CALT-68-2067, QUIC-96-002

Efficient Computations of Encodings for Quantum Error Correction

Richard Cleve*

Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4

Daniel Gottesman†

California Institute of Technology, Pasadena, CA 91125

Abstract

We show how, given any set of generators of the stabilizer of a quantum code, an efficient gate array that computes the codewords can be constructed. For an n-qubit code whose stabilizer has d generators, the resulting gate array consists of $O(nd)$ operations, and converts k-qubit data (where $k = n - d$) into n-qubit codewords.

02.70.Rw, 03.65.Bz, 89.80.+h

*cleve@cpsc.ucalgary.ca

†gottesma@theory.caltech.edu

I. INTRODUCTION

Recently, significant progress has been made in the development of error-correction schemes for quantum information systems [1–10]. This includes methods for converting classical error-correcting codes to into quantum error-correcting codes [2,3], formalizations of necessary and sufficient conditions for sets of states to form quantum codes [11,7,12], and a mathematical framework for a large class of quantum codes, known as *stabilizer codes* [8,9].

In order to actually use quantum codes in quantum information systems, constructive methods for performing encodings, error-correction, and decodings are required. Towards this end, gate arrays that perform these computations are helpful. Methods for producing gate arrays that perform error-correction for any stabilizer code have been presented [13]. For computing encodings, the only gate arrays that have been proposed apply either to one specific code (such as one that encodes one qubit as five qubits and protects against a one-qubit error) [4,7,14], or to restricted classes of stabilizer codes [3,10]. In the present paper, we show how to efficiently construct a gate array that computes encodings for *any* stabilizer code. In the case of an n-qubit code defined in terms of d generators, our gate array consists of at most nd operations (which are all one- or two-qubit gates and performed in-place), and it converts k-qubit data (where $k = n - d$) into n-qubit codewords. The gate array can be used for decoding by running it backwards (on a correct codeword). The method is illustrated throughout the paper using an $n = 8$, $d = 5$, $k = 3$ code (that can handle a one-qubit error) [8–10]; however, it works for any stabilizer code.

We conjecture that the sizes of our gate arrays are asymptotically optimal for *general* stabilizer codes, though they may not be optimal for some specific stabilizer codes.

In Section II, we review the basics of stabilizer codes, and introduce new terminology that will be useful later. In Section III, we explain how to use Gaussian elimination to convert codes into a "standard form." In Section IV, we show how to produce a gate array from a code in standard form.

II. STABILIZER CODES

In [8,9] it has been shown that many quantum codes can be described in terms of their *stabilizers*. The stabilizer of a code is the set of all operators that:

(a) are formed by taking tensor products of matrices of the form

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad Y = X \cdot Z = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}; \text{ and}$$

(b) fix every codeword.

Thus, the codewords form the $+1$-eigenspace of the operators in the stabilizer. An error that anticommutes with an operator M in the stabilizer will take a codeword from the $+1$-eigenspace of M to the -1-eigenspace of M. The new state can be distinguished from the old state, and hopefully corrected, by measuring the eigenvalue of M. If the operators in the stabilizer are chosen carefully, a large class of errors can be corrected — for instance, all errors operating on a single qubit or up to t qubits.

The stabilizer of a code is always a group. It can be most easily described by a set of *generators* $G_1, \ldots, G_d$. The other elements of the stabilizer are products of various G_i's. The actual choice of generators is somewhat arbitrary, as long as none of them is the product of other generators. Any tensor product of I, X, Y, and Z will square to ± 1, but in order to have eigenvalues $+1$, the generators must actually square to $+1$. In addition, in order to have a nontrivial joint $+1$-eigenspace, the generators must commute with each other, so the stabilizer is an Abelian group.

Generators for an eight qubit code that protects a three qubit state with up to one error (as explained in [8–10]) are

$$\begin{aligned} G_1 &= X \otimes X \otimes X \otimes X \otimes X \otimes X \otimes X \otimes X \\ G_2 &= Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \otimes Z \\ G_3 &= X \otimes I \otimes X \otimes I \otimes Z \otimes Y \otimes Z \otimes Y \\ G_4 &= X \otimes I \otimes Y \otimes Z \otimes X \otimes I \otimes Y \otimes Z \\ G_5 &= X \otimes Z \otimes I \otimes Y \otimes I \otimes Y \otimes X \otimes Z. \end{aligned} \tag{1}$$

Once we know the generators $G_1, \ldots, G_d$, we have most of the vital information about the code. If there are n qubits and d generators, we can encode $k = n - d$ data qubits. By measuring the eigenvalue of each of the d generators, we can learn what, if any, error has occured, and fix it. In [13], it is explicitly shown how to do this in a fault-tolerant way. If we do not require strict fault-tolerance, the array to identify the error syndrome has $O(nd)$ gates. Once the error syndrome is determined, it may take a long classical computation to determine the actual error, depending on the code used.

