_0	X_1	Z_1	X_2	Z_2	X_3	Z_3
$\pm$	$+$	$+$	$+$	$+$	$-$	$+$	$+$	$+$
0	Z	X	$\_\_$	Z	$\_\_$	Z	$\_\_$	Z
1	$\_\_$	Y	Y	Z	$\_\_$	$\_\_$	$\_\_$	Z
2	$\_\_$	X	$\_\_$	$\_\_$	X	Y	$\_\_$	Y
3	$\_\_$	X	X	$\_\_$	X	$\_\_$	X	Z

Now that the propagation is correct across all inputs, we have to modify the conjugation after the circuit.

Where fixing the propagation involved inspecting the columns of the tableau, fixing the conjugations requires inspecting the rows of the tableau. For each $(X'_j,\ Z'_j)$ we isolate the output on qubit j. For example, $(X'_0,\ Z'_0) = (Z_0,\ X_0)$ in Eq. (7). Looking at Eq. (3) we see this should be $(X_0,\ Z_0)$ and so we append a Hadamard onto qubit 0. Repeating this process for qubits 1, 2 and 3 we see that we need to apply a H_{XY} to qubit 1, H_{YZ} to qubit 2 and qubit 3 remains untouched. Skipping ahead to the final circuit in terms of S and $\sqrt{X}$:



$$=$$

	X_0	Z_0	X_1	Z_1	X_2	Z_2	X_3	Z_3
$\pm$	+	+	+	+	+	−	+	−
0	X	Z	_	X	_	X	_	X
1	_	X	X	Z	_	_	_	Z
2	_	X	_	_	X	Z	_	Z
3	_	X	X	_	X	_	X	Z

. (8)

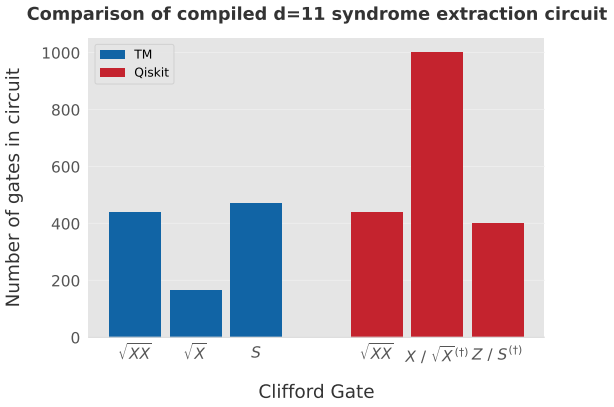


FIG. 1: Gate distribution when compiling a syndrome extraction circuit on a rotated surface code of distance $d = 11$. The original circuit is 6 layers of gates, defined in terms of H and CX. Tableaux Manipulation (left, dark blue) and Qiskit (right, dark red) both compiled into a gateset with $\sqrt{XX}$ as the native two-qubit gate.

This circuit requires a frame correction of X_2 to be exactly equivalent to Eq. (3).

B. Comparison to open-source software

Here we provide some comparisons to the Qiskit compiler functionality [17]. We will be comparing a single round of syndrome extraction on a distance d rotated surface code. For all code distances this circuit has depth-6 where four layers of CX gates are sandwiched by a layer of Hadamards on either side. The circuits are generated with `stim` [16] using the `stim.Circuit.generated` functionality, with kwargs `code_task=surface_code:rotated_memory_z, distance=`d`, rounds=1`. The circuit is identical regardless if the memory basis is chosen as X or Z.

We compare compilation procedures using both the $\sqrt{XX}$ gate and the ECR gate. When compiling with the Tableaux Manipulation method, compilation was performed on `stim.Circuit` objects using S and $\sqrt{X}$ as single qubit gates. We do not include frame-fixing Paulis when compiling with TM. When compiling with Qiskit, we convert the circuit from `stim.Circuit` into `qiskit.QuantumCircuit`, and call the `qiskit.compiler.transpile` function. The basis gates provided are `basis_gates=['rxx', 'rx', 'rz']` for $\sqrt{XX}$ compilation and `basis_gates=['ecr', 'rx', 'rz']` for ECR compilation.

