Build your own quantum computers, Part 3: Quantum error correction [1]

D. Candela

November 28, 2016

[1] These handouts were created for the honors colloquium for Physics 424. The material in Parts 1 and 2 can also be found in the published article: American Journal of Physics **83**, 688 (2015). Equations before Eq. 43 and figures before Fig. 11 are in Parts 1 and 2, also in the AJP article.

Introduction

Large-scale quantum computation will only be possible with an effective scheme for correcting errors. The difficulty is doing this without wrecking the superposition of states used by quantum algorithms.

This handout is a quick tour of quantum error correction, with numerical projects you can do. For more in-depth understanding look at the article by Devitt, et al.,[2] the text by Nielsen and Chuang,[3] or some of the lecture notes that have been posted online.[4]

There is a long section here on the "surface code." This can be skipped, proceeding directly to the last section on magic states.

[2] Simon J. Devitt, William J. Munro, and Kae Nemoto. Quantum error correction for beginners. 2013. URL `http://arxiv.org/abs/0905.2794v4`

[3] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge, 2000

[4] For a more mathematical viewpoint and interesting philosophical/historical asides see the notes posted by Dave Bacon at `http://courses.cs.washington.edu/courses/cse599d/06wi/`

Real quantum computers?

Will quantum computers ever be built that outperform the world's largest classical computers? Here are some impressions from talks given at the March, 2016, meeting of the American Physical Society:

- There has been progress in building physical qubits and quantum gates, using superconducting devices with Josephson junctions, trapped ions, single-atom defects in semiconductors, electromagnetic modes of cavities, and photons in free space or waveguides.

  Less developed at this point are qubits made from topological excitations of fractional quantum hall states or unconventional superconductors.

- A computer is commercially available from D-Wave Systems that uses "quantum annealing" (QA), not discussed in these handouts. It's not yet clear if QA can be scaled up usefully.[5]

  For details on the D-Wave computer look at Fig. 10 in:

  James King et al. Benchmarking a quantum annealing processor with the time-to-target metric. 2015. URL `http://arxiv.org/abs/1508.05087`

- Current physical qubits and gates are nearly good enough for large-scale calculations, but error correction requires many more physical qubits than the "logical" qubits in the algorithm – probably about 10^6 physical qubits for a world-beating computation. Building and interconnecting this many qubits is still a ways off.

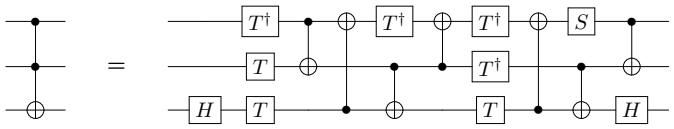
  [5] Troels F. Rønnow et al. Defining and detecting quantum speedup. *Science*, 345:420–424, 2014

- "Adaptive quantum computing" uses a quantum computer in a feedback loop with a classical computer to solve problems in condensed-matter physics and quantum chemistry.[6] The quantum circuits use relatively few gates and do not need error correction. A computer with 50 - 100 physical qubits might be competitive with the largest classical computers for this type of problem.

  [6] Bela Bauer, Dave Wecker, Andrew J. Millil, Matthew B. Hastings, and M. Troyer. Hybrid quantum-classical approach to correlated materials. 2015. URL `https://arxiv.org/abs/1510.03859`

Unlike classical computers, the algorithms that can be carried out by a quantum computer are limited. Shor's algorithm is a convenient

(a)



(b)

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(c)

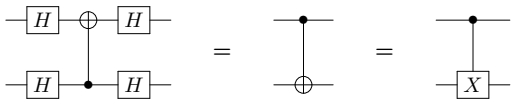


Figure 11: (a) The symbol for the three-qubit Toffoli gate, and an equivalent circuit made from one- and two-qubit gates. The T and S gates are phase-shift gates given by Eq. 16. Specifically $\hat{S} = \hat{R}_{\pi/2}$, $\hat{T} = \hat{R}_{\pi/4}$ and $\hat{T}^\dagger = \hat{R}_{-\pi/4}$ is the adjoint of $\hat{T}$. This decomposition of the Toffoli gate is from P. Selinger, Phys. Rev. A **87**, 042302 (2013).

(b) Matrix representation of the Toffoli gate. The Toffoli gate inverts its target qubit if and only if *both* control qubits are 1. Note the similarity to the matrix for the CNOT, Eq. 30.

(c) Four Hadamards can be used to flip the direction of a CNOT. Also note that a CNOT gate is the same as a controlled-X (controlled bit flip) gate, see Eq. 43.

benchmark, but it may have little practical value breaking future encryption methods. However, quantum algorithms are being developed for other difficult problems such as machine learning.[7,8]

The necessity of error correction

Scalable computation requires error correction. The CPU chip in my 2012-era PC runs at 3 GHz and contains 100 million logic gates. In a day-long calculation (as on the last page of this handout) it completes 2×10^{22} gate operations. This calculation would fail if small errors in each operation were not removed before they accumulated.

What would a quantum computer of useful size look like? Fowler, et al.[9] describe using Shor's algorithm to factor a 2000-bit number. As in Table 5, $2L = 4000$ qubits are needed. The modular multiplication of Fig. 8 is carried out by three-qubit "Toffoli" gates, each constructed from 17 one- and two-qubit gates (Fig. 11). $40L^3$ Toffoli gates are needed, which require 5×10^{12} one- and two-qubit gates.

Assume that the probability for any one of these gates to give the wrong result is p. For the Shor's-algorithm calculation to succeed with 90% probability, we need $p < 0.1/(5 \times 10^{12}) = 2 \times 10^{-14}$, since an error in any one gate would spoil the calculation. Current physical quantum gates have $p \approx 10^{-2}$ - 10^{-3}, and future developments will reduce p further – but clearly some sort of error correction is needed if a quantum computer is ever to factor a 2000-bit number.

Exercise: Fowler, et al. estimated that a quantum computer built from superconducting qubits with error rate $p = 10^{-3}$ using the "surface code" to correct errors could factor a 2000-bit number in one

[7] Nathan Wiebe, Ashish Kapoor, and Krysta M. Svore. Quantum deep learning. 2015. URL `https://arxiv.org/abs/1412.3489`

[8] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Phys. Rev. Lett.*, 113:130503, 2014. Also online at `http://arxiv.org/abs/1307.0471`

"Scalable" means the size of the computation can be increased without prohibitive (e.g. exponential) increases in the amount or precision of the resources needed.

As in all classical computers the logic gates in this PC have nonlinear transfer functions that push the outputs closer to the values representing 0 and 1, preventing small errors from accumulating. This type of error correction in each physical gate is not possible in a quantum computer, because quantum gates are carried out by *linear* operations.

[9] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86:032324, 2012. Also online at `http://arxiv.org/abs/1208.0928`

These authors compute that with physical-gate error probability $p = 10^{-3}$ factoring a 2000-bit number would require 2×10^8 interconnected superconducting qubits, far more than can be constructed at present. For comparison, my 2012-era CPU chip has 4×10^8 interconnected transistors.

day. **(a)** Compute how big an integer can be factored by the world's largest supercomputer in one year. **(b)** Compute how long it would take this supercomputer to factor a 2000-bit integer. Assume that the number of classical-computer operations required to factor an L-bit number is $(1.5 \times 10^{-7}) \exp((64L/9)^{1/3}(\ln L)^{2/3})$, and that the supercomputer can carries out 3.4×10^{16} operations per second.

This estimate is for the best publicly known factoring algorithm, the "general number field sieve". Factoring the 768-bit number RSA-768 required 1.6×10^{20} operations which gives the prefactor. As of 2015 the largest publicly-disclosed supercomputer could do 34 petaFLOPS.

Programming project 9: Toffoli gate decomposition.

Show that the decomposition of the Toffoli gate of Fig. 11(a) is correct by having your code generate and multiply the 17 3-qubit (8×8) matrices for the gates on the right – you should get the Toffoli matrix in Fig. 11(b). Also show that the identity shown in Fig. 11(c) is correct by multiplying the 4×4 matrices for the four Hadamards and CNOT with qubit 2 controlling qubit 1—you should end up with the CNOT matrix of Eq. 30. We will use this identity several times below.

For Shor's algorithim it is assumed that every gate operation is carried out correctly and this leads to the requirement for a very low per-gate error probability p. There are other types of quantum algorithm, for example those used for "adaptive quantum computing" referenced above, that can generate useful results in the presence of errors provided p is not so large as to make the result completely random. This type of algorithm can be carried out with many fewer physical qubits, and thus may be used for the first quantum computations that exceed that capacity of classical supercomputers – if that ever happens.

Quantum errors, part 1: Inaccurate gates

The physical operations used for gates like $\hat{H}$ and $\hat{C}_{\rm NOT}$ will never be exactly as they should be theoretically (Eqs. 12, 30). After many gates are applied, these errors can add up eventually turning the result of the computation into nonsense. Consider three new one-qubit gates,

$$\hat{X} \leftrightarrow \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \hat{Y} \leftrightarrow \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \hat{Z} \leftrightarrow \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \tag{43}$$

which you recognize as the Pauli matrices. These matrices are unitary, so they can be used as quantum gates. Each of these gates makes a big change to a qubit it operates on:

They are also Hermitian. The eigenvalues of a matrix that is both Hermitian and unitary are all ± 1.

- $\hat{X}$ is called the bit-flip operator, because $\hat{X}|0\rangle = |1\rangle$ and $\hat{X}|1\rangle = |0\rangle$. Make sure you understand this by multiplying the matrix for $\hat{X}$ by the vectors representing $|0\rangle$ and $|1\rangle$.
- $\hat{Z}$ is called the phase-flip operator. It is equal to the phase-shift gate $\hat{R}_\theta$ with $\theta = \pi$.
- $\hat{Y}$ can be written in terms of the other two operators, $\hat{Y} = i\hat{X}\hat{Z}$. This means $\hat{Y}$ is equivalent to a phase flip followed by a bit flip.

The factor i in $\hat{Y}$ is an overall phase that is often omitted. Such factors multiply the final result of the quantum computation by a phase, and so have no effect on the measured qubit values.

To represent *small* perturbations to a qubit pick a small number $\epsilon \ll 1$ and define two more gates

$$\hat{E}_X = e^{-i\epsilon\hat{X}} \quad \leftrightarrow \quad \begin{bmatrix} \cos\epsilon & -i\sin\epsilon \\ -i\sin\epsilon & \cos\epsilon \end{bmatrix} \approx \begin{bmatrix} 1 & -i\epsilon \\ -i\epsilon & 1 \end{bmatrix}, \tag{44}$$

$$\hat{E}_Z = e^{-i\epsilon\hat{Z}} \quad \leftrightarrow \quad \begin{bmatrix} e^{-i\epsilon} & 0 \\ 0 & e^{i\epsilon} \end{bmatrix} \approx \begin{bmatrix} 1-i\epsilon & 0 \\ 0 & 1+i\epsilon \end{bmatrix}. \tag{45}$$

Optional exercise. Derive these equations.

Exercise. (a) Show by matrix multiplication that $\hat{E}_X$ and $\hat{E}_Y$ are unitary, and therefore can represent operations in a quantum computation (that is, show that $\hat{E}_X^\dagger \hat{E}_X = \hat{I}$, etc.). **(b)** If the initial state $|\psi\rangle$ of a qubit is $|0\rangle$ or $|1\rangle$, show that the probability of finding the modified state $\hat{E}_X|\psi\rangle$ in the opposite (flipped) state is $p = (\sin\epsilon)^2$. **(c)** If again the initial state $|\psi\rangle$ is $|0\rangle$ or $|1\rangle$, show that the probability of finding the modified state $\hat{E}_Z|\psi\rangle$ in the opposite state is 0.

Recall that the probability of finding a system known to be in state $|a\rangle$ in some other state $|b\rangle$ is computed as $|\langle b|a\rangle|^2$, and that the states $|0\rangle$, $|1\rangle$ are written in our usual basis as in Eq. 13.

The error operators $\hat{E}_X$ and $\hat{E}_Y$ are close to $\hat{I}$ (assuming $\epsilon \ll 1$) so they should have only a small effect on a qubit. Specifically, applying $\hat{E}_X$ to a qubit flips it between $|0\rangle$ and $|1\rangle$ with the small probability $p = (\sin\epsilon)^2$, while applying $\hat{E}_Z$ has a different effect (a small phase shift) that cannot be represented by a bit-flip.

Programming project 10: Gate errors

Figure 12(a) shows a quantum computation you have simulated before (Fig. 3(c)): preparation in the state $|0\rangle$, two Hadamards which leave the qubit back in the state $|0\rangle$, followed by measurement which should therefore always give the result 0.

The double line after the measurement symbol denotes a *classical wire* carrying one bit of classical information, the result of the measurement.

For this project you should simulate the computation of Fig. 12(a) and also Fig. 12(b-f), which show the same computation with various gates representing errors added. By simulate we mean run the calculations many times, keeping track of the frequencies of the results. For these simulations you should use a not-to-small ϵ so $p = (\sin\epsilon)^2$ is in the range 5-10%. Here's what should happen:

As usual, you could simply calculate the quantum state up to the point just before the measurement and use this quantum state to predict what the probabilities of the various outcomes will be, rather than simulating the measurement process using random numbers. However, simulating the measurement here is a good warm-up for the more complicated partial measurements you will be doing later.

- If the error operator $\hat{E}_X$ is placed at location e_1, right after initializing the qubit to $|0\rangle$ (Fig. 12(b)), the result of the computation should be erroneous (1 rather than 0) with probability p.
- If instead the full bit-flip operator $\hat{X}$ is placed at e_1 (Fig. 12(c)), the result should be erroneous 100% of the time. You should get the same results if you put a $\hat{E}_X$ or $\hat{X}$ at location e_3, just before the measurement. **Putting the small-flip operator $\hat{E}_X$ at some point in the circuit has the same effect on the results as *probabilistically* (with probability p) putting the full-flip operator $\hat{X}$ at the same point in the circuit.** In either case, the result of the calculation is erroneous with probability p.
- Putting a *phase*-flip error $\hat{E}_Z$ or $\hat{Z}$ at e_1 or e_3 will have no effect, and similarly putting a bit-flip error between the two Hadamards at e_2 will have no effect. However, putting at phase-flip error at e_2 will cause computation errors. Simulate the computations of Fig. 12(d)-(e) to show this.

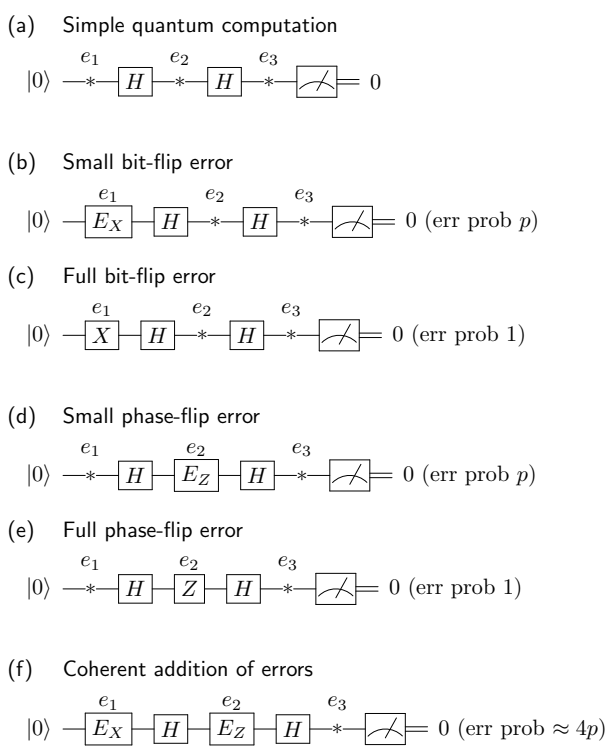


Figure 12: (a) A simple quantum computation, which should always give the result 0 in the absence of errors. The points marked e_j are places where error gates can be added to represent inaccuracies in the state preparation, the quantum wires, the Hadamard gates, or the state measurement. The double line to the right of the meter stands for a classical bit.

(b-f) Some specific things to try (see text). In these diagrams gates representing small or full spin-flip or phase-flip errors have been added to diagram (a) at one or more of the points e_j. As a result the measured output will be erroneous (not equal to 0) with some probability.

Here $p = (\sin\epsilon)^2$ where ϵ is the number used to calculate $\hat{E}_X$ and $\hat{E}_Z$.

- If we put two error gates in the computation with no intervening measurement, the errors can add *coherently*, which is usually bad. This means the gate errors ϵ add, rather than the probabilities p. Simulate the computation of Fig. 12(f) to see this happening.

Although it is not used in this handout, the best way to describe the effects of errors on qubits is to use the *density operator* (or density matrix) $\hat{\rho}$. The density operator can describe both the pure quantum states we have been using, and probabilistic mixtures of pure states which are called mixed states. Each quantum error process is described by "Kraus operators" that transform the density operator.

The density operator is covered in Townsend, *A Modern Approach To Quantum Mechanics, Second Ed.* Sec. 5.7, but it is not covered in Griffiths, *Introduction to Quantum Mechanics, Second Ed.*. Most introductions to quantum computing cover the density operator since it is so useful for describing quantum error processes.

Quantum errors, part 2: Interactions with the environment

By the *environment* we mean everything in the universe apart from the qubits. Classically, the environment can affect data, for example environmental electromagnetic noise can induce voltages on wires carrying logic signals turning 0's into 1's and vice-versa.

Since a CNOT is the same as a controlled bit-flip gate (Fig. 11(c)), this process can be represented by letting a qubit representing the environment conditionally flip a computational qubit as in Fig. 13(a). Such environmentally-induced bit flips result in computational er-

Modeling the entire environment as a single qubit is a drastic simplification, but it captures some key effects.

rors, just as they would classically. **We could get the same effect on the quantum computation by omitting the environment altogether, and *probabilistically* inserting a bit-flip $\hat{X}$ gate at e_1 as in Fig. 12(c).**

In the density-matrix formalism this is described by tracing over the environment variables, leaving the the reduced density matrix for the computational qubit in a mixed state with probability p of being flipped.

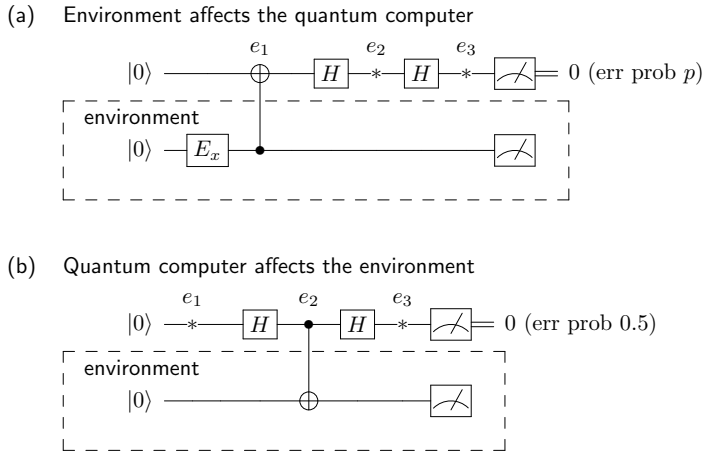


Figure 13: Modeling interactions with the environment, which is represented by the additional qubit in the dashed box.

(a) A bit flip error operator $\hat{E}_X$ is used to flip the environment qubit from $|0\rangle$ to $|1\rangle$ with probability p. Then a CNOT is used to flip the computational qubit depending upon the state of the environment qubit. As a result, the computation has probability p of giving the wrong result.

(b) The state of the computational qubit conditionally flips the environment qubit, using a CNOT going the other way. Classically this would have no effect on the computational qubit, but here it results in a large error probability (50%).

In Fig. 13(b), the direction of influence is reversed so that the *environment qubit* is flipped depending on the state of the computational qubit. A classical CNOT gate would only affect its target (the environment), but that's not how quantum controlled gates work – recall Fig. 6(d). Allowing the computational qubit to control an environmental degree of freedom causes a huge error rate (50%, so the result of the computation is completely random).

Comparing with Fig. 12(e), we see that **allowing the computational qubit to control an environmental degree of freedom as in Fig. 13(b) is equivalent to probabilistically inserting a *phase*-flip error $\hat{Z}$ in the quantum circuit at point e_2.**

Programming project 11: The environment

Simulate the two computations shown in Fig. 13. Then try moving the points in the computation where the environmental interaction occurs and see if you can understand the results.