In order to actually encode states using a quantum code, we need to decide which states will act as basis states for the coding space. In order to do this, it is convenient to use the language of binary vector spaces, as in [9]. Define the *X-vector* of the generator

$$G_i = G_{i1} \otimes G_{i2} \otimes \cdots \otimes G_{in},$$

as the n-bit vector, denoted as X_{G_i}, where

$$(X_{G_i})_j = \begin{cases} 1 & \text{if } G_{ij} = X \text{ or } Y \\ 0 & \text{if } G_{ij} = I \text{ or } Z. \end{cases}$$

The *Z-vector* of G_i, denoted as Z_{G_i}, is defined as

$$(X_{G_i})_j = \begin{cases} 1 & \text{if } G_{ij} = Z \text{ or } Y \\ 0 & \text{if } G_{ij} = I \text{ or } X. \end{cases}$$

Also, the *X-matrix* of generators $G_1, \ldots, G_d$ is defined as the $n \times d$ matrix, denoted as X_G, where

$$(X_G)_{ji} = \begin{cases} 1 & \text{if } G_{ij} = X \text{ or } Y \\ 0 & \text{if } G_{ij} = I \text{ or } Z, \end{cases}$$

(that is, the matrix whose columns are $X_{G_1}, \ldots, X_{G_d}$). The *Z-matrix* of $G_1, \ldots, G_d$, denoted as Z_G, is defined similarly. Note that the X- and Z-matrices together completely determine the sequence of generators that they correspond to.

The X- and Z-matrices for the aforementioned generators (1) of the eight-qubit code are

$$X_G = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad Z_G = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{2}$$

Note that columns 1, 3, 4, 5 of X_G are linearly independent, while column 2 is null. In general, we will call generators whose X-vectors are linearly independent *primary* generators ("type 1" in the language of [8]), and generators whose X-vectors are null *secondary* generators ("type 2" in the language of [8]). (It is possible to have sets of non-null X-vectors that are not linearly independent; however, as discussed in [8], the generators can always be transformed so that they consist of primary and secondary types.)

To choose the codewords, we augment these generators with a set of k *seed* generators (where $k = n - d$), which are chosen so that:

(a) the X-vectors of the seed and primary generators are linearly independent; and

(b) each seed generator commutes with each secondary generator.

Let $M_1, \ldots, M_b$, $L_1, \ldots, L_r$, and $N_1, \ldots, N_k$ be the primary, secondary, and seed generators, respectively (where $b + r = d$). Then, as shown in [8], each k-qubit basis state $|c_1 \ldots c_k\rangle$ can be associated with a quantum codeword

$$\frac{1}{\sqrt{2^b}} \sum_{a_1 \ldots a_b \in \{0,1\}^b} M_1^{a_1} \cdots M_b^{a_b} N_1^{c_1} \cdots N_k^{c_k} |\overbrace{0 \ldots 0}^{n}\rangle, \tag{3}$$

(and, by linear extension, this defines the codeword for an arbitrary k-qubit state). Note that these 2^k basic codewords are all valid quantum states and are mutually orthogonal, since condition (a) implies that the states $M_1^{a_1} \cdots M_b^{a_b} N_1^{c_1} \cdots N_k^{c_k} |0 \ldots 0\rangle$ for $a_1, \ldots, a_b, c_1, \ldots, c_k \in \{0, 1\}$ are all distinct basis states. Also, the basic codewords are all fixed by the stabilizer (i.e., they lie within the specified code). To see why this is so, it is useful to note that

$$\frac{1}{\sqrt{2^b}} (I + M_1) \cdots (I + M_b) N_1^{c_1} \cdots N_k^{c_k} |\overbrace{0 \ldots 0}^{n}\rangle \tag{4}$$