We permit arbitrary angle rotations in Qiskit as we found this provided the most compact compilation. Manual checks were performed to ensure all angles were within tolerance to $\pm\pi$ or $\pm\pi/2$, making the gate Clifford. Where possible the `optimization_level` kwarg was set to 3, however when compiling with $\sqrt{XX}$ on distances $d = 5$ and above this had to be dropped to `optimization_level=1` as non-Clifford gates were being introduced. By explicitly setting the `basis_gates` kwarg to be a list of Clifford operations, i.e. `['z', 's', 'sdg', 'x', 'sx', 'sxdg', 'ecr' / 'rxx']`, the compiled circuit had significantly more gates (in the case of `ecr`) or raised an error (`rxx`).

1. $\sqrt{XX}$

To provide insight on the gate distributions of differently compiled circuits, we start by displaying a bar chart of the gates present for a $d = 11$ rotated surface code syndrome extraction circuit, Fig. 1. The uncompiled circuit consists of 440 CX gates and 120 Hadamard gates. As Qiskit has access to arbitrary rotations in both X and Z bases it introduces a mix of S, $S^\dagger$ and Z (and their equivalent Pauli-X rotations). Tableaux Manipulation only introduces either S or $\sqrt{X}$. For both compiled circuits the number of two-qubit gates is 440 as expected. TM also introduces marginally more S gates than Qiskit, but significantly fewer $\sqrt{X}$ gates.

To quantify this difference as the circuit size changes, in Fig. 2 we plot the ratio of gate counts when compiling with TM against Qiskit. In the left hand plot we are displaying a breakdown of the single qubit Cliffords present in both circuits. Splitting them into their respective X-type and Z-type rotations, we can see that TM is consistently introducing

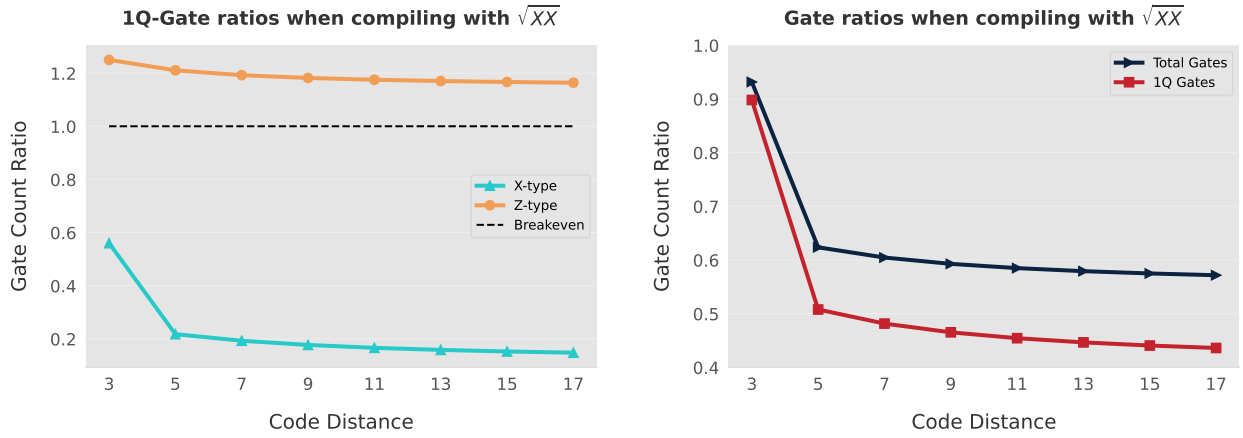


FIG. 2: *Left*: Ratio of single qubit gates present in the circuit when compiling with Tableaux Manipulation against Qiskit. Ratios are split into X-type (orange circles) and Z-type (blue triangles) single qubit gates. *Right*: Ratio of the total gates (navy arrowheads) and total single qubit gates (red squares) when compiling with Tableaux Manipulation against Qiskit.

