Tigon

- a quantum computing research project -

Prerequisites: I've had planned this project back in 2015 but "life happened". I will focus mostly on algorithm design and, if possible, circuit schematics. Given that I do not posses any physical device, our research will be somewhat limited. Yet, we should be able to touch the main points of interest via cloud services. Let us begin :).

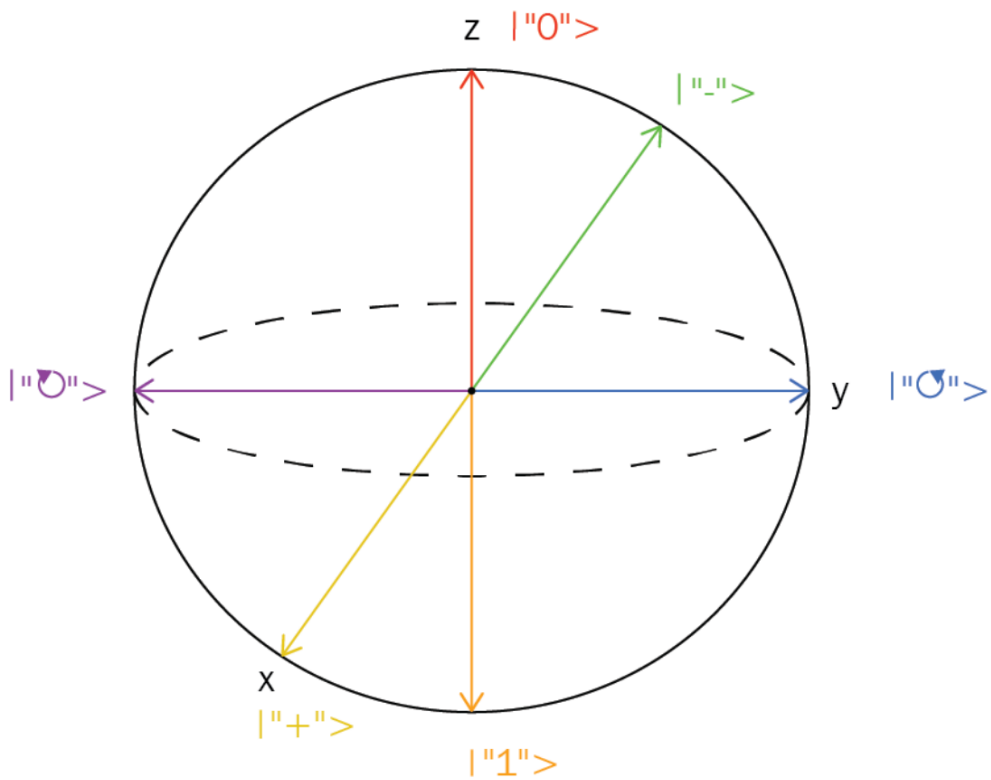
Start of the project - 5th of April

We will start from here: https://qiskit.org/ with the qiskit quantum SDK.

Took me more than an hour to configure everything… Finally I got my API key and made the first test using IBM's Q platform:

```
(base) mihai@mihai-Lenovo-V330-14IKB:~/Downloads$ python Hello\ Quantum\ World.py
[<IBMQBackend('ibmqx4') from IBMQ()>, <IBMQBackend('ibmqx2') from IBMQ()>, <IBMQBackend('ibmq_16_melbourne') from IBMQ()>, <IBMQBackend
('ibmq_qasm_simulator') from IBMQ()>]
[<QasmSimulator('qasm_simulator') from AerProvider()>, <StatevectorSimulator('statevector_simulator') from AerProvider()>, <UnitarySimu
lator('unitary_simulator') from AerProvider()>]
```

Qubit representation on the Boch sphere:



Followed a recipe from [1] to simulate quantum gates:

Identity gate:

Identity Gate, I

The identity gate leaves the qubit unchanged, and thus in the same position on the Bloch Sphere.

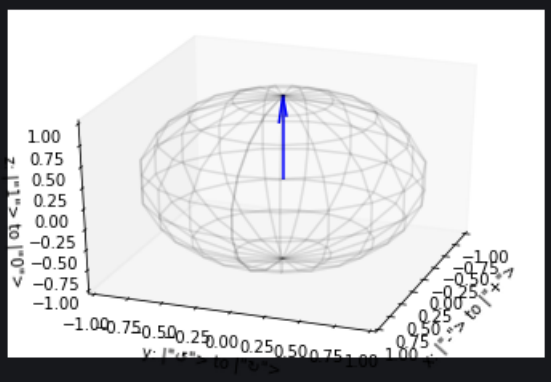
```
identity_gate=np.eye(2,2)
print(np.array_equal(zero_qubit,identity_gate*zero_qubit))
print(np.array_equal(one_qubit,identity_gate*one_qubit))
```

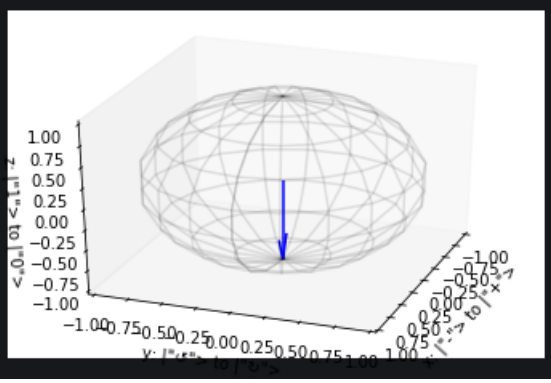
```
True
True
```

One qubit gates

```
plot_bloch(zero_qubit)
plot_bloch(one_qubit)
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f44010063c8>
```





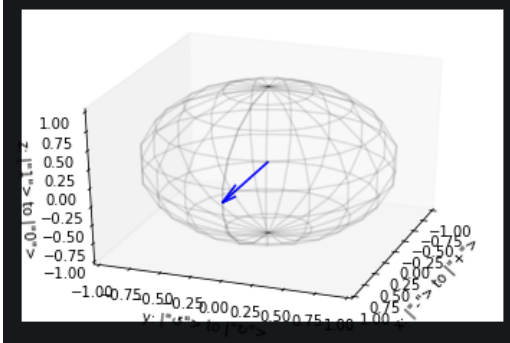
Hadamard gate:

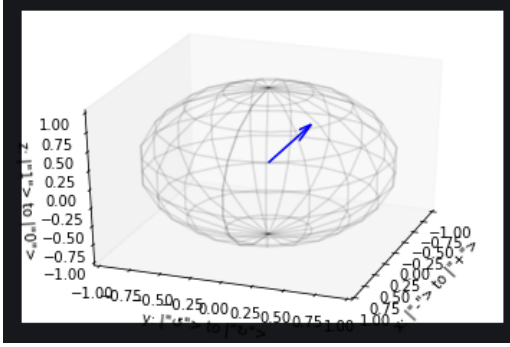
Hadamard gate, H

This gate rotates the qubit by 180° around the x-axis then 90° about the y-axis.

```
H=1./np.sqrt(2)*np.matrix('1 1; 1 -1')
plot_bloch(H*zero_qubit)
plot_bloch(H*one_qubit)
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f4400ee8630>
```





Pauli gates:

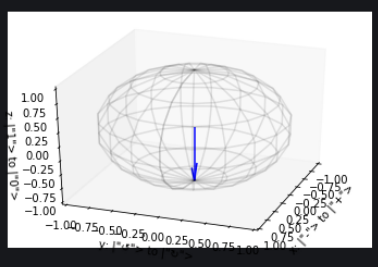
Pauli gates, X, Y, Z

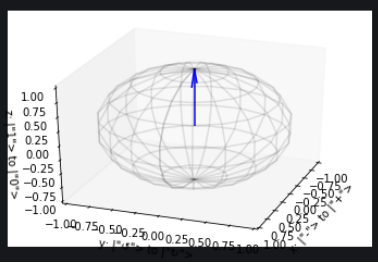
Each of these gates does a 180° flip of the qubit with respect to a different basis/axis on the Bloch sphere

```
X=np.matrix('0 1; 1 0')
Y=np.matrix([[0, -i ],[i , 0]])
Z=np.matrix([[1,0],[0,-1]])
```

```
plot_bloch(X*zero_qubit)
plot_bloch(X*one_qubit)
```

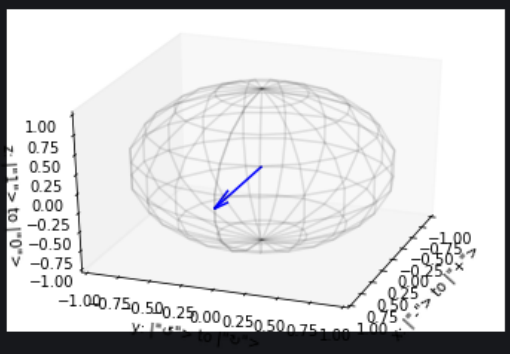
```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f4400d93358>
```

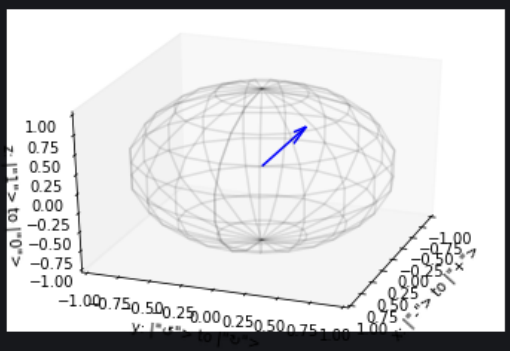




```
plot_bloch(plus_qubit)
plot_bloch(Y*plus_qubit)
```

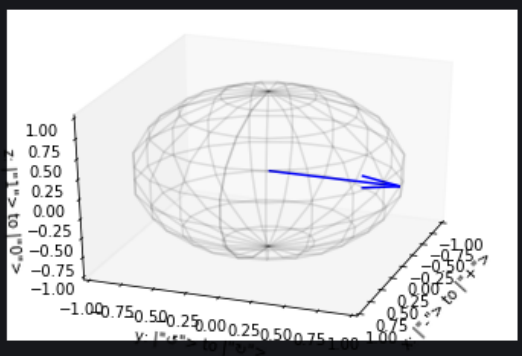
```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f4400cbc390>
```

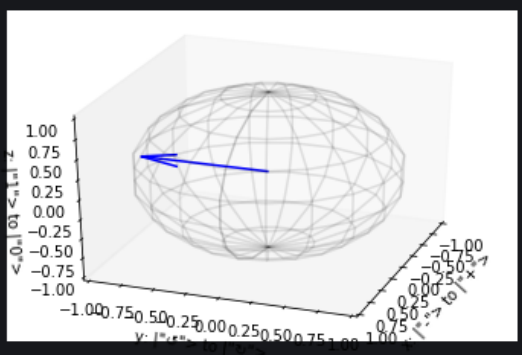





```
plot_bloch(clockwisearrow_qubit)
plot_bloch(Z*clockwisearrow_qubit)
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f4400b672e8>
```



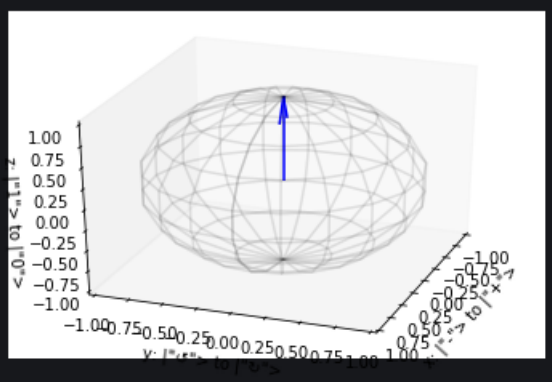


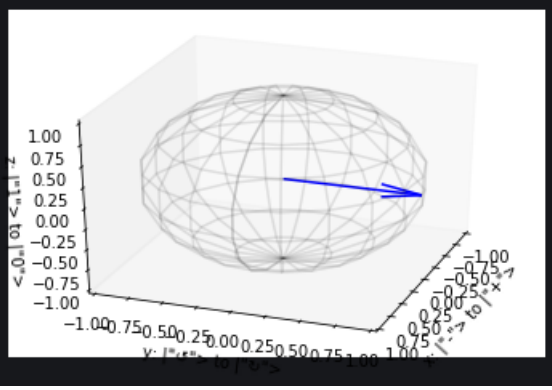
Phase gate:

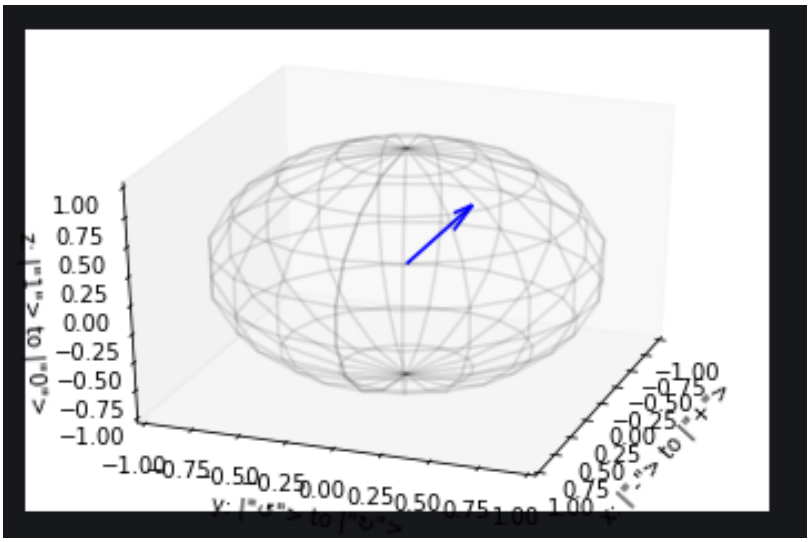
Phase gate, S

```
S=np.matrix([[1,0],[0,np.e**(i_*np.pi/2.)]])

plot_bloch(S*zero_qubit)
plot_bloch(S*plus_qubit)
plot_bloch(S*S*plus_qubit)
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f44009cddd8>
```



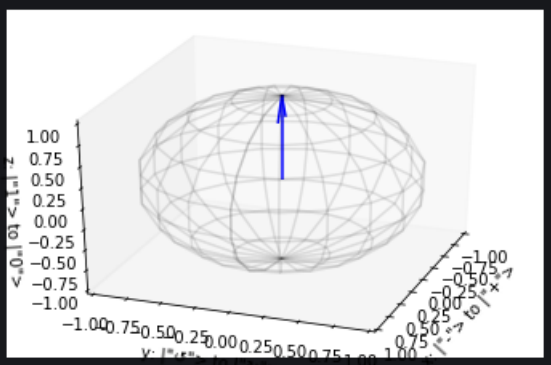


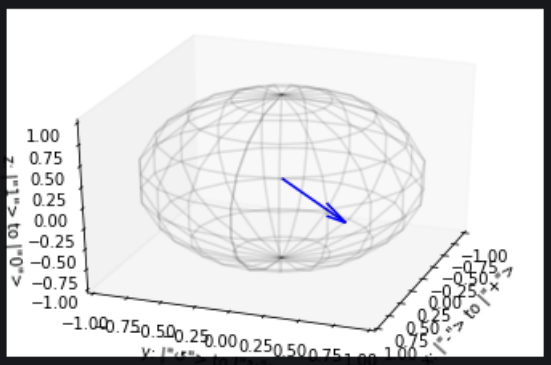


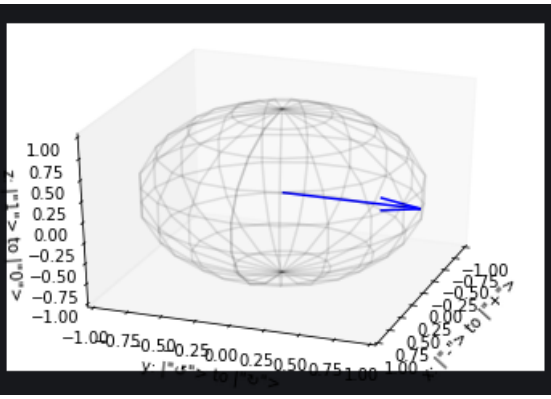
π/8 gate:

```
T=np.matrix([[1,0],[0, np.e**(i_*np.pi/4.)]])
plot_bloch(S*zero_qubit)
plot_bloch(T*plus_qubit)
plot_bloch(T*T*plus_qubit)
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f4400cd90f0>
```





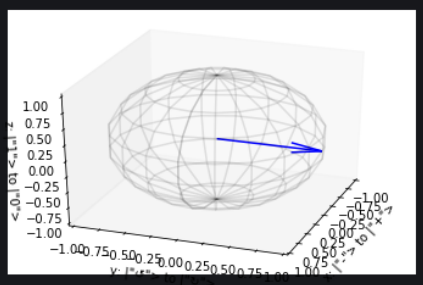


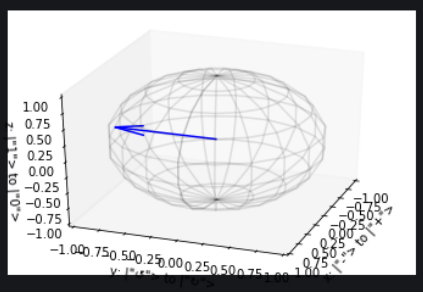
Dagger gates:

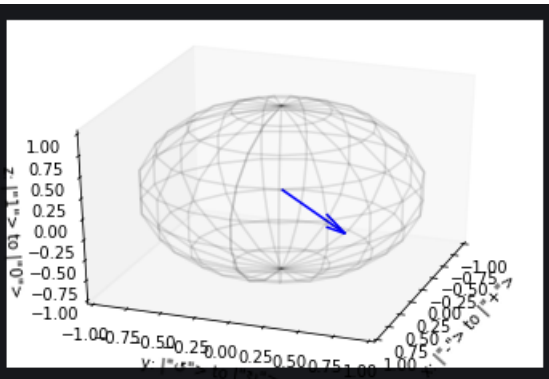
Dagger gates †

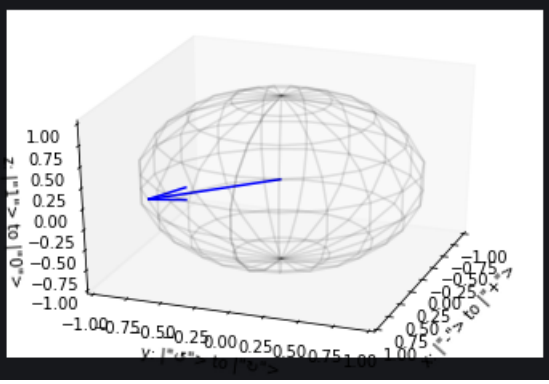
```
Sdagger = S.conjugate().transpose()
Tdagger= T.conjugate().transpose()
plot_bloch(S*plus_qubit)
plot_bloch(Sdagger*plus_qubit)
plot_bloch(T*plus_qubit)
plot_bloch(Tdagger*plus_qubit)
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f440092f5f8>
```









CNOT Gate

```
CNOT=np.matrix('1 0 0 0; 0 1 0 0; 0 0 0 1; 0 0 1 0')
starting_state=create_quantum_state([plus_qubit,zero_qubit])
final_state=CNOT*starting_state
probability_table_in_01_basis(final_state)
```

