Quantum Error Correction (QEC)

Submitted by: Guy Nir 036907251

The need for error correction
Every computer system has occasional, random failures. Even more so when treating a Quantum mechanincal system in which decoherence is a constant problem. Transference of information from memory to CPU and back to memory, or from one computer to another across some distance, is always subject to random faults.

To better understand error correction, we look first at classical computers:

Classical Error Correction
When transfering one bit of information, an error occurs as a 'bit flip': A zero bit turns into one bit or vice versa. Classical error correction implements redundancy to correct a flipped bit.

For example one bit is multiplied three times. The chance of an error occuring, p, is considered small, while the chance of no error occuring, q, is close to unity. The chance of one error occuring is pq^2. The chance for two errors is qp^2 and is considered small (negligible). We can now assume that only one error occurs for one of these bits while transfered. If upon transference one of the bits is different from the other bits, we assume it is wrong and take only the content of the two bits. This is called 'majority vote'.

No Cloning Theorem

When looking for a similar algorithm to treat Quantum information transer we have two problems. The first is the 'No Cloning theorem', which states that a quantum bit cannot be copied multiple times, because each such copy is actually a measurement of the information in the Qbit. Such measurement 'collapses' whatever superposition the Qbit was in and destroys all the Quantum information in it. This means we cannot implement redundancy in QEC. We also see that for the information in the transfered Qbit to be of any use to us it is essential that **the error correcting code does not measure the information in the Qbit** in any part of the algorithm.

Bit Flip and Sign Flip

Another problem in QEC arrises because an error can occur by bit flip:

$|0\rangle \rightarrow |1\rangle$ or $|1\rangle \rightarrow |0\rangle$ which is equivalent to classical bit flip.

(Its occurance can also be represented by the Pauli Matrix σ_x)
Or, it can also occur by sign flip:
example:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \rightarrow |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{1}$$

which is a phase change error, applicable by the Pauli σ_z Matrix.
It is also possible for both errors to occur, flipping both sign and the bit itself. This is represented by σ_y.

The Bit Flip Code

First we look for an algorithm to solve the basic problem of bit flip, without changing the transfered bit or in fact ever measuring it. We begin by using two CNOT gates with input $|0\rangle$ to entangle the Qbit with two more bits. We now have a highly entangled three Qbit array, which preserves the information of the original Qbit without measuring it:

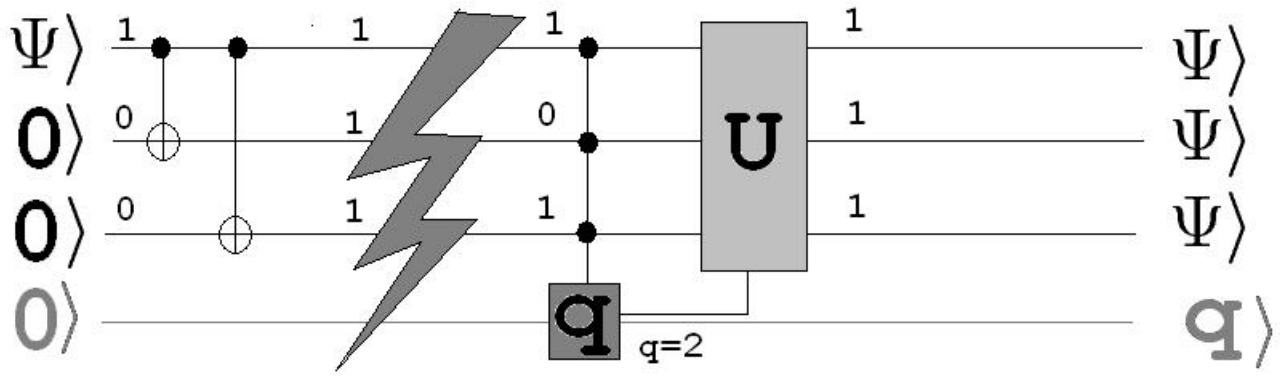
$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \rightarrow |\psi'\rangle = \alpha|000\rangle + \beta|111\rangle \tag{2}$$

Note we do not measure α or β at any point in this algorithm. These numbers contain the information we wish to transfer. Each of the Qbits in this array is transfered through a similar channel. After the error correcting algorithm the Qbits are used in further calculations. We assume an error can occur on no more than one channel each time data is transmitted (this is equivalent to the assumption that only one of the copied bits suffers an error, in classical error correction).

We may write the following statement mathematically:

$$|U_{\text{CPU}}\Psi\rangle \otimes |q\rangle = U_{\text{correction}} U_{\text{error}} (U_{\text{CPU}} \otimes I)(|\Psi\rangle \otimes |0\rangle) \tag{3}$$

Where we have taken the original state $|\Psi\rangle$ and coupled it to some initialized Von Neumann pointer $|0\rangle$, on which we have used the CPU as an encoding (the CNOT gates used in the example to prepare the input), which is represented by $U_{\text{CPU}} \otimes I$. This setup is subjected to an error (in the form of some unitary operation U_{error}) and then subjected to error diagnosis and correction ($U_{\text{correction}}$). The result must be, for correction to be considered successful, equal to the original encoded Qubits ($|U_{\text{CPU}}\Psi\rangle$), with the pointer still entangled ($... \otimes |q\rangle$) but irrelevant to the rest of the calculation.



example of bit flip circuit

In the above figure we have used q as the Von Neumann pointer, and use a set of operators to measure which erro occured (note the information itself is not measured). after the pointer is set to the error number (0 through 3) a Unitary operation U fixes the flipped bit (which is applicable by using the σ_x matrix on the appropriate channel). In the figure an example of value 1 bit is entered into the system and an error occurs on the second channel.