more S gates than Qiskit for all code distances: 125% of the Qiskit count for $d = 3$ and decreasing towards 116% as the distance increases. However, TM introduces far fewer $\sqrt{X}$ gates than Qiskit: 56% of the Qiskit count for $d = 3$ and trending to below 15% for increasing code distance. For some hardware architectures, such as superconducting qubits, Z-type rotations can be implemented 'virtually' making them effectively noiseless [18]. The right hand plot compares both the single qubit gate count and the total gate count ratios. The first point in both curves representing $d = 3$ is an outlier, as the circuit is quite small. For higher distance values, the single qubit gate count in a TM compiled circuit approaches 43% of that in one compiled by Qiskit. In terms of total number of quantum operations, TM only uses 57% of those used by Qiskit for a $d = 17$ syndrome extraction circuit.

With regards to circuit depth, for both Qiskit and TM-compiled circuits across all code distances, the resulting circuits had 14 layers of gates.

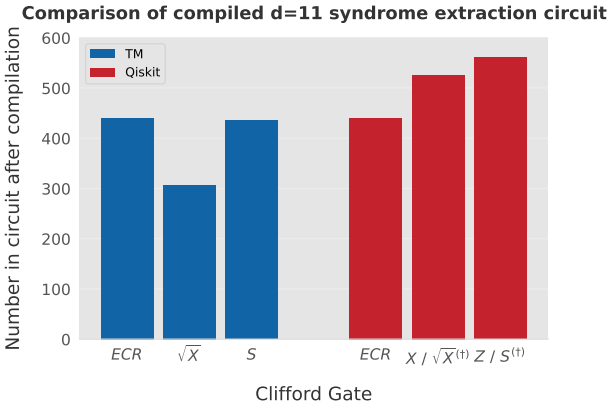
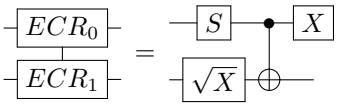


FIG. 3: Gate distribution when compiling a syndrome extraction circuit on a rotated surface code of distance $d = 11$. The original circuit is 6 layers of gates, defined in terms of H and CX. Tableaux Manipulation (left, dark blue) and Qiskit (right, dark red) both compiled into a gateset with ECR as the native two-qubit gate.

2. *ECR*

The ECR gate, while Clifford, is not a native gate to `stim`. When compiling with this gate in `stim` we used the equivalent circuit in alternative Clifford gates:



(9)

Where the subscripts 0, 1 in the ECR gate diagram represents the role of the qubits, as the gate is anti-symmetric. The tableau for ECR is shown in Eq. (4). This circuit was treated as a single two-qubit gate and gates were only added before or after ECR layers.

In Fig. 3 we display the equivalent bar chart as in Fig. 1 for compilation with an ECR gate. While the effect is perhaps more muted than that of $\sqrt{XX}$, TM is consistently introducing fewer single qubit gates than Qiskit. This reduction is quantified across all code distances in Fig. 4, where we plot the ratio of gate counts of TM against Qiskit. Across all code distances we introduce fewer single qubit gates than Qiskit, with the effect more pronounced for X-type rather than Z-type gates. For Z-type gates, we reduce the total count down to 75% of Qiskits total for $d = 17$; for X-type gates TM introduces only 57% of gates compared to Qiskit.

Looking at the right hand plot, the ratio of single qubit gates introduced by TM against Qiskit drops from $< 82\%$ for the smallest code size and approaches 66% for the highest distance considered. When compiling with an ECR gate for $d = 17$, a circuit compiled by TM will only have about 76% of the gates compared to one compiled by Qiskit.