Together, the environmental effects shown in Figs. 13(a) and (b) are called "decoherence". It may seem that the environmental influence on the computation of Fig. 13(a) is a purely classical effect, distinct from the odd quantum effect that influencing the environment as in Figs 13(b) scrambles the quantum phase. This is false dichotomy, since you have shown that the direction of a CNOT can be reversed by surrounding it with Hadamards (Fig. 11(c)). An important take-away is that **quantum information (including errors) moves in both directions through a controlled gate between qubits.**

It is difficult to protect qubits from decoherence. For example, the nuclei of most atoms have spin angular momentum. Due to the weakness of their interactions, the effect of nuclear spins on today's classical computers is completely negligible. Conversely, if interactions with the qubits change the state of a single nuclear spin, a quantum calculation will be ruined.

Quantum errors: Summary

- Various quantum computing errors can be modeled by randomly, with probability p, inserting $\hat{X}$ (bit-flip), $\hat{Z}$ (phase-flip) or $\hat{Y} = i\hat{X}\hat{Z}$ (bit+phase-flip) operators at each point in the quantum circuit diagram. This model covers both errors in the quantum gates and interactions with a simple model of the environment.
- *Measurement* of the error plays a key role in this error model. Error processes like $\hat{E}_{X,Y,Z}$ depend on the continuous variable ϵ, and it requires a measurement in the appropriate basis to collapse the quantum state into one of four discrete possibilities: $\hat{I}$ (no error), $\hat{X}$ (bit flip), $\hat{Z}$ (phase-flip) or $\hat{Y}$ (bit+phase-flip).
- If multiple errors occur with no intervening measurement, as in Fig. 12(f), it is possible for the errors to add coherently rather than probabilistically, and this is usually bad.
- Some types of quantum error are not covered by this error model:
  1. "Loss" errors in which the qubit leaves the quantum computer. This can happen when photons are used as the physical qubits.
  2. "Leakage" errors in which the qubit transitions to a state other than $|0\rangle$ or $|1\rangle$. This can happen with trapped ions used as qubits, since the ions have more than two atomic levels.
  3. Errors in which a single environmental variable interacts coherently with more than one qubit, or with the same qubit at separate times in the quantum computation.

  Although they are not covered in this handout, there are techniques for dealing with loss and leakage errors which are similar to the techniques discussed here. Conversely, multi-qubit errors could be quite difficult to characterize and deal with.

This model is called the "depolarizing channel" for quantum errors. It is attractive because it is relatively straightforward to analyze and it can be simulated for systems of many qubits as discussed in the last section of this handout. However, there is an active discussion in the current literature about its sufficiency to capture the true nature of quantum error processes, and how to analyze and simulate more general error models. See, for example:

Yu Tomita and Krysta M. Svore. Low-distance surface codes under realistic quantum noise. *Phys. Rev. A*, 90:062320, 2014. Also online at `https://arxiv.org/abs/1404.3747`; and P. Jouzdani et al. Fidelity threshold of the surface code beyond single-qubit error models. *Physical Review A*, 90:042315, 2014. Also online at `https://arxiv.org/abs/1401.6540`

Detecting bit-flip errors: The repetition code

Now that we have a model for quantum errors (randomly inserting $\hat{X}$, $\hat{Y}$, or $\hat{Z}$ operators into the quantum circuit), we need a strategy to prevent them from ruining the computation.

A common strategy with classical information is to include additional bits, which together with the data bits are used to detect and correct errors. One scheme is to represent each "logical" bit by three physical bits, which are 000 if the logical bit is 0 and 111 if the logical bit is 1. If a physical bit is flipped by an error this can be detected and corrected using a majority rule, e.g. $010 \rightarrow 000$.

Information sent to and from cell phones, laptops, televisions, etc. always includes more bits than the data itself, so errors can be corrected.

A logical bit is one of the data bits we are trying to compute with or transmit without errors. A physical bit is one of the actual physical variables stored or transmitted.

This same "repetition code" can be implemented for qubits, representing each logical qubit (shown with subscript L) by three physical

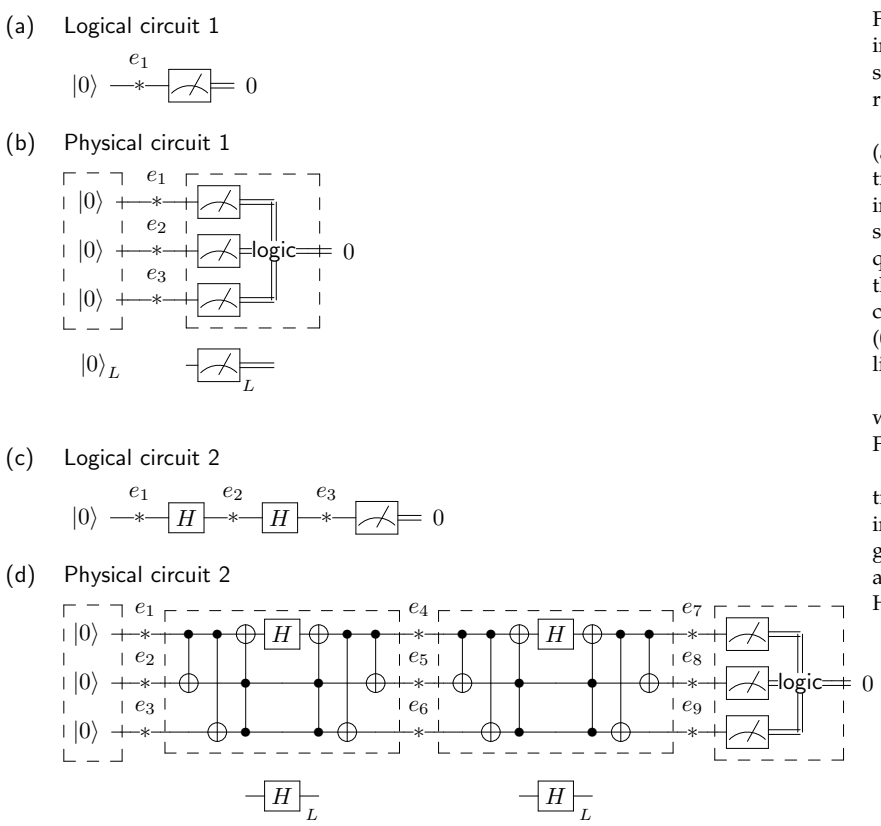


Figure 14: (a) Logical computation in which a qubit is initialized to the state $|0\rangle$ and then measured, giving the result 0.

(b) Implementation of the circuit of (a) using the repetition code. Initialization of the state $|0\rangle_L$ is accomplished by initializing three physical qubits to the state $|000\rangle$. Measurement of the logical qubit is accomplished by measuring the three physical qubits, then applying classical logic to compute which result (0 or 1) is in the majority. (Recall double lines represent classical bits, not qubits).

(c) This logical computation with two Hadamards is the same as Fig. 12(a).

(d) To implement this computation in the repetition code we have introduced a block of seven physical gates (four CNOTs, two Toffolis and a Hadamard) to carry out the logical Hadamard $\hat{H}_L$.

qubits (shown without subscript):

$$|0\rangle_L = |000\rangle, \quad |1\rangle_L = |111\rangle. \tag{46}$$

Then, for example, if a bit-flip operator is applied to second physical qubit, $\hat{X}^{(2)}|000\rangle = |010\rangle$, the result $|010\rangle$ must be recognized as a logical $|0\rangle_L$, or even better corrected back to $|000\rangle$.

This is the same notation used in the earlier handouts: A superscript in parentheses on a gate operator indicates which qubit the gate is applied to. Thus $\hat{X}^{(2)}$ flips qubit 2.

One way to correct errors is to measure the three physical qubits, then apply classical logic to compute whether 0 or 1 is in the majority. Figure 14 shows this scheme, for two different quantum computations. For the second computation, a block of seven physical gates is used to carry out a logical Hadamard gate on the encoded qubit.

Optional exercise. Show that this block of seven physical gates does indeed carry out a logical Hadamard. First, show that the block operates correctly on the two logical basis states, i.e. that $\hat{H}_L|0\rangle_L = (|0\rangle_L + |1\rangle_L)/\sqrt{2}$, and $\hat{H}_L|1\rangle_L = (|0\rangle_L - |1\rangle_L)/\sqrt{2}$. Then, if you like, also show that $\hat{H}_L$ operates correctly on input states with a bit-flip error. For example, $\hat{H}_L|001\rangle = (|001\rangle + |110\rangle)/\sqrt{2}$ (both input and output have a bit-flip error on qubit 3).

Programming project 12: Repetition code

Simulate the physical circuits of Figs. 14(b) and (d), and run them both with and without added errors (some suggestions are below). Here are some things to try:

- You should be able to put a *single* bit-flip error $\hat{X}$ at any of the error points $e_1 \ldots e_3$ in (b) or $e_1 \ldots e_9$ in (d), and the final result of the calculation (output of the majority logic) should always be correct. This means the error correction works!
- There are many places where you can put *two* bit-flip errors (e_1 and e_6 in (d), for example) which will send two incorrect bits to the majority logic, causing the computation to fail. **The repetition code can correct one error, but not two.**
- If you put a single *phase*-flip error $\hat{Z}$ at a point like e_5 in (d), the computation will fail. **The repetition code corrects bit-flip errors, but it does not correct phase-flip errors.**
- Finally, there are points inside the $\hat{H}_L$ blocks where a *single* bit-flip error will cause the computation to fail. Show this by putting an $\hat{X}$ inside one of the $\hat{H}_L$ blocks, e.g. in the first $\hat{H}_L$ on qubit 1 after the first CNOT. This means this implementation of $\hat{H}_L$ is not "fault tolerant" – a bit-flip error at these points can propagate from qubit 1 to other qubit(s) over the CNOTs, resulting in two or more bit-flip errors (which cannot be corrected).

Full quantum error correction codes

In 1995 Shor[10] described a quantum error code capable of detecting and correcting both bit-flip and phase-flip errors, using nine physical qubits to represent each logical qubit. Quantum error codes are often described by the triplet $[[n, k, d]]$ where n physical qubits are used to encode k logical qubits, and d is the "distance" of the code – the minimum number of errors needed to convert one value for the encoded qubits into a different value. Up to $(d-1)/2$ errors can be corrected, since this number of errors takes us less than halfway to the closest erroneous logical value. The Shor code is a $[[9, 1, 3]]$ quantum code.

Here we will look at the **seven-qubit Steane code**. The Steane code uses seven physical qubits to encode one logical qubit and can correct up to one error, so it is a $[[7, 1, 3]]$ code. As in Eq. 46, the two logical qubit states are represented by certain states of the seven

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

The shortest block that encodes one logical qubit and can correct any single $\hat{X}$, $\hat{Y}$ or $\hat{Z}$ error requires at least five physical qubits, and such a $[[5, 1, 3]]$ code exists. In an n-qubit code for one logical qubit there must be $3n + 1$ distinct syndromes: three possible errors on each of the physical qubits, or no error at all. For each of these syndromes, there must be two different logical states $|0\rangle_L$, $|1\rangle_L$. With n qubits there are 2^n independent states, so we need $2(3n + 1) \leq 2^n$ which gives $n \geq 5$.

physical qubits:

$$\begin{aligned}|0\rangle_L &= \tfrac{1}{\sqrt{8}}(|0000000\rangle + |1010101\rangle + |0110011\rangle + |1100110\rangle \\ &\quad +|0001111\rangle + |1011010\rangle + |0111100\rangle + |1101001\rangle), \\ |1\rangle_L &= \tfrac{1}{\sqrt{8}}(|1111111\rangle + |0101010\rangle + |1001100\rangle + |0011001\rangle \\ &\quad +|1110000\rangle + |0100101\rangle + |1000011\rangle + |0010110\rangle). \quad (47)\end{aligned}$$

Where do these come from? The Steane code is derived from a famous classical error code, the $[7,4,3]$ Hamming code, using the CSS construction to derive a $[[7,1,3]]$ quantum error code. See Nielsen and Chuang Sec. 10.4 if you'd like to learn more about this.

Figure 15 shows preparation of $|0\rangle_L$, construction of logical $\hat{H}_L$ and $\hat{Z}_L$ gates, and measurement of a logical qubit for the Steane code. The $|0\rangle_L$ block using three Hadamards and eight CNOTs is a bit mysterious, but the logical gates are particularly simple in the Steane code: To build a $\hat{H}_L$, you simply put an $\hat{H}$ on each of the seven physical qubits. The same construction works for a number of other gates, $\hat{X}$, $\hat{Z}$ and $\hat{S}$ (which however gives $\hat{S}_L^\dagger$), but not $\hat{T}$.

I copied the $|0\rangle_L$ circuit from:

Hayato Goto. Minimizing resource overheads for fault-tolerant preparation of encoded states of the Steane code. *Sci. Rep.*, 5:19578, 2016

This "transversal" construction of logical gates in the Steane code even works for CNOTs: To build a logical CNOT between two seven-bit code blocks you simply put seven physical CNOTs from qubit 1 of one block to qubit 1 of the other block, etc. As a result, it is easier to make quantum computation fault-tolerant with the Steane code than with some other codes.

$\hat{S}$ and $\hat{T}$ were introduced in Fig. 11. As discussed towards the end of this handout $\hat{T}$ has some magic properties not shared by the other gates.

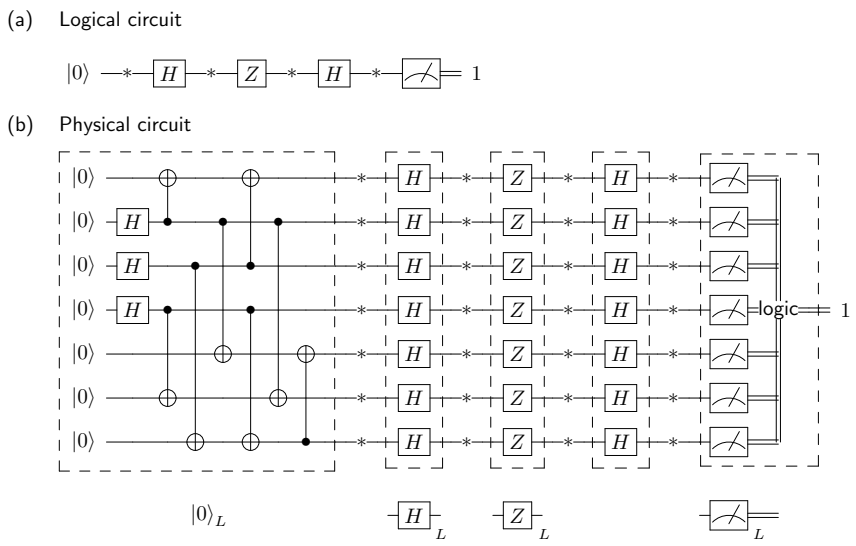


Figure 15: (a) A quantum computation like Fig. 3(d). Due to the $\hat{Z} = \hat{R}_\pi$ gate the qubit is flipped so the result is 1.
(b) Physical implementation of the computation using the Steane code.

Programming project 13: The Steane code

The quantum circuit shown in Fig. 15 is similar to Fig. 14(d) except that it uses the seven-qubit Steane code rather than the three-qubit repetition code, and a Z gate has been added. As you did with the repetition code, simulate Fig. 15(b) with and without errors at some of the marked error points. Here's what you should find:

- Now you can put a single $\hat{X}$, $\hat{Y}$ or $\hat{Z}$ error gate at any one of the marked error points without causing the calculation to fail. This is different from the repetition code, which could only handle $\hat{X}$ (bit-flip) errors. **The Steane code can correct any one-qubit error.**

This means any of $\hat{I}, \hat{X}, \hat{Y}, \hat{Z}$ and therefore also any linear combination of these errors. More generally, the Steane code can correct any transformation to the density matrix for a qubit, including putting the qubit into a completely unknown quantum state, described by the density operator $\hat{\rho} = \hat{I}/2$.

- There are many pairs of points where you can put *two* errors that will cause the computation to fail. **The Steane code can correct one error, but not two.**
- Unlike the repetition-code $\hat{H}_L$ blocks in Fig. 14, there are no points in the $\hat{H}_L$ blocks in Fig. 15 where a single error will propagate into multiple errors and cause the computation to fail. **The Steane code is naturally fault-tolerant for some types of gate.**

There are such points inside the $|0\rangle_L$ block, so this is not a fault-tolerant construction for $|0\rangle_L$. The referenced article by Goto shows how a fault-tolerant Steane-code $|0\rangle_L$ can be constructed.

For this project you need the logic in Fig. 15(b) that converts the seven physical measurements into a logical output:

Call the measured values for the seven qubits $b_1 \ldots b_7$, each 0 or 1.

Compute $S_4 = 1(0)$ if an odd(even) number of b_4, b_5, b_6, b_7 are 1.

Compute $S_5 = 1(0)$ if an odd(even) number of b_2, b_3, b_6, b_7 are 1.

Compute $S_6 = 1(0)$ if an odd(even) number of b_1, b_3, b_5, b_7 are 1.

Compute $Z_L = 1(0)$ if an odd(even) number of $b_1, \ldots b_7$ are 1.

If any of $S_4, S_5, S_6 = 1$, the output is NOT Z_L; otherwise it is Z_L.

NOT $Z_L = 1$ if $Z_L = 0$ and vice versa.

We will see that S_4, S_5, S_6 are part of the "syndrome" for the Steane code, and that Z_L measures $\hat{Z}$ for the encoded logical qubit.

Quantum error correction: Ancilla qubits, syndromes, and all that

The error correction in Figs. 14 and 15 is done by classical logic after the qubits are measured. Unfortunately this is useless for a long computation, since we can't measure the qubits without destroying the coherence needed for quantum computations. A key insight for quantum error correction is that it is possible to measure if an error has occurred without measuring the state of the logical qubits. For the repetition code, rather than measuring the three physical qubits as in Fig. 14, instead we measure the following two quantities:

$$\begin{aligned} S_1 = &\quad 0 \quad \text{if qubit 1 and qubit 2 are the same, 1 otherwise,} \\ S_2 = &\quad 0 \quad \text{if qubit 1 and qubit 3 are the same, 1 otherwise.} \end{aligned} \tag{48}$$

"Qubit 1 and qubit 2 are the same" means that their quantum state is $|00\rangle$, $|11\rangle$, or a superposition of these two states. Note that this does not tell us the state of either qubit.

Fig. 16(a) shows how S_1 and S_2 can be measured, without measuring physical qubits 1-3 which are used to encode the logical qubit. We need two additional qubits initialized to $|0\rangle$, called "ancilla qubits".

Exercise. Show that $S_{1,2}$ measured as in Fig. 16(a) agree with Eq. 48. Assume there may be bit-flip errors at $e_1 \ldots e_3$ so qubits 1-3 can be in any of the basis states $|000\rangle \ldots |111\rangle$.

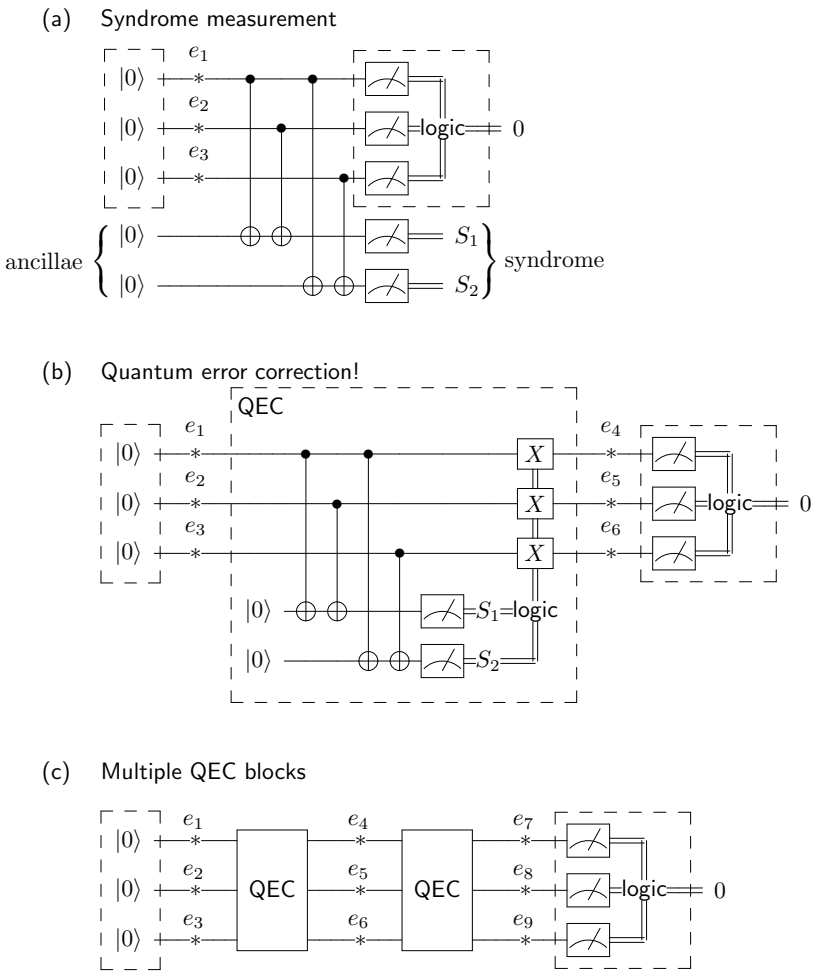


Figure 16: (a) Quantum circuit to measure the error syndrome for the repetition code. The syndrome variables $S_{1,2}$ measure which qubit (if any) has experienced a bit-flip error, but do not measure the logical qubit encoded in physical qubits 1-3.