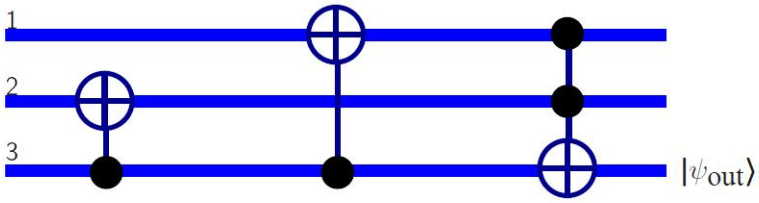
The operators used to determine which error occured are as follows. Notice any kind of three Qbit combination remains unchanged (unmeasured) though we do get information regarding which error occured.

$$
\begin{aligned}
P_0 &= |000\rangle\langle 000| + |111\rangle\langle 111| \\
P_1 &= |100\rangle\langle 100| + |011\rangle\langle 011| \\
P_2 &= |010\rangle\langle 010| + |101\rangle\langle 101| \\
P_3 &= |001\rangle\langle 001| + |110\rangle\langle 110|
\end{aligned}
\tag{4}
$$

Each projector can give a 0 or 1 result, according to the error. If there is no error, we get P_0. If the error is a bit flip on the first bit, we get P_1, and the same for the other two bits. Once we know which bit has been flipped, the value of $|q\rangle$ is set and we can use U (the appropriate σ_x) to flip it back to position. We may then use the corrected information for further calculations.

Practical circuit for Bit Flip detection and correction

We now show a slightly different approach for solveing the same problem. Where before we have shown that all three Qbits were equivalent in our detection of the error, in this circuit we define the two first bits as syndrom subsystem, and the remaining bit as information carrying bit. The concept is the same as before, the data in any of the bits is not measured, but we may use the correlation between the two syndrom bits to decide whether the third bit should be flipped or not:



$\|000\rangle$ $\|001\rangle$ $\|010\rangle$ $\|100\rangle$	$\|000\rangle$ $\|011\rangle$ $\|010\rangle$ $\|100\rangle$	$\|000\rangle$ $\|111\rangle$ $\|010\rangle$ $\|100\rangle$	$\|00\rangle\|0\rangle$ $\|11\rangle\|0\rangle$ $\|01\rangle\|0\rangle$ $\|10\rangle\|0\rangle$
$\|111\rangle$ $\|110\rangle$ $\|101\rangle$ $\|011\rangle$	$\|101\rangle$ $\|110\rangle$ $\|111\rangle$ $\|001\rangle$	$\|001\rangle$ $\|110\rangle$ $\|011\rangle$ $\|101\rangle$	$\|00\rangle\|1\rangle$ $\|11\rangle\|1\rangle$ $\|01\rangle\|1\rangle$ $\|10\rangle\|1\rangle$

In the figure above we see the first bit is connected via CNOT gate with two others, followed by a Toffoli gate which controls whether the last bit is flipped to correct an error in the end. The chart underneath it shows the different input options (after an error occured), and the final step corrects the error. Notice it doesn't matter which bit is chosen as the information carrying bit, but once chosen and entered into the system, only that bit would be taken for the rest of the calculation process.

To initially create such a system of three Qbits we use the same circuit only reversed. Since the two syndrom bits would be initialized to begin with, the Toffoli gate, now on the first step, is unnecessary. This circuit brings us back to the basic principle of entangling additional Qbits using CNOT gates, just as before.

The advantage of this method over our general discussion is that the code already includes correction, not just detection, and it is quite easy to implement (using standard gates, instead of projector operators which are mathematically simple but need to be realized in the actual circuit). Also notice this same circuit can be reversed when sending information, so the same channel can be used for two way transfer of data.

Error Syndrom Diagnosis

The major issue in QEC is, as clear from the above example, error diagnosis. The idea is to learn what the error was, without measuring the Qbit involved. If we are able to identify each error, we can learn how to reverse it by using the appropriate correction matrix. We also see now that the method of Syndrom Diagnosis is capable of detecting any general error in the channel. Since every unitary transformation the Qbit might undergo in the channel can be represented by the linear combination:

$$U = c_0 I + c_1 \sigma_x + c_2 \sigma_y + c_3 \sigma_z \tag{5}$$

We see that a general error correction code can fix **any general error in the channel**, as long as we assume only one error occurs in the three channels (including a mixed error).

Shor Code: general error correction

The Shor code is used to correct any general error, bit flip, sign flip, or both. In fact it corrects arbitrary one bit errors.

The code splits the input bit into nine entangled bits, and sets them up as shown:

$$|0_S\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle)$$

$$|1_S\rangle = \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \tag{7}$$

If a bit flip occurs, it is corrected for each group of three bits individualy.
If a sign flip occurs, each of the three groups is treated as a $|+\rangle$ or $|-\rangle$ bit and treated as simple sign flip error.

Since each single bit error is a linear combination of Pauli matrices, each discribing an error, the Shor code, or and equivalent general error correcting code (which is a linear operation in itself), can detect and correct any arbitrary error combination, so long as it only acts on one bit.