With regards to circuit depth we take into account that when compiling with the circuit equivalency shown in Eq. (9) a single layer is being represented by three layers. Correcting for this, across all code distances for both Qiskit and TM-compiled cir-

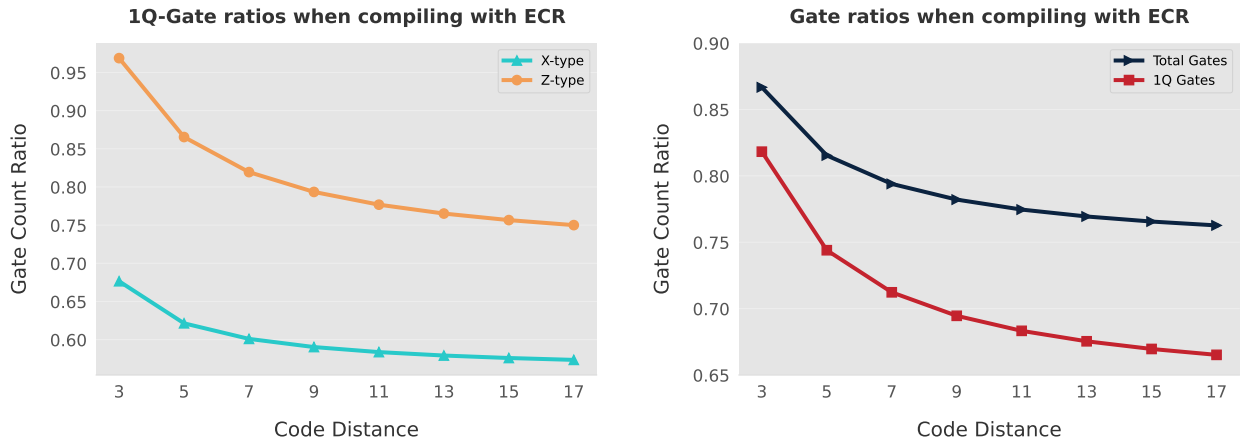


FIG. 4: *Left*: Ratio of single qubit gates present in the circuit when compiling with Tableaux Manipulation against Qiskit. Ratios are split into X-type (orange circles) and Z-type (blue triangles) single qubit gates. *Right*: Ratio of the total gates (navy arrowheads) and total single qubit gates (red squares) when compiling with Tableaux Manipulation against Qiskit.

cuits, the compiled circuits had 15 layers of gates.

Qiskit was the best performing open-source compiler we tested. A future version of this paper may include comparisons to other compilers. We compare favorably to Qiskit as it is *not* a Clifford compiler: it is designed for arbitrary quantum gates. As mentioned, restricting Qiskit to only use Clifford operations resulted in either a failure or degrading performance, indicating that this process (or one similar to it) is not being employed.

C. Implications for logical error

Here we will briefly highlight how the above results affect the logical error probability of quantum memory experiments. We present the results for a $d = 11$ quantum memory being compiled into a gateset with $\sqrt{XX}$. As this gate is native to `stim` it is straightforward for noisy simulations. For context, at this distance TM uses about 58.5% of the quantum operations compared to Qiskit (see Fig. 2).

The circuits are generated with `stim` as described in Sec. III B, setting `distance=11` and `rounds=11`. The syndrome extraction subcircuit (4 layers of CX sandwiched by 2 layers of H) is extracted and individually compiled using both TM and Qiskit into a gateset with $\sqrt{XX}$ as the two-qubit gate. The single qubit gates used for compilation are as in Sec. III B. The compiled circuits are then reinserted into the `stim` circuit in place of the $H + CX$ subcircuit, such that we have three copies of the same $d = 11$ quantum memory experiment defined in `stim`: one using $H + CX$, one compiled by TM and one compiled by Qiskit.

We apply a depolarizing noise model to each circuit, such that:

- Two-qubit gates have a two-qubit depolarizing channel applied with probability p.
- Single qubit gates have a single qubit depolarizing channel applied with probability $p/10$.
- Reset operations in the Z basis have a single qubit depolarizing channel applied with probability $p/10$.
- Measurement outcomes are flipped with probability p.
- For each layer in the circuit, any idle qubits have a single qubit depolarizing noise channel applied with probability $p/10$.