is equivalent to (3). Expression (4) is fixed by each primary generator M_i, because $M_i \cdot (I + M_i) = (M_i + I)$, and is fixed by each secondary generator L_j, because

$$\begin{aligned} L_j \cdot \left(\frac{1}{\sqrt{2^b}} (I + M_1) \cdots (I + M_b) N_1^{c_1} \cdots N_k^{c_k} |0 \ldots 0\rangle \right) \\ &= \frac{1}{\sqrt{2^b}} (I + M_1) \cdots (I + M_b) N_1^{c_1} \cdots N_k^{c_k} \cdot L_j |0 \ldots 0\rangle \\ &= \frac{1}{\sqrt{2^b}} (I + M_1) \cdots (I + M_b) N_1^{c_1} \cdots N_k^{c_k} |0 \ldots 0\rangle, \end{aligned}$$

where we are using the fact that L_j commutes with each seed generator (condition (b)).

In [8], methods for constructing generators and seeds for a variety of codes are given, including the following seeds for the eight qubit code (1), (2):

$$\begin{aligned} N_1 &= X \otimes X \otimes I \otimes I \otimes I \otimes I \otimes I \otimes I \\ N_2 &= X \otimes I \otimes X \otimes I \otimes I \otimes I \otimes I \otimes I \\ N_3 &= X \otimes I \otimes I \otimes I \otimes X \otimes I \otimes I \otimes I \,. \end{aligned} \tag{5}$$

It has been shown [15] that any set of primary and secondary generators can be efficiently augmented with seed generators. We shall present a related method for constructing seed generators that takes advantage of the generators being in a special form.

The resulting basic codewords for the eight qubit code are written out in full in [8,10]. For large codes, it is impractical to explicitly write out the basic codewords. For instance, one of the codes from [8] is a sixteen qubit code that protects ten qubits against one error. The code is described by six generators and ten seed generators. If written out in full, this code consists of 1,024 codewords, each of which is a superposition of 32 basis states (which amounts to 32,768 basis states in total!).

An alternative is to produce a gate array that transforms basis states into codewords. This, in addition to specifying the code, indicates how encodings might be *computed* by quantum computers. The orthogonality of the codewords implies that the mapping of k-qubit states to n-qubit codewords can, in principle, be implemented unitarily (technically, the unitary transformation would map n-qubit states of the form $|c_1 \ldots c_k\rangle \otimes |\overbrace{0 \ldots 0}^{d}\rangle$ to n-qubit codewords). This does not imply that the unitary transformation can be implemented *efficiently* by a gate array (for example, with a polynomial number of operations with respect to n). A direct conversion of the 2^k basic codewords (each of which is a superposition of 2^b basis states) into a gate array would generally result in an exponential number of operations.

The expression (4) is equivalent to (3) and is apparently simpler in that it contains no exponentially large sums; however, it is not clear how to translate this into a gate array, since operations of the form $\frac{1}{\sqrt{2}}(I + M_i)$ are not unitary.

III. CONVERTING GENERATORS INTO STANDARD FORM

Let $G_1, \ldots, G_d$ be generators of the stabilizer of some n-qubit code. We shall show how to systematically convert the matrices X_G and Z_G into a *standard form* which is very useful for producing codewords. From this form, a set of primary and secondary generators, as well as a suitable set of seed generators, are readily available. More importantly, we shall show in the next section how to convert a set of generators in standard form into a gate array of size $O(nd)$ that transforms k-qubit states (where $k = n - d$) into codewords.

Our conversion will involve transformations which change a generator G_i into $G_i \cdot G_j$, where $j \neq i$. Since the stabilizer is a group, it is unchanged by such a transformation on the generators. In terms of the X- and Z-matrices, such an operation adds the jth column to the ith column in both matrices (in modulo 2 arithmetic), which is a basic step in Gaussian elimination. We shall also allow the n qubit positions to be reordered, which corresponds to a reordering of the rows in both matrices (another basic step in Gaussian elimination). This reordering is not strictly necessary, but is convenient for notational purposes; it clearly does not change the characteristics of the code.