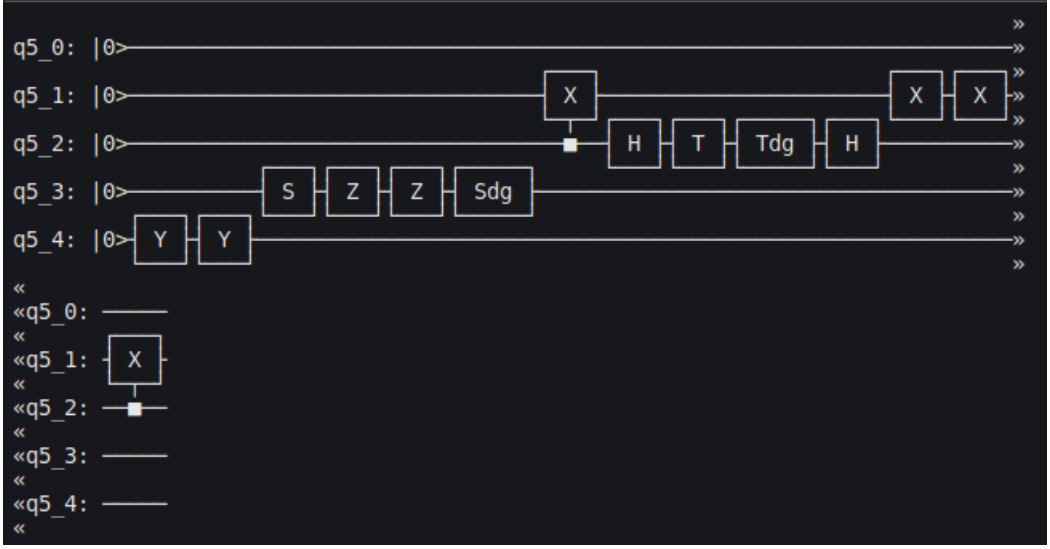
```
|"00"> 48%
|"11"> 52%
```

<u>Notice that in quantum computing, every gate is reversible.</u>

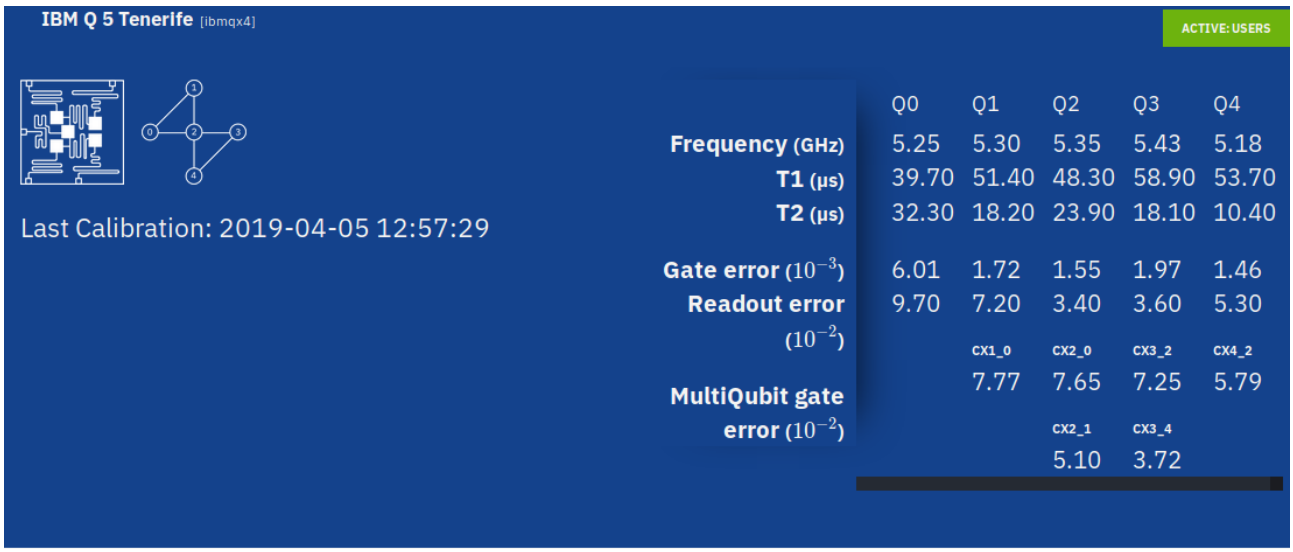
Example of reversible circuit from [1]:

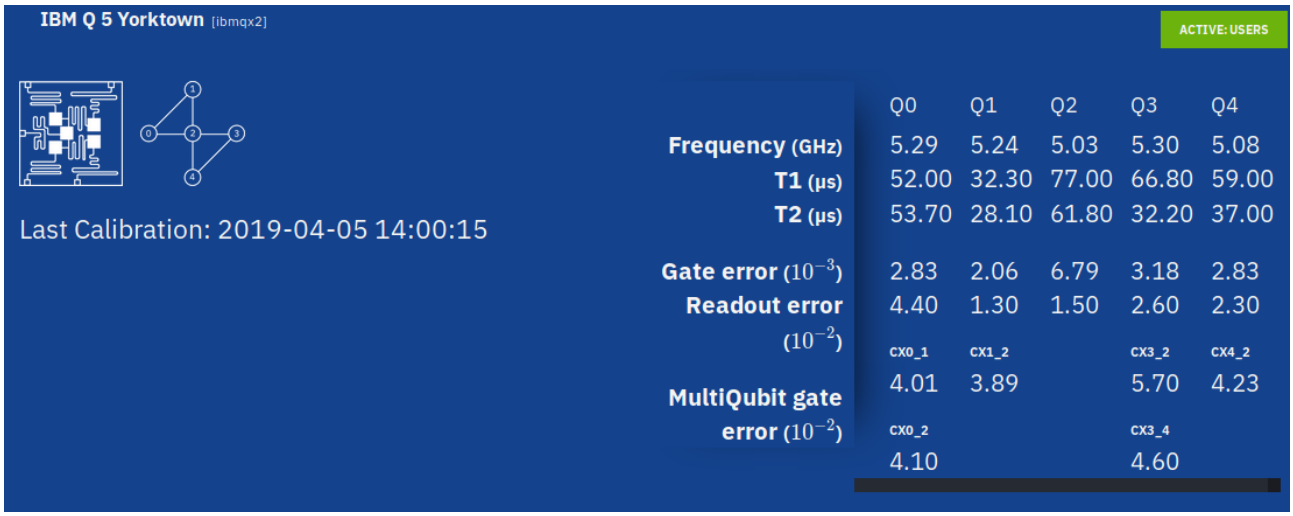
Reversible Circuit Example

```
qr = QuantumRegister(5)
cr = ClassicalRegister(5)
circuit = QuantumCircuit(qr)
circuit.cx(qr[2],qr[1])
circuit.x(qr[1])
circuit.h(qr[2])
circuit.s(qr[3])
circuit.y(qr[4])
circuit.z(qr[3])
circuit.t(qr[2])
circuit.tdg(qr[2])
circuit.z(qr[3])

circuit.y(qr[4])
circuit.sdg(qr[3])
circuit.h(qr[2])
circuit.x(qr[1])
circuit.cx(qr[2],qr[1])
circuit.draw()
```



Let us check the online IBM composer:





<u>T1 and T2 quantify how long a quantum computer can maintain coherence, that is how long we can compute without expecting errors from decoherence.</u>

Emulated the circuit on the IBM's Q platform. Given the recipe that I've followed, IBM prompted me to use the cached results. This is a great way of finding if someone else used your schematic :).

I'll insert the screenshot on the next page:

I'll stop here today. Tomorrow I will continue with OpenQASM.

6th of April

IBM QX headers:

```
OPENQASM 2.0;
include "qelib1.inc";
```

Quantum register initialization:

```
qreg q[1];
```

QASM code and representation of the gates [1]:

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[5];
creg c[5];
cx q[2],q[1];
x q[1];
h q[2];
s q[3];
y q[4];
t q[2];
z q[3];
tdg q[2];
```

z q[3];
x q[1];
h q[2];
sdg q[3];
y q[4];
cx q[2],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];

Resetting a qubit:

reset q[0];

Github repo for the referenced book by Christine Corbett:

https://github.com/PacktPublishing/Mastering-Quantum-Computing-with-IBM-QX

SAT:

https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

Results after running their Grover Algorithm implementation:

Testing two qubit quantum AND
inputs forward reverse

abc a'b'c' a"b"c"

000 000 000
001 001 001
010 010 010
011 011 011
100 100 100
101 101 101
110 111 110
111 110 111

Testing two qubit quantum OR
inputs forward reverse
abc a'b'c' a"b"c"
000 000 000
001 001 001
010 011 010
011 010 011
100 101 100
101 100 101
110 111 110
111 110 111

Testing three qubit quantum AND
inputs forward reverse
abcd a'b'c'd' a"b"c"d"
0000 0000 0000
0001 0001 0001
0010 0010 0010
0011 0011 0011
0100 0100 0100
0101 0101 0101
0110 0110 0110
0111 0111 0111
1000 1000 1000
1001 1001 1001
1010 1010 1010
1011 1011 1011
1100 1100 1100
1101 1101 1101
1110 1111 1111
1111 1110 1110

Testing three qubit quantum OR
inputs forward reverse
abcd a'b'c'd' a"b"c"d"
0000 0000 0000
0001 0001 0001
0010 0011 0010
0011 0010 0011
0100 0101 0100
0101 0100 0101

0110 0111 0110
0111 0110 0111
1000 1001 1000
1001 1000 1001
1010 1011 1010
1011 1010 1011
1100 1101 1100
1101 1100 1101
1110 1111 1110
1111 1110 1111

(0, 0, 0) -> 0
(0, 0, 1) -> 0
(0, 1, 0) -> 0
(0, 1, 1) -> 0
(1, 0, 0) -> 0
(1, 0, 1) -> 0
(1, 1, 0) -> 1
(1, 1, 1) -> 0

(0, 0, 0) -> 0 (measured bits: 00001001111111111011010)
(0, 0, 1) -> 0 (measured bits: 00100011111111111011010)
(0, 1, 0) -> 0 (measured bits: 01011110001111111101010)
(0, 1, 1) -> 0 (measured bits: 01111110100111111101010)
(1, 0, 0) -> 0 (measured bits: 10011111111010011110100)
(1, 0, 1) -> 0 (measured bits: 10111111111000111110100)
(1, 1, 0) -> 1 (measured bits: 11011111111111101111111)
(1, 1, 1) -> 0 (measured bits: 11111111111111100111100)

(0, 0, 0) -> 0 (measured bits: 00000000000000000000000)
(0, 0, 1) -> 0 (measured bits: 00100000000000000000000)
(0, 1, 0) -> 0 (measured bits: 01000000000000000000000)
(0, 1, 1) -> 0 (measured bits: 01100000000000000000000)
(1, 0, 0) -> 0 (measured bits: 10000000000000000000000)
(1, 0, 1) -> 0 (measured bits: 10100000000000000000000)
(1, 1, 0) -> 0 (measured bits: 11000000000000000000000)
(1, 1, 1) -> 0 (measured bits: 11100000000000000000000)

(0, 0, 0) -> 0 (measured bits: 00000000000000000000000)
(0, 0, 1) -> 0 (measured bits: 00100000000000000000000)
(0, 1, 0) -> 0 (measured bits: 01000000000000000000000)
(0, 1, 1) -> 0 (measured bits: 01100000000000000000000)
(1, 0, 0) -> 0 (measured bits: 10000000000000000000000)
(1, 0, 1) -> 0 (measured bits: 10100000000000000000000)
(1, 1, 0) -> 1 (measured bits: 11000000000000000000001)
(1, 1, 1) -> 0 (measured bits: 11100000000000000000000)

Running Grover's algorithm for 0 iterations

{'000000000000000000000001': 2, '000000000000000000000000': 10, '000000000000000000000100': 4, '000000000000000000000010': 3, '000000000000000000000101': 7, '000000000000000000000110': 7, '000000000000000000000111': 9, '000000000000000000000011': 8}

Running Grover's algorithm for 2 iterations
{'000000000000000000000001': 1, '000000000000000000000011': 48, '000000000000000000000010': 1}

Running Grover's algorithm for 1 iterations
{'000000000000000000000001': 3, '000000000000000000000000': 1, '000000000000000000000100': 1, '000000000000000000000010': 1, '000000000000000000000101': 2, '000000000000000000000111': 4, '000000000000000000000011': 38}

Running Grover's algorithm for 3 iterations
{'000000000000000000000001': 6, '000000000000000000000000': 3, '000000000000000000000100': 5, '000000000000000000000010': 5, '000000000000000000000101': 4, '000000000000000000000110': 7, '000000000000000000000111': 1, '000000000000000000000011': 19}

Quantum Fourier Transform:

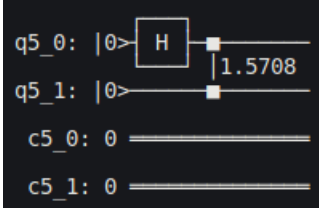
QFT one qubit

```
import qiskit
from qiskit import Aer
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
qr = QuantumRegister(1)
cr = ClassicalRegister(1)
circuit = QuantumCircuit(qr, cr)
circuit.h(qr[0])
circuit.measure(qr[0],cr[0])
# Run on local simulator
backend = Aer.get_backend('qasm_simulator')
sim = qiskit.execute(circuit,backend=backend)
result = sim.result()
print(result.get_counts(circuit))
```

```
{'1': 544, '0': 480}
```

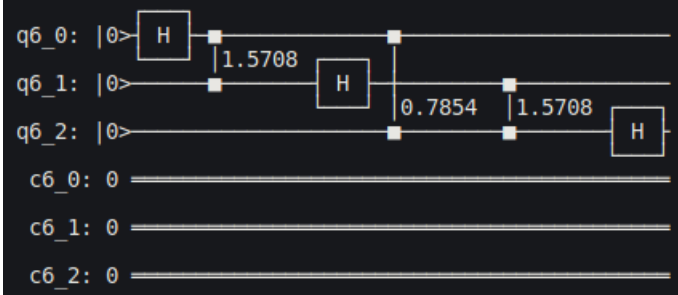
QFT two qubits

```
import math
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
qr = QuantumRegister(2)
cr = ClassicalRegister(2)
circuit = QuantumCircuit(qr, cr)
circuit.h(qr[0])
circuit.cu1(math.pi/2.,qr[1],qr[0])
circuit.draw()
```



QFT three qubits

```
import math
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
qr = QuantumRegister(3)
cr = ClassicalRegister(3)
circuit = QuantumCircuit(qr, cr)
circuit.h(qr[0])
circuit.cu1(math.pi/2.,qr[1],qr[0])
circuit.cu1(math.pi/4.,qr[2],qr[0])
circuit.h(qr[1])
circuit.cu1(math.pi/2.,qr[2],qr[1])
circuit.h(qr[2])
circuit.draw()
```



QFT N qubits

```
def qft(qr,qc,n_qubits):
    # Get the circuit and the quantum register by name
    for i in range(n_qubits):
        for j in range(i):
            qc.cu1(math.pi/float(2**(i-j)), qr[i], qr[j])
        qc.h(qr[i])
```

Shor's Algorithm transcript:

Choose an integer random number less than the integer *N* we wish to factor.
Compute the GCD between our random number and *N*:

> **If the GCD between our random number and *N* isn't one, great news—we've found a factor of *N*, which is the GCD we just computed. A second factor is the integer we want to factor divided by this GCD. The algorithm returns.**
> **Otherwise, if the GCD is one, then we consider the function: our random number to the *x* power modulo *N*, and compute its period (when the function repeats itself):**

If the period we find is odd, we're out of luck and need to choose another random number. We go back to step 1 and start over with a new random number.
Else if the period we find to the power of our random number divided by two modulo *N* is the same as *-1* modulo *N* (they are congruent modulo *N*), we are also out of luck, so go back to the very beginning of the algorithm, and start over with a new random number.
Else we are finally at a potential solution. We have found two factors of *N*.

- **The first factor we have found can be computed by computing the GCD between the following:**
  - ***N* the power of the period we found divided by two that quantity plus one, and N**
- **The second factor can be computed by taking the GCD (greatest common divisor) between the following:**
  - ***N* to the power of the period we found divided by two that quantity minus one, and N**
- **If these two factors are simply *N* and *1*, the solution isn't the one we are looking for, so we aren't done yet and go back to step 1 and choose a new random number. Otherwise, we're done and the algorithm returns the two factors we just found.**

Check out the following link: https://www.scottaaronson.com/blog/?p=208

Example:

Let's say $N = 837$. Let's trace through this example:

Pick a random integer a, so that $a < 837$. Our code picked 802 in this example.
Compute $g = gcd(a,N) = gcd(802,837) = 1$:
Since $g = 1$, we have to keep going!
Find the period of $f(x) = a^x \pmod N = 802^x \pmod{837}$. This period is $r = 30$:
$r = 30$ is even, so we keep going!
$a^{r/2} = 802^{30/2} \pmod{837} = 433$ and $433 \neq -1 \pmod{837} = 836$, so we keep going!
We've found two factors of 837. They are $gcd(a^{r/2} + 1, N) = gcd(802^{30/2} + 1837) = 31$ and $gcd(a^{r/2} + 1, N) = gcd(802^{30/2} - 1837) = 27$.

Christine Corbett's python implementation:

```
import math
import random
def shors_algorithm_classical(N):
```

```
assert(N>0)
assert(int(N)==N)
while True:
    a=random.randint(0,N-1)
    g=math.gcd(a,N)
    if g!=1 or N==1:
        first_factor=g
        second_factor=int(N/g)
        return first_factor,second_factor
    else:
        r=period_finding_classical(a,N)
        if r % 2 != 0:
            continue
        elif a**(int(r/2)) % N == -1 % N:
            continue
        else:
            first_factor=math.gcd(a**int(r/2)+1,N)
            second_factor=math.gcd(a**int(r/2)-1,N)
            if first_factor==N or second_factor==N:
                continue
            return first_factor,second_factor
```

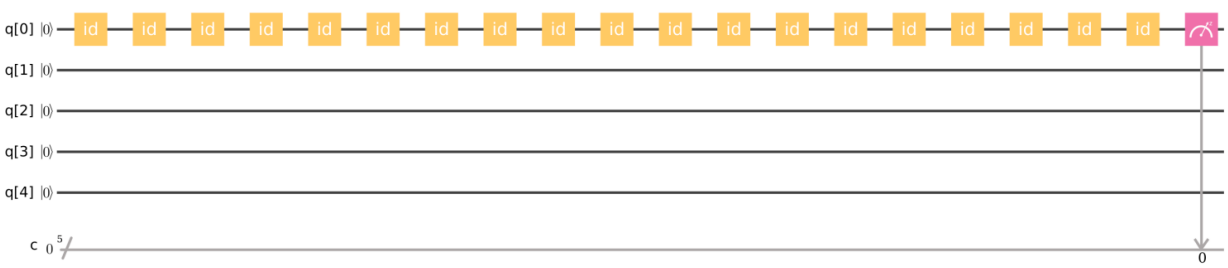
I'll stop here for today.

7th of April

Quantum error correction algorithms (QEC):

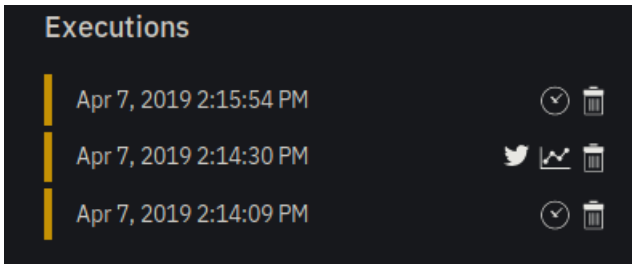
"*T_1 helps quantify how quickly the qubits on a given hardware experience energy loss due to environment interaction (energy loss results in a change in frequency, causing decoherence), which can cause a bit flip, and T_2 helps quantify how quickly the qubits experience a phase difference due to interaction with the environment, again a cause of decoherence. The bigger T_1 and T_2, the more robust a quantum computation will be to errors.*" [1]

Quantum error proof – test circuit:



Funny enough, on this execution I got 100 % accuracy. Will try with 1o this time. Still 100 % .

Let's try 30. Haha, I was dumb enough to not realize that I was simulating them, not actually running on the quantum computer. **All my queries are pending execution on the real machine**.



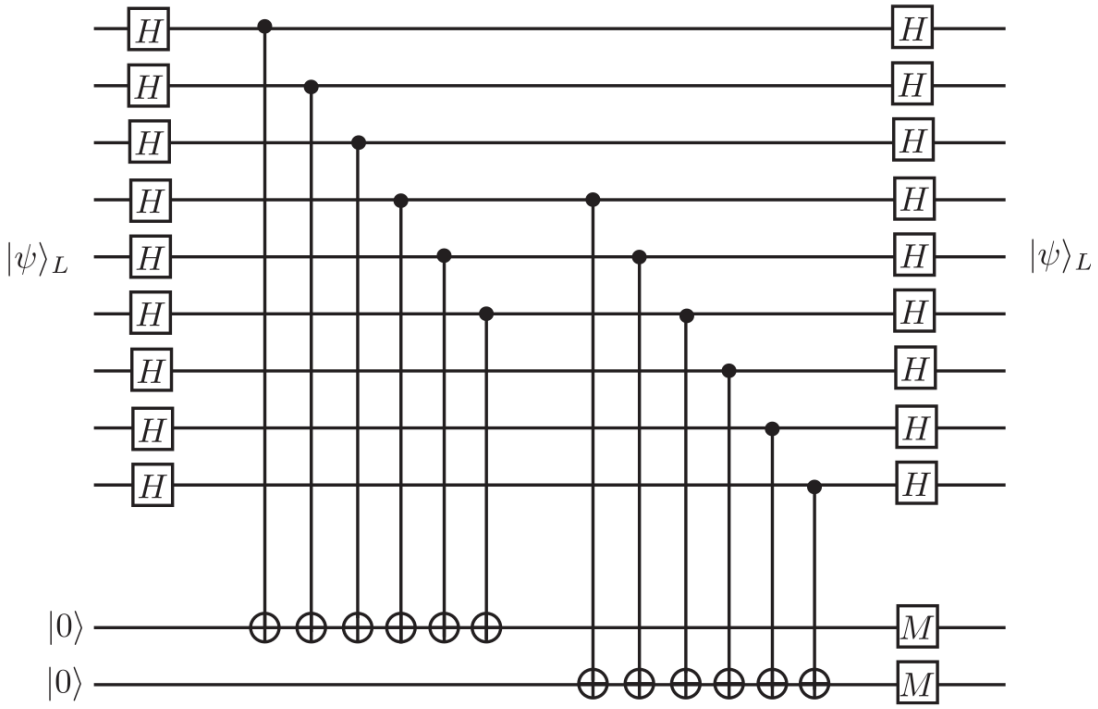
I really hope that they will finish executing today.

Discovered that Qiskit Aer offers the possibility of simulating noise, thus emulating a real execution in a simulated environment.

"*If we use qubits to correct errors in other qubits, the error correcting qubits are themselves subject to errors. We might be worried that the errors might accumulate, resulting in there being no way to effectively error correct a quantum computation in general, but luckily this is not the case. An important concept in quantum computing is the* ***threshold theorem****, which shows that* ***with enough error correcting qubits, we can always create an accurate computation, so long as the error in the gates is below a particular threshold****. This threshold varies depending on how the error in the gates is measured as well as the choice of universal gates, but the important thing is that the theorem shows that it is possible to correct errors faster than creating them under the right conditions. The threshold theorem shows that, although the error correcting qubits are themselves subject to quantum error, they too can be error corrected with other qubits, and their error correcting qubits can themselves be error corrected, and so on (in logarithmic levels). Since each level of error correction requires fewer error correcting qubits than the original number of qubits in the computation, eventually we reach a point where all the levels of qubits work together to sufficiently error correct the original qubits.*

Classical error correction involves duplication of information so that if part of it is corrupted by noise, the true value can be recovered. Quantum error correction can't rely on that strategy as ***qubits can't be copied****. One strategy for quantum error correction, which we will outline in this chapter, involves spreading out the quantum information from one qubit to multiple qubits, and then performing measurements that can diagnose a potential error without affecting the quantum state. Apart from a few simple cases, a quantum error correction code will not restore exactly the original state. Instead, the quantum error correction code increases the probability of obtaining the correct result.*" [1]

Here is a screenshot of a circuit required to perform phase correction for a 9 qubit code[4]:



Another good snippet from [4]:

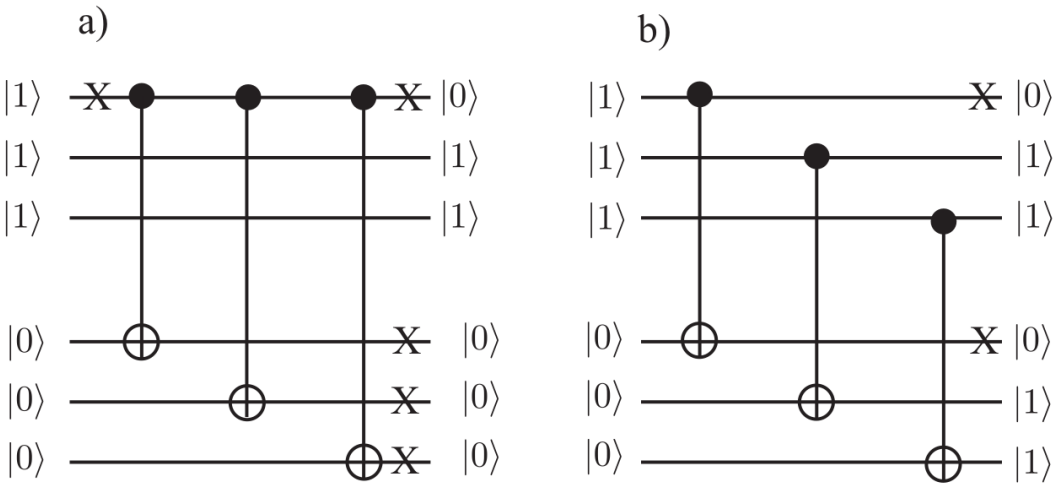


FIG. 10 Two circuits to implement the transformation $|111\rangle|000\rangle \rightarrow |111\rangle|111\rangle$. a) shows a version where a single X error can cascade into four errors while b) shows an equivalent circuit where the error only propagates to a second qubit.

Will also attach their table of transformation:

$K^i \otimes K^j$	K^1	K^2	K^3	K^4	K^5	K^6
K^1	$K^1 \otimes I$	$K^1 \otimes K^1K^2$	$K^1 \otimes K^1K^3$	$K^1K^4 \otimes K^1K^4$	$K^1K^5 \otimes K^1K^5$	$K^1K^6 \otimes K^1K^6$
K^2	$K^2 \otimes K^1K^2$	$K^2 \otimes I$	$K^2 \otimes K^2K^3$	$K^2K^4 \otimes K^2K^4$	$K^2K^5 \otimes K^2K^5$	$K^2K^6 \otimes K^2K^6$
K^3	$K^3 \otimes K^3K^1$	$K^3 \otimes K^3K^2$	$K^3 \otimes I$	$K^3K^4 \otimes K^3K^4$	$K^3K^5 \otimes K^3K^5$	$K^3K^6 \otimes K^3K^6$
K^4	$K^4 \otimes K^1$	$K^4 \otimes K^2$	$K^4 \otimes K^3$	$I \otimes K^4$	$K^4K^5 \otimes K^5$	$K^4K^6 \otimes K^6$
K^5	$K^5 \otimes K^1$	$K^5 \otimes K^2$	$K^5 \otimes K^3$	$K^5K^4 \otimes K^4$	$I \otimes K^5$	$K^5K^6 \otimes K^6$
K^6	$K^6 \otimes K^1$	$K^6 \otimes K^2$	$K^6 \otimes K^3$	$K^6K^4 \otimes K^4$	$K^6K^5 \otimes K^5$	$I \otimes K^6$

TABLE V Transformations of the $[[7,1,3]]$ stabilizer set under the gate operation $U = \text{CNOT}^{\otimes 7}$, where $\mathcal{G} \rightarrow U^\dagger \mathcal{G} U$. Note that the transformation does not take any stabilizer outside the group generated by $K^i \otimes K^j$ $(i,j) \in [1,..,6]$, therefore $U = \text{CNOT}^{\otimes 7}$ represents a valid operation on the codespace.

We will thoroughly analyze those papers in depth later on anyway, but I just want to save a couple of them here for quick reference.

Will save here the key concepts of qc from [1]:

Key concepts of quantum computing

Some key concepts under quantum computing are described as follows:

- **Quantum computers**: These exploit the properties of quantum physics, superposition and entanglement, to perform some computations much more efficiently than on a classic computer.
- **Qubit**: This is a quantum piece of information used in quantum computing that represents one possible combination of two values. Manipulating qubits of information is what powers many modern quantum computers, including IBM QX.
- **Quantum state**: A quantum state is a grouping of one or more qubits.
- **Entanglement**: When qubits that compose a quantum state are related, they are entangled. Entanglement is key to the power of quantum computing.
- **Coherence**: Coherence is the property of waves, and if waves are coherent they can in some senses "work together." Qubits interact with the environment and this process causes the correlations they had due to coherence to be lost, which hinders the ability to perform quantum computation. The longer the coherence time of a quantum computer, the better it will function.
- **Quantum gate**: In the gate model of quantum computing, taking an input quantum state and transforming it in the desired way is done with quantum gates. The right set of gates can perform any quantum computation. In IBM QX, these are called the I, X, Y, Z, H, S, $S^\dagger$, T, $T^\dagger$ and CNOT gates.
- **Quantum circuit**: This is a quantum circuit is a sequence of one or more quantum gates grouped together.
- **Quantum circuit diagram**: A method to visualize quantum circuits. The IBM QX Quantum Composer is a way to program IBM QX using the quantum circuit visualization.
- **IBM QX programming**: Can be programmed via the quantum composer, a way of visually representing quantum circuits. Can be programmed via a Python API interacting with the

computer and simulation tools via the Python qiskit module. Can be programmed with a general purpose quantum programming language called OpenQASM.

- **Quantum algorithms**: Grover's algorithm works to efficiently find the one solution to a function out of all possible inputs. Shor's algorithm factors numbers. The Quantum Fourier Transform is a useful subroutine of many quantum algorithms, including Shor's algorithm.
- **Quantum supremacy**: This is when a quantum computing algorithm offers significant speedup compared to the best known algorithm on a classic computer, and this speedup is demonstrated on a quantum computer. As of the writing of this book, many algorithms offer the promise of quantum supremacy, but the hardware of quantum computing lags behind the theory and quantum supremacy has yet to be clearly demonstrated.

I enjoyed Christine's closing line:

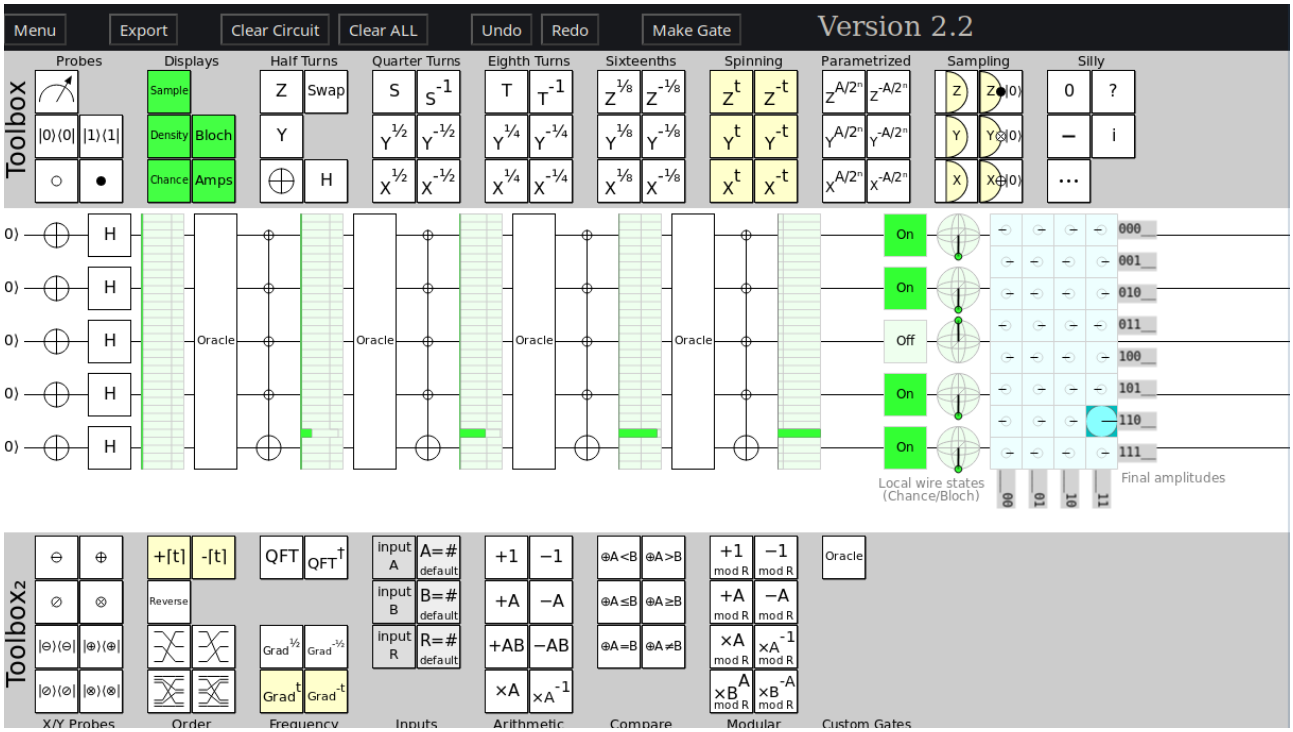
"*Although it is a general concern that the field is overhyped, some experts are optimistic that the field in general is so close to being useful that the hype is warranted.*"

Additional resources:

http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.eecs.umich.edu/~jhayes/JPH_DACslides_Jun03.pdf

[6] is a very good read indeed. Highly recommended.

Found another good website for circuit generation. Grover search implementation shown:



https://github.com/Strilanc/Quirk/wiki/How-to-use-Quirk

I'll save me a note from [14]: "*Noise and Decoherence:Noise and decoherence are key obstacles in implementing quantum computers devices. Recent results show that theoretically, fault tolerant computation exists[11, 1, 10, 9]. Still, in the task of realizing quantum computers, it is likely that more theoretical work will be needed in this direction .A key problem in this inter-face between quantum physics and quantum computation models is the fact that quantum noise, and in particular, decoherence, are non-unitary operations that cause a pure state to become a mixed state. Incorporating quantum noise, which was impossible in the standard unitary model, is naturally done in our model.*"

I highly recommend taking your time to read about Quipper [28].

As of 5:55 PM, my simulations are still pending execution on the quantum device. I'll stop here today.

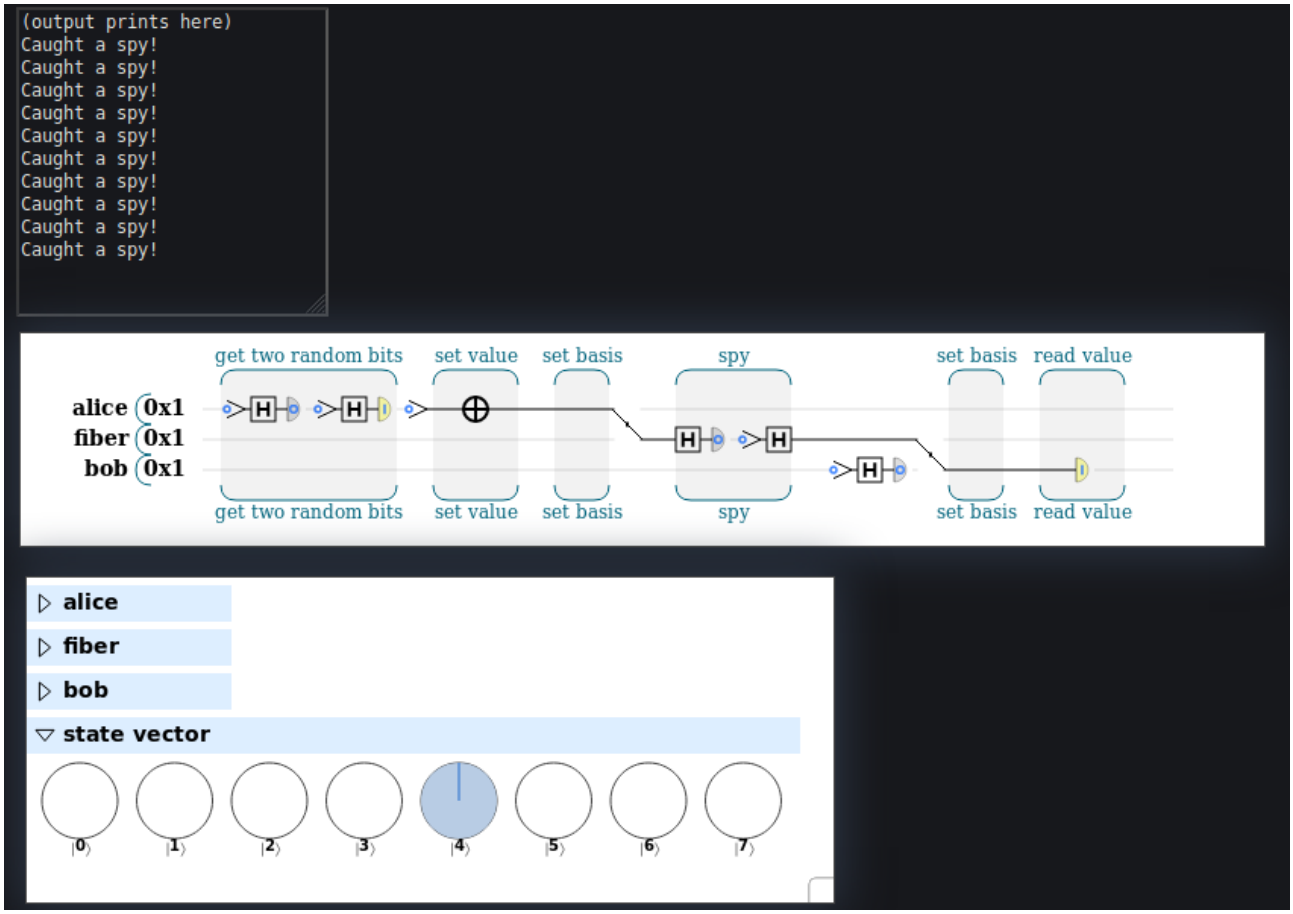
8th of April

Started with a great example from [31]:

```
// Run this sample online: http://oreilly.machinelevel.com?p=ex_1qubit_spy


qc.reset(3);
qc.discard();
a = qint.new(1, 'alice');
fiber = qint.new(1, 'fiber');
b = qint.new(1, 'bob');

function random_bit(q) {
   q.write(0);
   q.had();
   return q.read();
}

// Generate two random bits
qc.codeLabel('get two random bits');
send_basis = random_bit(a);
send_value = random_bit(a);
qc.codeLabel('');

// Prepare Alice's qubit
a.write(0);
qc.codeLabel('set value');
qc.nop();
if (send_value)
   a.not();
qc.nop();
qc.codeLabel('');
```

```
qc.nop();
qc.codeLabel('set basis');
qc.nop();
if (send_basis)
    a.had();
qc.nop();
qc.codeLabel('');

// Send the qubit!
fiber.exchange(a);

// Activate the spy
var spy_is_present = true;
if (spy_is_present) {
    var spy_basis = 1;
    qc.nop();
    qc.codeLabel('spy');
    if (spy_basis)
        fiber.had();
    stolen_data = fiber.read();
    fiber.write(0);
    if (stolen_data)
        fiber.not();
    if (spy_basis)
        fiber.had();
    qc.codeLabel('');
    qc.nop();
}

// Receive the qubit!
recv_basis = random_bit(b);
fiber.exchange(b);
qc.codeLabel('set basis');
qc.nop();
if (recv_basis)
    b.had();
qc.nop();
qc.codeLabel('');
qc.nop();
qc.codeLabel('read value');
qc.nop();
recv_val = b.read();
qc.nop();
qc.codeLabel('');
qc.nop();

// Now Alice emails Bob to tell
// him her basis and value.
// If the basis matches and the
// value does not, there's a spy!
```