Decoding was then performed using the `PyMatching` package [19], with each circuit being sampled with $100p^{-2}$ shots. We ran simulations for varying noise values around the threshold, which for this noise model on logical-Z memory using a $H + CX$ gateset is 0.00927(9). When compiling with Qiskit, the threshold drops to 0.00747(23); for TM the threshold is 0.00797(54). The results for the $d = 11$ case are reported in Table II.

We see that compiling into a native gateset causes the logical error probability to jump significantly, and that this effect is suppressed by using Tableaux Manipulation. At $p = 0.01$, we see that TM is reducing the logical error probability by about $\sim 11\%$, and this reduction improves as the physical error decreases below threshold. At $p = 0.004$, we are achieving $\sim 18\%$ fewer logical errors by compiling with TM versus Qiskit. Beyond this point the TM / Qiskit ratio begins to rise again, as we enter the regime where single qubit gates become essentially noiseless in comparison to two-qubit gates and measurements. Far below threshold, say at $p = 10^{-4}$, it is likely that the logical error probability would be equivalent across all three circuit implementations. However, we argue that it is still advantageous to employ TM as communicating instructions to physical qubits still entails a non-zero heat cost, and TM allows you to achieve the same circuit implementations with fewer quantum operations.

Depolarizing Noise p	H+CX	TM Compiled	Qiskit Compiled	TM / Qiskit
0.002	$(6.12 \pm 2.81) \times 10^{-6}$	$2.07(43) \times 10^{-5}$	$2.49(41) \times 10^{-5}$	0.846(187)
0.004	$4.13(42) \times 10^{-4}$	$1.39(8) \times 10^{-3}$	$1.70(7) \times 10^{-3}$	0.819(48)
0.006	$4.68(24) \times 10^{-3}$	$1.41(3) \times 10^{-2}$	$1.7(0) \times 10^{-2}$	0.828(25)
0.008	0.023(1)	0.06(1)	0.07(1)	0.851(15)
0.01	0.0683(12)	0.152(2)	0.172(2)	0.886(13)

TABLE II: Logical error values for a $d = 11$ quantum memory experiment experiencing a depolarizing noise model decoded with minimum weight perfect matching. We compare the original circuit defined in terms of $H + CX$ to a circuit compiled into a gateset using $\sqrt{XX}$ with both TM and Qiskit.

IV. DISCUSSION

We have presented a simple, intuitive method for compiling a Clifford interaction into another by running a find-and-replace protocol for entangling operations, and fixing up the instantaneous Pauli conjugation as necessary. In comparing to open-source alternatives, compilation of syndrome extraction circuits for the rotated surface code resulted in a reduction of required quantum operations across all code distances.

For the class of circuits we have considered, experimentally this method provides a net positive with no clear downsides. For near-term devices exhibiting a noise profile roughly around 1% depolarizing, this method of compilation will improve logical fidelity in experiments. When hardware improves to the point that single qubit gates are effectively noiseless, this protocol maintains an advantage over existing compilers by requiring fewer quantum operations communicated to physical qubits.

Practicality aside, the procedure outlined above is useful for building intuition around Clifford gates and circuits, and accelerating certain aspects of research when dealing with alternative Clifford gatesets beyond $\{H,\ S,\ CX\}$. Directly dealing with tableaux allows extraneous detail of the Clifford interaction to be abstracted away and reduced to ensuring the conjugation is correct.

We have only considered circuits employing one kind of entangling Clifford, and focused on compiling into gatesets using the same kind of entangling Clifford; we provide some discussion about replacing a CX-like gate with an $iSWAP$-like gate in Appendix A. However, this is insufficient to determine how this process would perform on arbitrary Clifford circuits. Given a gateset with both CX- and $iSWAP$-like Cliffords, the initial condition can be easily adapted to replacing each entangling gate in an arbitrary circuit with its corresponding type in our gateset. If we only have access to one type of entangling Clifford, compiling an arbitrary stabilizer circuit would likely require further algorithms to ensure the minimal number of entangling gates is employed, such as that in Ref. [11].

[1] Daniel Litinski. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum*, 3:128, March 2019. ISSN 2521-327X. doi: 10.22331/q-2019-03-05-128. URL `https://doi.org/10.22331/q-2019-03-05-128`.