Let $X^{(0)}$ and $Z^{(0)}$ denote the original X- and Z-matrices (X_G and Z_G) of the generators (they are each $n \times d$ matrices). We shall perform a suitable Gaussian elimination on these matrices, and then augment them with columns corresponding to seed generators leading to what we shall call a *standard form.*

By performing Gaussian elimination on $X^{(0)}$, we can obtain matrices of the form

$$X^{(1)} = \begin{matrix} \\ r+k \\ b \end{matrix} \overbrace{\quad}^{r} \overbrace{\quad}^{b} \begin{pmatrix} 0 & A \\ 0 & I \end{pmatrix} \qquad Z^{(1)} = \begin{matrix} \\ r+k \\ b \end{matrix} \overbrace{\quad}^{r} \overbrace{\quad}^{b} \begin{pmatrix} B & C \\ D & E \end{pmatrix},$$

where b is the rank of $X^{(0)}$, $r = d - b$, and $k = n - d$. At this stage, the Z-matrix $Z^{(1)}$ has no special form. Next, by performing Gaussian elimination on the first $r + k$ rows and the first r columns of $Z^{(1)}$, we can transform B, the $(r+k) \times r$ submatrix of $Z^{(1)}$, into B' of the

form

$$B' = \begin{array}{c} \\ k\{ \\ r_2\{ \\ r_1\{ \end{array} \overbrace{\begin{array}{cc} 0 & B_1 \\ 0 & B_2 \\ 0 & I \end{array}}^{\begin{array}{cc} r_2 & r_1 \end{array}},$$

where r_1 is the rank of B and $r_2 = r - r_1$. Note that this does not affect the last b rows or the first r columns of $X^{(1)}$. Thus, the resulting forms of the X- and Z-matrices (blocked with the new partition) are

$$X^{(2)} = \begin{array}{c} \\ k\{ \\ r_2\{ \\ r_1\{ \\ b\{ \end{array} \overbrace{\begin{array}{ccc} 0 & 0 & A_1 \\ 0 & 0 & A_2 \\ 0 & 0 & A_3 \\ 0 & 0 & I \end{array}}^{\begin{array}{ccc} r_2 & r_1 & b \end{array}} \qquad Z^{(2)} = \begin{array}{c} \\ k\{ \\ r_2\{ \\ r_1\{ \\ b\{ \end{array} \overbrace{\begin{array}{ccc} 0 & B_1 & C_1 \\ 0 & B_2 & C_2 \\ 0 & I & C_3 \\ D_1 & D_2 & E \end{array}}^{\begin{array}{ccc} r_2 & r_1 & b \end{array}}.$$

The first $r = r_1 + r_2$ columns correspond to secondary generators, and the last b columns correspond to primary generators (it is clear that the last b columns of $X^{(2)}$ are linearly independent).

Next, we shall augment these matrices with k columns corresponding to k seed generators. Recall that the properties that the seed generators must have are: their X-vectors are linearly independent of those of the primary generators; and, they commute with the secondary generators. Let the X- and Z-matrices of the seed generators be

$$X^{(s)} = \begin{array}{c} \\ k\{ \\ r_2\{ \\ r_1\{ \\ b\{ \end{array} \overbrace{\begin{array}{c} I \\ 0 \\ B_1^T \\ 0 \end{array}}^{k} \qquad Z^{(s)} = \begin{array}{c} \\ k\{ \\ r_2\{ \\ r_1\{ \\ b\{ \end{array} \overbrace{\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \end{array}}^{k},$$

where B_1^T is the transpose of B_1. It is clear that the X-vectors of these seed generators are linearly independent of those of the primary generators (that is, the columns of $X^{(s)}$

are linearly independent of the last b columns of $X^{(2)}$). To see why these seed generators commute with the secondary generators, note that this condition is equivalent to having an even number of 1's in their X-vectors in common with the Z-vector of each secondary generator. This is equivalent to

$$\overset{}{}_{k}\{\begin{pmatrix} \overbrace{I}^{k} & \overbrace{0}^{r_2} & \overbrace{B_1}^{r_1} & \overbrace{0}^{b} \end{pmatrix} \begin{pmatrix} \overbrace{0}^{r_2} & \overbrace{B_1}^{r_1} \\ 0 & B_2 \\ 0 & I \\ 0 & D_2 \end{pmatrix} = {}_{k}\{\begin{pmatrix} \overbrace{0}^{r_2} & \overbrace{0}^{r_1} \end{pmatrix},$$

which holds because $B_1 + B_1 = 0$ (modulo 2).