(b) The results of the syndrome measurement are used to classically control the application of a quantum bit-flip ($\hat{X}$) gate to zero or one of the qubits, to undo any detected bit-flip error. Unlike Figs 14 and 15, this is true *quantum* error correction (although only for bit-flip errors), as the encoded quantum state of the logical qubit is not measured by the QEC block.

(c) Multiple QEC blocks can be included in a computation, and up to one error before each QEC block will be successfully corrected by the blocks.

Taken together the variables $S_{1,2}$ are called the "syndrome". The syndrome tells us what error if any has occurred in the physical qubits, and therefore what we need to do to undo the error:

S_1	S_2	Error	Action to correct error
0	0	none	none
0	1	qubit 3 bit-flip	$\hat{X}$ on qubit 3
1	0	qubit 2 bit-flip	$\hat{X}$ on qubit 2
1	1	qubit 1 bit-flip	$\hat{X}$ on qubit 1

In Fig. 16(b) the results of the syndrome measurement are used to classically control conditional $\hat{X}$ gates on the physical data qubits to carry out any needed correction. **Finally, we have a prescription for true *quantum error correction* (QEC) that can be applied many times as the quantum computation proceeds, without coherence-destroying measurement of the logical qubits.**

In Fig. 16(a) physical qubits 1-3 are also measured, but this is separate from the measurement of $S_{1,2}$ and need not be done at the same time, see Fig. 16(b).

Quantum mechanically the measurement of S_1, S_2 in Fig. 16 must have an effect on the three data qubits. In fact, by projecting the quantum state of the data qubits onto definite values of S_1, S_2 the measurement alone reduces the accumulation of errors, even if no corrective action is taken. If the syndrome could be measured repetitively at a sufficiently rapid rate (without introducing more errors), no errors would occur – an example of the "quantum Zeno effect".

Programming project 14: Quantum error correction!

(a) Simulate the syndrome measurement of Fig. 16(a). Insert zero, one, or more bit-flip ($\hat{X}$) errors at some of the locations $e_{1,2,3}$ and check that the measured syndrome $S_{1,2}$ agrees with the table above.

(b) Simulate the quantum error correction of Fig. 16(b). Check that a single bit-flip error inserted at one of $e_{1,2,3}$ is corrected by the QEC block. As a consequence, you should be able to put a single bit-flip error at at one of $e_{1,2,3}$ and a second bit-flip error at at one of $e_{4,5,6}$ and still get the correct result, 0, for the quantum computation. Finally, put two successive QEC blocks in the computation as in Fig. 16(c). The computation should succeed with single bit-flip errors at any one of $e_{1,2,3}$, any one of $e_{4,5,6}$, *and* any one of $e_{7,8,9}$.

The QEC block of Fig. 16(b) is not fault tolerant, as you may want to show in your simulation: A single error can spread to multiple qubits along the CNOTs. The block can be made fault-tolerant, but only at the expense of greater complexity, see Fig. 17(d).

Part (b) presents a new programming challenge: measuring some of the qubits while leaving the remaining qubits undisturbed. Consider measuring qubit 4 in Fig. 16(b), which gives S_1. An ideal measurement in quantum mechanics "collapses" the quantum state so that it agrees with the measured value. This means that measuring S_1 *projects* the full 2^N dimensional state space onto the 2^{N-1} dimensional subspace in which S_1 has the measured value. While projecting onto a lower-dimensional subspace may seem a bit abstract, it is at the heart of quantum error correction – this is the process that "digitizes" whatever inaccuracy has crept into the quantum state into one of two discrete outcomes: Either S_1 is measured to be 0, in which case the projection *eliminates the error measured by S_1 from the quantum state*, or S_1 is measured to be 1, in which case the projection *modifies the quantum state to a state in which this error has definitely occurred.* The programming hints in the Appendix explain how to carry this out, and also how to initialize the ancillae to $|0\rangle$.

Here is an interesting way to think of the QEC in Fig. 16(b): The unpredictable errors at points e_j "heat up" the physical qubits (add entropy to them) by reducing our knowledge of which quantum state they are in. The QEC block is a sort of refrigerator that cools the qubits back down. By the second law the added entropy cannot be destroyed, so as with any refrigerator the QEC moves the undesired entropy somewhere else, in this case into the ancilla bits (which start in a definite state, i.e. zero entropy, but end up with probabilities to be in various states.) Just as an air conditioner needs to be in a window so it can dump the entropy removed from a room outside, a functioning quantum computer will need to continually remove the entropy from the ancillae to reset them to $|0\rangle$ and dump this entropy somewhere else.

The stabilizers for a quantum error code

To carry out quantum error correction for the Steane code, we need to measure a syndrome that indicates which error has occurred, if any (like $S_{1,2}$ for the repetition code). This can be done using the "stabilizer" formalism, introduced by Daniel Gottesman in 1997.

Nielsen and Chuang Sec. 10.5 covers the stabilizer formalism.

We start by writing down without explanation the stabilizers and logical operators X_L, Z_L first for the three-qubit repetition code:

S_1	ZZI
S_2	ZIZ
X_L	XXX
Z_L	ZII

and then for the seven-qubit Steane code:

S_1	$IIIXXXX$
S_2	$IXXIIXX$
S_3	$XIXIXIX$
S_4	$IIIZZZZ$
S_5	$IZZIIZZ$
S_6	$ZIZIZIZ$
X_L	$XXXXXXX$
Z_L	$ZZZZZZZ$

Note that the stabilizers $S_{1,2}$ for the repetition code correspond to the syndrome variables for this code. Each line in these tables is an operator on the physical qubits encoding a logical qubit, written in a shorthand form. For example, for the repetition code the stabilizer $S_2 = ZIZ$ is more formally written

$$\hat{S}_2 = \hat{Z} \otimes \hat{I} \otimes \hat{Z} \tag{49}$$

We could also write $\hat{S}_2 = \hat{Z}^{(1)}\hat{Z}^{(3)}$ in our usual notation $\hat{Z}^{(1)} = \hat{Z} \otimes \hat{I} \otimes \hat{I}$, etc. (Eq. 18).

which means apply $\hat{Z}$ to qubit 1, $\hat{I}$ to qubit 2, and $\hat{Z}$ to qubit 3. Thus

$$\begin{aligned} \hat{S}_2|111\rangle &= (-1 \times 1 \times -1)|111\rangle = |111\rangle, \\ \hat{S}_2|011\rangle &= (1 \times 1 \times -1)|011\rangle = -|011\rangle \end{aligned} \tag{50}$$

since $\hat{Z}|0\rangle = |0\rangle$ and $\hat{Z}|1\rangle = -|1\rangle$. The syndrome variable S_2 measures whether or not qubits 1 and 3 are equal, and the stabilizer operator $\hat{S}_2$ measures exactly the same thing: a state like $|111\rangle$ in which qubits 1 and 3 are equal is an eigenvector of $\hat{S}_2$ with eigenvalue $+1$, while a state like $|011\rangle$ in which qubits 1 and 3 are unequal is an eigenvector of $\hat{S}_2$ with eigenvalue -1 (Eq. 50).

Here are some properties (presented without proof) of the stabilizers of a quantum error code:

- The eigenvalues of the stabilizers are all ± 1.
- The stabilizers all commute with each other, and with $\hat{X}_L$ and $\hat{Z}_L$ (more on these two below).
- By simultaneously measuring the full set of stabilizers (which is possible because they all commute with each other), we are measuring the syndrome for the code. In particular, a "no-error" state is a simultaneous $+1$ eigenvector of all of the stabilizers, while if the measured value (eigenvalue) of one or more of the stabilizers is -1 an error has been detected.
- $\hat{X}_L$ and $\hat{Z}_L$ are logical operators for the encoded qubit. These operators obey the same commutation and anticommutation relations as the one-qubit $\hat{X}$ and $\hat{Z}$ matrices, Eq. 43:

$$[\hat{Z}_L, \hat{X}_L] = 2i\hat{Y}_L \text{ where } \hat{Y}_L = i\hat{X}_L\hat{Z}_L, \text{ and } \{\hat{X}_L, \hat{Z}_L\} = 0. \tag{51}$$

To commute with each other, every X-type stabilizer or logical operator must have an *even* number of non-I qubits in common with every Z-type stabilizer or logical operator. For example, $S_3 = XIXIXIX$ and $S_4 = IIIZZZZ$ both have non-I operators at qubits 5 and 7, while S_3 and $Z_L = ZZZZZZZ$ both have non-I operators at qubits 1, 3, 5 and 7. Conversely, X_L and Z_L must have an *odd* number of non-I qubits in common.

We will use these properties later when discussing the surface code.

The commutator $[\hat{A}, \hat{B}] \equiv \hat{A}\hat{B} - \hat{B}\hat{A}$ and the anticommutator $\{\hat{A}, \hat{B}\} \equiv \hat{A}\hat{B} + \hat{B}\hat{A}$.

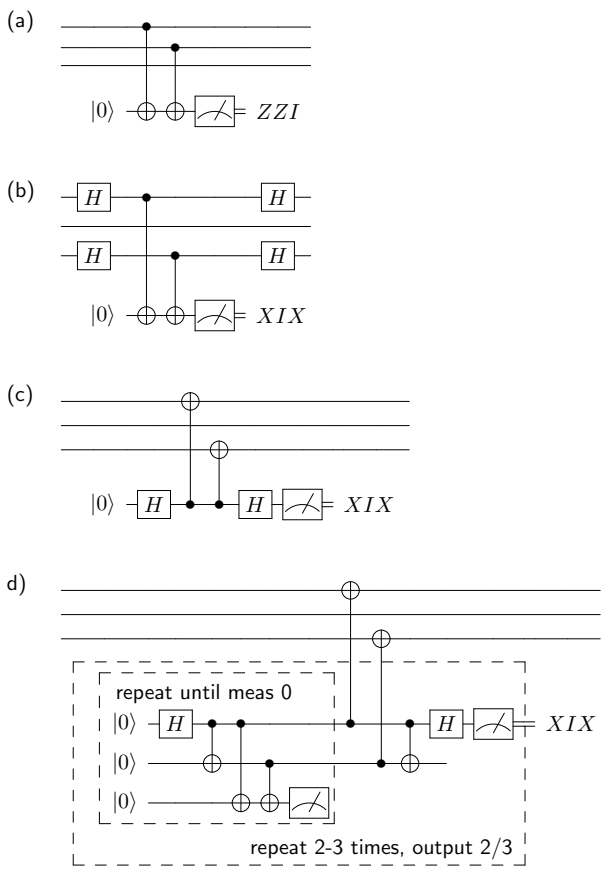


Figure 17: (a) Measuring a Z-type stabilizer. There is a CNOT from every qubit that has a Z in the stabilizer to the ancilla qubit, so the ancilla ends up flipped to $|1\rangle$ if an odd number of these qubits are $|1\rangle$.

(b) Measuring an X-type stabilizer. A Hadamard gate on each qubit to be measured rotates the qubit from the z basis to the x basis, then the measurement proceeds as in (a). Finally, a second Hadamard rotates the qubit back to its original basis.

(c) An alternate way to measure the X-type stabilizer. Here we have applied the identity of Fig. 11(c) to the circuit of (b) to flip the direction of the CNOTs, and used $\hat{H}^2 = \hat{I}$ to simplify.

(d) Fault-tolerant version of the circuit in (c). This circuit is designed so that the probability of introducing two or more errors is of order p^2, rather than p as it is for circuits (a)…(c). (p is the probability of error for a single gate, state preparation, or measurement.)

How do we measure the stabilizers? Fig. 17 shows the general procedure, and you can see this is exactly what was used to measure $S_1 = ZZI$ and $S_2 = ZIZ$ for the repetition code in Fig. 16. Using the circuits of Figs. 17(a) and (c) to measure the stabilizers of the Steane code, we end up with the QEC block shown in Fig. 18. If you like, you can program this QEC block and put one it into a quantum computation like Fig. 15 to see it in action. However, you may want to save your coding energy for the surface code, which is different in some interesting ways from the "block" codes we have been discussing so far. First we discuss an important theorem that shows that scalable quantum computation is possible at all.

Fault tolerance and the threshold theorem

Here we outline one of the most significant results in quantum computing, the *threshold theorem*. This theorem, proved around 1997 by several researchers, states that if we have a sufficient supply of qubits with error probability per gate less than some threshold value p_{th}, it

See Nielsen and Chuang Ch. 10 for the history of the threshold theorem and a more rigorous discussion.

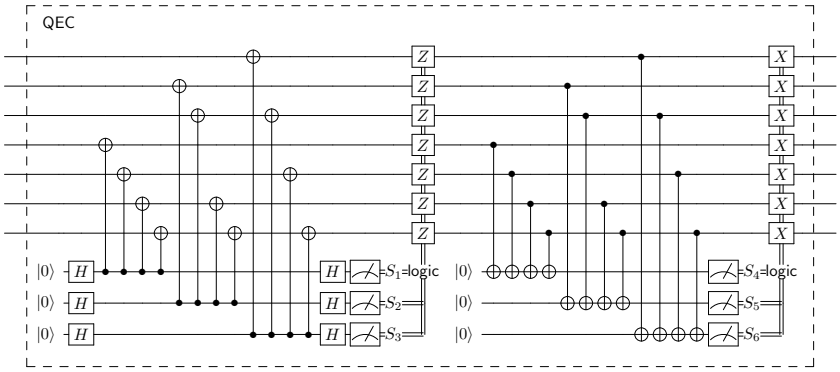


Figure 18: Quantum error correction block for the seven-bit Steane code. Like the QEC block in Fig. 16, this implementation is not fault tolerant: A single qubit error can use the CNOTs to spread to multiple qubits.

The classical logic used to decide when to apply Z and X correction gates is given in the Appendix.

is possible to carry out an arbitrarily large quantum computations.

Let's apply quantum error correction to some large quantum computation. As shown in Fig. 19 we encode each logical qubit in a block of seven physical qubits using the Steane code, and follow each logical gate block by a QEC block.

The QEC blocks must be fault-tolerant, unlike Fig. 18, meaning any single gate error in the QEC block results in at most one error on the seven output qubits. Let's estimate that this can be done with 100 gates in each QEC block, rather than 32 gates as in Fig. 18.

Specifically, a fault-tolerant method of measuring the syndrome like Fig. 17(d) must be used.

For the logical circuit of Fig. 19(a), an error means having *two or more errors* in the physical qubits encoding the logical qubit, since a single error can be corrected by a QEC block. At each point in the physical circuit of Fig. 19(b) let

$p_1 =$ probability of exactly one error in the seven physical qubits, and

$p_2 =$ probability of two or more errors in the seven physical qubits.

We assume that p, p_1 and p_2 are all much less than one, and compute each to lowest non-zero order in p.

Referring to Fig. 19(b), the probability of one error right after each QEC block is $p_1 = 100p$ because one-qubit errors at the input to the QEC block will be corrected, but there are 100 points inside the QEC block where a new error can occur. After one of the logical gate blocks, assumed to require seven physical gates, the probability of a single error increases to $p_1 = 107p$, but then it goes back down to $p = 100p$ after the following QEC block.

Some logical gates like the T gate require more than seven physical gates for fault-tolerant implementation in the Steane code, but we ignore this.

Since the QEC block cannot correct two-qubit errors, p_2 only increases as we move through the circuit. Starting as in Fig. 19(b) with $p_2 = p_{2a}$ at a point just after some QEC block, the probability to have two errors after the following Z gate is

$$p_{2b} = p_{2a} + 100p \cdot 7p + (7 \cdot 6/2)p^2 = p_{2a} + 721p^2. \qquad (52)$$

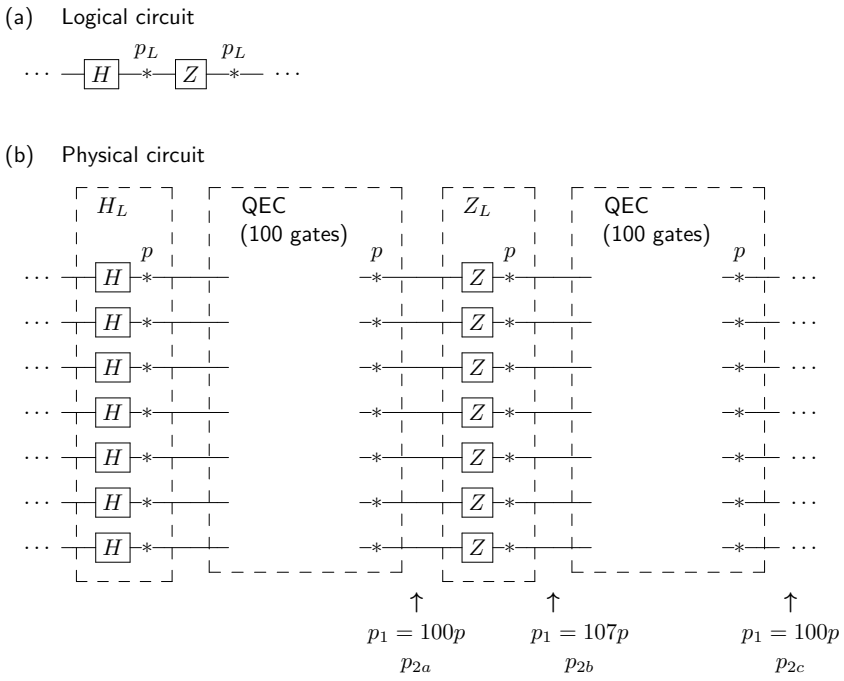


Figure 19: (a) Fragment of a large quantum computation (Hadamard gate followed by Z gate) at the logical level, with a probability p_L of having an error inserted after each logical gate. The logical error rate p_L must be calculated starting from the physical circuit of (b).

(b) Physical implementation of the computation fragment, with a QEC block after each logical gate block. It is assumed that there is probability p of inserting an error after each of the seven physical gates in each logical gate block, and after each of the 100 physical gates in the QEC bock.

In this equation p_{2a} is the probability of two errors coming into the Z gate, $100p \cdot 7p$ is the probability of of a new pair of errors consisting of a single error with $p_1 = 100p$ coming into the Z gate plus a single error in one of the seven physical gates in the gate block, and $(7 \cdot 6/2)p^2$ is the probability of a new pair of errors both in the gate block. Using exactly the same logic the probability of two errors after the following QEC block is

Since there are 7 possible error points in the gate block, there are $7 \cdot 6/2$ distinct possible pairs of error points.

$$p_{2c} = p_{2b} + 107p \cdot 100p + (100 \cdot 99/2)p^2 = p_{2b} + 15650p^2. \qquad (53)$$

Finally, we can compute the *logical* error probability p_L occurring after each logical gate in Fig. 19(a) as the total increase in p_2 over one cycle of a gate followed by a QEC block,

$$p_L = p_{2c} - p_{2a} = 16371p^2. \qquad (54)$$

The upshot of this analysis is that by encoding the qubits and using fault-tolerant gate and QEC blocks:

- We have gone from an error probability per physical gate p to an error probability per logical gate $p_L = cp^2$, with a rough estimate $c \approx 1.6 \times 10^4$.
- The number of physical qubits needed has increased by a factor of about 11, counting 7 to encode the logical qubit plus about 4 ancilla qubits (fault-tolerant version of Fig. 18) that must be reset to $|0\rangle$ after each logical gate operation.

- The total number of physical gate operations has increased by a factor of 107 (7 gates in each logical gate block plus 100 gates in the QEC block following each logical gate).

Is this good for anything? The Shor's algorithm calculation requires $p_L = 2 \times 10^{-14}$; using Eq. 54 we get $p = 1.1 \times 10^{-9}$. This is much larger than the unencoded error requirement p_L, but it still is probably unachievably small for physical qubits and gates.