[2] Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, dec 2012. doi:10.1088/1367-2630/14/12/123011. URL `https://dx.doi.org/10.1088/1367-2630/14/12/123011`.

[3] Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, Feb 2023. ISSN 1476-4687. doi:10.1038/s41586-022-05434-1. URL `https://doi.org/10.1038/s41586-022-05434-1`.

[4] S. A. Moses et al. A race-track trapped-ion quantum processor. *Phys. Rev. X*, 13:041052, Dec 2023. doi: 10.1103/PhysRevX.13.041052. URL `https://link.aps.org/doi/10.1103/PhysRevX.13.041052`.

[5] Dolev Bluvstein et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, Feb 2024. ISSN 1476-4687. doi:10.1038/s41586-023-06927-3. URL `https://doi.org/10.1038/s41586-023-06927-3`.

[6] Klaus Mølmer and Anders Sørensen. Multiparticle entanglement of hot trapped ions. *Phys. Rev. Lett.*, 82:1835–1838, Mar 1999. doi: 10.1103/PhysRevLett.82.1835. URL `https://link.aps.org/doi/10.1103/PhysRevLett.82.1835`.

[7] Moein Malekakhlagh, Easwar Magesan, and David C. McKay. First-principles analysis of cross-resonance gate operation. *Phys. Rev. A*, 102:042605, Oct 2020. doi:10.1103/PhysRevA.102.042605. URL `https://link.aps.org/doi/10.1103/PhysRevA.102.042605`.

[8] Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574 (7779):505–510, Oct 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1666-5. URL `https://doi.org/10.1038/s41586-019-1666-5`.

[9] Sergey Bravyi and Dmitri Maslov. Hadamard-free circuits expose the structure of the clifford group. *IEEE Transactions on Information Theory*, 67(7): 4546–4563, July 2021. ISSN 1557-9654. doi: 10.1109/tit.2021.3081415. URL `http://dx.doi.org/10.1109/TIT.2021.3081415`.

[10] Ewout van den Berg. A simple method for sampling random clifford operators, 2021. quant-ph/2008.06011.

[11] Sergey Bravyi, Ruslan Shaydulin, Shaohan Hu, and Dmitri Maslov. Clifford Circuit Optimization with Templates and Symbolic Pauli Gates. *Quantum*,

5:580, November 2021. ISSN 2521-327X. doi: 10.22331/q-2021-11-16-580. URL https://doi.org/10.22331/q-2021-11-16-580.

[12] Vadym Kliuchnikov and Dmitri Maslov. Optimization of clifford circuits. *Phys. Rev. A*, 88:052307, Nov 2013. doi:10.1103/PhysRevA.88.052307. URL https://link.aps.org/doi/10.1103/PhysRevA.88.052307.

[13] Daniel Gottesman and Isaac L. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402(6760):390–393, Nov 1999. ISSN 1476-4687. doi:10.1038/46503. URL https://doi.org/10.1038/46503.

[14] Daniel Gottesman. Stabilizer codes and quantum error correction. 1997. quant-ph/9705052.

[15] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), November 2004. ISSN 1094-1622. doi: 10.1103/physreva.70.052328. URL http://dx.doi.org/10.1103/PhysRevA.70.052328.

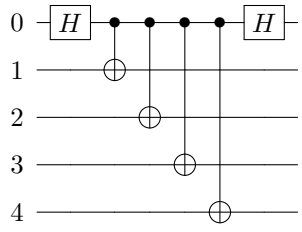
[16] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, July 2021. ISSN 2521-327X. doi: 10.22331/q-2021-07-06-497. URL https://doi.org/10.22331/q-2021-07-06-497.

[17] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023. 10.5281/zenodo.2573505.

[18] David C. McKay, Christopher J. Wood, Sarah Sheldon, Jerry M. Chow, and Jay M. Gambetta. Efficient z-gates for quantum computing. *Physical Review A*, 96(2), August 2017. ISSN 2469-9934. doi: 10.1103/physreva.96.022330. URL http://dx.doi.org/10.1103/PhysRevA.96.022330.