The augmented matrices $X^{(*)}$ and $Z^{(*)}$, which include the seed, secondary, and primary generators, are as follows.

$$X^{(*)} = \begin{matrix} \\ {}_{k}\{ \\ {}_{r_2}\{ \\ {}_{r_1}\{ \\ {}_{b}\{ \end{matrix} \begin{matrix} \overbrace{}^{k} & \overbrace{}^{r_2} & \overbrace{}^{r_1} & \overbrace{}^{b} \end{matrix} \begin{pmatrix} I & 0 & 0 & A_1 \\ 0 & 0 & 0 & A_2 \\ B_1^T & 0 & 0 & A_3 \\ 0 & 0 & 0 & I \end{pmatrix} \qquad Z^{(*)} = \begin{matrix} \\ {}_{k}\{ \\ {}_{r_2}\{ \\ {}_{r_1}\{ \\ {}_{b}\{ \end{matrix} \begin{matrix} \overbrace{}^{k} & \overbrace{}^{r_2} & \overbrace{}^{r_1} & \overbrace{}^{b} \end{matrix} \begin{pmatrix} 0 & 0 & B_1 & C_1 \\ 0 & 0 & B_2 & C_2 \\ 0 & 0 & I & C_3 \\ 0 & D_1 & D_2 & E \end{pmatrix} \tag{6}$$

Call any specification of generators in the above form a *standard form.* Note that there are $O(nd)$ 1's in each matrix.

As an example, consider the generators for the eight qubit code described in the previous section (represented by the matrices of equations (2)). A standard form for this code is

$$X^{(*)} = \begin{pmatrix} 1&0&0&0&1&1&1&0 \\ 0&1&0&0&1&1&0&1 \\ 0&0&1&0&1&0&1&1 \\ 1&1&1&0&0&1&1&1 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{pmatrix} \qquad Z^{(*)} = \begin{pmatrix} 0&0&0&1&0&0&0&0 \\ 0&0&0&1&0&1&0&1 \\ 0&0&0&1&1&0&1&0 \\ 0&0&0&1&1&1&0&0 \\ 0&0&0&1&1&1&1&1 \\ 0&0&0&1&1&0&0&1 \\ 0&0&0&1&0&1&1&0 \\ 0&0&0&1&0&0&1&1 \end{pmatrix} \tag{7}$$

(with $k = 3$, $r_1 = 1$, $r_2 = 0$, and $b = 4$), where the fourth and fifth qubit positions (rows) have been transposed in the Gaussian elimination process. Modulo this transposition, the

last five columns generate the same stabilizer as the original five generators. The first three columns correspond to seed generators, the fourth column corresponds to a secondary generator, and the last four columns correspond to primary generators. The seed generators are not the same as those presented in Eq. (5), so the basis codewords will be different. However, since the stabilizer is the same, the full coding space is the same as before.

IV. CONSTRUCTION OF GATE ARRAYS

We shall construct a n-qubit gate array with $O(nd)$ operations that computes the mapping

$$|c_1 \dots c_k\rangle \otimes |\overbrace{0 \dots 0}^{d}\rangle \mapsto \tfrac{1}{\sqrt{2^b}} \sum_{a_1 \dots a_b \in \{0,1\}^b} M_1^{a_1} \cdots M_b^{a_b} N_1^{c_1} \cdots N_k^{c_k} |\overbrace{0 \dots 0}^{n}\rangle. \tag{8}$$

This mapping (8) is the composition of two mappings,

$$|c_1 \dots c_k\rangle \otimes |\overbrace{0 \dots 0}^{d}\rangle \mapsto |c_1 \dots c_k\rangle \otimes |\overbrace{0 \dots 0}^{r}\rangle \otimes \tfrac{1}{\sqrt{2^b}} \sum_{a_1 \dots a_b \in \{0,1\}^b} |a_1 \dots a_b\rangle \tag{9}$$

and

$$|c_1 \dots c_k\rangle \otimes |\overbrace{0 \dots 0}^{r}\rangle \otimes |a_1 \dots a_b\rangle \mapsto M_1^{a_1} \cdots M_b^{a_b} N_1^{c_1} \cdots N_k^{c_k} |\overbrace{0 \dots 0}^{n}\rangle. \tag{10}$$