A way out of this is to use **concatenation of error codes**. We take 11 *logical* qubits, each made from 11 physical qubits (7 to encode plus 4 ancillae), and use these to encode one *second-level* logical qubit. The error probability per second-level logical gate, following the same analysis that led to Eq. 54 will be

$$p_L^{(2)} = c p_L^2 \tag{55}$$

Here the superscript (2) is used to denote the second level of concatenation. With a third level we have $p_L^{(3)} = c(p_L^{(2)})^2$.

with the same $c \approx 1.6 \times 10^4$ as Eq. 54. Unfortunately, each second-level logical qubit requires $11^2 = 121$ physical qubits and each second-level logical gate block requires $107^2 = 1145$ physical gate operations, but that's the price of doing business.

Provided the lowest-level (physical) per-gate error probability meets the condition

$$p < p_{\text{th}} \quad \text{with} \quad p_{\text{th}} \equiv 1/c \tag{56}$$

each successive logical error probability $p, p_L, p_L^{(2)}, p_L^{(3)} \ldots$ is smaller than the previous (less concatenated) one. Thus we have the famous

> **Threshold theorem**: Provided the physical per-gate error probability p can be made less than some threshold value p_{th}, the logical per-gate error probability $p_L^{(n)}$ can be made as low as desired by using sufficiently many levels of concatenation n.

As usual we have omitted all the technical details. In particular, it is important to show that the physical resources required (numbers of physical qubits and gates) do not increase so fast with concatenation as to negate the advantages of quantum computation over classical computation.

Using the rough arguments above we estimate $p_{\text{th}} = 1/c \approx 6 \times 10^{-5}$ for the Steane code, which is similar (perhaps by luck) to the original estimates of p_{th} in the late 1990s. Remarkably the threshold theorem showed that scalable quantum computation should indeed be possible, but the required error rate $p < 10^{-4}$ seemed far from realizable in a real system.

Since that time, physically-achievable error rates have decreased (to perhaps 10^{-2} in 2016) while more detailed analysis and newer types of error code (like the surface code discussed below) have increased the theoretical error threshold (also to perhaps 10^{-2} in 2016). So at present it appears that the main obstacle, still formidable, to large-scale quantum computation is the fabrication and interconnection of the large numbers of physical qubits that will be required.

Not everyone agrees with this rosy assessment. If physical quantum computers have more complicated types of errors not accounted for by the error model used here, it may be much more difficult to make them work.

Exercise: Using the numbers estimated above (11 physical qubits per logical qubit, 107 physical gates per logical gate plus QEC block,

$c = 1.6 \times 10^4$), compute (i) how many levels of concatenation will be needed, (ii) how many physical (lowest-level) qubits will be required, and (iii) how many physical (lowest-level) gate operations will be required to carry out the example 2000-bit Shor's computation. Do this for two different values of the physical error rate p, (a) $p = 0.05 p_{\text{th}}$ and (b) $p = 0.5 p_{\text{th}}$.

The surface code

This section relies heavily on the following article, which you should consult for a more detailed and rigorous treatment:

Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86:032324, 2012. Also online at `http://arxiv.org/abs/1208.0928`

The $[[n, k, d]]$ block codes described above must be concatenated to achieve the low logical error rates required for large-scale quantum computation. Here we describe a non-concatenated quantum error code, the "surface code"[11,12] which has the following features:

- The physical qubits are laid out in a regular pattern on a two-dimensional surface (hence the name), and physical gates are only required between neighboring physical qubits.
- The code distance d ($\approx$ twice the number of physical qubit errors needed to cause a logical qubit error) can be freely chosen.
- The error threshold is rather large, $p_{\text{th}} \approx 6 \times 10^{-3}$.
- The surface code is a stabilizer code, as described above.
- The surface code has fascinating *topological* properties:
  - Logical qubits are formed by punching pairs of holes in the surface – the bigger and farther apart the holes, the bigger the code distance d.
  - Physical errors only lead to logical errors when they happen to string together into paths leading about halfway from one edge or hole to another, or about halfway around a hole.
  - A CNOT operation is accomplished by "braiding" the logical qubit holes – moving them around each other in paths that leave them back where they started.

[11] A. Yu Kitaev. Fault-tolerant quantum computation by anyons. *Annals Phys.*, 303:2–30, 2003. Originally online at `https://arxiv.org/abs/quant-ph/9707021` (1997)

[12] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *J. Math Phys.*, 43: 4452–4505, 2002. Also online at `http://arxiv.org/abs/quant-ph/0110143` The surface code is also called the "toric code", and was originally designed to occupy the surface of a torus rather than an open surface like a rectangle.

One can at least imagine how real large-scale quantum computer could be built using the surface code. Fowler, et al. discuss the physical parameters of such a computer fabricated from superconducting devices; other authors discuss large-scale surface code computers made from solid-state qubits[13] or trapped ions.[14] Thus, the surface code is a popular benchmark for estimating the practicability of large-scale computing with various types of qubits.

To see how a superconducting surface-code computer might be built, see

A. D. Córcoles et al. Demonstration of a quantum error detection code using a square lattice of four superconducting qubits. *Nature Comm.*, 6:6979, 2015

Fig. 20(a) shows a surface-code quantum computer with 49 qubits. The computer consists of a regular two-dimensional lattice made up of three types of qubits:

[13] Charles D. Hill et al. A surface code quantum computer in silicon. *Sci. Adv.*, 1:e1500707, 2015

[14] B. Lekitsch et al. Blueprint for a microwave ion trap quantum computer. 2015. URL `http://arxiv.org/abs/1508.00420`

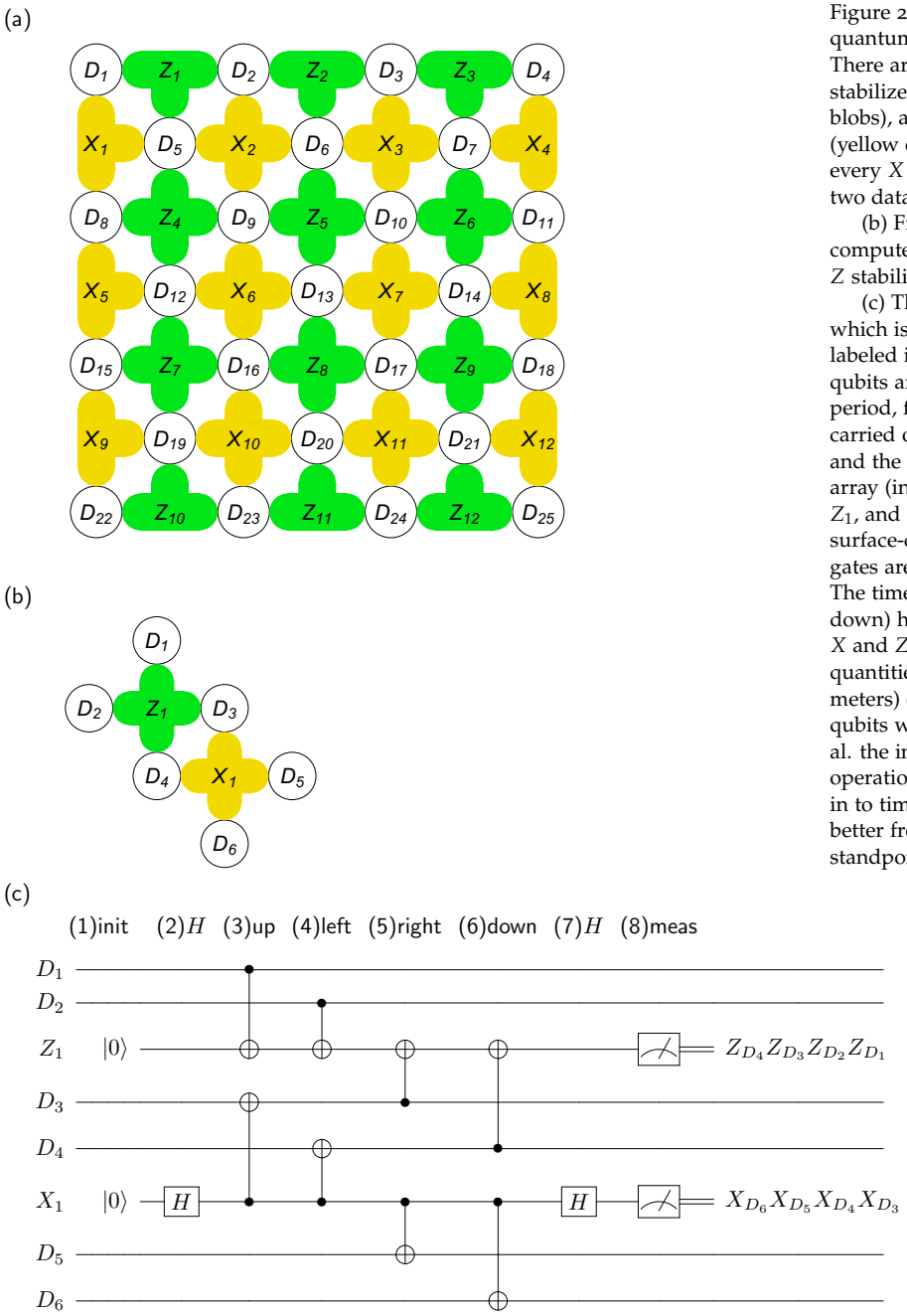


Figure 20: (a) A complete surface-code quantum computer with 49 qubits. There are 25 data qubits (circles), 12 Z-stabilizer ancilla qubits (green or darker blobs), and 12 X-stabilizer ancilla qubits (yellow or lighter blobs). Note that every X stabilizer shares either zero or two data qubits with every Z stabilizer.

(b) Fragment of a surface-code computer showing six data qubits, one Z stabilizer, and one X stabilizer.

(c) The surface-code stabilizer cycle, which is divided into eight time periods labeled init, H, up, ... H, meas. The qubits are labeled as in (b). In the "up" period, for example, a CNOT gate is carried out between every stabilizer and the data qubit that is above it in the array (in (b) note that D_1 is "up" from Z_1, and D_3 is "up" from X_1). In a real surface-code computer, all of the "up" gates are carried out simultaneously. The time sequence (up, left, right, down) has been cleverly chosen so the X and Z stabilizers measure the desired quantities (shown to the right of the meters) even though they share data qubits with each other. (In Fowler, et al. the initialization and measurement operations for Z stabilizers are moved in to time periods 2 and 7, which is better from an error-accumulation standpoint.)

- Data qubits, which hold the encoded quantum information.
- Ancilla qubits used to measure Z stabilizers (like Fig. 17(a)).
- Ancilla qubits used to measure X stabilizers (like Fig. 17(c)).

The surface-code syndrome. Each stabilizer measures products of $\hat{X}$ or $\hat{Z}$ operators on the data qubits shown touching the stabilizer.

For example, in Fig. 20(a),

$$\hat{X}_1 = \hat{X}_{D_8}\hat{X}_{D_5}\hat{X}_{D_1}, \quad \hat{Z}_1 = \hat{Z}_{D_5}\hat{Z}_{D_2}\hat{Z}_{D_1}. \tag{57}$$

Every X stabilizer shares zero or two data qubits with every Z stabilizer, which makes the stabilizers commute with each other.

After a measurement of all of the stabilizers the computer will be in a simultaneous eigenstate of the stabilizers, which can be labeled by the eigenvalues (± 1) of each stabilizer – this is the syndrome, as usual. In Fig. 20(a), measuring all 24 stabilizers gives us a 24-bit syndrome with 2^{24} possible different values.

For the repetition and Steane codes discussed above, a syndrome of all zeros (all $+1$ eigenvalues of the stabilizers) was the "no-error" state, while any other syndrome indicated an error that should be corrected. **For the surface code we will allow the syndrome to have any value, and look for *changes* in the syndrome to detect errors.**

Possible point of confusion: Our usual measurements in the z basis are measurements of the operator $\hat{Z}$, which has eigenvalues ± 1. Since we have always considered $|0\rangle$ to be the spin-up state, an eigenvalue of $+1$ corresponds to a measurement result of 0 and and eigenvalue of -1 corresponds to a measurement result of 1. We switch freely between the eigenvalue $(+1, -1)$ and measurement result $(0, 1)$ languages.

The surface-code cycle. To operate a surface-code computer a cycle is repeated in which all of the stabilizers are measured, giving the current syndrome. On the very first cycle, a random syndrome will be measured. On subsequent cycles, the same syndrome will be measured unless errors occur. The types and locations of the errors are inferred from the pattern of syndrome changes.

We could do this with any stabilizer error code. Although a surface-code computer can have any syndrome value and still be in a no-error state, this does not mean it can be in any quantum state. The surface-code cycle puts the computer into a simultaneous eigenvector of the stabilizers, which gives the quantum state very specific and unusual correlations. Fowler, et al. call this a "quiescent state".

If all the data qubits start out at $|0\rangle$, on the first cycle the X bits of the syndrome will be random while the Z bits will all be 0.

The quantum circuit used to measure the Z and X stabilizers is shown in Fig. 20(c). This is almost identical to the circuits of Figs. 17(a) and (c) used previously to measure stabilizers. However, the time sequence up-left-right-down shown in Fig. 20(c) must be followed, with all Z and X stabilizers completing their "up" measurements before any "left" measurements are started.

For more on the reasons for this specific sequence of stabilizer measurements, see Appendix B of Fowler, et al., or the paper by Dennis, et al. cited at the start of this section.

In a surface-code computer, the eight-part stabilizer measurement cycle shown in Fig. 20 is repeated continuously for the entire array. Logical qubit operations such as qubit creation and initialization, logical gates operations, and measurement are carried out by temporarily changing the operation of a few of the stabilizers while most of the stabilizers continue with the cycle.

Continuous, periodic error correction will be needed for any quantum error code and is also used for classical computation, for example in the dynamic RAM chips currently used for most computer memory – these contain tiny capacitors, one for each bit, that must be read out and restored to the 0 or 1 charge value every 64 ms.

Surface code error detection

Fig. 21 shows the "signatures" of syndrome changes due to various kinds of data-qubit error:

- An $\hat{X}$ (bit-flip) error on a data qubit causes the Z value for the data qubit to change sign, but has no effect on the X value. Therefore, this error will make the result for any Z stabilizers connected to the data qubit change, but will have no effect on any X stabilizers connected to the data qubit. In Fig. 21, a bit-flip error on qubit

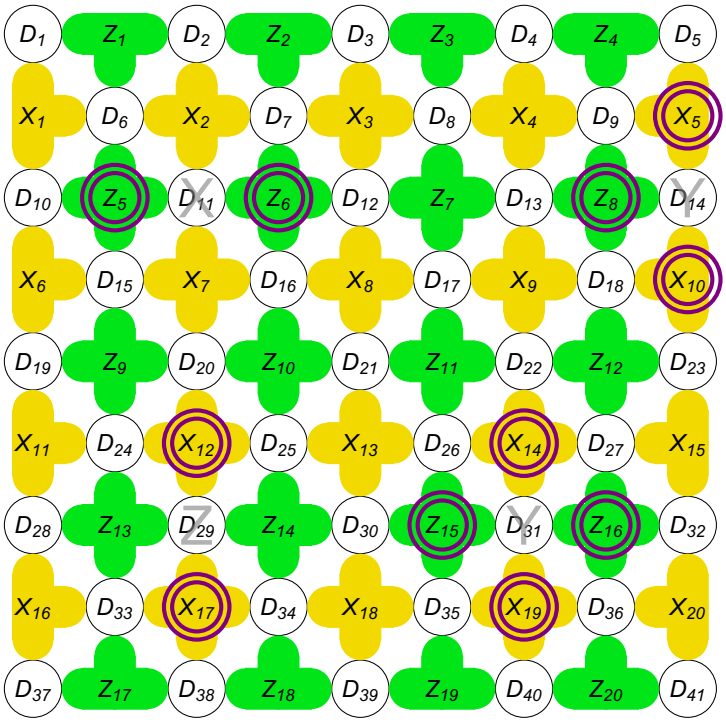


Figure 21: Error signatures (doubly-circled stabilizers) due to various errors on the data qubits (light gray X, Y, Z). The error signatures consist of stabilizer measurements that change as time proceeds, e.g. successive measurement results $\ldots 1,1,1,1,0,0,0,\ldots$

The classical control software must make the best estimate of which errors have occurred based only on changes in the stabilizer measurement results.

D_{11} (shown as a faint X, as it is not directly observable) causes changes for the two Z stabilizers connected to D_{11}, Z_5 and Z_6. These changes are shown as heavy double circles on Z_5 and Z_6 since they are observed.

- Similarly, a $\hat{Z}$ (phase-flip) error on a data qubit (like D_{29}) causes the measured values for the adjacent X stabilizers to change (X_{12} and X_{17}), but has no effect on the adjacent Z stabilizers.
- Finally, a $\hat{Y}$ (bit plus phase flip) error (like D_{14} or D_{31}) causes measured values to change for adjacent X stabilizers *and* adjacent Z stabilizers.

The only data available to the classical control software for error detection and correction are the syndrome changes shown by circled stabilizers in Fig. 21. Based on these data, the classical software must estimate as well as possible which errors occurred, and correct for these errors. Error detection is nearly unambiguous when the qubit errors are well separated, as they are in the figure, but it becomes more difficult as the density of errors increases (or more relevantly, as the tolerable probability for a mistake in error detection decreases).

Other kinds of errors, such as errors in the stabilizer-measurement circuit of Fig. 20(c), will also have characteristic syndrome-change signatures – to keep things simple we ignore such errors here. In all

Some errors have temporal rather than spatial signatures. For example, an error just before the stabilizer measurement period will make the syndrome bit change for one cycle and then change back. For a detailed discussion of the syndrome-change signatures for different kinds of error see Ref. 17 in Fowler, et al.

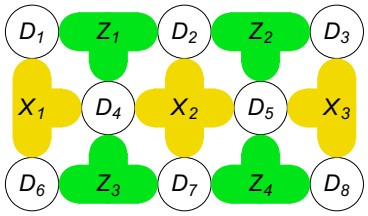


Figure 22: A 15-qubit surface code computer that you can simulate.

If you have sufficient computer power, you can substitute the 17-qubit computer shown in Fig. 34, which does a better job of distinguishing between errors than this 15-qubit computer.

cases the classical control software estimates which errors occurred, and the failure rate of these estimates will determine the failure rate for quantum computations with the surface-code computer.

Programming project 15: Surface code syndrome and errors

You can't simulate a surface-code array like Fig. 21 which has 81 qubits. Fig. 22 shows a 15-qubit surface-code array which otherwise is similar to the larger array of Fig. 21. (The Appendix shows an even smaller surface-code computer you can program as a warm-up.)

As discussed later, it actually is possible to simulate a large array of qubits like Fig. 21 provided only "Clifford group" gates are used. This includes all the gates used for the stabilizer measurements and to model errors, but not some other gates like $\hat{T}$ that are needed for universal quantum computation.

- **Syndrome measurement:** Build this array, and starting with the qubits all at $|0\rangle$ execute several stabilizer cycles reporting the measured 7-bit syndrome. The first measurement should give a random value (with however all of the Z syndrome bits 0), but all subsequent measurements should give the same syndrome.

Better: For your initial state, start with $|00\ldots0\rangle$ then with 50% probabilty bit-flip each qubit. With this initial state the entire syndrome will be random.

- **Error signatures:** Add a single X, Y, or Z error to a data qubit between two cycles. The syndrome should change to a new value, which should then remain stable. The various types of errors should have the signatures shown in Fig. 21. Optional: Make your software deduce as well as possible which error(s) occurred based on measured syndrome changes, and test it by randomly adding errors. This is how a real surface-code computer would work.

See the Appendix for a suggestion on how to deduce errors from syndrome changes.

Logical qubits

We haven't yet shown how the surface code can be used to store or operate on quantum information. Recall that for the repetition and Steane codes, in addition to the stabilizers (which measured the syndrome) there were operators $\hat{X}_L$, $\hat{Z}_L$ that measured the logical qubit (the encoded quantum information).

In Fig. 23 the left and right edges of the array are "X boundaries" (alternating data and X-stabilizer qubits), while the top and bottom edges are Z-boundaries. It turns out that we can use a *chain* of $\hat{X}$ operators on the data qubits extending from one X boundary to another X boundary as $\hat{X}_L$: the blue chain in the Fig. 23(a) is $\hat{X}_L = \hat{X}_{D_8}\hat{X}_{D_9}\hat{X}_{D_{10}}\hat{X}_{D_{11}}$ (the order doesn't matter because the $\hat{X}_{D_j}$

Any of the four edges can be either an X boundary or a Z boundary, depending on where we choose to start and end the array. The surface array does not need to be rectangular; for example we can punch holes in it.

