Name: Duc Manh Nguyen
Email: nguyenmanhduc18@gmail.com

**Task 3**

Write script which takes as inputs:

- quantum program (e.g. in QASM or QUIL)
- errors/noise parameters (coherence time, 1&2 qubit errors, measurement error)
- or no errors

and calculates probability of getting a correct result. I

Some requirements:

- Use realistic values for the parameters.
- You don't have to support all gates - just enough to run QFT.

Test it on QFT (Quantum Programming Studio is a good tool for generating input circuits).

**My Solution:**

1) Explain my solution:

+I consider the error as Bit-flip, phase-Flip, and their combination Bit-Phase flip error. It is a novel model and many textbooks or papers have mentioned that. Hence, I will not give detail here.

+We assume to transfer a qubit $\alpha|0) + \beta|1)$ over the quantum channel. We will protect this qubit via the repetition of extension 1 qubit system to 3 qubits system or 9 qubits system with the help of ancilla qubits (|0)).

+As literature, the bit flip error can be solved by repetition code of three qubits. The phase flip will be the same as bit flip error correction with Hadamard gate. The combination error will be solved by combination of these two solutions, a code with 9 qubits (Its name is Shor code). Then I will simulate them by quantum circuit model.

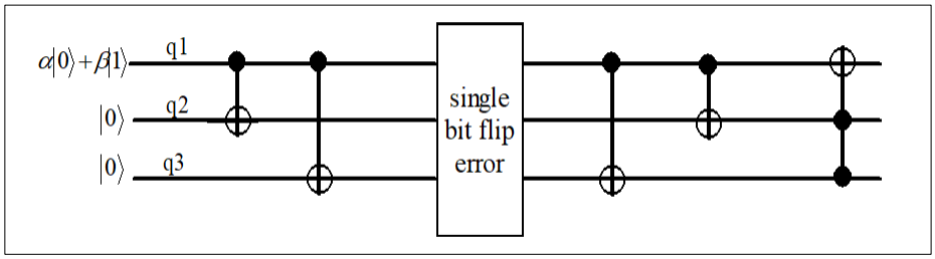


**Fig. 1.** Quantum bit flip error correction code.

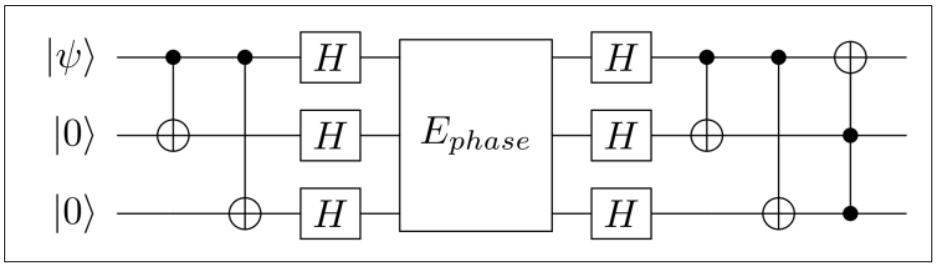


**Fig. 2.** Quantum phase flip error correction code.

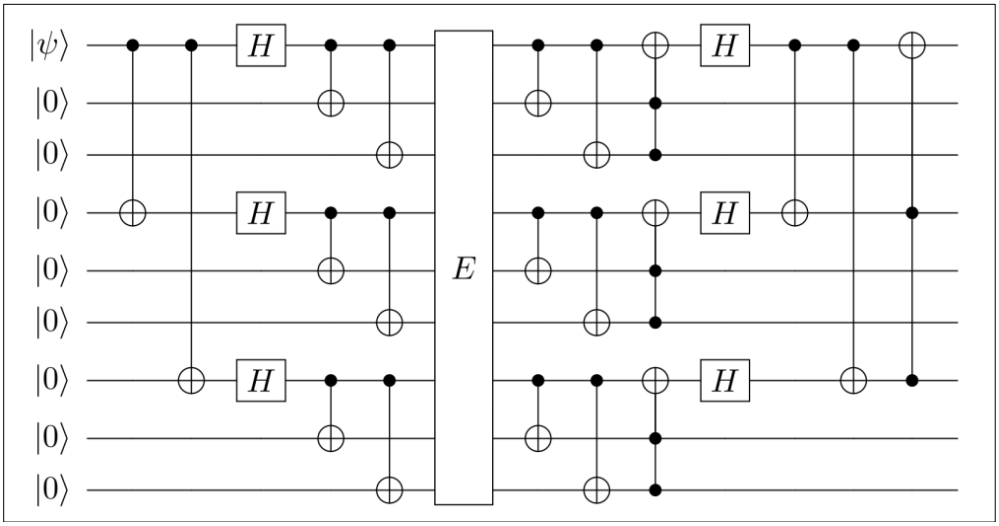


**Fig. 3.** Quantum full bit flip, phase flip error correction.

+The framework to simulate quantum circuit is MAGMA calculation tool. It is free, open source, and web-based to do many tasks.

+MAGMA is a software tool which we can install in PC or we can used as web-based for the purpose of computation in number theory, algebra, algebraic geometry, and combinatorics. It is open access for study, research purpose and provides a comfortable defined environment for many problems of mathematical such as graph, group, fields, code designs, and many others [1]. Using MAGMA, we can do the working with quantum computation since it offers some basic tool for defining the quantum state, Hilbert space, Galois field, and unitary transformation of quantum states. In this study, we use the web-based MAGMA tool, it is free and can be found at [2].