```
if (send_basis == recv_basis)
   if (send_value != recv_val)
      qc.print('Caught a spy!\n');
```



The basic idea is that if a spy is eavesdropping on the communication channel, it automatically changes the qubit's state, thus giving away his presence.

This will come into play later on when quantum cryptography will be more widely deployed.

Speaking of quantum cryptography, here is a good article from 2010 explaining one attack vector against such communication systems:

http://www.vad1.com/lab/hacking-commercial-quantum-cryptography-2010/

The lab's official page – including news regarding one of the commercial prototypes:

http://www.vad1.com/lab/

http://news.ifmo.ru/en/science/photonics/news/7084/

Very quality presentation:

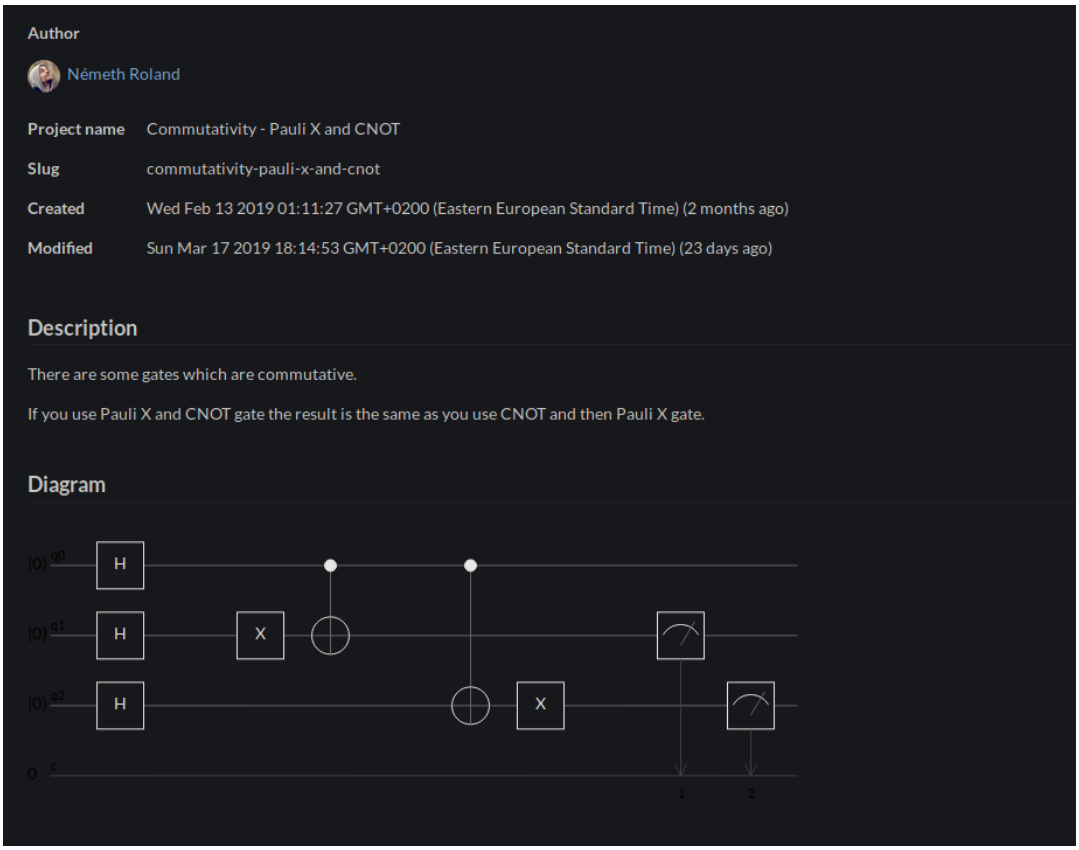
https://www.youtube.com/watch?v=Z5TcRS1zlio

Another one: https://www.youtube.com/watch?v=wF-BWgnpYmI

9th of April

Today we will explore additional QC simulators available online.

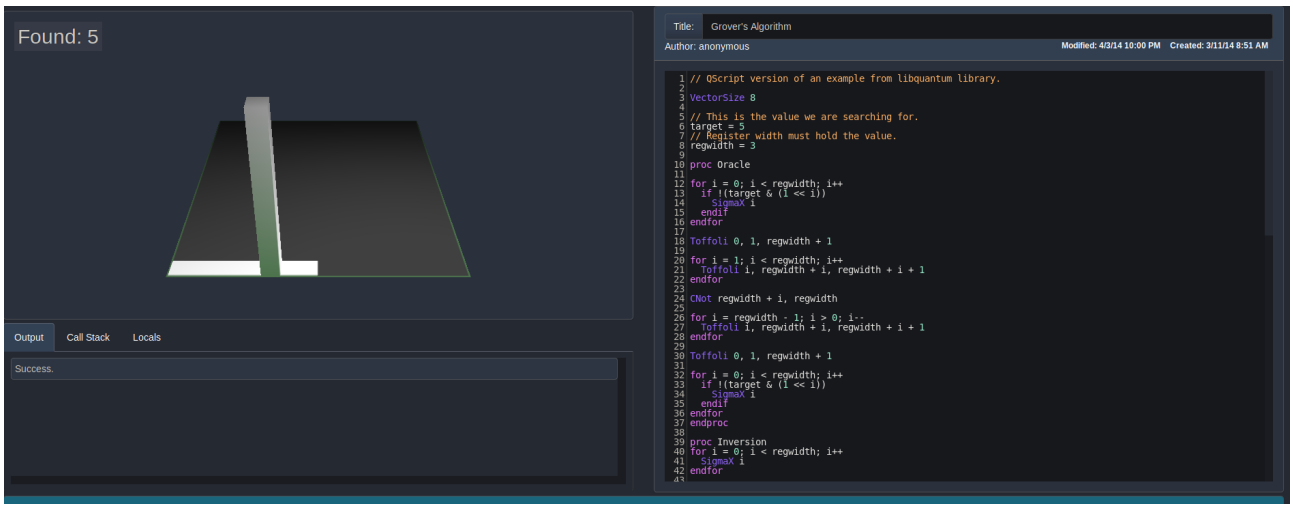
First one: quantum-circuit.com



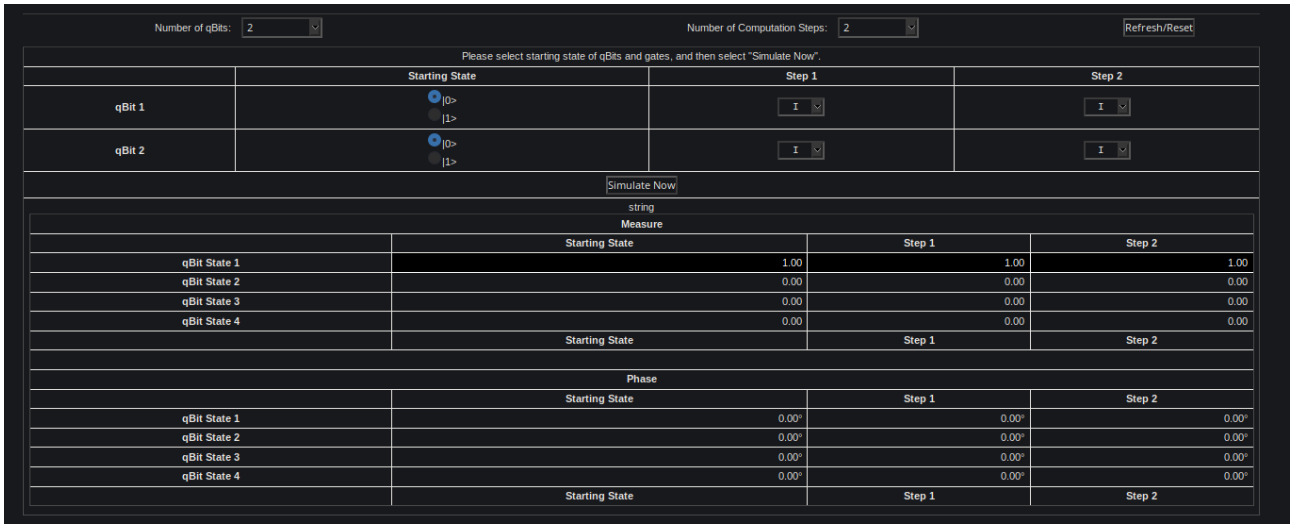
Yesterday we discussed about the "spy eavesdropping" case. Here is a related paragraph explaining the phenomenon in scientific terms[33]:

> One particular watched pot effect that occurs in quantum systems is the following. Suppose a system, known to occupy at time t a quantum state Ψ_1, is left alone over the time interval $[t, t+\Delta t]$, and is then measured at time $t+\Delta t$. For such a system, we call $P_{1\leftarrow 1}(\Delta t)$ the probability that at time $t+\Delta t$, the system will be found to be still lying in state Ψ_1. Suppose that $P_{1\leftarrow 1}(\Delta t) \to 1$ continuously as $\Delta t \to 0$ and that τ is a positive real number that is small enough for $P_{1\leftarrow 1}(\tau)$ to be fairly close to 1. Suppose that the system is measured at times $t = \frac{i}{N}\tau$ for $i = 1, 2, \ldots, N$, where N is some positive integer. The probability that the system will be found to be in state Ψ_1 in every one of those N measurements is then $[P_{1\leftarrow 1}(\frac{\tau}{N})]^N$. As we show in the proof below, this probability tends to 1 as N tends to infinity. Thus, each measurement exerts a small influence on the system. These influences incline the system not to stray far away from its initial state Ψ_1. It's not clear who was the first person to discover this effect. The authors of Ref.[3] were the first to call it "Quantum Zeno's Paradox". The effect has been observed experimentally[4] in atoms executing induced transitions between energy levels.

quantumplayground.net :



Quantum computer simulator (University of Patras):



Python libraries. The most promising so far seems to be Cirq:

https://github.com/quantumlib/Cirq

I also signed up on Rigetti.com . Their SDK sounds promising.

http://forest.rigetti.com/

Besides Cirq, projectQ is also on my list.

https://github.com/ProjectQ-Framework/ProjectQ

I am also adding qubiter:

https://github.com/artiste-qb-net/qubiter

The last one I'm adding to the list (for now) is qutip.

https://github.com/qutip/qutip

I'm stopping here.

11th of April

The execution of my previous circuits on IBM QX is still pending.

On error correction [35]:

"Our architecture achieves system-level efficiencies through code teleportation, quantum memory refresh units, dynamic compilation of quantum programs, and scalable error correction. Our work indicates that reliability of the underlying technology is crucial; **practical architectures will require quantum technologies with error rates between 10^{-6} and 10^{-9}**."

"We conducted this work as part of the Quantum Architecture Research Center (http://feynman.media.mit.edu/quanta/qarc/), funded by the DARPA Quantum Information Science and Technology program. John Kubiatowicz contributed to the initial discussions of this work. John Black provided comments on the state of the art in breaking cryptosystems. Thanks also to Matt Farrens and Gary Tyson"

A great thought provoking paper [37]: "In conclusion, we have proved a no-go theorem that shows **that an unknown arbitrary unitary cannot be controlled in a quantum circuit**, even modulo a global phase. This control is, however, possible for any physical implementation of a unitary transformation. This shows that the language of quantum circuits should be extended in order to capture all information processing possibilities allowed by quantum physics."

ScaffCC is added to the list as well [41] : https://github.com/epiqc/ScaffCC

Today I will finally decide Tigon's objective: **Tigon will be a quantum-based random number generator.**

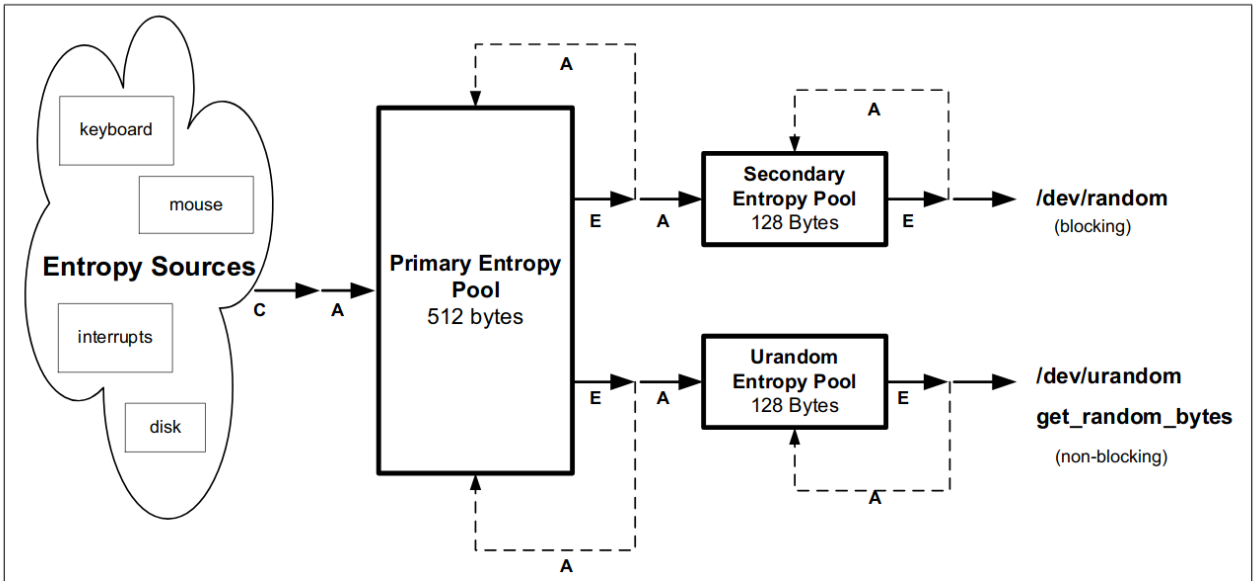
I think this is the only realistic objective given the current progress in this field. I guarantee you that it will be great fun :) .

Let us proceed with the additional research in this direction.

"What do we want from a pseudo-random sequence generator? Ideally, we would like a pseudo-random sequence generator to quickly produce, from short seeds, long sequences (of bits) that appear in every way to be generated by successive flips of a fair coin." [47]

On the Linux Random Number Generation [56]: "The LRNG is part of an open source project and therefore one might assume that its source code is available for public scrutiny and that its security can be easily analyzed (or at least, is not based on "security by obscurity"). However, the LRNG is not well documented and there is no clear description of the implemented algorithm. The LRNG is composed of about 2500 lines of code, and in addition, hundreds of code patches were applied to the code during the last five years (and consequently, the available documentation does

not always reflect the current code). One example of the complexity of the LRNG code is the fact that for 17 months the LRNG code included a bug in which entropy addition used a vector of size 4×n instead of n. We also note that throughout our analysis we were not helped by any of the LRNG authors. “



[56]

Keyboard	Mouse	Hard Drive	Interrupts
8	12	3	4

Table 1: The number of unknown bits in operating system events.

[56]

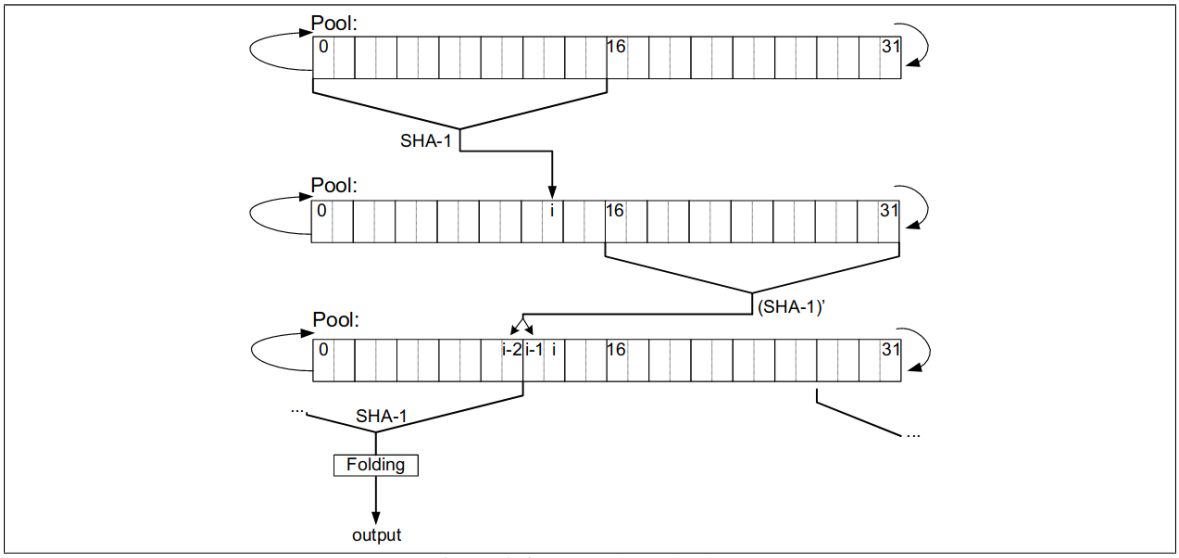
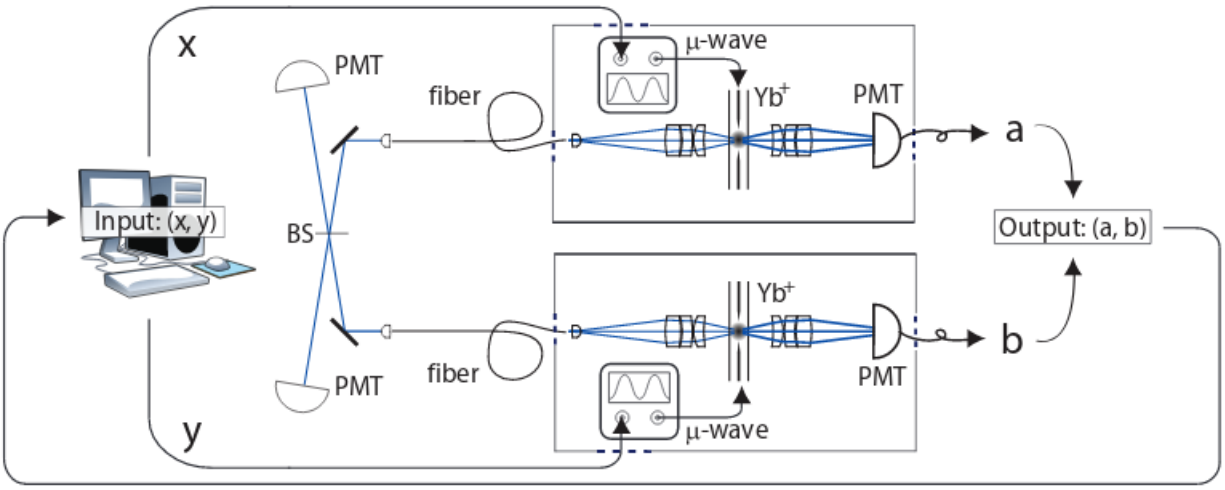


Figure 2.4: Extraction algorithm

[56]

“Quantum theory is, on the other hand, fundamentally random; yet, in any real experiment the intrinsic randomness of quantum systems is necessarily mixed-up with an apparent randomness that results from noise or lack of control of the experiment. It is therefore unclear how to certify or quantify unequivocally the observed random behaviour even of a quantum process.” [59]



[59]

[60] got me thinking about: https://www.youtube.com/watch?v=CeCePH_HL0g

https://www.youtube.com/watch?v=EF5Wvi_Iiy4

https://www.youtube.com/watch?v=JnlkKdDXk-I

https://www.youtube.com/watch?v=aVvS_tWel54

15th of April

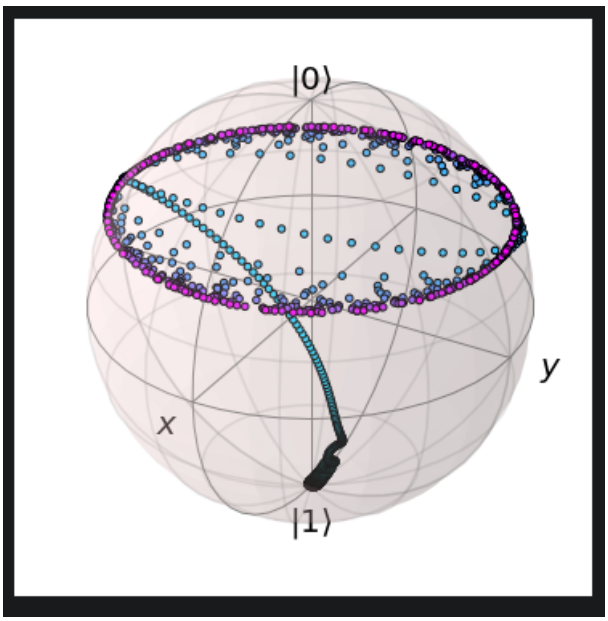
Today I will decide which library I will use. I have 4 on the list:

- ProjectQ
- Qubiter
- Qutip
- Cirq

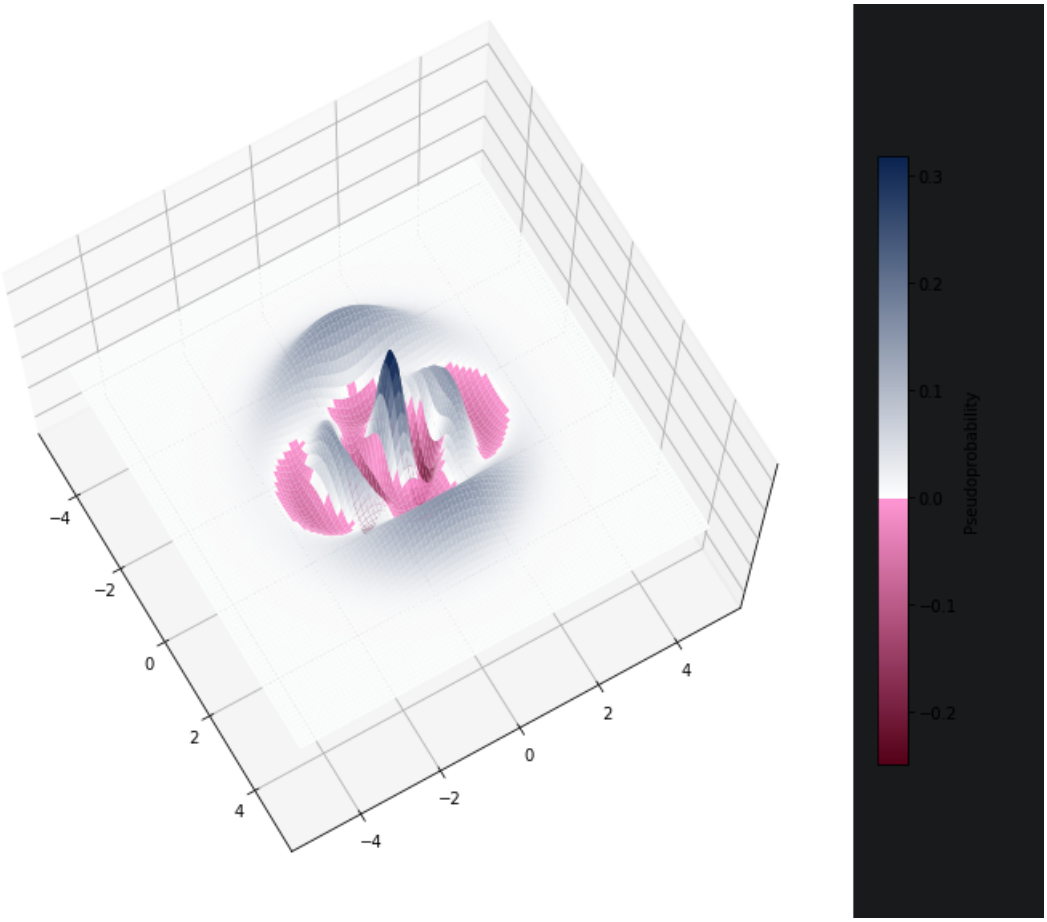
I have eliminated Cirq. 3 left on the table.

Took a gander at Qutip’s notebooks.

Boch Sphere generation:

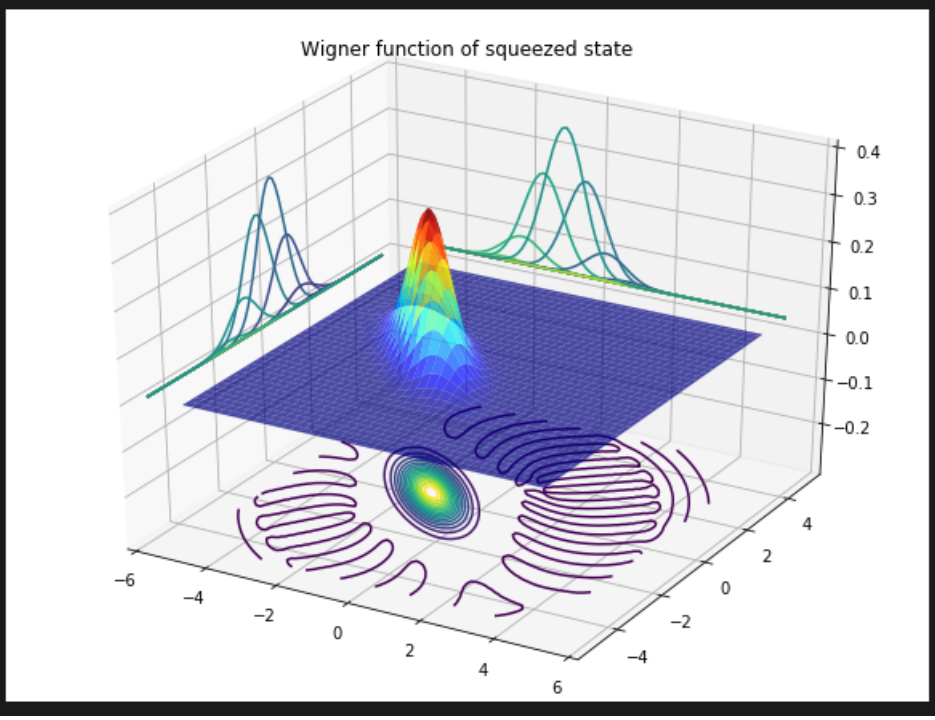


What interested me the most was the pseudo probability functions examples:



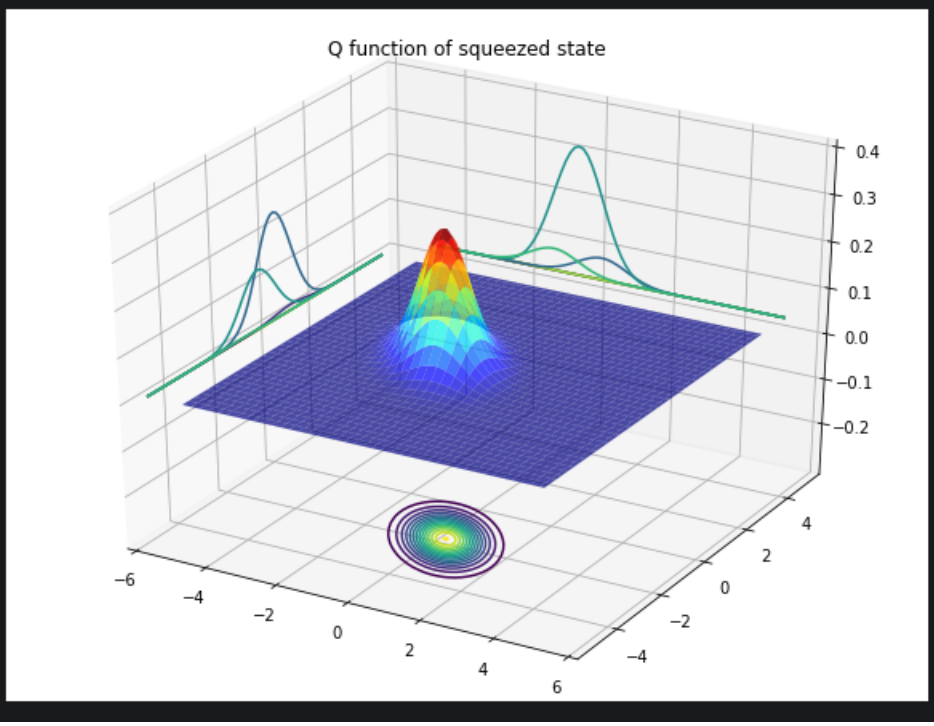
Wigner function