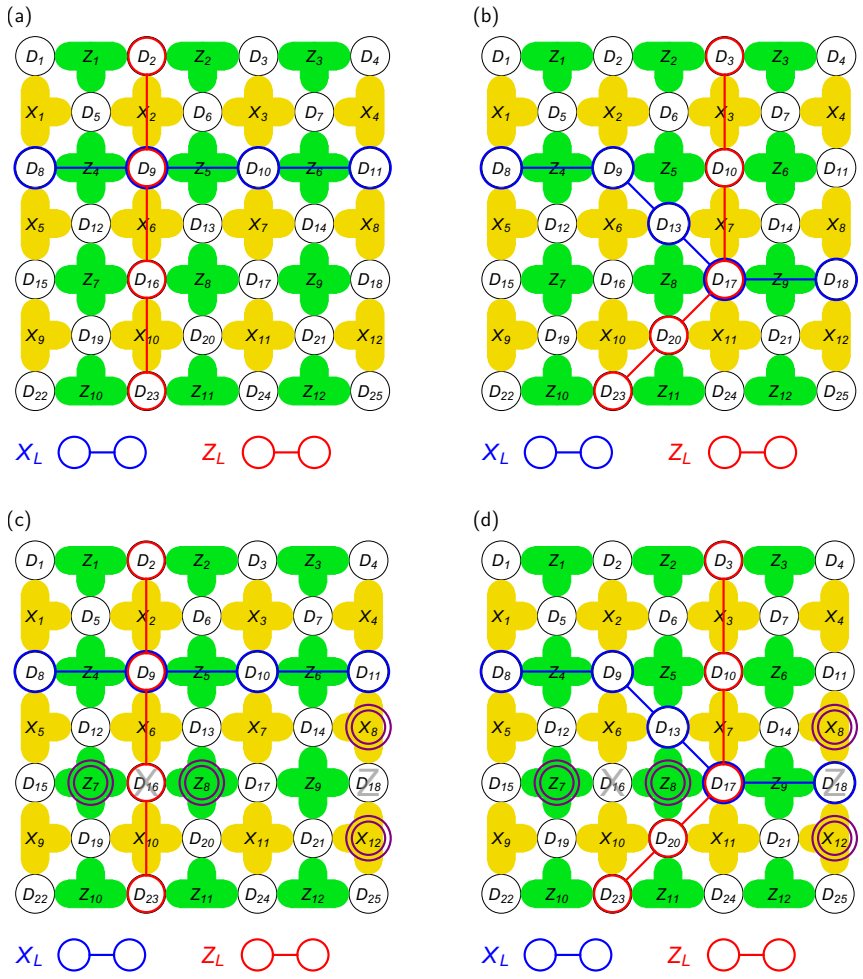


Figure 23: (a), (b) Two different ways to compute logical qubit operators $\hat{X}_L$, $\hat{Z}_L$ for a 7×7 surface code computer. The logical operators in (b) are equal to the logical operators in (a) up to sign changes determined by the syndrome.

(c), (d) The effect of two data qubit errors on the logical qubit operators (see text).

commute). This $\hat{X}_L$ shares an *even* number of data qubits with every Z stabilizer, for example it shares $\hat{X}_{D_8}$ and $\hat{X}_{D_9}$ with the stabilizer $\hat{Z}_4$. This means $\hat{X}_L$ commutes with all the stabilizers, as it should.

Similarly we we can use a chain of $\hat{Z}$ operators on the data qubits extending from one Z boundary to another Z boundary as $\hat{Z}_L$: the red chain in the Fig. 23(a) is $\hat{Z}_L = \hat{Z}_{D_2}\hat{Z}_{D_9}\hat{Z}_{D_{16}}\hat{Z}_{D_{23}}$, and $\hat{Z}_L$ also commutes with all the stabilizers. Finally, $\hat{X}_L$ and $\hat{Z}_L$ share one data qubit, D_9. This means that $\hat{X}_L$ and $\hat{Z}_L$ have the right commutation and anticommutation relations, Eq. 51, to serve as the $\hat{X}$ and $\hat{Z}$ operators for a logical qubit. In summary, a logical qubit with operators $\hat{X}_L, \hat{Z}_L$ in a surface-code array has the following *topological* properties:

- $\hat{X}_L$ is the product of $\hat{X}$ operators on a chain of data qubits extending from one X boundary to another X boundary, with every Z stabilizer in the array measuring an even number of data qubits in the $\hat{X}_L$ chain (usually zero or two).

If the chain stopped partway across the array, for example $\hat{X}_L = \hat{X}_{D_8}\hat{X}_{D_9}\hat{X}_{D_{10}}$, it would share only one data qubit (D_{10}) with Z_6 and $\hat{X}_L$ would not commute with stabilizer $\hat{Z}_6$.

- $\hat{Z}_L$ is the product of $\hat{Z}$ operators on a chain data qubits extending from one Z boundary to another Z boundary, with every X stabilizer in the array measuring an even number of data qubits in the $\hat{Z}_L$ chain (usually zero or two).
- If $\hat{X}_L$ and $\hat{Z}_L$ belong to the same logical qubit, their operator chains will intersect at an odd number of data qubits (usually one).

There are many different chains of $\hat{X}_{D_j}$ operators that connect two given X boundaries (e.g Fig. 23(a) and (b)). These do not represent independent $\hat{X}_L$-type operators. Rather, they are all equal to each other up to factors of ± 1, with the sign determined by the current syndrome. **A rectangular surface code array as in Figs. 20-23 encodes exactly one logical qubit, no matter how big the array is.**

The sign change between two different $\hat{X}_L$ chains is given by the product of the Z stabilizers enclosed by a loop that represents the difference between the chains. See Fowler, et al. for details.

In the 2002 article by Dennis, et al.[15] a surface-code computer was conceived as a 3D stack of 2D sheets, each encoding one logical qubit. Conversely in Fowler, et al. the surface-code computer is conceived as a single large 2D sheet, with multiple logical qubits provided by punching holes in the sheet, as we describe below.

[15] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *J. Math Phys.*, 43: 4452–4505, 2002. Also online at `http://arxiv.org/abs/quant-ph/0110143`

Error detection vs error correction

We have seen how errors on the data qubits can be detected by their characteristic signatures of changes in the syndrome (Fig. 21). What should we do with this information? One possibility is to apply $\hat{X}$ and/or $\hat{Z}$ gates to correct the errors, as in Figs. 16(b) and 18.

A much better procedure for a real quantum computer is to use the error information for purely *classical* error correction after a data qubit is measured. To do this, the classical control logic maintains two bits (not qubits) of information for each data qubit, indicating whether the data qubit has been bit-flipped, phase-flipped, both, or neither. If data qubit D_j has been bit-flipped when $\hat{Z}_{D_j}$ is measured, then the measurement result is reversed before being used. Similarly a phase-flip error reverses the result of a measurement $\hat{X}_{D_j}$.

This is better because (at least at present) classical computation is cheaper, faster, and more accurate than quantum computation per gate operation.

Figs. 21(c),(d) show how this works. In both figures there is a bit-flip error on D_{16} (detected by changes in Z_1 and Z_8), and a phase-flip error on D_{18} (detected by changes in X_8 and X_{12}). These errors are stored classically for use when the data qubits are measured.

If a second bit-flip error were detected on D_{16} before this qubit was measured, this would cancel the first error and the classical error bits for D_{16} would be reset to show no bit flip.

If we are using the logical qubit chains shown in Fig. 21(c), then a result from measuring measuring $\hat{Z}_L = \hat{Z}_{D_2}\hat{Z}_{D_9}\hat{Z}_{D_{16}}\hat{Z}_{D_{23}}$ must be flipped, since a bit-flip error has been recorded for D_{16}. A result from measuring $\hat{X}_L$ in Fig. 21(c) is not modified, since there are no phase-flip errors recored for D_8, D_9, D_{10}, or D_{11}. If instead we use the logical qubit chains shown in Fig. 21(d), then a $\hat{Z}_L$ measurement

result is not modified but an $\hat{X}_L$ measurement result must be flipped due to the phase-flip error recorded for D_{18}.

Classical implementation of $\hat{X}_L$ and $\hat{Z}_L$

Here we quote Fowler, et al:

> We now must inform the reader of a curious aspect of the surface code: The logical operators $\hat{X}_L$ and $\hat{Z}_L$ that we have spent so much time discussing are not actually implemented in the surface code hardware. These operations are handled entirely by the classical control software... whenever a particular quantum algorithm calls for an $\hat{X}_L$ or $\hat{Z}_L$ operator, the operator is commuted through each subsequent logical operation in the algorithm until a second identical operator is called for, in which case the two cancel (since $\hat{X}_L^2 = \hat{Z}_L^2 = \hat{I}_L$), or until a measurement of the logical qubit is performed, in which case the operator is applied to a the measurement outcome (hence, an $\hat{X}_L$ would be applied by reversing the sign of a $\hat{Z}_L$ measurement, while it would have no effect on an $\hat{X}_L$ measurement, and similarly for a $\hat{Z}_L$ operator combined with an $\hat{X}_L$ or a $\hat{Z}_L$ measurement, respectively).

Figure 24 shows some of the rules used to move $\hat{X}_L$ and $\hat{Z}_L$ operators past other gates in a quantum circuit diagram, and Fig. 25 shows how these rules are used in practice.

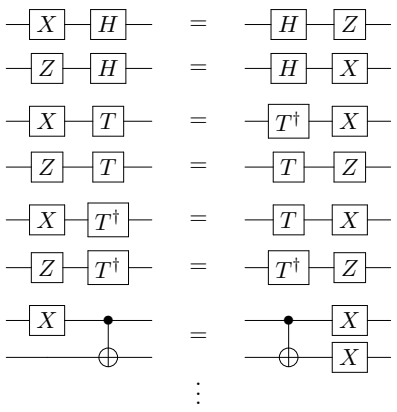


Figure 24: Partial set of rules showing how X and Z gates can be moved later in a quantum circuit diagram (not shown are three additional rules needed for moving an X or Z past a CNOT, or the rules for moving an X or Z past an S gate). Because $\hat{X}^2 = \hat{Z}^2 = \hat{I}$, at any given point on a qubit line there will be at most one X and one Z. Using these rules X and Z gates can be moved all the way to the time when the qubits are measured, and used to classically correct the measurement results. Thus, there is no need to actually apply X or Z gates to the qubits as quantum gate operations.

A possible point of confusion: To carry out these rules, the classical control software must maintain two bits of classical information for each *logical* qubit indicating any pending X or Z gates for that logical qubit. This is distinct from and in addition to the two bits of classical information maintained for each *physical* data qubit indicating any bit-flip or phase-flip errors for that physical qubit, as discussed above.

In a real surface-code computer, there will be many more physical qubits than logical qubits.

Exercise. (a) Prove the third and fourth rules in Fig. 24 by multiplying some 2×2 matrices. **(b)** Similarly show that for the circuit of Fig. 25(e) the probability of measuring 0 is indeed 0.854.

Each identity in Fig. 24 is true up to an overall phase factor, which as usual has no effect on the quantum computation.

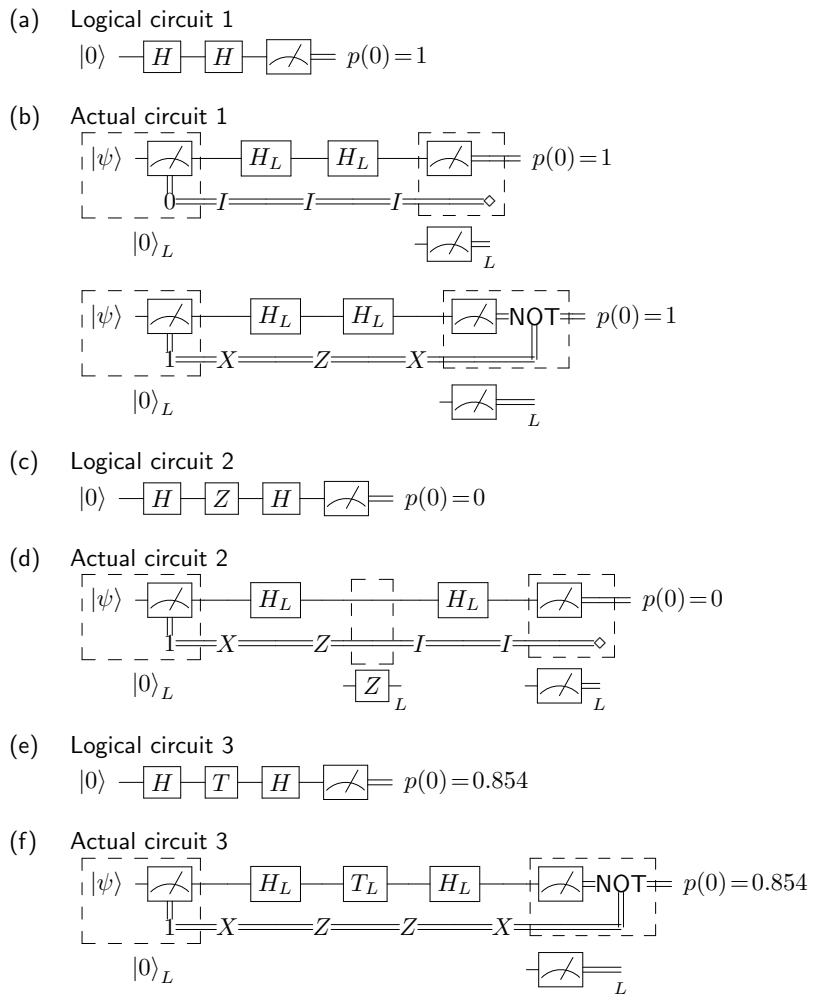


Figure 25: Three quantum calculations showing how X and Z gates are carried out using classical rather than quantum operations. Below the single line representing the logical qubit, there is a double line representing classical information on whether an X, a Y, an XY, or an I gate is pending for this qubit (this line represents *two* bits as there are four possibilities).

In all three calculations, the logical qubit is prepared in the state $|0\rangle$ by measuring it in the z basis, then applying a bit-flip $\hat{X}$ operator if the measurement result is 1. In the top diagram of (b), the result is 0 so no X is needed and the classical information is set to I (identity). In the bottom diagram of (b), the result is 1. Rather than carrying out the required X on the logical qubit, the classical information is set to X to indicated a pending X operation.

The classical X,Z information is updated according to the rules of Fig. 24 for each gate applied to the logical qubit. Thus, in the bottom diagram of (b) the first Hadamard converts the pending X to a pending Z using the first rule in Fig. 24. If there is a pending X when a qubit is measured, a classical NOT gate (flipping 0 to 1 and 1 to 0) is applied to the measurement result.

For circuits 2 and 3, only the case when the initial measurement is 1 is shown. Note that the Z gate in logical circuit 2 is never actually applied, so the quantum part of the computation is the same for circuits 1 and 2. In (d) $\hat{Z}^2 = \hat{I}$ is used, combining a pending Z with the Z in the logical circuit to give I.

Programming project 16: Surface code computing and error correction

Figure 26(a) shows the same 15-qubit surface-code computer as Fig. 22, but now with some chains marked for logical qubit operators $\hat{X}_L, \hat{Z}_L$ (you can use different chains if you prefer).

For this project you should use the surface-code computer to carry out at least the three logical computations shown in Fig. 25. Your code should maintain classical information indicating any pending X or Z gates for the single logical qubit, as in Fig. 25.

As an optional additional part, set your code up to deal with an errors on the eight physical data qubits, and show that the computation continues to work correctly except when errors are confused because they have identical syndrome signatures. To do this, your code will maintain classical information on any bit-flip or phase-flip errors on the eight data qubits, and update this information based only upon syndrome changes that occur when errors are inserted.

With the 15-qubit array of Fig. 26 there are many possibilities for confusing even single errors because the array is so small. For example, a Z error on D_2 has the same signature as a Z error on D_7. If you simulate the 17-qubit array of Fig. 34 you should find improved error correction.

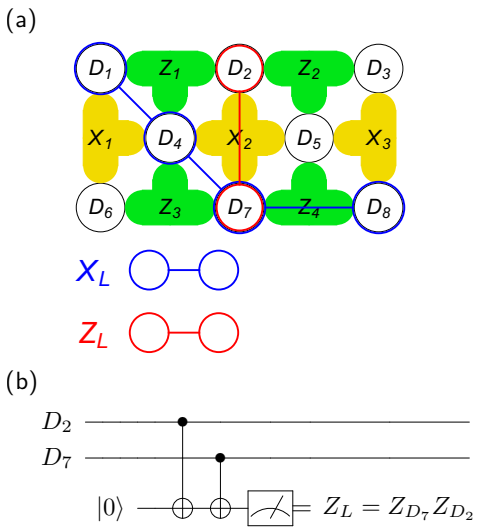


Figure 26: (a) A 15-qubit surface code array with possible operator chains for $\hat{X}_L$, $\hat{Z}_L$ marked.

(b) How to measure the logical qubit in the array, i.e to measure Z_L. You can re-use one of the stabilizer ancillae for the ancilla in this circuit.

As with the previous project, you can subsitute the 17-qubit array shown in Fig. 34 if you have sufficient computer power.

To carry out this project, you need to be able to (a) initialize the qubit to $|0\rangle_L$, (b) apply gates like $\hat{H}_L$, $\hat{Z}_L$, or $\hat{T}_L$ to the qubit, and (c) measure the qubit, i.e. measure $\hat{Z}_L$. The stabilizer sequence should be applied before any of these logical steps, and between them. Unless errors are added, the syndrome should remain constant throughout the computation.

- To measure $\hat{Z}_L$, use the circuit shown in Fig. 26(b) which is similar to a Z stabilizer circuit. Because $\hat{Z}_L$ commutes with all of the stabilizers, you should be able to measure $\hat{Z}_L$ without changing the syndrome, and similarly carrying out the surface-code cycle should not change the measured value of $\hat{Z}_L$.

  Below, when "defect qubits" are discussed, we will see a nifty way to measure $\hat{Z}_L$ using the stabilizers themselves.

- To apply an $\hat{X}_L$, $\hat{Z}_L$ or $\hat{Y}_L = i\hat{X}\hat{Z}$ to the logical qubit, use classical logic as in Fig. 25. Adding these gates does not change the quantum calculation, except that $\hat{T}$ may change to $\hat{T}^\dagger$ and vice versa.

  As usual the i in the expression for $\hat{Y}_L$ is an overall phase factor that can be omitted as it has no effect on the results.

- To initialize the qubit to the state $|0\rangle_L$, measure $\hat{Z}_L$ and then apply a (classically implemented) bit-flip gate $\hat{X}_L$ if the measurement result is $|1\rangle_L$, as shown in Fig. 25.

- To compute the $2^{15} \times 2^{15}$ matrix for a gate like $\hat{H}_L$ or $\hat{T}_L$, first compute the matrices for $\hat{X}_L = \hat{X}_{D_8}\hat{X}_{D_7}\hat{X}_{D_4}\hat{X}_{D_1}$ and $\hat{Z}_L = \hat{Z}_{D_7}\hat{Z}_{D_2}$ from logical chains in Fig. 26. Then use $\hat{H}_L = (\hat{X}_L + \hat{Z}_L)/\sqrt{2}$, $\hat{T}_L = A_+\hat{I}_L + A_-\hat{Z}_L$, or $\hat{T}_L^\dagger = A_+^*\hat{I}_L + A_-^*\hat{Z}_L$ to calculate the desired gate. Here $A_\pm = (1 \pm e^{i\pi/4})/2$. (See Appendix for the effect of errors on these gates.)

  For a real surface-code computer you cannot simply write down the matrix for any gate you want to apply like this, and it is more difficult (but still possible) to carry out $\hat{H}_L$ and $\hat{T}_L$ operations as shown in Fowler, et al.