The first mapping (9) is trivially computed by independently applying a Q operation to each of the last b qubits, where

$$Q = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

To compute the second mapping (10), it is helpful to consider the eight qubit code, whose standard form is given by (7) in the previous section. Recall that the first three columns specify the seed generators and the last four columns specify the primary generators. Let us temporarily consider a simplified version of the generators specified in (7), where phase shifts are ignored. This is equivalent to keeping $X^{(*)}$ as is and setting $Z^{(*)}$ to all zeros (thus, Z's become I's, Y's become X's, and X's and I's are unchanged in the tensor products that

make up the generators). With respect to these generators, the mapping (10) is given by matrix $X^{(*)}$ as

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ 0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ 0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}, \tag{11}$$

where we are denoting the input state $|c_1c_2c_3\rangle \otimes |0\rangle \otimes |a_1a_2a_3a_4\rangle$ and the output state as column vectors (the phases are all $+1$ here). This mapping is computed by the following gate array (where we are using notation of [16]).

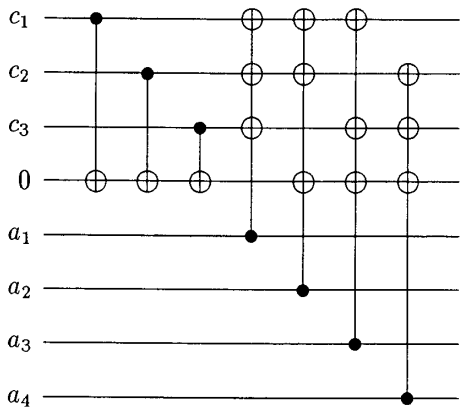

Figure 1

By inspection, we can see how the operations in the gate array correspond to the entries of matrix $X^{(*)}$. What we have done is to apply each column conditioned on one of its elements. For instance, if $a_3 = 1$, we apply the third primary generator by flipping the first, third, and fourth qubits. We can only do this if the seventh qubit, which initially has the value a_3, has not been changed before we get around to applying that generator. If an earlier column had a 1 in the seventh row, that qubit might have been changed before it could be used. This is why we needed to put the X-matrix in standard form — each column of the matrix must be conditioned on the input of the corresponding qubit, so the diagonal elements must be first in their rows. A matrix in standard form has this property (at least for columns not corresponding to secondary generators), while a more general matrix does not.

Returning to our goal of implementing the mapping (10) *with phase shifts included*, we make the following observations about phase shifts and conditional phase shifts acting on basis states with phase ± 1.

(a) The action of a conditional phase shift or conditional negation depends on the state, but not on the phase of the state, to which it is applied.

(b) The action of a phase shift, or conditional phase shift, may affect the phase, but does not change the state on which it acts.

This implies that the phase shifts can be essentially inserted into Figure 1 as they occur in matrix $Z^{(*)}$; they do not affect the evolution of the state, but they do affect an evolving phase factor of ± 1. In the case of the eight qubit code, we insert conditional phase shifts into Figure 1 in accordance with the matrix $Z^{(*)}$ of (7) to obtain the following gate array.

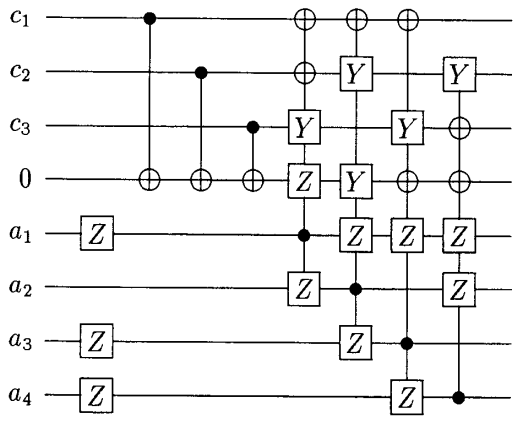

Figure 2

Of course, the fact that $X \cdot Z = Y$ is used here. We have done essentially the same thing here as with the X-matrix, conditioning the phases on specific qubits. Notice, for example, that the phase factors conditioned on a_1 correspond to the generator of the fifth column of $Z^{(*)}$. Also, recall that we exclude the secondary generators, so that the fourth column of $Z^{(*)}$ is ignored. Strictly speaking, the initial Z's for a_1, a_3, and a_4 are also conditioned on those qubits. However, since Z has no effect on $|0\rangle$, we can apply it as an unconditional operation.

The gate array of Figure 2 computes the mapping (10) for the eight qubit code, and

such a gate array can be constructed in general from any matrices $X^{(*)}$ and $Z^{(*)}$ in standard form.