```
xvec = np.arange(-40.,40.)*5./40
X,Y = np.meshgrid(xvec, xvec)

W = wigner(psi, xvec, xvec)

fig1 = plt.figure(figsize=(8,6))
ax = Axes3D(fig1)
ax.plot_surface(X, Y, W, rstride=2, cstride=2, cmap=cm.jet, alpha=0.7)
ax.contour(X, Y, W, 15,zdir='x', offset=-6)
ax.contour(X, Y, W, 15,zdir='y', offset=6)
ax.contour(X, Y, W, 15,zdir='z', offset=-0.3)
ax.set_xlim3d(-6,6)
ax.set_xlim3d(-6,6)
ax.set_zlim3d(-0.3,0.4)
plt.title('Wigner function of squeezed state');
```



Q-function

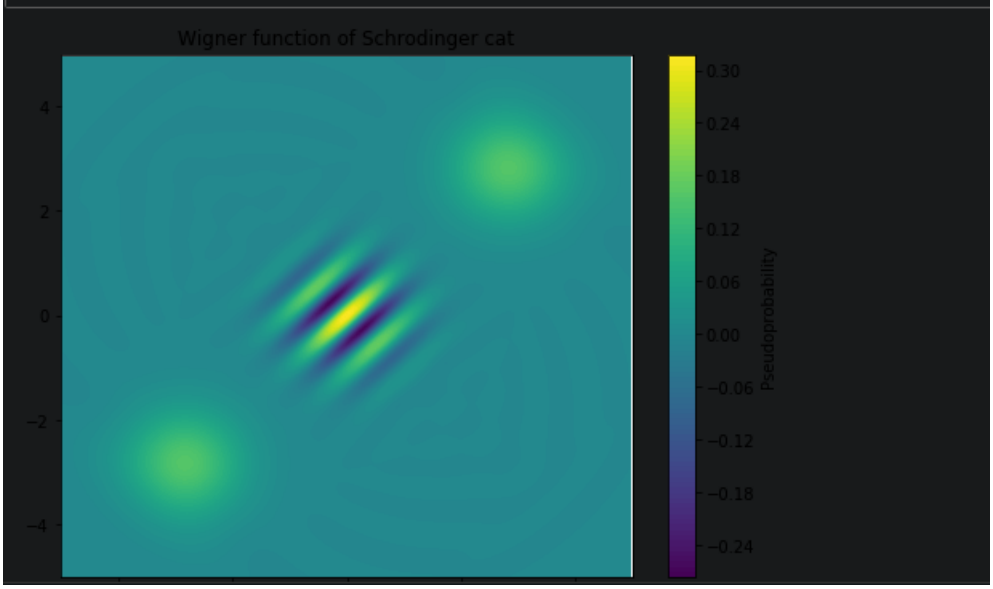
```
Q = qfunc(psi, xvec, xvec, g);

fig2 = plt.figure(figsize=(8,6))
ax = Axes3D(fig2)
ax.plot_surface(X, Y, Q, rstride=2, cstride=2, cmap=cm.jet, alpha=0.7)
ax.contour(X, Y, Q,zdir='x', offset=-6)
ax.contour(X, Y, Q,zdir='y', offset=6)
ax.contour(X, Y, Q, 15,zdir='z', offset=-0.4)
ax.set_xlim3d(-6,6)
ax.set_xlim3d(-6,6)
ax.set_zlim3d(-0.3,0.4)
plt.title('Q function of squeezed state');
```



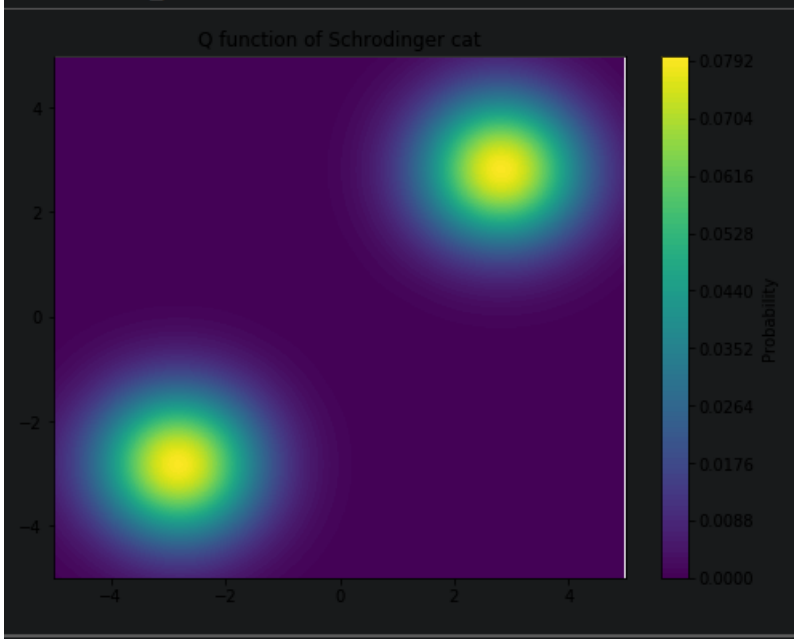
Schrodinger cat state

```
N = 20;
#amplitudes of coherent states
alpha1=-2.0-2j
alpha2=2.0+2j
#define ladder oeprators
a = destroy(N);
#define displacement oeprators
D1=(alpha1*dag(a)-np.conj(alpha1)*a).expm()
D2=(alpha2*dag(a)-np.conj(alpha2)*a).expm()
#sum of coherent states
psi = np.sqrt(2)**-1*(D1+D2)*basis(N,0); # Apply to vacuum state
```

```
#calculate Wigner function
yvec = xvec = np.arange(-100.,100.)*5./100
g=2.
W = wigner(psi, xvec, yvec)
fig = plt.figure(figsize=(8,6))
c = plt.contourf(xvec, yvec, np.real(W), 100)
plt.xlim([-5,5])
plt.ylim([-5,5])
plt.title('Wigner function of Schrodinger cat')
cbar = plt.colorbar(c)
cbar.ax.set_ylabel('Pseudoprobability');
```



```
#calculate Q function
Q = qfunc(psi,xvec,yvec)
fig = plt.figure(figsize=(8,6))
qplt = plt.contourf(xvec, yvec, np.real(Q), 100)
plt.xlim([-5,5])
plt.ylim([-5,5])
plt.title('Q function of Schrodinger cat')
cbar = plt.colorbar(qplt)
cbar.ax.set_ylabel('Probability');
```



BTW, I have discovered http://artiste-qb.net/bayesforge/

It is available on AWS: https://aws.amazon.com/marketplace/pp/B06Y6BNHD3

“Bayesforge integrates Python, R and Octave kernels which can be used interchangeably within a notebook with "Jupyter Magic". (The bash kernel is also included, because it makes for neat utility notebooks).

This image is ideal for Bayesian Big Data and Quantum Computing practitioners, but can also serve as a basis for Deep Learning.

The major software packages for Classical Bayesian Analytics are bnlearn (R), pyStan, pyMC, BNT (Octave), pyTorch and Tensor Flow.

All major Open Source development environments that allow for redistribution are included in this image. This includes our own software, Quantum-Fog and Qubiter, as well as IBM's QISKit products, D-Wave's QBSolve and chimera-embedding software, as well as Rigetti Computers, PyQuil and Grove products.”

It’s definitely worth your time.

I have decided to settle for qutip.

16th if April

The primary property of quantum computing that Tigon will use is reversibility of the circuit.

Started researching in this direction [64]:

"Reversible hardware computation, that is, performing logic signal transformations in a way that allows the original input signals to be recovered from the produced outputs, is helpful in diverse areas such as quantum computing, low-power design, nanotechnology, optical information processing, and bioinformatics. We propose a paradigm for performing such reversible computations in a manner that renders a wide class of circuit faults readily detectable at the circuit's outputs. More specifically, we introduce a class of reversible logic gates (consisting of the well-known Fredkin gate and a newly defined Feynman double-gate) for which the parity of the outputs matches that of the inputs. Such parity-preserving reversible gates, when used with an arbitrary synthesis strategy for reversible logic circuits, allow any fault that affects no more than a single logic signal to be detectable at the circuit's primary outputs. We show the applicability of our design strategy by demonstrating how the well-known, and very useful, Toffoli gate can be synthesized from parity-preserving gates and apply the results to the design of a binary full-adder circuit, which is a versatile and widely used element in digital arithmetic processing"

"Conventional logic is not reversible. For example, knowing the values of both A∨B and A∧B is inadequate for deducing the values of the inputs A and B. Reversible logic, which allows the reproduction of the circuit's inputs from observed outputs [Benn73], [Haye06], finds applications in quantum computing [Rief00], low-power design, nanotechnology, optical information processing, and bioinformatics. Elaborating on the relevance of reversible logic to low-power circuits, we note that the loss of 1 bit of information dissipates at least kTln2 joules of energy, where k=1.3806505×10–23m2kgs–2K–1 (joule/kelvin) is the Boltzmann constant and T is the operating temperature [Land61]. Thus, the only way to avoid power waste, and hence to curtail the production of heat, is to build circuits from reversible elements. "

"Any reversible gate has the same number of input and output lines, and it implements a permutation from input values to output values. For example, a 3-input, 3-output reversible gate may permute input values 0,1,2,3,4,5,6,7 to output values 0,1,4,6,2,3,5,7, corresponding to a permutation definable by its cycle structure (2 4)(3 6 5). Neither feedback nor fan-out is allowed in reversible logic. Consequently, making a reversible circuit robust or fault-tolerant is much more difficult than a conventional logic circuit. For example, the commonly used method of duplication and result checking via comparison implies the additional overhead of accommodating input fan-out, to feed the two copies of the circuit, and output fan-out, to allow comparison, besides requiring a reversible comparator. "

"With reversible logic, the use of standard methods of error detection presents some problems, given the requirement for fan-out and the associated increase in "garbage bits," that is, extra bits that are produced to maintain the reversibility property, without themselves being a necessary part of the computation itself or of its final results. If computation is performed in such a way that the parity of the input data persists throughout the computation (and, thus, at the output), no intermediate checking would be required. Such results can be forwarded to subsequent modules on the data path, while they are checked in a manner that is off the computation's critical path, and thus not subject to stringent performance or reversibility requirements. Any erroneous result tends to

propagate through the downstream modules without a danger of corrupting additional information in the absence of multiple compensating faults. ”

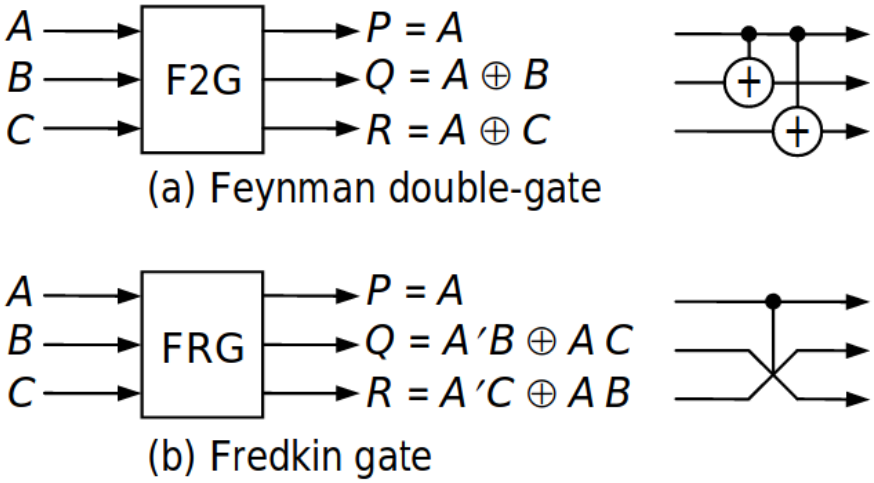


Fig. 2. Parity-preserving reversible gates, satisfying $A \oplus B \oplus C = P \oplus Q \oplus R$: Block diagrams and symbolic forms.

[64]

Table I. Potential truth tables for a 3-input, 3-output, parity-preserving gate with $P = A$.

A	B	C	$P = A$	Q	R
0	0	0	0	a	a
0	0	1	0	b	b'
0	1	0	0	b'	b
0	1	1	0	a'	a'
1	0	0	1	c	c
1	0	1	1	d	d'
1	1	0	1	d'	d
1	1	1	1	c'	c'

[64]

On reversible computation - “Take for example an AND gate. Suppose its output is |0〉. Can we infer what its inputs were? The answer is no — they could have been |00〉, |01〉, or |10〉. The AND process is not reversible: information sometimes needs to be deleted; “entropy” is lost. According to the 2nd Law of Thermodynamics, a physical system consisting of a single AND gate cannot be “closed”; its operation must dissipate some energy — typically as escaping heat. On the

other hand, a NOT gate is theoretically “reversible”: its output can be determined from its input; no information is created or destroyed in switching|0〉 to a|1〉 or vice versa. Thus, in principle, it is possible to construct a completely closed physical system implementing a NOT gate, without the need for energy dissipation.” [67]

Tigon will make use of the Pauli-X gate as the primary building block of our circuit.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Take a look at some random number generators attempts:

https://www.youtube.com/watch?v=GLUFMd0kLvU

https://makezine.com/projects/really-really-random-number-generator/

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf

https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software

https://www.youtube.com/watch?v=4sYawx70iP4

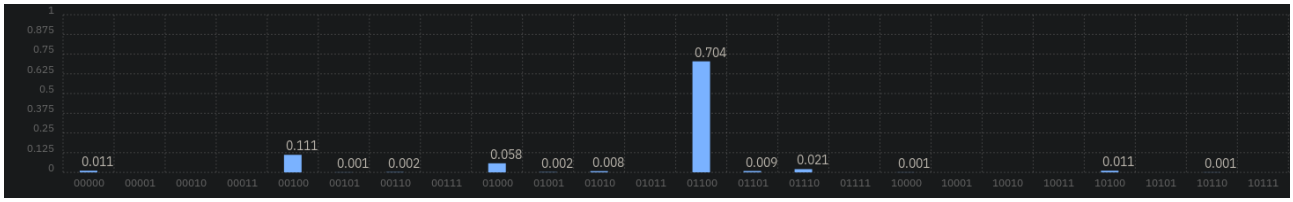
https://www.wired.com/story/quantum-mechanics-could-solve-cryptographys-random-number-problem/

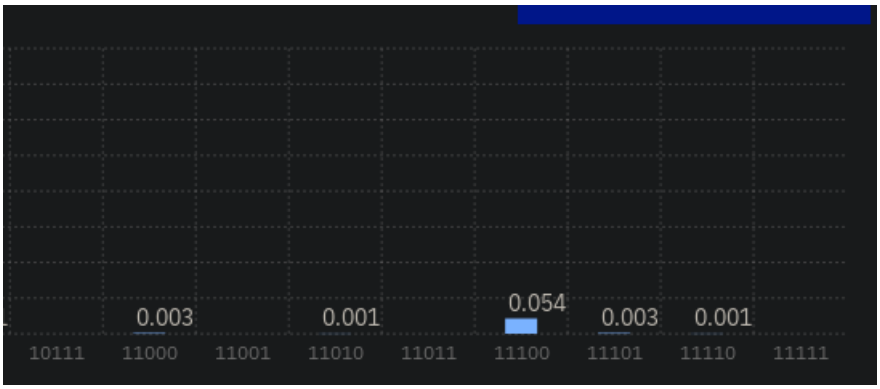
“1. The Frequency (Monobit) Test,
2. Frequency Test within a Block,
3. The Runs Test,
4. Tests for the Longest-Run-of-Ones in a Block,
5. The Binary Matrix Rank Test,
6. The Discrete Fourier Transform (Spectral) Test,
7. The Non-overlapping Template Matching Test,
8. The Overlapping Template Matching Test,
9. Maurer's "Universal Statistical" Test,
10. The Linear Complexity Test,
11. The Serial Test,
12. The Approximate Entropy Test,
13. The Cumulative Sums (Cusums) Test,
14. The Random Excursions Test, and
15. The Random Excursions Variant Test” [70]

Our purpose is just exploratory. We will not try to make Tigon pass all the required tests. This is a hobby project, not endorsed or sponsored by any organization.

All we meed to do is have fun :) .

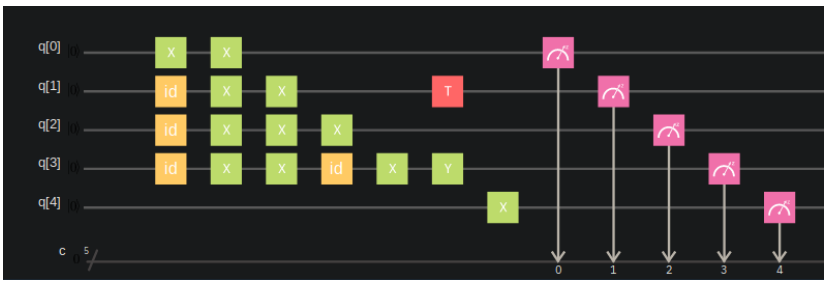
Finally, the execution of one of my experiments has been executed on the IBM back end (not simulated) :





c[5]	n
0	11
100	114
101	1
110	2
1000	59
1001	2
1010	8
1100	721
1101	9
1110	21
10000	1
10100	11
10110	1
11000	3
11010	1
11100	55
11101	3
11110	1

This is the csv file. We will use this format from now on, it is less distracting than the screenshots I usually take.



c[5]	n
0	20
1	116
11	5
100	1
1000	110
1001	615
1010	2
1011	31
1100	6
1101	9
1110	1
1111	1
10001	8
10010	1
10011	1
10100	1
10101	1
11000	11
11001	68
11010	1
11011	11
11100	2
11101	1
11111	1

We cannot create a RNG on a simulated quantum computer, **we need to make use of its natural occurring entropy**. In case of Tigon, we are left with 2 options: either use a real quantum computer like IBM, Rigetti etc or **find a way of simulating noise**.

Given the monetary constraints – I cannot make a partnership with IBM, we will ultimately look for ways/ libraries that give us the capability of generating imperfections.

We will resume next time.

18th of April

As discussed last time, we have to find a way of generating realistic noise. Went back to qiskit. Chose the aer lib.