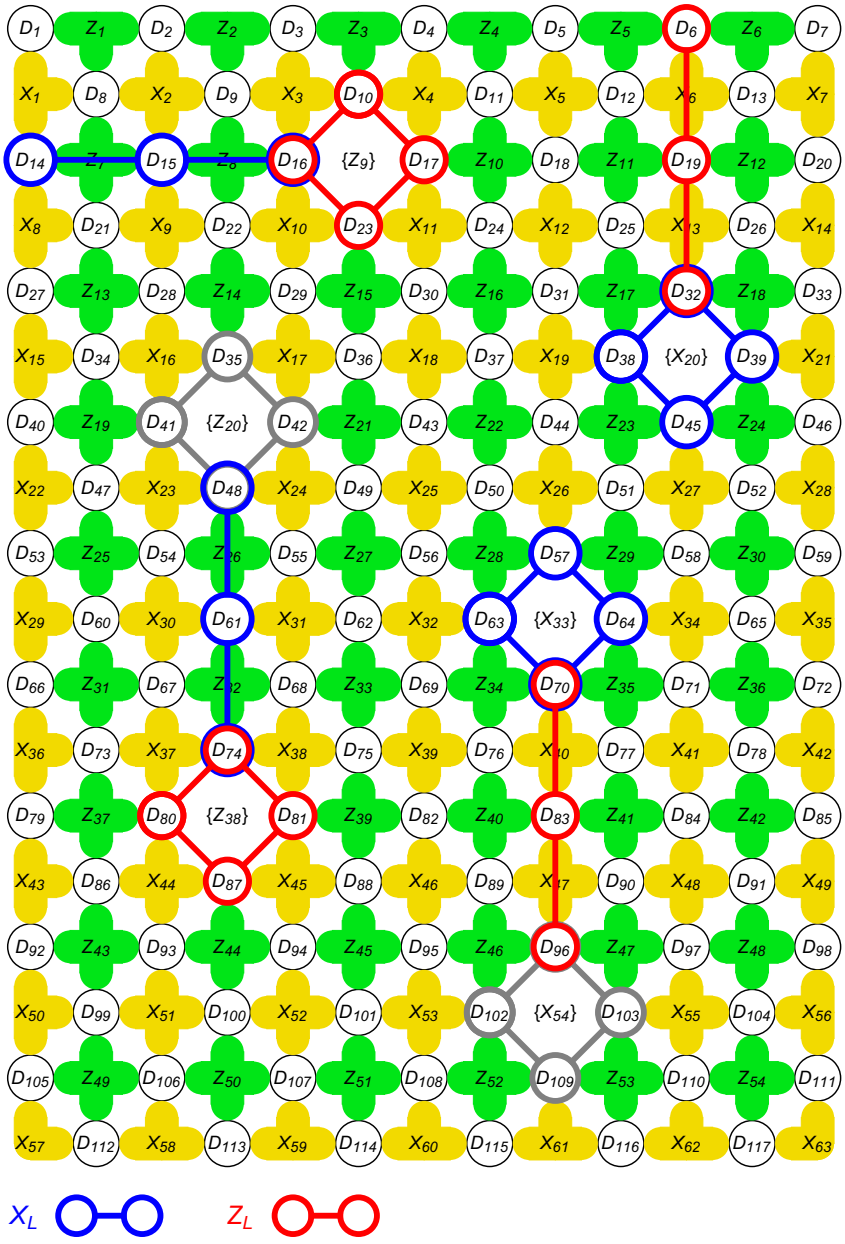


Figure 27: Four different types of defect logical qubit.

On the upper left, a "Z-cut qubit" has been formed by removing (turning off) the stabilizer Z_9, creating an internal X-boundary around the turned-off stabilizer. For this logical qubit the $\hat{X}_L$ operator is a chain of physical $\hat{X}$ operators from the external X-boundary on the left to this new internal X boundary, and the $\hat{Z}_L$ operator is a chain of physical $\hat{Z}$ operators in a closed loop around the turned-off stabilizer.

On the upper right, an "X-cut qubit" has been formed by turning off the stabilizer X_{20}. Now $\hat{Z}_L$ is a chain of $\hat{Z}$ operators from the newly-formed internal Z boundary to an external Z boundary, while the $\hat{X}_L$ operator is a closed loop of $\hat{X}$ operators around the turned-off stabilizer.

Below there are a "double Z-cut qubit" (left) and "double X-cut qubit" (right), each formed by turning off *two* stabilizers, to create two internal boundaries. These double-cut qubits would be used in a real, large-scale surface-code computer because they do not need to be near an array boundary. For the double-cut qubits the $\hat{X}_L, \hat{Z}_L$ operators are formed by a chain around one of the turned-off stabilizers and a chain between the two turned-off stabilizers; the chain around the second turned-off stabilizer (gray) is not used.

Defect qubits in the surface code

A surface array like Fig. 23 only encodes one logical qubit, no matter how big the array is made, because there is only one pair of X boundaries for an $\hat{X}_L$ chain to connect and only one pair of Z boundaries for a $\hat{Z}_L$ chain to connect.

To make additional boundaries (and thus the possibility of additional logical qubits), we can punch holes in the array by removing one or more interior stabilizers. These holes are called defects (in the

In a rectangular array like Fig. 23 the number of data qubits (25 in Fig. 23) is one greater than the total number of stabilizers (24). This means there is the equivalent of one unstabilized qubit distributed through the array – this is the one logical qubit. By removing stabilizers from the interior of the array, we create more degrees of freedom for logical qubits.

language of solid state physics), and the resulting qubits are called defect qubits. Fig. 27 shows several different ways this can be done:

- By removing a single X stabilizer (X_{20} in Fig. 27) we create an closed interior Z boundary. We can use the closed chain of data-qubit $\hat{X}$ operators around this boundary as our $\hat{X}_L$ operator, and a chain of data-qubit $\hat{Z}$ operators from the hole to a Z boundary as our $\hat{Z}_L$ operator:

$$\hat{X}_L = \hat{X}_{D_{38}}\hat{X}_{D_{32}}\hat{X}_{D_{39}}\hat{X}_{D_{45}}, \quad \hat{Z}_L = \hat{Z}_{D_{32}}\hat{Z}_{D_{19}}\hat{Z}_{D_6}. \tag{58}$$

  These $\hat{X}_L, \hat{Z}_L$ operators have the necessary properties to commute with the stabilizers and anticommute with each other: $\hat{X}_L$ shares an even number of data qubits with every Z stabilizer, $\hat{Z}_L$ shares an even number of data qubits with every X stabilizer, and $\hat{X}_L$ and $\hat{Z}_L$ share one data qubit. This is called an "X-cut qubit" since an X stabilizer was removed to create it.

- Analogously a Z-cut qubit can be created by removing a Z stabilizer (Z_9 in Fig. 27), using the closed chain of $\hat{Z}$ operators around the hole for $\hat{Z}_L$ and a chain of $\hat{X}$ operators from the hole to an X boundary for $\hat{X}_L$.

- If we don't want our logical qubits to be stuck near a boundary of the array, we can remove *two* X stabilizers (X_{33} and X_{54} in Fig. 27), use the boundary around one of the holes for $\hat{X}_L$, and a chain between the two holes for $\hat{Z}_L$. This is a "double X-cut qubit", and clearly we can also make an analogous double Z-cut qubit. In a large surface-code quantum computer as envisaged by Fowler, et al., these two-hole qubits would be used for all the logical qubits.

With this scheme, the chain of operators around the second hole goes unused.

Programming project 17: Defect qubit

It is just barely possible to fit a single-cut defect qubit into a surface-code-computer small enough to simulate using the techniques developed in these handouts, Fig. 28.

This assumes you can easily simulate a 15 qubit system, but not a 25 qubit system.

With a defect qubit, you can try out some methods to initialize, apply gates to, and measure the logical qubit that could actually be carried out in a real surface-code computer:

- **Initializing a defect qubit.** To create a Z-cut qubit, turn off the associated Z stabilizer (Z_1 in Fig. 28). It turns out that the initial value of Z_L for the qubit is the measured value for this stabilizer (the syndrome bit) just before it is turned off. If you want to initialize the qubit to $|0\rangle_L$ and the measured value is $|1\rangle_L$, you apply the (classically implemented) bit-flip operator $\hat{X}_L$ to the qubit.

To turn off a stabilizer, after a complete surface cycle you stop carrying out the operations shown in Fig. 20(c) – or as a minimum you "disconnect" the stabilizer by no longer carrying out the CNOTs associated with it.

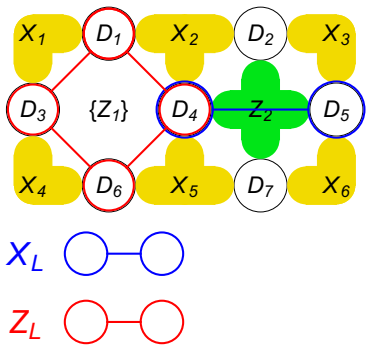


Figure 28: 15-qubit surface-code array with a Z-cut qubit. Note the array has been positioned to have X boundaries on all four sides, unlike earlier examples. The stabilizer qubit Z_1 is needed to initialize and measure the logical qubit.

- **Measuring a defect qubit.** To measure the Z cut qubit in Fig. 28, turn the stabilizer Z_1 back on (destroying the qubit). The measured syndrome bit for this stabilizer after it is turned on is the measured value of the logical qubit.
- **Applying $\hat{X}_L$, $\hat{Y}_L$, or $\hat{Z}_L$ gates.** As already described, these gates are all implemented in the classical control software rather than the quantum computation.
- **Applying other one-qubit gates.** To apply these gates in your simulation, you can calculate the required matrices as explained earlier. It is more complicated to apply $\hat{H}_L$, $\hat{S}_L$ or $\hat{T}_L$ gates in a real surface-code computer, and they are needed for universal quantum computation. Fowler, et al. show pictures of the somewhat hair-raising manipulations required.
- **Applying a CNOT between two defect qubits**. A hole that is part of a logical qubit can be *moved* by an appropriate sequence of stabilizer operations. One of the most fascinating surface-code operations is the CNOT. This is accomplished by moving one of the holes of an Z-cut qubit in a path around one of the holes of a X-cut qubit returning to its starting position. Unfortunately you cannot easily simulate this process since it requires a much larger surface-code computer than is shown in Fig. 28.

The path used to carry out a CNOT cannot be shrunk to nothing without passing through the other qubit. This topological aspect of the path is described as a "braid" of the two qubits. For some fun pictures of braided surface-code qubits, see P. Mishra and A. Fowler, `http://arxiv.org/abs/1406.4948` (2014).

Decreasing the logical error rate

If the error detection based on syndrome changes works perfectly, there will be no logical-qubit errors and the quantum computation will be successful. Therefore, to determine the actual logical error rate we must examine the situations in which error detection fails.

In Fig. 29(a), bit-flip errors on the data qubits D_{15} and D_{16} result in a single syndrome change, Z_8 (since Z_7 is flipped by both errors, the net result is that it does not change). If the error-detection algorithm decides that a syndrome change on Z_8 indicates bit flips on D_{15} and

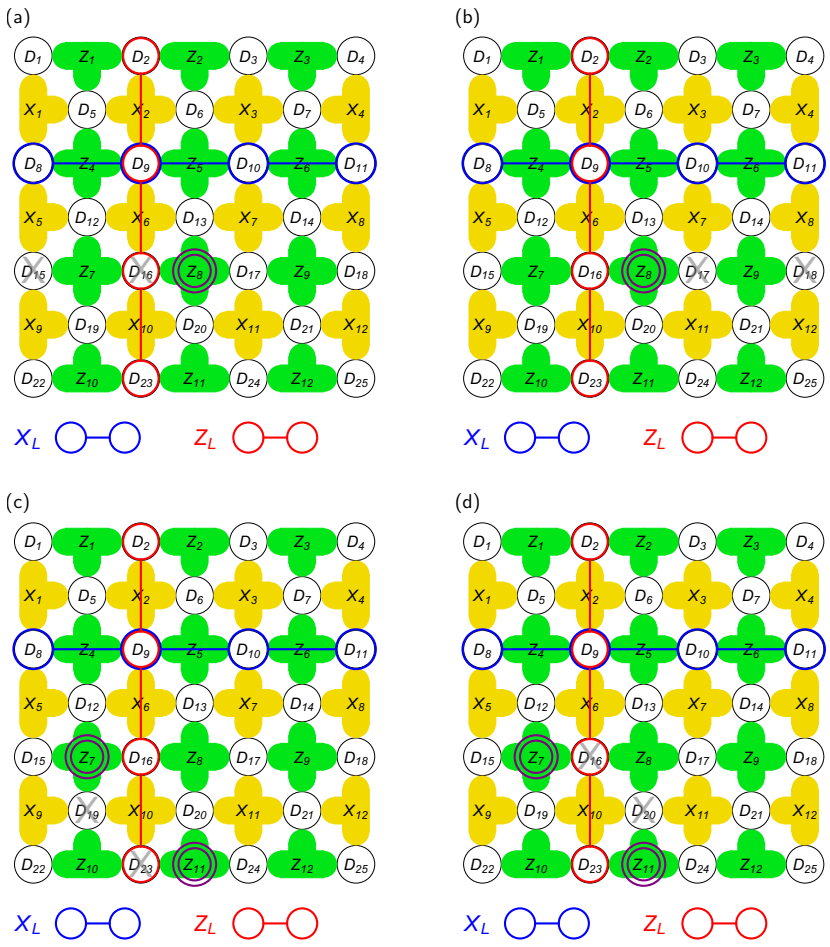


Figure 29: (a),(b) Two different sets of bit-flip errors on the data qubits that result in the same syndrome signature, a change in Z_8. Confusing these two situations will result in a logical-qubit error.

(c),(d) Again two different sets of bit-flip errors result in the same syndrome changes, now Z_7 and Z_{11}. Confusing these two situations will *not* result in a logical-qubit error.

D_{16}, then a measured value for $\hat{Z}_L$ will be flipped because the $\hat{Z}_L$ chain includes D_{16}.

In Fig. 29(b), bit-flip errors on D_{17} and D_{18} result in a the same syndrome signature, Z_8. With this interpretation of the signature, a measured value for $\hat{Z}_L$ will not be flipped. The error detection software has no way to distinguish between these two situations (it only knows about the syndrome changes), so there is a 50% (i.e huge) likelihood for a logical qubit error.

Figs. 29(c),(d) show two other sets of bit-flip errors (D_{19}, D_{23} in (c), D_{16}, D_{20} in (d)) which again give the same syndrome signature (changes in Z_7 and Z_{11}). In this case, however, confusing the two sets of data qubit errors will not cause a logical qubit error – in either case, a bit-flip error is stored for one of the data qubits in the $\hat{Z}_L$ chain and a measured value for $\hat{Z}_L$ will be flipped.

The essential difference between the data qubit errors in (a),(b) and those in (c),(d) is *topological*. Taken together the bit-flip errors in

(a) plus the bit-flip errors in (b) form a chain extending from the left X boundary of the array to the right X boundary of the array, just like an $\hat{X}_L$ chain. Therefore, the *difference* between (a) and (b), which the error-detection software cannot detect, is equivalent to an $\hat{X}_L$ – a bit-flip on the logical qubit.

Conversely, the bit-flip errors in (c) plus the bit-flip errors in (d) form a chain that is a closed loop around the stabilizer X_{10} which is topologically trivial, i.e. equivalent to no chain at all. The lesson suggested by this example is: **Indistinguishable sets of data qubit errors, leading to logical qubit errors, occur when the errors fill up about half of an $\hat{X}_L$ or $\hat{Z}_L$-type chain.**

If X_{10} were turned off for a defect qubit, a loop surrounding X_{10} would no longer be topologically trivial.

To make logical qubit errors less likely, the minimum lengths of $\hat{X}_L$ or $\hat{Z}_L$-type chains must be made large. Specifically, the distance d for the surface is the defined as the minimum length of an $\hat{X}_L$ chain or a $\hat{Z}_L$ chain, whichever is less. The logical qubits shown in Fig. 27 all have $d = 3$, and are susceptible to logical errors caused by two or more physical errors as in Fig. 29.

To make d bigger so logical errors are less likely, both the size and distance between the qubit holes must be increased. By doing detailed simulations and comparing with statistical arguments, Fowler, et al. estimate that

$$p_L \approx 0.03(p/p_{\text{th}})^{d/2}, \quad \text{with } p_{\text{th}} \approx 5.7 \times 10^{-3}. \tag{59}$$

Factoring a 2000 bit number with 90% success requires $p_L < 2 \times 10^{-14}$. With this p_L and an assumed per-gate error rate $p = 10^{-3}$, Eq. 59 gives $d \geq 32$. Figure 30 shows a double-cut qubit with $d = 32$.

Fowler, et al. do this more carefully getting $d \geq 34$.

The surface code as a passive error-correcting medium

The surface code has been presented here as a stabilizer-type quantum error code, in which measurements of the syndrome are used to actively eliminate detected errors. This is similar to the error correction discussed above for block codes like the Steane code.

If you look at the papers that introduced the surface code[16,17] or a more recent paper discussing the effects of correlated errors[18] you will see a different way of looking at the surface code, as a model for a two-dimensional material. The data qubits are the degrees of freedom in this material and the Hamiltonian is proportional to minus the sum of the X and Z stabilizers,

$$\hat{\mathcal{H}} = -J_x \sum_s (\prod_{j \in s} \hat{X}_j) - J_z \sum_p (\prod_{j \in p} \hat{Z}_j). \tag{60}$$

Here J_x, J_z are parameters of the model, and s and p list the X and Z stabilizers which are computed as products of $\hat{X}$ and $\hat{Z}$ operators on

[16] A. Yu Kitaev. Fault-tolerant quantum computation by anyons. *Annals Phys.*, 303:2–30, 2003. Originally online at `https://arxiv.org/abs/quant-ph/9707021` (1997)

[17] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *J. Math Phys.*, 43: 4452–4505, 2002. Also online at `http://arxiv.org/abs/quant-ph/0110143`

[18] I. S. Tupitsin, A. Kitaev, N. V. Prokof'ev, and P. C. E. Stamp. Topological multicritical point in the phase diagram of the toric code model and three-dimensional lattice gauge Higgs model. *Phys. Rev. B*, 82:085114, 2010

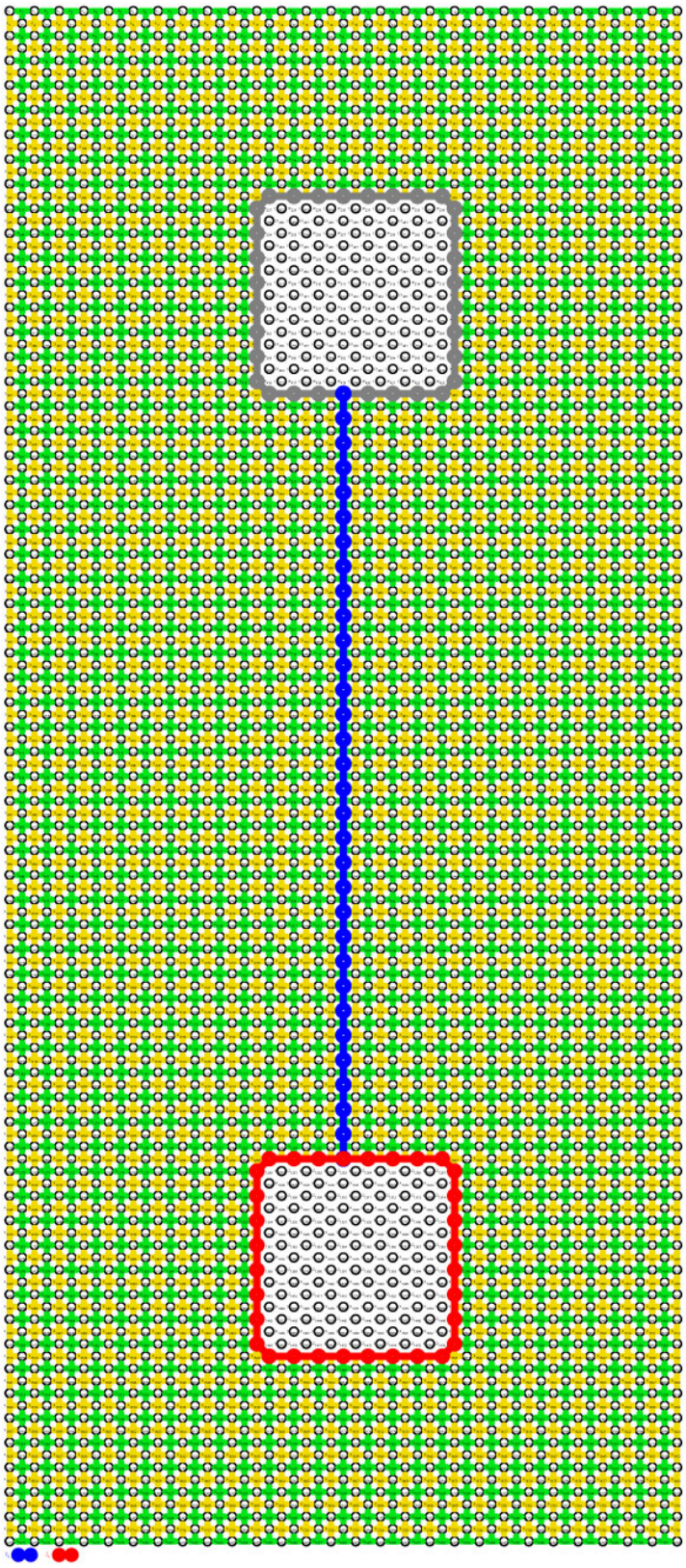


Figure 30: A double Z cut qubit with distance $d = 32$. This means both the $\hat{X}_L$ chain between the two holes and the $\hat{Z}_L$ chain around the bottom hole are 32 data qubits long.