[19] Oscar Higgott. Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching, 2021. quant-ph/2105.13082.

Appendix A: Caveats for the broader Clifford class



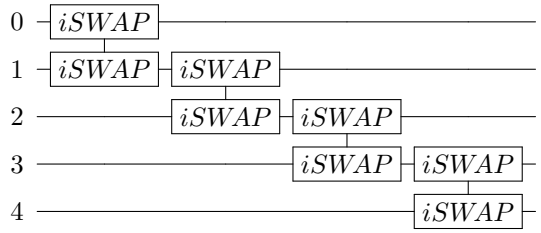
(A1)

	X_0	Z_0	X_1	Z_1	X_2	Z_2	X_3	Z_3	X_4	Z_4
$\pm$	+	+	+	+	+	+	+	+	+	+
0	X	Z	_	X	_	X	_	X	_	X
= 1	_	X	X	Z	_	_	_	_	_	_
2	_	X	_	_	X	Z	_	_	_	_
3	_	X	_	_	_	_	X	Z	_	_
4	_	X	_	_	_	_	_	_	X	Z

.

In the main text we are focused on circuits defined on CX-like entangling Cliffords being compiled into a gateset which also uses CX-like entangling Cliffords. The same process can be directly applied to circuits defined on only $iSWAP$-like entangling Cliffords, being compiled into a gateset which also uses $iSWAP$-like entangling Cliffords. As the protocol relies on a heuristic of replacing entangling gates with entangling gates (implicitly between the same qubits), it applies equally to both these cases. However, if we wish to compile a circuit defined on a CX-like gate into a gateset that only uses an $iSWAP$-like gate, or vice versa, this heuristic breaks down.

Consider the weight-4 X-stabiliser circuit shown in Eq. (A1). If we replace each CX gate with an $iSWAP$ between the same qubits (such that qubit 0 is involved in four $iSWAP$ operations) we will not be within single qubit rotations to the desired Clifford interaction. Due to the swapping aspect of these gates we must be aware of which qubit states have been swapped, tracking an 'instantaneous qubit index' so to speak. Consider the following as an alternative initial circuit for TM compilation:



(A2)

	X_0	Z_0	X_1	Z_1	X_2	Z_2	X_3	Z_3	X_4	Z_4
$\pm$	+	+	+	+	+	+	+	+	+	+
0	Z	_	Y	Z	_	_	_	_	_	_
= 1	Z	_	_	_	Y	Z	_	_	_	_
2	Z	_	_	_	_	_	Y	Z	_	_
3	Z	_	_	_	_	_	_	_	Y	Z
4	X	Z	Z	_	Z	_	Z	_	Z	_

This circuit can be made equivalent to Eq. (A1) using only single qubit Clifford gates, up to some permutation of rows. Qubit 0 is no longer the qubit to be measured, and has been swapped into qubit 4. This is a relaxation of the requirement that two stabilizer tableaux be equal to ensure compilation, but comes at the cost of qubit state shuffling. Indeed, on certain qubit architectures the connectivity graph dictates that the circuit in Eq. (A1) is valid and the circuit in Eq. (A2) is not. In this case we can then either a) introduce further qubits to the tableaux to act as intermediaries, wherein our goal is to replicate the target tableau in a *subset* of rows in the new tableau, up to some permutation, or b) introduce further $iSWAP$ and single qubit gates within the original set of qubits to undo the state swapping.

In the cases where an informed initial condition is not immediately obvious, we propose employing the initial condition as normal and then inserting $SWAP$ gates after each $iSWAP$, between the same qubits, such that $(iSWAP + SWAP)$ approximates a CX. Compilation can then proceed as normal. Once the tableaux are equivalent, iterate through the circuit removing each $SWAP(i,\ j)$ gate sequentially, and updating following qubit indices $(i \leftrightarrow j)$ as necessary. This will not correct for all possible edge cases, such as the restriction in connectivity mentioned above.