[1]http://magma.maths.usyd.edu.au/magma/handbook/text/1942#21811
[2] http://magma.maths.usyd.edu.au/calc/

2) Script for my solution:

+Bit flip error correction model.

```
F<i> := ComplexField(4);
H1 := HilbertSpace(F, 3);
f := 3/5 * H1![0,0,0] + 4/5 * H1![1,0,0];
f;
```

```
ControlledNot(~f, {1}, 2);
ControlledNot(~f, {1}, 3);
f;
BitFlip(~f, 2);
f;
ControlledNot(~f, {1}, 2);
ControlledNot(~f, {1}, 3);
ControlledNot(~f, {2,3}, 1);
f;
%Output
-->
0.6000|000> + 0.8000|100>
0.6000|000> + 0.8000|111>
0.6000|010> + 0.8000|101>
0.6000|010> + 0.8000|110>
```

+Phase flip error correction model.

```
F<i> := ComplexField(4);
H1 := HilbertSpace(F, 3);
f := 3/5 * H1![0,0,0] + 4/5 * H1![1,0,0];
f;
ControlledNot(~f, {1}, 2);
ControlledNot(~f, {1}, 3);
f;

f1 := BitFlip(f, 1);
f2 := PhaseFlip(f, 1);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f1 := BitFlip(f, 2);
f2 := PhaseFlip(f, 2);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f1 := BitFlip(f, 3);
f2 := PhaseFlip(f, 3);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;

PhaseFlip(~f, 3);

f1 := BitFlip(f, 1);
f2 := PhaseFlip(f, 1);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f1 := BitFlip(f, 2);
f2 := PhaseFlip(f, 2);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f1 := BitFlip(f, 3);
f2 := PhaseFlip(f, 3);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;

ControlledNot(~f, {1}, 2);
ControlledNot(~f, {1}, 3);
```

```
ControlledNot(~f, {2,3}, 1);
f;
-->
%Output
0.6000|000> + 0.8000|100>
0.6000|000> + 0.8000|111>
0.5999|001> + 0.8000|101>
```

+Shor code model.

```
F<i> := ComplexField(4);
H1 := HilbertSpace(F, 9);
f := 3/5 * H1![0,0,0,0,0,0,0,0,0] + 4/5 * H1![1,0,0,0,0,0,0,0,0];
f;

ControlledNot(~f, {1}, 4);
ControlledNot(~f, {1}, 7);
f;

f1 := BitFlip(f, 1);
f2 := PhaseFlip(f, 1);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f1 := BitFlip(f, 4);
f2 := PhaseFlip(f, 4);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f1 := BitFlip(f, 7);
f2 := PhaseFlip(f, 7);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f;

ControlledNot(~f, {1}, 2);
ControlledNot(~f, {1}, 3);
ControlledNot(~f, {4}, 5);
ControlledNot(~f, {4}, 6);
ControlledNot(~f, {7}, 8);
ControlledNot(~f, {7}, 8);
f;

PhaseFlip(~f, 3);
BitFlip(~f, 3);
f;

ControlledNot(~f, {1}, 2);
ControlledNot(~f, {1}, 3);
ControlledNot(~f, {4}, 5);
ControlledNot(~f, {4}, 6);
ControlledNot(~f, {7}, 8);
ControlledNot(~f, {7}, 8);
ControlledNot(~f, {2,3}, 1);
ControlledNot(~f, {5,6}, 4);
```

```
ControlledNot(~f, {8,9}, 7);
f;

f1 := BitFlip(f, 1);
f2 := PhaseFlip(f, 1);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f1 := BitFlip(f, 4);
f2 := PhaseFlip(f, 4);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f1 := BitFlip(f, 7);
f2 := PhaseFlip(f, 7);
f := 1/SquareRoot(2)*f1 + 1/SquareRoot(2)*f2;
f;

ControlledNot(~f, {1}, 4);
ControlledNot(~f, {1}, 7);
ControlledNot(~f, {4,7}, 1);
f;
%Output
-->
0.6000|000000000> + 0.8000|100000000>
0.6000|000000000> + 0.8000|100100100>
0.4949|000000000> - 0.07074|100000000> - 0.07074|000100000> + 0.4949|100100000>
- 0.07074|000000100> + 0.4949|100000100> + 0.4949|000100100> -
0.07074|100100100>
0.4949|000000000> - 0.07074|111000000> - 0.07074|000111000> + 0.4949|111111000>
- 0.07074|000000100> + 0.4949|111000100> + 0.4949|000111100> -
0.07074|111111100>
0.07074|110000000> + 0.4949|001000000> - 0.4949|110111000> - 0.07074|001111000>
- 0.4949|110000100> - 0.07074|001000100> + 0.07074|110111100> +
0.4949|001111100>
0.4949|001000000> + 0.07074|101000000> - 0.07074|001100000> - 0.4949|101100000>
- 0.07074|001000100> - 0.4949|101000100> + 0.4949|001100100> +
0.07074|101100100>
0.5999|101000000> + 0.8000|001100100>
0.5999|001100100> + 0.8000|101100100>
```