This is approximately the size of the logical qubits that will be needed for a 2000-bit Shor's algorithm factoring computation if the physical per-gate error rate is $p = 10^{-3}$.

The spacing between any two qubits and between any qubit and the edge of the array will need to be at least d (the length of the X_L chain shown in blue), and at least $d = 32$ surface-code cycles will be required to carry out a measurement with sufficient reliability.

the j^{th} data qubit as usual. In the ground state of the model all of the stabilizers are $+1$, while J_x, J_z give the energy cost to flip an X or Z stabilizer to -1. This ground state is an example of a "topological" phase. The excitations are "anyons", which means that they pick up a quantum phase factor when they are braided around each other. This leads to the implementation of CNOT in the surface code by braiding defect qubits around each other.

At low temperatures, the excitations (errors to our usual way of thinking) are exponentially suppressed by the Boltzmann factor $e^{-J_{x,z}/kT}$. Thus, coupling the system to a thermal bath at sufficiently low temperature "passively" controls errors, requiring no no specific error detection/correction actions.

This passive Hamiltonian way of looking at the surface code is not pursued further here, other than to note that it establishes a connection between the surface code and some other possibilities for creating physical qubits as anyonic excitations in materials like fractional quantum Hall states and unconventional superconductors.

Magic states and universal quantum computation

The boundary between classical and quantum computation

What types of computation are *scalable* on quantum computer, but not on a classical computer? That is, where is the boundary between classical and quantum computation? Factoring large numbers seems to lie on the quantum side of the divide, as we have seen.

Here scalable will mean requiring time and memory resources that grow no faster than a power of the input size, and therefore not exponentially.

Naively the types of quantum simulations we have been doing throughout these handouts are also on the quantum side of the divide. To simulate an N-qubit computation we have been storing quantum states with $2^N = e^{N \ln 2}$ complex amplitudes, and applying CNOT gates that can arbitrarily entangle the qubits.

Therefore the number of qubits that can be simulated in this way is limited to about $N < 20$ on a small computer, or about $N < 40$ on a supercomputer – not scalable!

Remarkably, this is not true: *Some* of these types of quantum computation, specifically any that can be built up from $\hat{H}$, $\hat{X}$, $\hat{Y}$, $\hat{Z}$, $\hat{S} = \hat{R}_{\pi/2}$ and $\hat{C}_{\text{NOT}}$ gates along with state preparation and measurement in the usual (z) basis can indeed be scalably simulated (e.g. for N in the thousands or more) on a classical computer.

This result is known as the Gottesman-Knill theorem, `https://arxiv.org/abs/quant-ph/9807006` (1998). Quantum states that have limited entanglement (e.g. only within limited-size blocks of qubits) can also be scalably simulated on a classical computer. Thus both widespread entanglement and non-Clifford operations are necessary conditions for "true" quantum computation but are they sufficient? Both are present in Shor's algorithm. See

Richard Jozsa and Noah Linden. On the role of entanglement in quantum-computational speed-up. *Proc. R. Soc. Lond. A*, 458:2011–2032, 2003. Also online at `https://arxiv.org/abs/quant-ph/0201143`

The quantum gates that can be built up from these gates are called the "Clifford gates", and they include rotation of a spin representing a qubit through angles that are multiples of $\pi/2$ about any axis, as well as arbitrary entangling of the qubits using CNOTs. Some of the Cifford gates can be written in terms of other ones, for example $\hat{X} = \hat{H}\hat{Z}\hat{H}$ as you saw in Figs. 3(d) and 15. In fact all of the Clifford gates can be built up from $\hat{H}$, $\hat{S}$, and $\hat{C}_{\text{NOT}}$.

Operations consisting of Clifford gates along with preparation

and measurement of qubits in the standard z basis are collectively called "Clifford operations". The Gottesman-Knill theorem shows that quantum computations made up only of Clifford operations can be efficiently simulated on a classical computer.

When a qubit is measured as part of a quantum computation using only Clifford operations, there are only three possible probabilities for getting 0 or 1: 0, $\frac{1}{2}$, or 1. Conversely, if the T gate is included other probabilities are possible as Fig. 25(e) shows.

Given that Shor's algorithm is not scalable on a classical computer, it must require some non-Clifford operations. It turns out that a *universal* set of quantum gates (capable of efficiently and accurately approximating any quantum computation on N qubits) can be formed by supplementing the Clifford operations with just one non-Clifford gate, such as the the T gate:

Confusingly the T gate is often called the $\pi/8$ gate.

$$\hat{T} = \hat{R}_{\pi/4} \leftrightarrow \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}. \tag{61}$$

The T gate rotates a spin representing a single qubit by $\pi/4 = 45°$ about the z axis, and it is interesting that the ability to carry out such $\pi/4$ rotations is all that separates classically scalable computations from computations that require a quantum computer.

Since $\hat{S} = \hat{T}^2$, the set $\{\hat{H}, \hat{T}, \hat{C}_{\text{NOT}}\}$ is a universal quantum gate set, sufficient to build up any possible quantum computation.

The Solovay-Kitaev Theorem

You may wonder about the IQFT used in Shor's algorithm (Fig. 10), which requires controlled phase-shift operators $\hat{R}_{\pi/2}, \hat{R}_{\pi/4}, \hat{R}_{\pi/8}, \hat{R}_{\pi/16} \ldots$, i.e. rotations through angles much smaller than $\pi/4$.

An interesting result called the Solovay-Kitaev theorem shows that sequences of gates from the universal set $\{\hat{H}, \hat{T}, \hat{C}_{\text{NOT}}\}$ can efficiently approximate (that is, using not too many gates) any rotation gate to any required accuracy. Colloquially speaking, the Clifford gates can only take us from the $+z$ direction on the Bloch sphere (corresponding to $|0\rangle$, see Fig. 31) to the six cardinal directions $\pm x, \pm y, \pm z$ since the Clifford gates only include rotations of $n\pi/2$ about the axis directions. But once you throw in a $\pi/4$ rotation about the z axis (the T gate), you can quickly get near any point on the Bloch sphere.

See *The Solovay-Kitaev Algorithm* by C. M Dawson and M. A. Nielsen, `http://arxiv.org/abs/quant-ph/0505030` (2005), or Nielsen and Chuang Appendix 3.

Magic states and their distillation

Rather than explicitly introducing T gates, there is an even simpler-sounding addition to the Clifford operations that will give us universal quantum computation: the ability to *initialize* a qubit in either of the following "magic states":[19]

[19] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A*, 71:22316–1–13, 2005. Also online at `https://arxiv.org/abs/quant-ph/0403025`

$$\begin{aligned} |H\rangle &\leftrightarrow \begin{bmatrix} \cos(\pi/8) \\ \sin(\pi/8) \end{bmatrix}, \text{ or} \\ |T\rangle &\leftrightarrow \begin{bmatrix} \cos\beta \\ e^{i\pi/4}\sin\beta \end{bmatrix} \text{ with } \beta = \frac{1}{2}\cos^{-1}\left(\frac{1}{\sqrt{3}}\right). \end{aligned} \tag{62}$$

$|H\rangle$ and $|T\rangle$ are one-qubit states, not operators, and they should not be confused with the $\hat{H}$ and $\hat{T}$ operators.

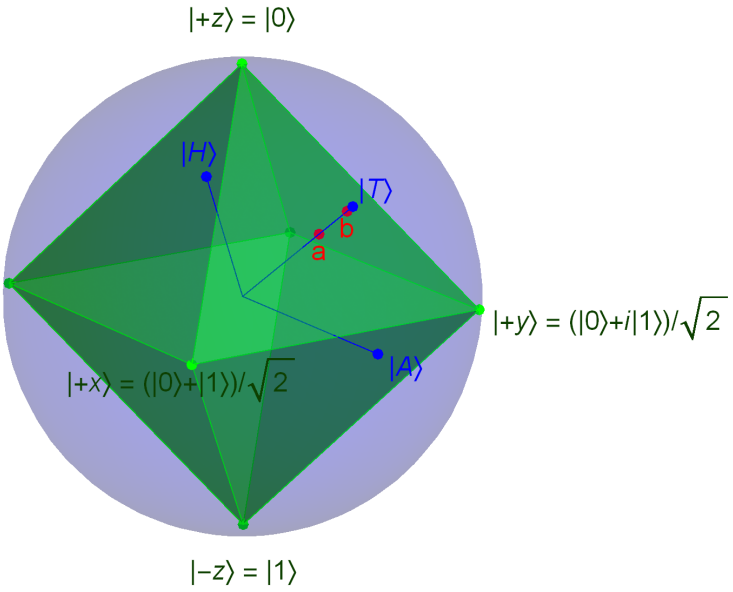


Figure 31: Bloch-sphere representation of the quantum states of a qubit. Pure states are on the surface of the blue sphere, while mixed states are in the interior.

(green) The states accessible via Clifford operations are the six pure states in the $\pm x, \pm y, \pm z$ axis directions, along with the surface and interior of the octahedron spanned by linear combinations of these points.

(blue) The pure magic states $|H\rangle, |A\rangle, |T\rangle$ are not accessible via Clifford operations.

(red) Magic-state distillation starts with several copies of a high-error mixed state near one of the magic states (a), and produces a single lower-error mixed state (b). The input state (a) must lie outside the octahedron of Clifford-accessible states (by a minimum distance for $|T\rangle$, but only infinitesimally for $|H\rangle$ or $|A\rangle$).

Bravyi and Kitaev showed that several relatively error-prone copies of $|H\rangle$ or $|T\rangle$ could be "distilled" into a single $|H\rangle$ or $|T\rangle$ with a lower error probability. They termed these states magic, because the distillation process only works for certain states but these are just the states needed for universal quantum computation.

A different way to achieve universal quantum computation is to supplement the Clifford operations with *measurement* in an arbitrary basis. This used in "one-way" quantum computation in which Clifford operations are used to prepare an entangled state called a cluster or graph state. Then the quantum computation is accomplished by making a series of non-Clifford measurements on this state. See

Dan E. Brown and Hans J. Briegel. One-way quantum computation - a tutorial introduction. 2006. URL `http://arxiv.org/abs/quant-ph/0603226`

Bloch sphere representation of the magic states

Any pure quantum state of a qubit (i.e. a spin-1/2 system) can be written in the form

$$|\theta, \phi\rangle \leftrightarrow \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)e^{i\phi} \end{bmatrix} \tag{63}$$

where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$ are the standard angles for spherical coordinates, so we can plot the state as a point on the surface of the Bloch sphere (Fig. 31). For example, the state $|0\rangle = |+z\rangle = |0, \phi\rangle$ (the north pole), while $\hat{H}|0\rangle = |+x\rangle = |\frac{\pi}{2}, 0\rangle$ (on the equator).

See Prob. 3.2 in Townsend, *A Modern Approach to Quantum Mechanics, Second Ed.*, or Eqs. 4.163-4.166 in Griffiths, *Introduction to Quantum Mechanics, Second Ed.*

Since the Clifford gates carry out rotations by integer multiples of $\pi/2$ about the three axes, the only pure states that can be obtained by applying these gates starting with $|0\rangle$ are the six states $|\pm x\rangle$, $|\pm y\rangle$, $|\pm z\rangle$, shown as green points in Fig. 31. Conversely, the magic states $|H\rangle$, $|T\rangle$ (blue points in Fig. 31) cannot be reached from $|0\rangle$ using Clifford gates.

To proceed, we must discuss *mixed* states, that is probabilistic mixtures of pure states. An example would be to start with the pure state $|0\rangle$, then add a small probability p for bit-flip error. Such a

mixed state is represented by a point slightly *inside* the Bloch sphere, near the point $|0\rangle$. As the error probability p is made smaller the point representing this mixed state gets closer to $|0\rangle$.

Using the randomness provided by measurement, probabilistic mixtures of pure states can be formed. In this way, mixed states corresponding to the entire surface and interior of the green octahedron shown in Fig. 31 can be created using Clifford operations. This set of operations can be efficiently simulated on a classical computer, and is not thought to be sufficient for universal quantum computation (e.g. for Shor's algorithm).

Formally, a mixed state is described by the density operator $\hat{\rho}$. If we express the density operator as a superposition of the Pauli operators $\hat{\rho} = \frac{1}{2}(\hat{I} + \rho_x\hat{X} + \rho_y\hat{Y} + \rho_z\hat{Z})$ then the vector (ρ_x, ρ_y, ρ_z) describes a point in or on the Bloch sphere as in Fig. 31. States with $\rho_x^2 + \rho_y^2 + \rho_z^2 = 1$ (on the surface of the Bloch sphere) have $\mathrm{Tr}(\hat{\rho}^2) = 1$ and so are pure states, while states with $\rho_x^2 + \rho_y^2 + \rho_z^2 < 1$ (in the interior of the Bloch sphere) have $\mathrm{Tr}(\hat{\rho}^2) < 1$ and are mixed states.

One way to achieve universal quantum computation is to supplement the Clifford operations with the ability to prepare qubits in a pure state outside of $|\pm x\rangle$, $|\pm y\rangle$, $|\pm z\rangle$ – for example, in either of the magic states $|H\rangle$ or $|T\rangle$. It is shown below that how a qubit in one of these magic states can be combined with Clifford operations to make a non-Clifford gate such as T, and this will enable universal quantum computation.

The line connecting the origin to $|T\rangle$ is a three-fold symmetry axis of the Clifford-operation octahedron – there are eight such states, which can be transformed into each other by Clifford gates and so are functionally equivalent. Likewise there are twelve functionally equivalent states like $|H\rangle$, which is connected to the origin by a two-fold symmetry axis of the octahedron. In Fowler, et al. one of these $|H\rangle$-like states is used to create T gates for the surface code,

$$|A\rangle \leftrightarrow \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ e^{i\pi/4} \end{bmatrix}. \tag{64}$$

Quantum teleportation for quantum computing

Both the algorithm for creating a T gate from an $|A\rangle$ magic state, and the procedure for distilling several error-prone $|A\rangle$ states into a single less error-prone $|A\rangle$, use a procedure you may not associate with quantum computing: quantum teleportation.

The no-cloning theorem states that starting with an arbitrary quantum state $|\psi\rangle$ we cannot created two unentangled copies of the state $|\psi\rangle \otimes |\psi\rangle$. We can, however "teleport" the state $|\psi\rangle$ onto a another qubit if we are willing to destroy the original state (Fig. 32(a)).

To review the no-cloning theorem and quantum teleportation, see Townsend, *A Modern Approach To Quantum Mechanics, Second Ed.* Sec. 5.6, or Griffiths, *Introduction to Quantum Mechanics, Second Ed.* Sec. 12.3.

Fig. 32(b) shows how quantum teleportation is used to implement the non-Clifford T gate using an $|A\rangle$ state as a consumable resource. Fig. 32(c) shows how quantum teleportation is used to distill $|A\rangle$ states. An essential part of the distillation process is a quantum error code, but used in a non-standard way on logical rather than physical qubits.

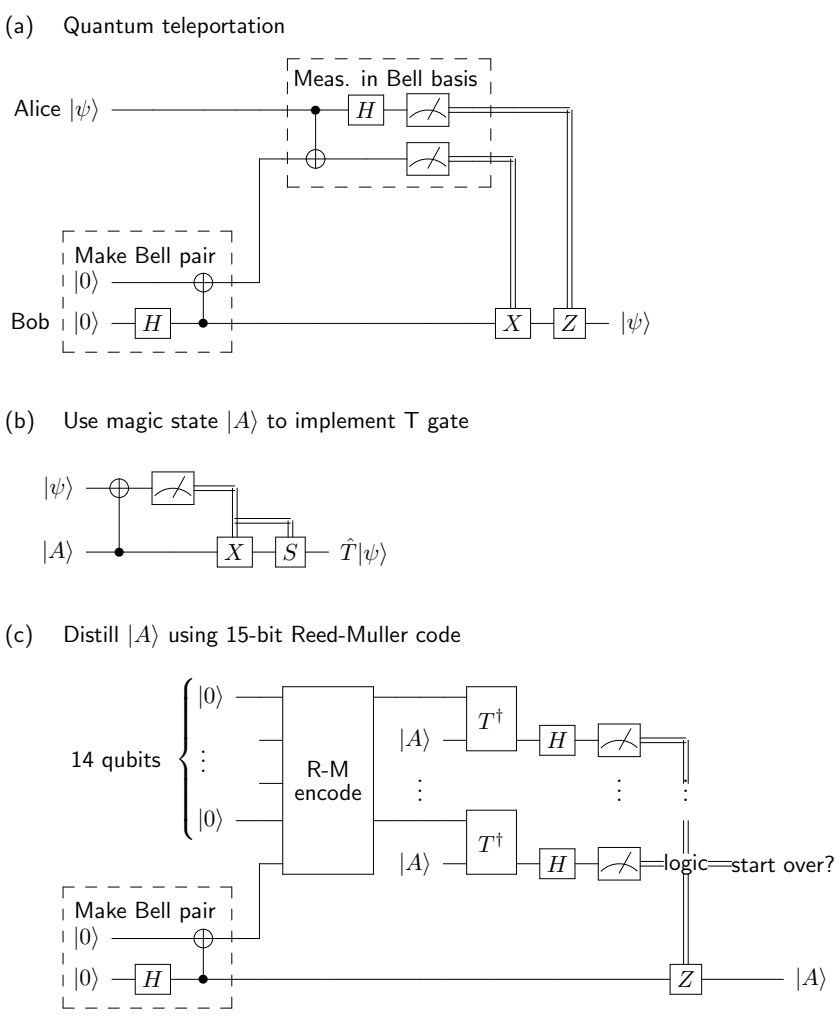


Figure 32: (a) Circuit enabling Alice to teleport the quantum state $|\psi\rangle$ to Bob, who may be some distance away. First Bob creates an entangled pair of qubits (a cat state, which is one of the four Bell states), and he sends one of the qubits to Alice. Next Alice measures the joint state of $|\psi\rangle$ and the qubit she received from Bob in the Bell-state basis, which measures correlations between her two qubits without revealing the state $|\psi\rangle$. Alice sends the measurement results (two classical bits) to Bob, who uses them to decide whether to apply bit-flip or phase-flip gates to the qubit he kept.

(b) In the circuit for a T gate, the input state $|\psi\rangle$ is destroyed by measurement (as in teleportation), but again the classical measurement result is used to decide whether to apply gates to a state that was entangled with $|\psi\rangle$. As a result the state $\hat{T}|\psi\rangle$ is teleported onto the second qubit. (Based on Fig. 30 in Fowler, et al.)

(c) Alice encodes the qubit Bob sent her with the Reed-Muller code, and sends the resulting qubits through 15 $\hat{T}^\dagger$ gates, each similar to (b) and consuming an error-prone $|A\rangle$ state, and finally measures the 15 qubits in the x basis. Using the resulting classical information, Alice tells Bob whether or not to apply a Z gate to the qubit he kept, or to throw away his qubit and start over. When the process finally succeeds an $|A\rangle$ state will have been teleported to Bob which has lower error probability than the 15 $|A\rangle$ states Alice used up, provided the input states were outside the Clifford-operation octahedron, Fig. 31. (Based on Fig. 33 in Fowler, et al.)

Postscript

We end this long handout by circling back to the example from Fowler, et al. introduced in the first few pages: using Shor's algorithm to factor a 2000-bit number on a surface-code computer. These authors estimate that the computation could be accomplished in about one day using 2.2×10^8 interconnected superconducting qubits with per-gate error probability $p = 10^{-3}$.

Interestingly, only about 10% of the physical qubits would be used to create the needed 4000 logical qubits. The other 90% of the physical qubits would be used to prepare and distill the magic $|A\rangle$ states needed to make Toffoli gates. Perhaps this is not surprising, as the Gottesman-Knill theorem shows that the ability to prepare $|A\rangle$ states is all that separates classical from quantum computation.