```
device = IBMQ.get_backend('ibmq_16_melbourne')
noise_model = noise.device.basic_device_noise_model(properties)
basis_gates = noise_model.basis_gates
job_sim = execute(qc, backend,
                  coupling_map=coupling_map,
                  noise_model=noise_model,
                  basis_gates=basis_gates)
sim_result = job_sim.result()
```

Made the first tests:

{'01': 90, '00': 450, '11': 400, '10': 84}
{'01': 96, '00': 415, '11': 441, '10': 72}
{'01': 92, '00': 424, '11': 425, '10': 83}
{'01': 95, '00': 405, '11': 423, '10': 101}
{'01': 97, '00': 400, '11': 436, '10': 91}
{'01': 99, '00': 412, '11': 392, '10': 121}
{'01': 94, '00': 435, '11': 404, '10': 91}
{'01': 96, '00': 426, '11': 409, '10': 93}
{'01': 97, '00': 419, '11': 412, '10': 96}
{'01': 95, '00': 427, '11': 405, '10': 97}
{'01': 92, '00': 402, '11': 414, '10': 116}
{'01': 106, '00': 411, '11': 391, '10': 116}
{'01': 100, '00': 415, '11': 410, '10': 99}
{'01': 91, '00': 430, '11': 416, '10': 87}
{'01': 77, '00': 443, '11': 416, '10': 88}
{'01': 101, '00': 422, '11': 404, '10': 97}
{'01': 112, '00': 408, '11': 409, '10': 95}
{'01': 88, '00': 427, '11': 427, '10': 82}
{'01': 107, '00': 411, '11': 400, '10': 106}
{'01': 106, '00': 415, '11': 415, '10': 88}
{'01': 97, '00': 446, '11': 391, '10': 90}
{'01': 111, '00': 414, '11': 428, '10': 71}
{'01': 85, '00': 383, '11': 454, '10': 102}
{'01': 102, '00': 401, '11': 433, '10': 88}
{'01': 77, '00': 449, '11': 399, '10': 99}
{'01': 101, '00': 402, '11': 412, '10': 109}
{'01': 103, '00': 396, '11': 430, '10': 95}
{'01': 93, '00': 424, '11': 423, '10': 84}
{'01': 109, '00': 393, '11': 427, '10': 95}
{'01': 101, '00': 380, '11': 432, '10': 111}
{'01': 75, '00': 435, '11': 413, '10': 101}
{'01': 90, '00': 422, '11': 403, '10': 109}
{'01': 100, '00': 426, '11': 416, '10': 82}
{'01': 108, '00': 430, '11': 397, '10': 89}
{'01': 98, '00': 413, '11': 413, '10': 100}
{'01': 88, '00': 442, '11': 394, '10': 100}
{'01': 89, '00': 419, '11': 442, '10': 74}
{'01': 97, '00': 419, '11': 424, '10': 84}
{'01': 93, '00': 430, '11': 412, '10': 89}
{'01': 79, '00': 418, '11': 445, '10': 82}
{'01': 78, '00': 432, '11': 422, '10': 92}
{'01': 91, '00': 398, '11': 424, '10': 111}
{'01': 89, '00': 430, '11': 419, '10': 86}
{'01': 74, '00': 412, '11': 439, '10': 99}
{'01': 101, '00': 415, '11': 409, '10': 99}
{'01': 117, '00': 431, '11': 402, '10': 74}
{'01': 103, '00': 399, '11': 429, '10': 93}

{'01': 94, '00': 410, '11': 442, '10': 78}
{'01': 102, '00': 420, '11': 404, '10': 98}
{'01': 98, '00': 411, '11': 414, '10': 101}
{'01': 80, '00': 413, '11': 431, '10': 100}
{'01': 100, '00': 453, '11': 396, '10': 75}
{'01': 102, '00': 399, '11': 433, '10': 90}
{'01': 92, '00': 424, '11': 416, '10': 92}
{'01': 95, '00': 426, '11': 412, '10': 91}
{'01': 109, '00': 402, '11': 410, '10': 103}
{'01': 93, '00': 420, '11': 413, '10': 98}
{'01': 78, '00': 434, '11': 404, '10': 108}
{'01': 106, '00': 414, '11': 402, '10': 102}
{'01': 91, '00': 423, '11': 414, '10': 96}
{'01': 96, '00': 401, '11': 439, '10': 88}
{'01': 98, '00': 418, '11': 424, '10': 84}
{'01': 101, '00': 400, '11': 425, '10': 98}
{'01': 105, '00': 451, '11': 370, '10': 98}
{'01': 82, '00': 441, '11': 400, '10': 101}
{'01': 106, '00': 420, '11': 414, '10': 84}
{'01': 96, '00': 418, '11': 410, '10': 100}
{'01': 88, '00': 399, '11': 423, '10': 114}
{'01': 108, '00': 373, '11': 433, '10': 110}
{'01': 99, '00': 432, '11': 401, '10': 92}
{'01': 102, '00': 388, '11': 426, '10': 108}
{'01': 83, '00': 406, '11': 438, '10': 97}
{'01': 91, '00': 423, '11': 415, '10': 95}
{'01': 96, '00': 411, '11': 425, '10': 92}
{'01': 95, '00': 440, '11': 397, '10': 92}
{'01': 105, '00': 447, '11': 392, '10': 80}
{'01': 98, '00': 426, '11': 400, '10': 100}
{'01': 100, '00': 404, '11': 426, '10': 94}
{'01': 84, '00': 411, '11': 442, '10': 87}
{'01': 103, '00': 438, '11': 393, '10': 90}
{'01': 97, '00': 435, '11': 397, '10': 95}
{'01': 96, '00': 401, '11': 409, '10': 118}
{'01': 98, '00': 427, '11': 410, '10': 89}
{'01': 82, '00': 428, '11': 435, '10': 79}
{'01': 96, '00': 424, '11': 397, '10': 107}
{'01': 79, '00': 407, '11': 431, '10': 107}
{'01': 91, '00': 414, '11': 436, '10': 83}
{'01': 87, '00': 417, '11': 432, '10': 88}
{'01': 87, '00': 424, '11': 426, '10': 87}
{'01': 90, '00': 423, '11': 423, '10': 88}
{'01': 79, '00': 439, '11': 420, '10': 86}
{'01': 97, '00': 423, '11': 416, '10': 88}
{'01': 86, '00': 393, '11': 444, '10': 101}
{'01': 92, '00': 423, '11': 415, '10': 94}
{'01': 73, '00': 407, '11': 436, '10': 108}
{'01': 91, '00': 410, '11': 430, '10': 93}
{'01': 108, '00': 416, '11': 414, '10': 86}

{'01': 78, '00': 409, '11': 436, '10': 101}
{'01': 106, '00': 418, '11': 405, '10': 95}
{'01': 94, '00': 419, '11': 414, '10': 97}

I tried it 100 times here. Let us go for 300.

{'10': 71, '00': 422, '01': 70, '11': 461}
{'10': 76, '00': 440, '01': 58, '11': 450}
{'10': 70, '00': 431, '01': 94, '11': 429}
{'10': 62, '00': 460, '01': 67, '11': 435}
{'10': 71, '00': 464, '01': 74, '11': 415}
{'10': 81, '00': 418, '01': 79, '11': 446}
{'10': 73, '00': 438, '01': 80, '11': 433}
{'10': 74, '00': 400, '01': 103, '11': 447}
{'10': 81, '00': 425, '01': 71, '11': 447}
{'10': 79, '00': 403, '01': 92, '11': 450}
{'10': 69, '00': 441, '01': 66, '11': 448}
{'10': 75, '00': 432, '01': 62, '11': 455}
{'10': 63, '00': 449, '01': 62, '11': 450}
{'10': 62, '00': 420, '01': 90, '11': 452}
{'10': 79, '00': 480, '01': 78, '11': 387}
{'10': 80, '00': 440, '01': 75, '11': 429}
{'10': 60, '00': 440, '01': 79, '11': 445}
{'10': 81, '00': 460, '01': 71, '11': 412}
{'10': 80, '00': 419, '01': 57, '11': 468}
{'10': 71, '00': 449, '01': 77, '11': 427}
{'10': 64, '00': 429, '01': 89, '11': 442}
{'10': 78, '00': 434, '01': 56, '11': 456}
{'10': 77, '00': 448, '01': 77, '11': 422}
{'10': 63, '00': 433, '01': 73, '11': 455}
{'10': 62, '00': 419, '01': 80, '11': 463}
{'10': 78, '00': 416, '01': 75, '11': 455}
{'10': 83, '00': 451, '01': 80, '11': 410}
{'10': 86, '00': 425, '01': 92, '11': 421}
{'10': 62, '00': 446, '01': 76, '11': 440}
{'10': 80, '00': 436, '01': 57, '11': 451}
{'10': 75, '00': 443, '01': 68, '11': 438}
{'10': 66, '00': 446, '01': 74, '11': 438}
{'10': 79, '00': 474, '01': 60, '11': 411}
{'10': 78, '00': 438, '01': 81, '11': 427}
{'10': 79, '00': 424, '01': 72, '11': 449}
{'10': 77, '00': 410, '01': 73, '11': 464}
{'10': 77, '00': 435, '01': 77, '11': 435}
{'10': 76, '00': 460, '01': 74, '11': 414}
{'10': 89, '00': 438, '01': 63, '11': 434}
{'10': 87, '00': 423, '01': 86, '11': 428}
{'10': 73, '00': 434, '01': 79, '11': 438}
{'10': 75, '00': 425, '01': 80, '11': 444}
{'10': 75, '00': 459, '01': 65, '11': 425}
{'10': 66, '00': 421, '01': 66, '11': 471}

{'10': 79, '00': 422, '01': 61, '11': 462}
{'10': 91, '00': 458, '01': 53, '11': 422}
{'10': 63, '00': 436, '01': 77, '11': 448}
{'10': 69, '00': 423, '01': 64, '11': 468}
{'10': 64, '00': 443, '01': 61, '11': 456}
{'10': 66, '00': 436, '01': 81, '11': 441}
{'10': 60, '00': 448, '01': 72, '11': 444}
{'10': 83, '00': 450, '01': 92, '11': 399}
{'10': 83, '00': 476, '01': 82, '11': 383}
{'10': 70, '00': 455, '01': 62, '11': 437}
{'10': 67, '00': 446, '01': 69, '11': 442}
{'10': 61, '00': 450, '01': 83, '11': 430}
{'10': 79, '00': 455, '01': 75, '11': 415}
{'10': 72, '00': 416, '01': 87, '11': 449}
{'10': 59, '00': 443, '01': 89, '11': 433}
{'10': 68, '00': 449, '01': 78, '11': 429}
{'10': 74, '00': 430, '01': 72, '11': 448}
{'10': 87, '00': 430, '01': 81, '11': 426}
{'10': 80, '00': 415, '01': 69, '11': 460}
{'10': 70, '00': 435, '01': 88, '11': 431}
{'10': 62, '00': 477, '01': 72, '11': 413}
{'10': 57, '00': 434, '01': 85, '11': 448}
{'10': 71, '00': 422, '01': 84, '11': 447}
{'10': 68, '00': 441, '01': 80, '11': 435}
{'10': 70, '00': 448, '01': 73, '11': 433}
{'10': 81, '00': 423, '01': 82, '11': 438}
{'10': 76, '00': 425, '01': 69, '11': 454}
{'10': 65, '00': 433, '01': 71, '11': 455}
{'10': 78, '00': 445, '01': 71, '11': 430}
{'10': 64, '00': 446, '01': 75, '11': 439}
{'10': 75, '00': 441, '01': 66, '11': 442}
{'10': 72, '00': 446, '01': 72, '11': 434}
{'10': 70, '00': 456, '01': 68, '11': 430}
{'10': 77, '00': 456, '01': 71, '11': 420}
{'10': 61, '00': 444, '01': 74, '11': 445}
{'10': 81, '00': 421, '01': 71, '11': 451}
{'10': 68, '00': 448, '01': 95, '11': 413}
{'10': 70, '00': 460, '01': 62, '11': 432}
{'10': 79, '00': 423, '01': 70, '11': 452}
{'10': 74, '00': 425, '01': 77, '11': 448}
{'10': 76, '00': 417, '01': 70, '11': 461}
{'10': 92, '00': 430, '01': 79, '11': 423}
{'10': 64, '00': 449, '01': 82, '11': 429}
{'10': 68, '00': 442, '01': 69, '11': 445}
{'10': 90, '00': 421, '01': 68, '11': 445}
{'10': 80, '00': 415, '01': 82, '11': 447}
{'10': 78, '00': 429, '01': 80, '11': 437}
{'10': 65, '00': 435, '01': 70, '11': 454}
{'10': 65, '00': 430, '01': 81, '11': 448}
{'10': 83, '00': 433, '01': 63, '11': 445}

{'10': 75, '00': 450, '01': 79, '11': 420}
{'10': 83, '00': 418, '01': 62, '11': 461}
{'10': 65, '00': 468, '01': 69, '11': 422}
{'10': 74, '00': 440, '01': 85, '11': 425}
{'10': 63, '00': 419, '01': 85, '11': 457}
{'10': 66, '00': 431, '01': 93, '11': 434}
{'10': 85, '00': 465, '01': 58, '11': 416}
{'10': 75, '00': 439, '01': 80, '11': 430}
{'10': 72, '00': 446, '01': 73, '11': 433}
{'10': 73, '00': 428, '01': 86, '11': 437}
{'10': 75, '00': 446, '01': 64, '11': 439}
{'10': 69, '00': 435, '01': 81, '11': 439}
{'10': 87, '00': 437, '01': 75, '11': 425}
{'10': 87, '00': 438, '01': 71, '11': 428}
{'10': 81, '00': 428, '01': 84, '11': 431}
{'10': 69, '00': 440, '01': 87, '11': 428}
{'10': 86, '00': 405, '01': 82, '11': 451}
{'10': 82, '00': 397, '01': 83, '11': 462}
{'10': 79, '00': 428, '01': 84, '11': 433}