Finally, a full gate array for computing codewords is obtained by composing the operations for mappings (9) and (10). In the case of the eight qubit code, we obtain the following (where $R = Q \cdot Z$).

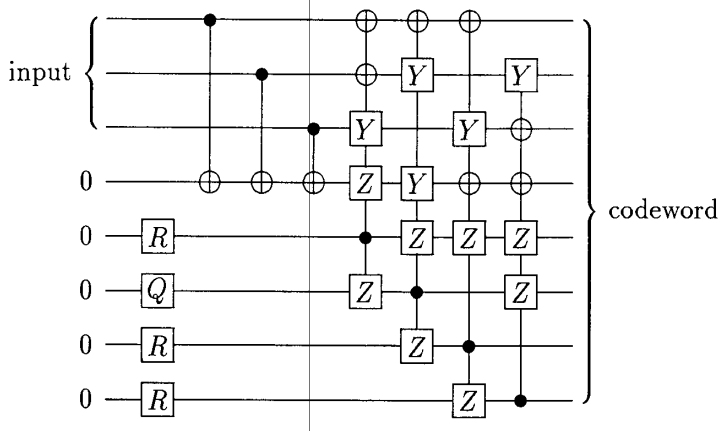

Figure 3

In general, the total number of two-qubit operations is bounded by

$$r_1 k + (n-1)b \leq (n-1)d$$

and the number of one-qubit operations is bounded by $b \leq d$. Thus, the total number of operations is bounded by nd, which is essentially the length of the description of the generators of the stabilizer. For the eight qubit code, there are four one-qubit operations and 23 two-qubit operations. For the sixteen qubit code mentioned in Section II, the gate array will have no more than 96 operations.

To recover messages from their codewords, it suffices to apply the codeword error-correction scheme proposed in [13] (without fault-tolerance, if not required) followed by the gate array for encoding (e.g. Figure 3) backwards.

ACKNOWLEDGMENTS

We would like to thank David Beckman, Sam Braunstein, David DiVincenzo, and Andrew Steane for helpful discussions about the computation of encodings. We are also grateful to the Institute for Scientific Interchange in Torino, and its director, Professor Mario Rasetti, for making possible the 1996 workshop on Quantum Computation at which some of this work was performed. R.C. is supported in part by NSERC of Canada. D.G. is supported by the U.S. Department of Energy under Grant No. DE-FG03-92-ER40701 and by DARPA through a grant to ARO.

REFERENCES

[1] P.W. Shor, "Scheme for reducing decoherence in quantum memory," Phys. Rev. A **52**, 2493 (1995).

[2] A.R. Calderbank and P.W. Shor, "Good quantum error-correcting codes exist," quant-ph/9512032.

[3] A.M. Steane, "Multiple particle interference and quantum error correction," quant-ph/9601029.

[4] R. Laflamme, C. Miquel, J.P. Paz, and W.H. Zurek, "Perfect quantum error correction code," quant-ph/9602019.

[5] L. Vaidman, L. Goldenberg, S. Wiesner, "Error prevention scheme with four particles", quant-ph/9603031.

[6] P.W. Shor and J.A. Smolin, "Quantum error-correcting codes need not completely reveal the error syndrome", quant-ph/9604006.

[7] C.H. Bennett, D.P. DiVincenzo, J.A. Smolin, and W.K. Wooters, "Mixed state entanglement and quantum error correcting codes," quant-ph/9604024.

[8] D. Gottesman, "A class of quantum error-correcting codes saturating the quantum Hamming bound," quant-ph/9604038.

[9] A.R. Calderbank, E.M. Rains, P.W. Shor, and N.J. Sloane, "Quantum error correction and orthogonal geometry," quant-ph/9605005.

[10] A.M. Steane, "Simple quantum error correcting codes," quant-ph/9605021.

[11] A. Ekert and C. Macchiavello, "Error correction in quantum communication", quant-ph/9602022.

[12] E. Knill and R. Laflamme, "A theory of quantum error-correcting codes," quant-ph/9604034.

[13] D.P. DiVincenzo and P.W. Shor, "Fault-tolerant error correction with efficient quantum codes," quant-ph/9605031.

[14] S.L. Braunstein, "Perfect quantum error correction in 26 laser pulses," quant-ph/9604036.

[15] D. Beckman, private communication, 1996.

[16] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," Phys. Rev. A **52**, 3457 (1995).