Recent work has reduced the number of qubits required for magic-state preparation, see e.g.

Cody Jones. Novel constructions for the fault-tolerant Toffoli gate. *Phys. Rev. A*, 87:022328, 2013. Also online at `http://arxiv.org/abs/1212.5069`

As of early 2016, it appeared that some labs were close to building 7×7 arrays of qubits with $p \approx 5 \times 10^{-3}$, and progress in both physical implementations and quantum algorithms was proceeding at a good clip. It still seems difficult to know if quantum computing will ever be a useful technology, but the answer to this question is slowly coming into focus.

Appendix: Programming hints, etc.

Convention used for numbering qubits

In a gate-array diagram like Fig. 1, qubit 1 is the top line and qubit N is the bottom line. For a basis state like $|011\rangle$, qubit 1 is the leftmost bit (0), and qubit N is the rightmost bit (1). It follows that basis states for an N-qubit register are read from top to bottom in a gate-array diagram. Thus in Fig. 1 the register is initialized to the state $|000b\rangle$.

The states $|g\rangle$, $|e\rangle$, $|+\rangle$, $|-\rangle$, and measurement in the x basis.

These notations were avoided in this handout, but you will see them in the references. The first two are alternative names for the standard basis states: $|0\rangle \equiv |g\rangle$, $|1\rangle \equiv |e\rangle$.

The basis states $|0\rangle$, $|1\rangle$ are the states of definite S_z (Eq. 2), and they are eigenstates of $\hat{Z}$ with eigenvalues ± 1.

Similarly, the states $|+\rangle$, $|-\rangle$ are the states of definite S_x, and they are eigenstates of $\hat{X}$ with eigenvalues ± 1. From the quantum mechanics of spin-$\frac{1}{2}$ we know

See Griffiths, *Introduction to Quantum Mechanics, Second Ed.* Eq. 4.151 or Townsend, *A Modern Approach To Quantum Mechanics, Second Ed.* Eqs. 2.6,2.15.

$$|+\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad |-\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{65}$$

Comparing with Eqs. 14,15 we see

$$\hat{H}|0\rangle = |+\rangle, \;\; \hat{H}|1\rangle = |-\rangle, \quad \hat{H}|+\rangle = |0\rangle, \;\; \hat{H}|-\rangle = |1\rangle. \tag{66}$$

A Hadamard gate takes us from the standard (z) basis to the x basis, and from the x basis back to the z basis. Therefore, wherever you see a qubit initialized in the state $|+\rangle$ you can substitute a qubit initialized in the state $|0\rangle$ followed by a Hadamard. This was done in Fig. 15(b) and several other places in this handout.

In a physical quantum computer it might be more efficient to directly initialize qubits to $|+\rangle$, $|-\rangle$ rather than using H gates, and it might be more efficient to directly measure qubits in the x basis.

Similarly, to measure a qubit in the x basis you can first apply an H gate, then measure the qubit in the standard z basis. This was used in Fig. 32(c), for example.

Hints for project project 10: Gate errors

When two gate errors add coherently as in Fig. 12(f), the error goes from $p = (\sin\epsilon)^2$ to $p_2 = (\sin(2\epsilon))^2$. The approximation $p_2 \approx 4p$ is

only accurate for $\epsilon \ll 1$. For example if $p = 0.1$, $p_2 = 0.36$.

Hints for project project 11: The environment

Here p is the probability for qubit 1 (the computational qubit) to have the wrong value *independent* of the value measured for qubit 2 (the environment). You may have already coded something like this for Shor's algorithm, to independently keep track of the measured results for the x-register and the f-register.

Hints for project project 12: Repetition code

Print out the measured values for the three qubits *and* the output of the classical logic for each measurement, to see what is going on.

To check that your $\hat{H}_L$ block is working, first simulate Fig. 14(d) with just one $\hat{H}_L$. The measurement results for the qubits should be either $|000\rangle = |0\rangle_L$ or $|111\rangle = |1\rangle_L$ with equal probability.

Hints for project project 13: The Steane code

To check your $|0\rangle_L$ block, feed it directly into the measurement block without intervening gates. The classical logic should always give 0, and the measured values for the seven qubits should vary randomly between the eight states shown in the first line of Eq. 47.

If you want to try out the *quantum* Steane-code QEC block of Fig. 18, here is the classical logic used to decide which corrections to apply:

			phase-flip				bit-flip
S_1	S_2	S_3	$\hat{Z}$ on	S_4	S_5	S_6	$\hat{X}$ on
0	0	0	(none)	0	0	0	(none)
0	0	1	qubit 1	0	0	1	qubit 1
0	1	0	qubit 2	0	1	0	qubit 2
0	1	1	qubit 3	0	1	1	qubit 3
1	0	0	qubit 4	1	0	0	qubit 4
1	0	1	qubit 5	1	0	1	qubit 5
1	1	0	qubit 6	1	1	0	qubit 6
1	1	1	qubit 7	1	1	1	qubit 7

Hints for project project 14(b): Error correction

For the first time, your code must be able to (a) initialize a single qubit to the state $|0\rangle$, and (b) measure a single qubit, in both cases leaving the remaining $N - 1$ qubits unmeasured.

Measuring a single qubit puts it into a definite, known state: either $|0\rangle$ or $|1\rangle$ depending on the measurement result. Therefore, you can

If you only have a single QEC block as in Fig. 16(b), the ancillae may already be $|0\rangle$ from the initialization of all the qubits at the start of the computation. But to reuse the same ancillae for multiple QEC blocks as in Fig. 16(c) you must be able to reset an ancilla from an arbitrary quantum state to $|0\rangle$.

accomplish initialization of qubit n to $|0\rangle$ by first measuring the qubit (in the usual z basis), then applying the bit-flip gate $\hat{X}^{(n)}$ if and only if the measurement result was 1. Thus, you can accomplish task (a) if you know how to do task (b), a single-qubit measurement.

Note this is *not* a two-qubit quantum controlled-X gate. Rather this is a one-qubit quantum X gate, which is either applied or not based on classical information (the measurement result).

First we describe a such a measurement in operator language. Let the normalized state of the N-qubit system before measurement be $|\Psi^{\rm in}\rangle$ and let the qubit to be measured be qubit n, with $1 \le n \le N$. Start by defining two projection operators that add up to the identity, $\hat{P}^{(n,0)} + \hat{P}^{(n,1)} = \hat{I}$. By definition $\hat{P}^{(n,0)}$ projects its argument onto the subspace in which (qubit n) = 0, while $\hat{P}^{(n,1)}$ projects its argument onto the subspace in which (qubit n) = 1. When qubit n is measured the probabilities of measuring 0 and 1 are

Colloquially $\hat{P}^{(n,0)}|\Psi\rangle$ keeps the parts of $|\Psi\rangle$ that have (qubit n) = 0, and discards the parts that have (qubit n) = 1.

$$\begin{aligned} p_0 &= \langle\Psi^{(0)}|\Psi^{(0)}\rangle \quad \text{with} \quad |\Psi^{(0)}\rangle \equiv \hat{P}^{(n,0)}|\Psi^{\rm in}\rangle, \\ p_1 &= \langle\Psi^{(1)}|\Psi^{(1)}\rangle \quad \text{with} \quad |\Psi^{(1)}\rangle \equiv \hat{P}^{(n,1)}|\Psi^{\rm in}\rangle. \end{aligned} \tag{67}$$

The result of the measurement $r = 0$ or 1 is decided randomly according to the probabilities p_0, p_1. Then the quantum state of the system after the measurement is

This is called "an ideal projective measurement". The state of the system is projected onto the subspace corresponding to the measurement result, which is often called the collapse of the wave function. The factor of $1/\sqrt{p_r}$ is necessary so $|\Psi^{\rm out}\rangle$ will be normalized.

$$|\Psi^{\rm out}\rangle = \begin{cases} |\Psi^{(0)}\rangle/\sqrt{p_0} & \text{if } r = 0, \\ |\Psi^{(1)}\rangle/\sqrt{p_1} & \text{if } r = 1. \end{cases} \tag{68}$$

Next we discuss how to implement Eqs. 67-68 in code. Let the normalized initial state be

$$|\Psi^{\rm in}\rangle \leftrightarrow \Psi^{\rm in}_j \tag{69}$$

where $\Psi^{\rm in}_j$ stands for the column vector of complex amplitudes with $0 \le j \le (2^N - 1)$. The $2^N \times 2^N$ matrices for the projection operators are computed as follows:

$$P^{(n,0)}_{j,k} = \begin{cases} \delta_{jk} & \text{if the } n^{\rm th} \text{ bit of } j \text{ is 0,} \\ 0 & \text{otherwise.} \end{cases} \tag{70}$$

$$P^{(n,1)}_{j,k} = \delta_{jk} - P^{(n,0)}_{j,k}. \tag{71}$$

Familiar manipulations are used to implement Eq. 70: In MATLAB or Mathematica (but not Python or C++) the array index is decremented so it runs from 0 to $2^N - 1$. Then it is converted to its binary digits so the $n^{\rm th}$ bit can be found. Note that $P^{(n,0)}_{j,k}$ and $P^{(n,1)}_{j,k}$ are diagonal matrices.

Typically you should compute the matrices $P^{(n,0)}_{j,k}$ for any qubits to be measured as well as other matrices that will be needed including bit-flips for qubits to be reset in advance of starting the quantum computation, to avoid re-computing matrices that are used repeatedly. However, there can be a trade-off between execution time and

Rather than storing $P^{(n,1)}_{j,k}$ you can use $\Psi^{(1)}_j = \Psi^{\rm in}_j - \Psi^{(0)}_j$.

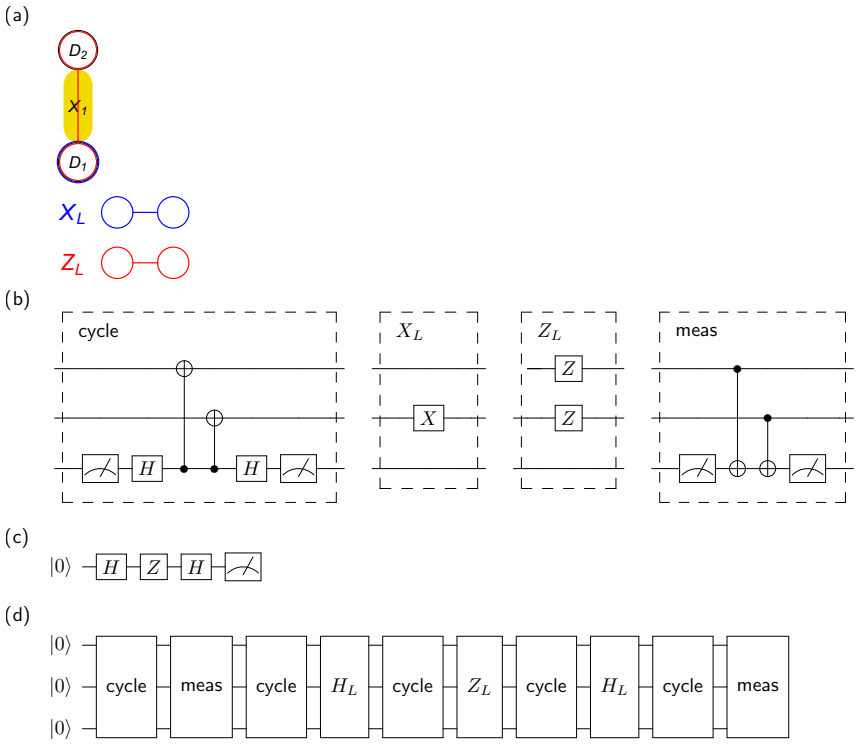


Figure 33: (a) A "toy" surface-code computer that you can simulate using only three qubits: Two data qubits and a single ancilla used to measure an X stabilizer and to measure the logical qubit (there are no Z stabilizers).

(b) How to construct the surface-code cycle, logical X and Z gates, and measurement of the logical qubit the z basis. For the cycle block the measured one-bit syndrome is 0 if the two measurements of the ancilla give the same result and 1 if they give different results; your block should print out the syndrome. For the measurement block the result is similarly 0 if the two measurements give the same result and 1 if they are different. The Hadamard gate is computed from $\hat{H}_L = (\hat{X}_L + \hat{Z}_L)/\sqrt{2}$.

(c),(d) A logical circuit and its physical implementation. The computation result is 0 if the two *logical* measurements in (d) are the same and 1 if they are different. The syndrome should be randomly 0 or 1 on the first cycle, then the same thereafter unless you add error gates (which of course you should try).

memory available to store the matrices.

There are several things you should try to see that your single-qubit measurement and QEC code is working correctly:

- Any of the quantum computations done up to now (measuring all of the qubits at the same time) should give the same results if the qubits are measured one at a time, in any order. As a minimum, check that this works for some of the computations of Figs. 3 and 6. A better test is to paste your qubit-at-a-time measurement routine into your codes for Grover's and Shor's algorithms.
- You should set up your QEC block so it prints out what it is doing:
  - Whether or not it is applying a bit flip to initialize each ancilla to $|0\rangle$ (this should agree with the way the ancilla was left by the previous QEC block, it any).
  - What quantum correction gates are applied based on the measured syndrome.

For example, the matrix for a Hadamard on qubit n is sparse and can be stored compactly. But if you try to multiply out the product of Hadamards on all N qubits in advance, you will get a huge, non-sparse matrix.

Hints for project project 15: Suface code syndrome and errors

These projects use 15 or 16 qubits, so you will probably need to use sparse matrices. (Fig. 33 shows a "toy" surface code computer you can program with only 3 qubits.)

It is very helpful to define a data structure that describes a surface code computer, consisting of: (a) a list of data qubits, (b) a list of X stabilizers, each listing which data qubit if any is in the up, left, right, and down director from this X stabilizer, and (c) a list of Z stabilizers with similar information. You can enter this information by hand for small SCC's like Fig. 20(b) and Fig. 22, or you can write code to generate this information automatically for larger SCC's (which however will be too big to simulate).

If your data structure additionally includes x, y coordinates on the surface for each type of qubit, you can write code to translate it into a picture of the SCC – that's what I did to make the figures for this handout. But this is not necessary for the simulation projects proposed here.

I suggest you get your code working with the tiny SCC of Fig. 20(b) before trying Fig. 22. Your code should print out the results of the syndrome measurement after each surface cycle.

The simplest way to add errors is the same as used for the earlier projects: Apply bit-flip or phase-flip gates of your choosing to one or more of the data qubits between surface cycles. More realistically you can define an error probability p, and before each surface cycle apply an error gate to each data qubit randomly with probability p. Your code should print out which errors are added so you can see if it is working correctly – the syndrome changes should be as expected for each type of error (Fig. 21).

For example, once a data qubit is selected to have an error you can pick an X, Y, or Z error with equal probabilities.

Note that this model unrealistically does not include errors in the syndrome measurement circuitry. Dealing with such errors, which I haven't tried, requires interpreting a wider set of syndrome-change signatures as discussed in Ref. 17 in Fowler, et al.

A more advanced project is to write code that deduces data qubit errors based on measured syndrome changes – this is how a real SCC would work. A relatively straightforward scheme is to make your code generate a table of the deduced bit-flip (X) and phase-flip (Z) errors on the data qubits for all $2^{N_X+N_Z}$ possible sets of changes in the syndrome. Here N_d is the number of data qubits, N_X is the number of X stabilizers, and N_Z is the number of Z stabilizers. Where there is more than one possible set of data qubit errors for a given signature of syndrome changes, the table should list one of the sets with the fewest number of errors since these would be more likely than sets with more errors. Error detection using such a table is imperfect, but in some sense it is as good as is possible.

To generate this table, your code will go through all 4^{N_d} possible sets of errors on the data qubits (I, X, Y or Z for each qubit) and for each set compute the signature of changes in the $N_X + N_Z$ syndrome bits. This set of errors goes in the table entry for this signature unless there is already a table entry for this signature that has less data qubit errors. This "brute force" method would need to be modified for a real SCC with more qubits than used in these projects.

Hints for project project 16: Surface code computing and error correction

Your code should print out the pending X_L and Z_L classical information at each stage of the calculation. It is helpful to build up to a calculation like Fig. 25(b) in stages, starting with no gates then including a single Hadamard and finally simulating the full circuit.

Subtle point, only needed if you combine logical-qubit computation with error correction: The operators $\hat{X}_L, \hat{Z}_L$ pick up factors of ± 1 as errors occur. These sign changes must be accounted for when $\hat{H}_L$, $\hat{T}_L$ and $\hat{T}_L^\dagger$ matrices are computed from $\hat{X}_L$, $\hat{Z}_L$ and $\hat{I}_L$, as follows:

Let E_Z be the total number of bit-flip errors detected on the data qubits in the $\hat{Z}_L$ chain, and E_X be the total number of phase-flip errors detected on the $\hat{X}_L$ chain. Whenever $E_Z + E_X$ is odd, the alternative operator $\hat{H}'_L = (\hat{X}_L - \hat{Z}_L)/\sqrt{2}$ must be used in place of $\hat{H}_L$. Similarly when E_Z is odd, $\hat{T}^\dagger$ must be used in place of $\hat{T}_L$ and vice versa.

You may want to try error correction as discussed in connection with Figs. 23(c),(d). Since the small SCC of Fig. 26 has code distance $d = 2$, it cannot always correct even a single error. Therefore the logical error rate over the physical error rate $p_L/p \to$ const. as $p \to 0$ from Eq. 59. Only for $d > 2$ do we have $p_L/p \to 0$ as $p \to 0$.

If you have sufficient computer power (and patience) you may

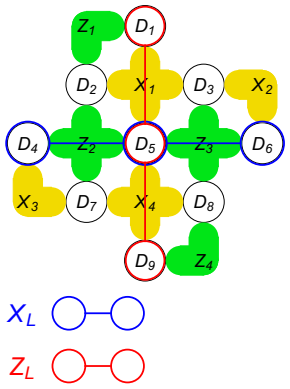


Figure 34: A 17-qubit surface-code computer with $d = 3$, copied from Tomita and Svore. This is a 5×5 array with a data qubit and stabilizer removed from each corner, and this is the smallest array that has $d = 3$ (unless stabilizer ancillae are used double-duty as shown in that article).

want to try the smallest possible $d = 3$ surface code array,[20] which is shown in Fig. 34. Unfortunately the error rate p must be set quite low (well below threshold) to see a significant improvement going from $d = 2$ to $d = 3$, making it time consuming to do sufficiently many simulations to estimate the logical error rate.

Here's what I found using my 2012-era PC running Mathematica, running the computation HHHHIHHHH (which has 10 surface cycles) with an error probability per data qubit per cycle $p = 0.02$:

	15-qubit array with $d = 2$	17-qubit array with $d = 3$
Setup time (calculate matrices and error signatures)	1.1 min	5.3 min
Memory used	118 MB	552 MB
Time to simulate 1000 computations	5.7 hr	28.2 hr
Erroneous computations	100/1000	20/1000
Logical error rate p_L	$(2.0 \pm 0.2) \times 10^{-2}$	$(6.0 \pm 1.3) \times 10^{-3}$

The logical error rate is computed as $p_L = dp_f$ where p_f is the computation failure rate per surface cycle. If it is assumed that $p_L \propto (p/p_{\text{th}})^{d/2}$ (ignoring the fact that this scaling is for $d \gg 1$) these data give $p_{\text{th}} \approx 0.2$. This number cannot be compared with p_{th} values in the literature because the error probability p used here is for an entire surface cycle and it does not include errors in the syndrome-measurement circuitry, which would reduce p_{th}.

Without error correction there would be many more erroneous computations, for example for $d = 3$ there are 50 error points on the X_L, Z_L chains (5 data qubits $\times$ 10 cycles) so $1 - 0.98^{50} = 63\%$ of the computations would have at least one potentially fatal error.

[20] Yu Tomita and Krysta M. Svore. Low-distance surface codes under realistic quantum noise. *Phys. Rev. A*, 90:062320, 2014. Also online at `https://arxiv.org/abs/1404.3747`

Production notes

These handouts were produced with Latex using the "Tufte latex" package available at

`https://tufte-latex.github.io/tufte-latex/`

The quantum circuits were drawn using the Q-circuit macro package by Bryan Eastin and Steven T. Flammia available at

`http://physics.unm.edu/CQuIC/Qcircuit/`

The surface-code arrays were drawn by the Mathematica program `2016-07-11 Surface code draw.nb`

The source file for this handout is `byo3.tex` with bib file `byo3bib.bib` and additional aux files `byo1.aux` and `byo2.aux` to enable cross-referencing the earlier handouts for Parts 1 and 2.