{'10': 79, '00': 457, '01': 75, '11': 413}
{'10': 86, '00': 484, '01': 63, '11': 391}
{'10': 77, '00': 428, '01': 67, '11': 452}
{'10': 87, '00': 441, '01': 73, '11': 423}
{'10': 80, '00': 457, '01': 63, '11': 424}
{'10': 80, '00': 448, '01': 72, '11': 424}
{'10': 69, '00': 439, '01': 79, '11': 437}
{'10': 66, '00': 433, '01': 75, '11': 450}
{'10': 85, '00': 445, '01': 66, '11': 428}
{'10': 77, '00': 439, '01': 83, '11': 425}
{'10': 76, '00': 428, '01': 70, '11': 450}
{'10': 79, '00': 457, '01': 76, '11': 412}
{'10': 72, '00': 457, '01': 72, '11': 423}
{'10': 78, '00': 421, '01': 67, '11': 458}
{'10': 80, '00': 461, '01': 76, '11': 407}
{'10': 90, '00': 409, '01': 82, '11': 443}
{'10': 76, '00': 428, '01': 82, '11': 438}
{'10': 68, '00': 478, '01': 76, '11': 402}
{'10': 65, '00': 444, '01': 60, '11': 455}
{'10': 81, '00': 404, '01': 84, '11': 455}
{'10': 80, '00': 429, '01': 73, '11': 442}
{'10': 85, '00': 429, '01': 62, '11': 448}
{'10': 74, '00': 428, '01': 68, '11': 454}
{'10': 83, '00': 424, '01': 75, '11': 442}
{'10': 71, '00': 464, '01': 62, '11': 427}
{'10': 67, '00': 430, '01': 66, '11': 461}
{'10': 96, '00': 452, '01': 60, '11': 416}
{'10': 75, '00': 431, '01': 73, '11': 445}
{'10': 71, '00': 453, '01': 90, '11': 410}
{'10': 74, '00': 448, '01': 71, '11': 431}
{'10': 81, '00': 428, '01': 79, '11': 436}

{'10': 78, '00': 413, '01': 74, '11': 459}
{'10': 84, '00': 442, '01': 73, '11': 425}
{'10': 85, '00': 424, '01': 82, '11': 433}
{'10': 80, '00': 418, '01': 79, '11': 447}
{'10': 75, '00': 436, '01': 65, '11': 448}
{'10': 61, '00': 441, '01': 78, '11': 444}
{'10': 75, '00': 440, '01': 77, '11': 432}
{'10': 65, '00': 429, '01': 75, '11': 455}
{'10': 57, '00': 447, '01': 92, '11': 428}
{'10': 79, '00': 445, '01': 78, '11': 422}
{'10': 86, '00': 431, '01': 88, '11': 419}
{'10': 94, '00': 415, '01': 79, '11': 436}
{'10': 76, '00': 436, '01': 71, '11': 441}
{'10': 75, '00': 422, '01': 88, '11': 439}
{'10': 86, '00': 435, '01': 75, '11': 428}
{'10': 74, '00': 444, '01': 59, '11': 447}
{'10': 65, '00': 456, '01': 81, '11': 422}
{'10': 66, '00': 423, '01': 77, '11': 458}
{'10': 82, '00': 443, '01': 71, '11': 428}
{'10': 91, '00': 419, '01': 78, '11': 436}
{'10': 68, '00': 439, '01': 76, '11': 441}
{'10': 82, '00': 441, '01': 61, '11': 440}
{'10': 65, '00': 455, '01': 64, '11': 440}
{'10': 69, '00': 477, '01': 71, '11': 407}
{'10': 76, '00': 423, '01': 70, '11': 455}
{'10': 73, '00': 448, '01': 88, '11': 415}
{'10': 84, '00': 433, '01': 67, '11': 440}
{'10': 75, '00': 442, '01': 65, '11': 442}
{'10': 64, '00': 460, '01': 74, '11': 426}
{'10': 78, '00': 428, '01': 78, '11': 440}
{'10': 76, '00': 440, '01': 79, '11': 429}
{'10': 62, '00': 460, '01': 86, '11': 416}
{'10': 74, '00': 434, '01': 91, '11': 425}
{'10': 80, '00': 437, '01': 92, '11': 415}
{'10': 79, '00': 452, '01': 69, '11': 424}
{'10': 80, '00': 450, '01': 70, '11': 424}
{'10': 69, '00': 458, '01': 96, '11': 401}
{'10': 73, '00': 439, '01': 77, '11': 435}
{'10': 72, '00': 444, '01': 82, '11': 426}
{'10': 70, '00': 440, '01': 86, '11': 428}
{'10': 65, '00': 463, '01': 63, '11': 433}
{'10': 74, '00': 411, '01': 80, '11': 459}
{'10': 77, '00': 441, '01': 71, '11': 435}
{'10': 65, '00': 425, '01': 74, '11': 460}
{'10': 83, '00': 445, '01': 89, '11': 407}
{'10': 78, '00': 445, '01': 82, '11': 419}
{'10': 82, '00': 435, '01': 77, '11': 430}
{'10': 70, '00': 432, '01': 67, '11': 455}
{'10': 87, '00': 403, '01': 86, '11': 448}
{'10': 72, '00': 459, '01': 68, '11': 425}

{'10': 76, '00': 435, '01': 75, '11': 438}
{'10': 87, '00': 409, '01': 76, '11': 452}
{'10': 84, '00': 421, '01': 69, '11': 450}
{'10': 78, '00': 432, '01': 76, '11': 438}
{'10': 82, '00': 457, '01': 69, '11': 416}
{'10': 68, '00': 453, '01': 77, '11': 426}
{'10': 76, '00': 454, '01': 70, '11': 424}
{'10': 81, '00': 444, '01': 75, '11': 424}
{'10': 82, '00': 441, '01': 64, '11': 437}
{'10': 92, '00': 411, '01': 71, '11': 450}
{'10': 71, '00': 431, '01': 77, '11': 445}
{'10': 80, '00': 415, '01': 81, '11': 448}
{'10': 66, '00': 455, '01': 70, '11': 433}
{'10': 81, '00': 468, '01': 70, '11': 405}
{'10': 63, '00': 447, '01': 78, '11': 436}
{'10': 74, '00': 447, '01': 74, '11': 429}
{'10': 73, '00': 422, '01': 83, '11': 446}
{'10': 66, '00': 465, '01': 68, '11': 425}
{'10': 74, '00': 416, '01': 75, '11': 459}
{'10': 86, '00': 427, '01': 71, '11': 440}
{'10': 85, '00': 453, '01': 76, '11': 410}
{'10': 62, '00': 461, '01': 74, '11': 427}
{'10': 80, '00': 408, '01': 77, '11': 459}
{'10': 82, '00': 447, '01': 60, '11': 435}
{'10': 86, '00': 437, '01': 77, '11': 424}
{'10': 87, '00': 440, '01': 77, '11': 420}
{'10': 79, '00': 434, '01': 83, '11': 428}
{'10': 74, '00': 438, '01': 73, '11': 439}
{'10': 76, '00': 430, '01': 83, '11': 435}
{'10': 58, '00': 446, '01': 83, '11': 437}
{'10': 75, '00': 428, '01': 62, '11': 459}
{'10': 84, '00': 427, '01': 61, '11': 452}
{'10': 59, '00': 435, '01': 74, '11': 456}
{'10': 90, '00': 428, '01': 70, '11': 436}
{'10': 70, '00': 420, '01': 78, '11': 456}
{'10': 78, '00': 459, '01': 70, '11': 417}
{'10': 82, '00': 407, '01': 86, '11': 449}
{'10': 75, '00': 456, '01': 75, '11': 418}
{'10': 79, '00': 434, '01': 66, '11': 445}
{'10': 71, '00': 429, '01': 69, '11': 455}
{'10': 98, '00': 418, '01': 79, '11': 429}
{'10': 53, '00': 415, '01': 74, '11': 482}
{'10': 82, '00': 455, '01': 81, '11': 406}
{'10': 68, '00': 420, '01': 71, '11': 465}
{'10': 71, '00': 415, '01': 68, '11': 470}
{'10': 75, '00': 453, '01': 81, '11': 415}
{'10': 73, '00': 435, '01': 78, '11': 438}
{'10': 77, '00': 428, '01': 86, '11': 433}
{'10': 74, '00': 436, '01': 75, '11': 439}
{'10': 78, '00': 455, '01': 77, '11': 414}

{'10': 64, '00': 432, '01': 69, '11': 459}
{'10': 77, '00': 479, '01': 55, '11': 413}
{'10': 74, '00': 455, '01': 65, '11': 430}
{'10': 71, '00': 438, '01': 78, '11': 437}
{'10': 81, '00': 445, '01': 72, '11': 426}
{'10': 76, '00': 444, '01': 71, '11': 433}
{'10': 69, '00': 440, '01': 79, '11': 436}
{'10': 57, '00': 455, '01': 78, '11': 434}
{'10': 76, '00': 427, '01': 81, '11': 440}
{'10': 56, '00': 450, '01': 74, '11': 444}
{'10': 74, '00': 448, '01': 71, '11': 431}
{'10': 64, '00': 450, '01': 78, '11': 432}
{'10': 70, '00': 461, '01': 66, '11': 427}
{'10': 72, '00': 440, '01': 95, '11': 417}
{'10': 73, '00': 443, '01': 68, '11': 440}
{'10': 87, '00': 440, '01': 72, '11': 425}
{'10': 83, '00': 429, '01': 64, '11': 448}
{'10': 76, '00': 429, '01': 72, '11': 447}
{'10': 70, '00': 439, '01': 87, '11': 428}
{'10': 80, '00': 399, '01': 86, '11': 459}
{'10': 85, '00': 422, '01': 82, '11': 435}
{'10': 85, '00': 422, '01': 64, '11': 453}
{'10': 78, '00': 440, '01': 68, '11': 438}
{'10': 66, '00': 450, '01': 68, '11': 440}
{'10': 79, '00': 423, '01': 83, '11': 439}
{'10': 70, '00': 422, '01': 77, '11': 455}
{'10': 70, '00': 433, '01': 83, '11': 438}
{'10': 71, '00': 419, '01': 88, '11': 446}
{'10': 72, '00': 439, '01': 76, '11': 437}
{'10': 64, '00': 440, '01': 77, '11': 443}
{'10': 78, '00': 463, '01': 67, '11': 416}
{'10': 87, '00': 445, '01': 71, '11': 421}
{'10': 63, '00': 424, '01': 69, '11': 468}
{'10': 68, '00': 467, '01': 78, '11': 411}
{'10': 87, '00': 454, '01': 63, '11': 420}
{'10': 87, '00': 416, '01': 82, '11': 439}
{'10': 82, '00': 399, '01': 81, '11': 462}
{'10': 65, '00': 430, '01': 85, '11': 444}
{'10': 72, '00': 427, '01': 64, '11': 461}
{'10': 78, '00': 437, '01': 84, '11': 425}
{'10': 85, '00': 439, '01': 74, '11': 426}
{'10': 72, '00': 441, '01': 64, '11': 447}
{'10': 86, '00': 436, '01': 80, '11': 422}
{'10': 94, '00': 410, '01': 97, '11': 423}
{'10': 68, '00': 441, '01': 76, '11': 439}
{'10': 78, '00': 452, '01': 81, '11': 413}
{'10': 65, '00': 480, '01': 74, '11': 405}
{'10': 60, '00': 442, '01': 59, '11': 463}
{'10': 80, '00': 452, '01': 61, '11': 431}
{'10': 85, '00': 399, '01': 72, '11': 468}

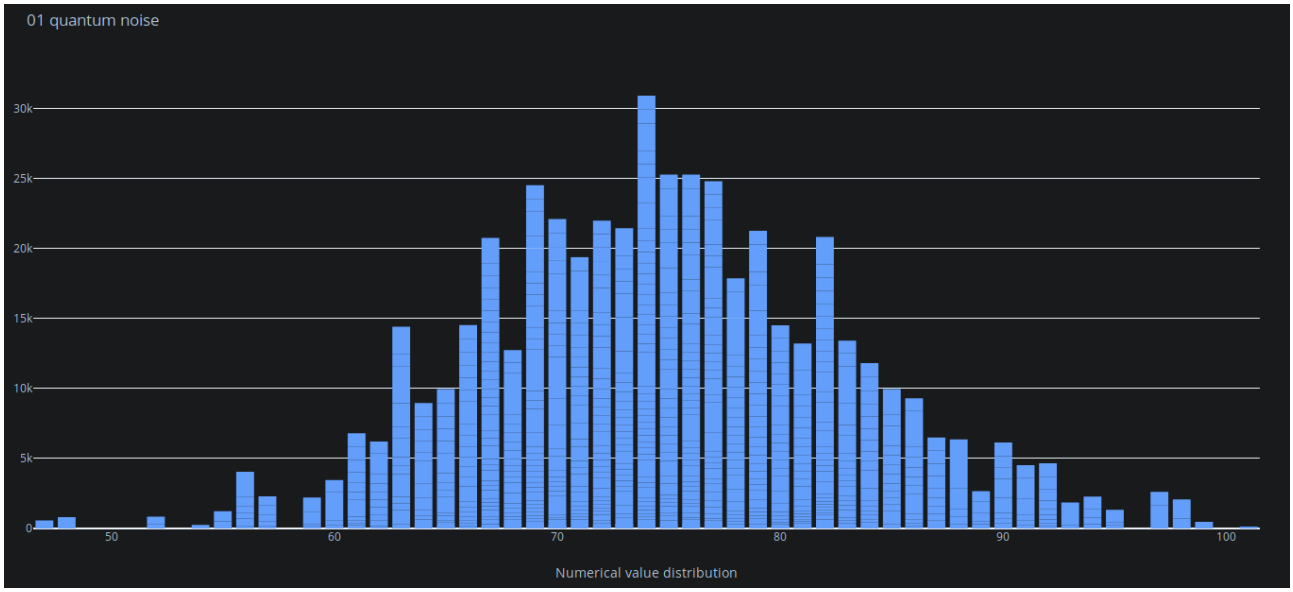
{'10': 81, '00': 469, '01': 75, '11': 399}
{'10': 55, '00': 446, '01': 87, '11': 436}
{'10': 83, '00': 460, '01': 60, '11': 421}
{'10': 61, '00': 444, '01': 81, '11': 438}
{'10': 72, '00': 437, '01': 62, '11': 453}
{'10': 62, '00': 430, '01': 89, '11': 443}

Good. Let us try to plot one of those states.

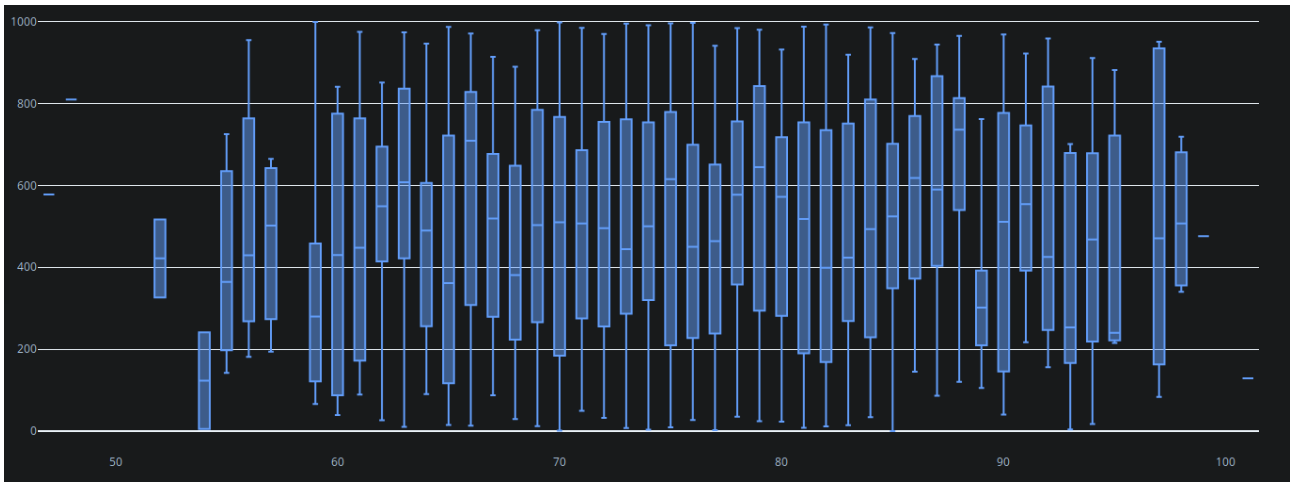
I'll choose 10.

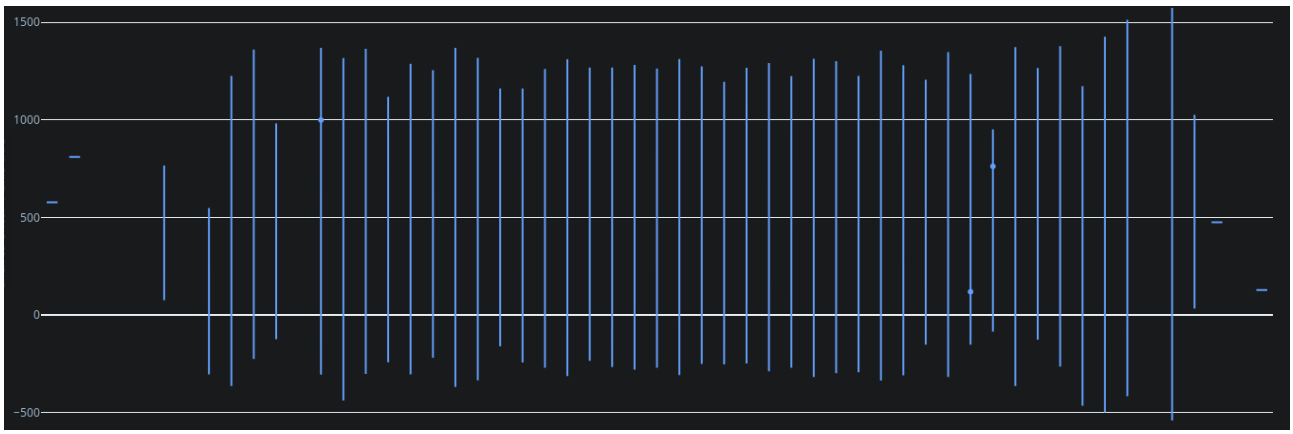
The distribution seems rather uniform from 53 to 98. Let us now generate 1000 samples. This will take a while.

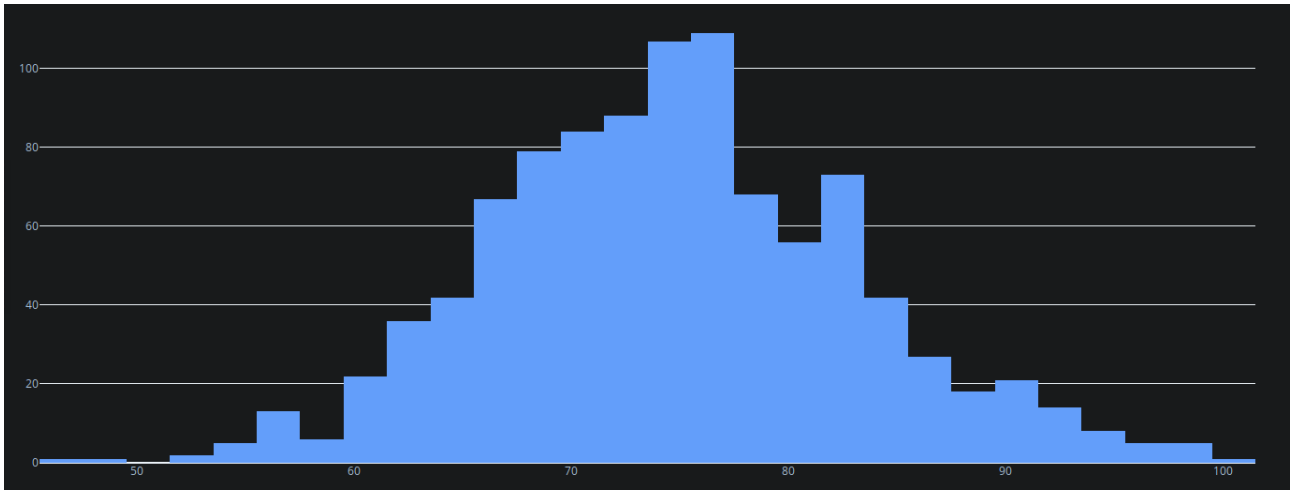
Looks like it is done. Now, the moment of truth :) :

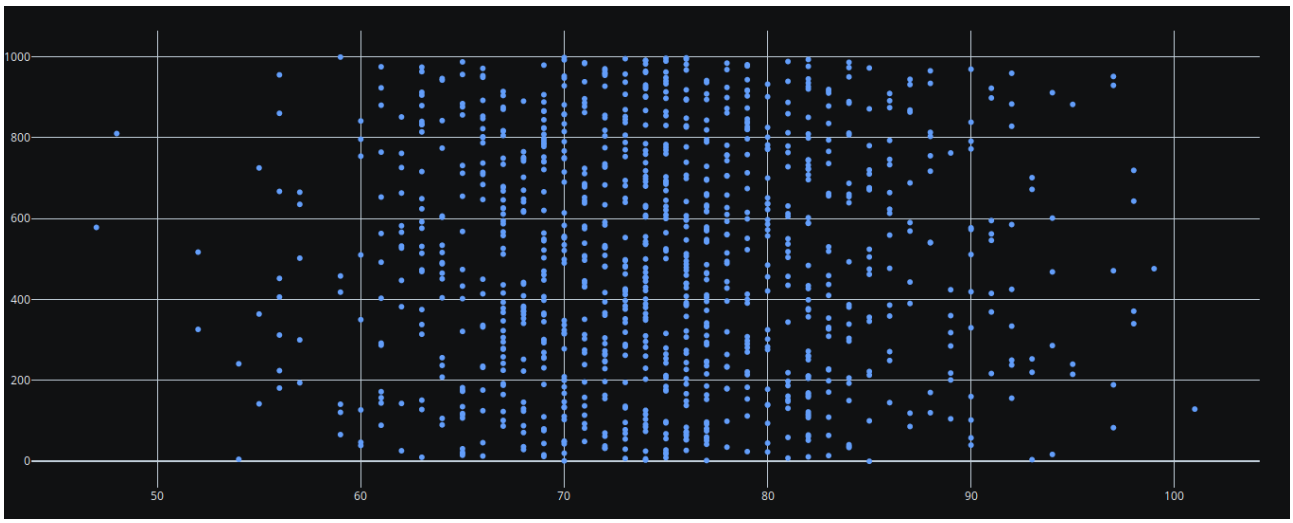


It resembles a Gaussian distribution really well.









Obviously, this is rather useless for a SRNG, but could be modified further to obtain a good enough RNG as long as it is not used for security purposes.

Here is a quote from the aer dev team: “You might notice that the counts returned in the above histogram don't look exactly like our simulation. **This is because the real errors that happen on a**

device can be very complicated, and characterizing them to create accurate models is an active area of quantum computing research. The *basic device noise model* we used for our simulation is **a simplified error model** that only takes into account *local* errors occurring on the qubits participating in each individual gate, and treats the errors as a relaxation process combined with an error operation which acts to randomize the state of the qubit with some probability *p* (the depolarizing probability). While this model is an approximation, due to its simplicity it is a useful starting point for studying the effects of noise on quantum computations."

They also give us the ability to print out the noise generated as a dict:

```
(base) mihai@mihai-Lenovo-V330-14IKB:~/Tigon$ python noise.py
/home/mihai/.local/lib/python3.6/site-packages/qiskit/providers/ibmq/credentials/_configrc.py:129: UserWarning: Credentials already present. Set overwrite=True to overwrite.
  warnings.warn('Credentials already present. Set overwrite=True to overwrite.')
{'00': 421, '11': 458, '01': 80, '10': 65}
{'errors': [{'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}],
[{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0013795774021903573, 0.0013795774021903573, 0.0013795774021903573, 0.9958612677934289],
'gate_qubits': [[0]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits':
[0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.005066270796948297, 0.005066270796948297, 0.005066270796948297,
0.9848011876091551], 'gate_qubits': [[1]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}],
[{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0022766487634409693, 0.0022766487634409693,
0.0022766487634409693, 0.993170053709677], 'gate_qubits': [[2]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}],
[{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0006198921307415117,
0.0006198921307415117, 0.0006198921307415117, 0.9981403236077755], 'gate_qubits': [[3]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions':
[[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities':
[0.00933402283095569, 0.00933402283095569, 0.00933402283095569, 0.971997931507133], 'gate_qubits': [[4]]}, {'type': 'qerror', 'operations':
['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities':
[0.0011480172218532037, 0.0011480172218532037, 0.0011480172218532037, 0.9965559483344404], 'gate_qubits': [[5]]}, {'type': 'qerror',
'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]],
'probabilities': [0.0007177691152206456, 0.0007177691152206456, 0.0007177691152206456, 0.9978466926543381], 'gate_qubits': [[6]]}, {'type':
'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits':
[0]}]], 'probabilities': [0.0008942467082432015, 0.0008942467082432015, 0.0008942467082432015, 0.9973172598752704], 'gate_qubits': [[7]]},
{'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id',
'qubits': [0]}]], 'probabilities': [0.0011814122130629767, 0.0011814122130629767, 0.0011814122130629767, 0.9964557633608111], 'gate_qubits':
[[8]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name':
'id', 'qubits': [0]}]], 'probabilities': [0.001427045935480875, 0.001427045935480875, 0.001427045935480875, 0.9957188621935573], 'gate_qubits':
[[9]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name':
'id', 'qubits': [0]}]], 'probabilities': [0.0006849271501655207, 0.0006849271501655207, 0.0006849271501655207, 0.9979452185495035],
'gate_qubits': [[10]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits':
[0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0010232133790341325, 0.0010232133790341325, 0.0010232133790341325,
0.9969303598628976], 'gate_qubits': [[11]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}],
[{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.002842129873611682, 0.002842129873611682, 0.002842129873611682,
0.9914736103791649], 'gate_qubits': [[12]]}, {'type': 'qerror', 'operations': ['u2'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}],
[{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0029379590353513507, 0.0029379590353513507,
0.0029379590353513507, 0.9911861228939459], 'gate_qubits': [[13]]}, {'type': 'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}],
[{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0027591548043807146,
0.0027591548043807146, 0.0027591548043807146, 0.9917225355868579], 'gate_qubits': [[0]]}, {'type': 'qerror', 'operations': ['u3'], 'instructions':
[[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities':
[0.010132541593896593, 0.010132541593896593, 0.010132541593896593, 0.9696023752183103], 'gate_qubits': [[1]]}, {'type': 'qerror', 'operations':
['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities':
[0.004553297526881939, 0.004553297526881939, 0.004553297526881939, 0.9863401074193542], 'gate_qubits': [[2]]}, {'type': 'qerror', 'operations':
['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities':
[0.0012397842614830235, 0.0012397842614830235, 0.0012397842614830235, 0.996280647215551], 'gate_qubits': [[3]]}, {'type': 'qerror',
'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]],
'probabilities': [0.01866804566191138, 0.01866804566191138, 0.01866804566191138, 0.9439958630142659], 'gate_qubits': [[4]]}, {'type': 'qerror',
'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]],
'probabilities': [0.0022960344437064073, 0.0022960344437064073, 0.0022960344437064073, 0.9931118966688808], 'gate_qubits': [[5]]}, {'type':
'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits':
[0]}]], 'probabilities': [0.0014355382304412911, 0.0014355382304412911, 0.0014355382304412911, 0.9956933853086761], 'gate_qubits': [[6]]},
{'type': 'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id',
'qubits': [0]}]], 'probabilities': [0.001788493416486403, 0.001788493416486403, 0.001788493416486403, 0.9946345197505408], 'gate_qubits':
[[7]]}, {'type': 'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name':
'id', 'qubits': [0]}]], 'probabilities': [0.0023628244261259534, 0.0023628244261259534, 0.0023628244261259534, 0.9929115267216222],
'gate_qubits': [[8]]}, {'type': 'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits':
[0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.00285409187096175, 0.00285409187096175, 0.00285409187096175, 0.9914377243871147],
'gate_qubits': [[9]]}, {'type': 'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits':
[0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0013698543003310415, 0.0013698543003310415, 0.0013698543003310415,
0.9958904370990069], 'gate_qubits': [[10]]}, {'type': 'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}],
[{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.002046426758068265, 0.002046426758068265, 0.002046426758068265,
0.9938607197257951], 'gate_qubits': [[11]]}, {'type': 'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}],
[{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.005684259747223364, 0.005684259747223364, 0.005684259747223364,
```

0.9829472207583299], 'gate_qubits': [[12]]}, {'type': 'qerror', 'operations': ['u3'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0058759180707027014, 0.0058759180707027014, 0.0058759180707027014, 0.9823722457878918], 'gate_qubits': [[13]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.003424327745117327, 0.9486350838232401], 'gate_qubits': [[1, 0]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.003655875843522211, 0.9451618623471668], 'gate_qubits': [[1, 2]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.003183477016834246 3, 0.9522478447474862], 'gate_qubits': [[2, 3]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.011848650156807299, 0.8222702476478905], 'gate_qubits': [[4, 3]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.00704667709663329, 0.8942998435505006], 'gate_qubits': [[4, 10]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.007409060410740479, 0.8888640938388928], 'gate_qubits': [[5, 4]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.005335335788714428, 0.9199699631692836], 'gate_qubits': [[5, 6]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': z', 'qubits': [1]}], [{'name': x', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': y', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': z', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': id', 'qubits': [0]}]], 'probabilities': [0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.009994895512139292, 0.9400765673179107], 'gate_qubits': [[5, 9]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': z', 'qubits': [1]}], [{'name': x', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': y', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': z', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': id', 'qubits': [0]}]], 'probabilities': [0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944,

0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.0031774118164736944, 0.9523388227528946], 'gate_qubits': [[6, 8]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.003081905524394193, 0.9537714171340871], 'gate_qubits': [[7, 8]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.003357810473558265, 0.9496328428966261], 'gate_qubits': [[9, 8]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.00326829799511994 1, 0.9509755300732009], 'gate_qubits': [[9, 10]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.0028383140651620877, 0.9574252890225687], 'gate_qubits': [[11, 3]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.0033842794382139296, 0.949235808426791], 'gate_qubits': [[11, 10]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.0037178758183439554, 0.9442318627248407], 'gate_qubits': [[11, 12]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.005473370749857394, 0.9178994387521391], 'gate_qubits': [[12, 2]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'z', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'z', 'qubits': [1]}], [{'name': 'id', 'qubits': [0]}]], 'probabilities': [0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.0097513175565092, 0.853730236652362], 'gate_qubits': [[13, 1]]}, {'type': 'qerror', 'operations': ['cx'], 'instructions': [[{'name': 'x', 'qubits': [0]}], [{'name': 'y', 'qubits': [0]}], [{'name': 'z', 'qubits': [0]}], [{'name': 'x', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'z', 'qubits': [0]}, {'name': 'x', 'qubits': [1]}], [{'name': 'y', 'qubits': [1]}], [{'name': 'x', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': 'y', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': z', 'qubits': [0]}, {'name': 'y', 'qubits': [1]}], [{'name': z', 'qubits': [1]}], [{'name': x', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': y', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': z', 'qubits': [0]}, {'name': z', 'qubits': [1]}], [{'name': id', 'qubits': [0]}]], 'probabilities': [0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993, 0.004337688796076993,

0.9349346680588451], 'gate_qubits': [[13, 12]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9724999999999999, 0.02750000000000008], [0.02750000000000008, 0.9724999999999999]], 'gate_qubits': [[0]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.915, 0.08499999999999996], [0.08499999999999996, 0.915]], 'gate_qubits': [[1]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9596, 0.04039999999999999], [0.04039999999999999, 0.9596]], 'gate_qubits': [[2]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9286000000000001, 0.07139999999999991], [0.07139999999999991, 0.9286000000000001]], 'gate_qubits': [[3]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.8053, 0.19469999999999998], [0.19469999999999998, 0.8053]], 'gate_qubits': [[4]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9444, 0.05559999999999998], [0.05559999999999998, 0.9444]], 'gate_qubits': [[5]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9219999999999999, 0.07800000000000007], [0.07800000000000007, 0.9219999999999999]], 'gate_qubits': [[6]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.7229, 0.2771], [0.2771, 0.7229]], 'gate_qubits': [[7]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9546, 0.04539999999999996], [0.04539999999999996, 0.9546]], 'gate_qubits': [[8]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9212, 0.07879999999999998], [0.07879999999999998, 0.9212]], 'gate_qubits': [[9]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9664, 0.03359999999999996], [0.03359999999999996, 0.9664]], 'gate_qubits': [[10]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.7768, 0.22319999999999995], [0.22319999999999995, 0.7768]], 'gate_qubits': [[11]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9475, 0.05249999999999999], [0.05249999999999999, 0.9475]], 'gate_qubits': [[12]]}, {'type': 'roerror', 'operations': ['measure'], 'probabilities': [[0.9508, 0.04920000000000002], [0.04920000000000002, 0.9508]], 'gate_qubits': [[13]]}], 'x90_gates': []}

20th if April

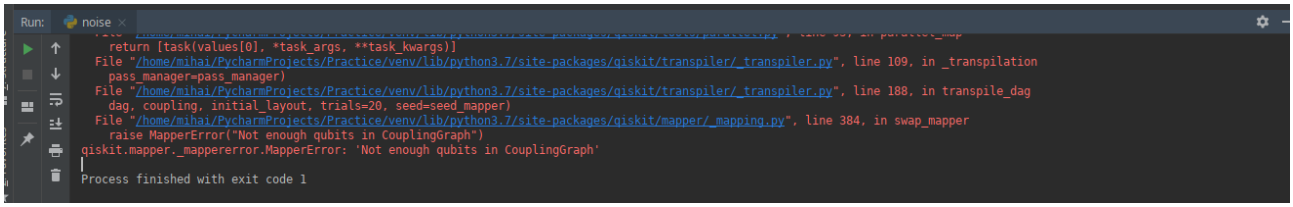
Tried the same thing like last time but this time with 4 quantum registers and 4 classical ones:

{'0110': 7, '0011': 246, '1001': 34, '0111': 6, '1011': 112, '0000': 234, '0001': 106, '0010': 95, '0101': 1, '1100': 8, '1110': 2, '1111': 3, '1010': 43, '0100': 14, '1000': 113}

I am really curious as to see how many qc and cc register Aer supports. We have 8 here:

{'11001001': 1, '01011000': 1, '00001000': 56, '10000100': 1, '01011011': 2, '11010010': 3, '00001011': 62, '00010000': 34, '01001001': 2, '10011011': 8, '11001000': 2, '10000111': 2, '01000110': 1, '00000111': 5, '00001100': 5, '00011000': 14, '10001001': 9, '10100011': 1, '00001111': 1, '00010001': 23, '00000011': 106, '10011000': 7, '01000000': 9, '10101000': 2, '01000011': 12, '00100101': 1, '10111011': 2, '11001011': 1, '10001111': 1, '10000000': 61, '00011011': 14, '01000010': 2, '11000001': 2, '11010110': 1, '10000011': 42, '10001100': 1, '11000000': 4, '00001010': 24, '00110011': 3, '10110000': 1, '10010101': 1, '01001000': 5, '10010100': 1, '01100000': 1, '00011001': 2, '10011010': 2, '00100001': 2, '10010011': 13, '01000001': 2, '10000010': 18, '00010010': 11, '10101011': 3, '10100010': 3, '11000011': 4, '00100010': 5, '10010111': 1, '10001000': 13, '01101000': 1, '00011100': 1, '10101010': 2, '01001010': 2, '10001010': 7, '00010011': 42, '00111011': 1, '00100000': 4, '00000001': 42, '01010011': 3, '00000100': 4, '10011001': 1, '00111010': 2, '00110010': 2, '10100000': 2, '01010000': 3, '00101000': 2, '00001001': 27, '11000010': 3, '10000001': 25, '10010000': 12, '11010000': 2, '01101011': 1, '01001011': 3, '01010001': 1, '00000000': 88, '01100001': 1, '00111001': 2, '00101011': 4, '00100011': 4, '00110000': 3, '00000101': 2, '11100000': 2, '00000010': 68, '10010010': 7, '10001011': 23, '10010001': 1, '00011010': 8}

Let’s try 16:



Well, we found out I guess :).

At the moment, I do not know in which direction I should proceed.

For one, there are actually very few resources on qc and the field is still experimental. Secondly, there is the budged issue – in order to create a reliable RNG, we need actual physical hardware to do so. Sure, a pseudo random number generator can still be used in use cases where security is not an issue. In my opinion, our project reached its limit. We have generated 1000 numbers, we've learned the basics of quantum computing and we have explored the available Python libraries. The Rigetti SDK looks promising, yet IBM is clearly the leader in this moment. Google's project lags behind a bit, being still in alpha. I did not touch Microsoft Q# because I was interested in Python libraries for convenience. Last but not least, this task cannot be accomplished by one person alone, a team effort is necessary in order to get satisfactory results. The aer library itself is just using a simplified noise model, thus of no use given our grandiose objective :) .

Please note again that this is a hobby project, and all this was done in the time span of 2 weeks. I hope that other enthusiasts and amateurs like me will find my brief research log useful.

Might resume it next year, who knows :)

Chelalau Ionut Mihai, R&D Engineer

c.ionutmihai@protonmail.ch

Bibliography:

1. Mastering Quantum Computing with IBM QX - Dr. Christine Corbett Moran
2. Realization of a scalable Shor algorithm - T. Monz, D. Nigg, E. A. Martinez, M. F. Brandl, P. Schindler, R. Rines, S. X. Wang, I. L. Chuang, and R. Blatt
3. Constant-Optimized Quantum Circuits for Modular Multiplication and Exponentiation - Igor L. Markov, Mehdi Saeedi
4. The idiots guide to Quantum Error Correction - Simon J. Devitt and Kae Nemoto, William J. Munro
5. Quantum advantage with shallow circuits – Robert Konig, Sergey Bravyi and David Gosset
6. Quantum Computing Circuits and Devices - Travis S. Humble, Himanshu Thapliyal, Edgard Munoz-Coreas,Fahd A. Mohiyaddin, Ryan S. Bennink
7. Experimental One-Way Quantum Computing – P. Walther, K. J. Resch, T. Rudolph, E. Schenck, H. Weinfurter, V. Vedral, M. Aspelmeyer & A. Zeilinger
8. Superconducting Circuits and Quantum Information - J. Q. You and Franco Nori
9. Superconducting quantum circuits at the surface code threshold for fault tolerance - Barends, R Kelly, J Megrant, et al.
10. Optical Quantum Computing - Jeremy L. O'Brien
11. A One-Way Quantum Computer - Robert Raussendorf and Hans J. Briegel
12. Digitized adiabatic quantum computing with a superconducting circuit - R. Barends, A. Shabani, L. Lamata et al.
13. Experimental quantum computing without entanglement - B. P. Lanyon, M. Barbieri, M. P. Almeida and A. G. White
14. Quantum Circuits with Mixed States - Dorit Aharonov, Alexei Kitaev, Noam Nisan
15. Superconducting Quantum Circuits, Qubits and Computing - G. Wendin and V.S. Shumeiko
16. Fault-Tolerant Quantum Computation - Peter W. Shor
17. A Hierarchical Approach to Computer-Aided Design of Quantum Circuits – Marek Perkowski, Martin Lukar, Pawel Kerntopf & Mikhail Pivtoraiko et al.
18. Experimental comparison of two quantum computing architectures - Norbert M. Linkea,b,1, Dmitri Maslovc, Martin Roetteler et al.
19. Computer Arithmetic Structures for Quantum Cellular Automata - K. Walus, G. A. Jullien, V. S. Dimitrov
20. Layered Architecture for Quantum Computing - N. Cody Jones,, Rodney Van Meter, Austin G. Fowler et al.
21. http://www.revlib.org/
22. RevLib: An Online Resource for Reversible Functions and Reversible Circuits - Robert Wille, Daniel Große, Lisa Teuber, Gerhard W. Dueck, Rolf Drechsler
23. Parallelizing quantum circuits – Anne Broadbenta, Elham Kashefi
24. Quantum Computing With Superconductors - KARL K. BERGGREN
25. Shor's quantum factoring algorithm on a photonic chip - Alberto Politi, Jonathan C. F. Matthews, and Jeremy L. O'Brien
26. Quantum computing:An IBM perspective - M. Steffen, D. P. DiVincenzo, J. M. Chow T. N. Theis, M. B. Ketchen
27. Universal quantum computation using only projective measurement, quantummemory, and preparation of the $|0\rangle$ state - Michael A. Nielsen
28. Quipper: A Scalable Quantum Programming Language - Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, Benoit Valiron

29. QCADesigner: A Rapid Design and Simulation Tool for Quantum-Dot Cellular Automata - Konrad Walus, Timothy J. Dysart, Graham A. Jullien, Arief R. Budiman
30. Toward a Scalable, Silicon-Based QuantumComputing Architecture - Dean Copsey, Mark Oskin, Francois Impens, Tzvetan Metodiev, Andrew Cross, Student Member, IEEE,Frederic T. Chong, Isaac L. Chuang, and John Kubiatowicz, Member, IEEE
31. Programming quantum computers – Mercedes Gimeno-Segovia, Nic Harrigan & Eric R. Johnston
32.Laser damage creates backdoors in quantum communications – Vadim Makarov, Jean-Philippe Bourgoin et al.
33. Quantum Fog Library of Essays – Robert R Tucci
34. A practical quantum instruction set architecture - Robert S. Smith, Michael J. Curtis, William J. Zeng
35. A Practical Architecture for Reliable Quantum Computers - Mark Oskin, Frederic T. Chong, Isaac L. Chuang
36. Quantum Reed–Solomon Codes - Markus Grassl, Willi Geiselmann, and Thomas Beth
37. Quantum circuits cannot control unknown operations - Mateus Araújo, Adrien Feix, Fabio Costa and Časlav Brukner
38. Test Generation and Fault Localization for Quantum Circuits - Marek Perkowski, Jacob Biamonte, and Martin Lukac
39. Open Quantum Assembly Language - Andrew W. Cross, Lev S. Bishop, John A. Smolin, Jay M. Gambetta
40. A Blueprint for Building a Quantum Computer - Rodney van Meter and Clare Horsman
41. ScaffCC: A Framework for Compilation and Analysisof Quantum Computing Programs - Ali Javadi Abhari, Shruti Patil, Daniel Kudrow, Je Heckey, Alexey Lvov, Frederic T. Chong, Margaret Martonosi
42. QWIRE: A Core Language for Quantum Circuits - Jennifer Paykin, Robert Rand, Steve Zdancewic
43. Quantum vs. Classical Communication and Computation - Harry Buhrman, Richard Cleve, Avi Wigderson
44. A Brief Survey Of Quantum Programming Languages - Peter Selinger
45. A portable high-quality random number generator for lattice field theory simulations – Martin Luscher
46. Coding the Lehmer Pseudo- random Number Generator - W. H. PAYNE, J. 1~. RABUNG, AND T. P. BOGYO
47. A SIMPLE UNPREDICTABLE PSEUDO-RANDOM NUMBER GENERATOR – L. Blum, M. Blum, M Shub
48. RANDOM NUMBER GENERATORS: GOOD ONES ARE HARD TO FIND - STEPHEN K. PARK AND KEITH W. MILLER
49. The art of computer programming, Volume 2 Seminumerical Algorithms – Donald E Knuth
50. A review of Pseudorandom Number Generators – F. James
51. A fast Normal Random Number Generator – JOSEPH L. LEVA
52. THE MCNP5 RANDOM NUMBER GENERATOR - Forrest B. Brown, Yasunobu Nagaya
53. THE INTEL® RANDOM NUMBER GENERATOR - Benjamin Jun and Paul Kocher
54. Recommendation for Random Number Generation Using Deterministic Random Bit Generators - Elaine Barker, John Kelsey
55. Towards a Universal Random Generator – George Marsaglia, Arif Zaman, Wai Wan Tsang
56. Analysis of the Linux Random Number Generator - Zvi Gutterman, Benny Pinkas, Tzachy Reinman
57. A model and architecture for pseudo-random generation with applications to/dev/random - Boaz Barak, Shai Halevi

58. A High Speed, Post-Processing Free, Quantum Random Number Generator - J. F. Dynes, Z. L. Yuan, A. W. Sharpe, and A. J. Shields
59. Random Numbers Certified by Bell's Theorem - S. Pironio1, A. Acın, S. Massar, A. Boyer de la Giroday, D. N. Matsukevich, P. Maunz, S. Olmschenk, D. Hayes, L. Luo, T. A. Manning, and C. Monroe
60. Chaos-Based Random Number Generators—Part I:Analysis - Toni Stojanovski and Ljupco Kocarev, Senior Member, IEEE
61. Unsupervised Machine Learning on a Hybrid Quantum Computer - J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E.Schuyler Fried, S. Hong, P. Karalekas, C. B. Osborn, A. Papageorge, E. C. Peterson, G. Prawiroatmodjo,N. Rubin, Colm A. Ryan, D. Scarabelli, M. Scheer, E. A. Sete, P. Sivarajah, Robert S. Smith, A. Staley,N. Tezak, W. J. Zeng, A. Hudson, Blake R. Johnson, M. Reagor, M. P. da Silva, and C. RigettiRigetti Computing, Inc., Berkeley, CA
62. ProjectQ: An Open Source Software Frameworkfor Quantum ComputingDamian S. Steiger, Thomas Häner, and Matthias Troyer
63. A Software Methodology for Compiling Quantum Programs - Thomas H¨aner, Damian S. Steiger, Krysta Svore, and Matthias Troyer
64. Fault-tolerant reversible circuits – Behriiz Oarhami
65. Synthesis of Reversible Logic Circuits - Vivek V. Shende, Aditya K. Prasad, Igor L. Markov, and John P. Hayes, Fellow, IEEE
66. Quantum computation is the unique reversible circuit model for which bits are balls - Marius Krumm, Markus P. Müller
67. Lecture 1: Introduction to the Quantum Circuit Model - Lecturer: Ryan O'Donnell, Scribe: Ryan O'Donnell
68. Chapter 2: Quantum gates – RP Feynman
69. From Reversible Logic to Quantum Circuits:Logic Design for an Emerging Technology - Robert Wille, Anupam Chattopadhyay, Rolf Drechsler
70. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications - Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks,Alan Heckert, James Dray, San Vo