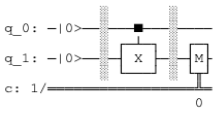
Qiskit Lab 1: Quantum Circuits

In the Qiskit book, Lab 1 is called Quantum Circuits and it focuses on implementing classical logic gates with quantum circuits. Do the lab plus the additional task below and submit the following.

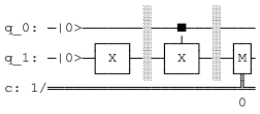
- In Part 1, show your code and circuit diagrams for XOR, AND, NAND, OR.
  - XOR

```
def XOR(inp1,inp2):
    """An XOR gate.

    Parameters:
        inpt1 (str): Input 1, encoded in qubit 0.
        inpt2 (str): Input 2, encoded in qubit 1.

    Returns:
        QuantumCircuit: Output XOR circuit.
        str: Output value measured from qubit 1.
    """

    qc = QuantumCircuit(2, 1)
    qc.reset(range(2))

    # For an input of '1', we do an x to rotate the |0) to |1)
    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)

    # barrier between input state and gate operation
    qc.barrier()

    # this is where your program for quantum XOR gate goes
    # Do an x to rotate the second qubit if the first qubit is 1
    qc.cx(0, 1)

    # barrier between input state and gate operation
    qc.barrier()

    qc.measure(1,0) # output from qubit 1 is measured

    #We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    #Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc, shots=1, memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

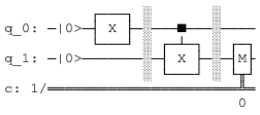
XOR with inputs 0 0 gives output 0



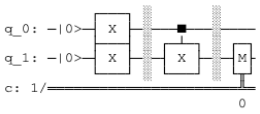
XOR with inputs 0 1 gives output 1



XOR with inputs 1 0 gives output 1



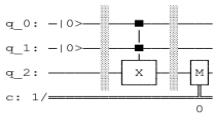
XOR with inputs 1 1 gives output 0



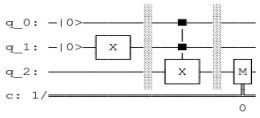
  - AND

```
def AND(inp1,inp2):
    """An AND gate.

    Parameters:
        inpt1 (str): Input 1, encoded in qubit 0.
        inpt2 (str): Input 2, encoded in qubit 1.

    Returns:
        QuantumCircuit: Output XOR circuit.
        str: Output value measured from qubit 2.
    """
    qc = QuantumCircuit(3, 1)
    qc.reset(range(2))

    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)

    qc.barrier()

    # this is where your program for quantum AND gate goes
    # Do an x to rotate the helper qubit from 0 to 1 if both circuit are 1
    qc.ccx(0, 1, 2)

    qc.barrier()
    qc.measure(2, 0) # output from qubit 2 is measured

    # We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    # Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc, shots=1, memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

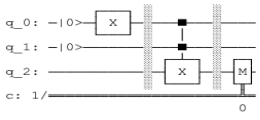
AND with inputs 0 0 gives output 0



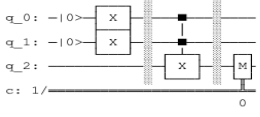
AND with inputs 0 1 gives output 0



AND with inputs 1 0 gives output 0



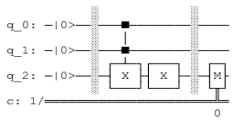
AND with inputs 1 1 gives output 1



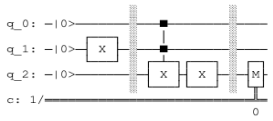
- NAND

```
def NAND(inp1,inp2):
    """An NAND gate.

    Parameters:
        inpt1 (str): Input 1, encoded in qubit 0.
        inpt2 (str): Input 2, encoded in qubit 1.

    Returns:
        QuantumCircuit: Output NAND circuit.
        str: Output value measured from qubit 2.
    """
    qc = QuantumCircuit(3, 1)
    qc.reset(range(3))

    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)

    qc.barrier()

    # this is where your program for quantum NAND gate goes
    # Do an x to rotate the helper qubit from 0 to 1 if both circuit are not 1
    qc.ccx(0, 1, 2)
    qc.x(2)

    qc.barrier()
    qc.measure(2, 0) # output from qubit 2 is measured

    # We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    # Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc,shots=1,memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

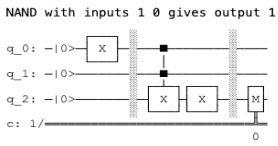
NAND with inputs 0 0 gives output 1



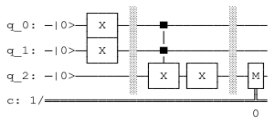
NAND with inputs 0 1 gives output 1



NAND with inputs 1 0 gives output 1



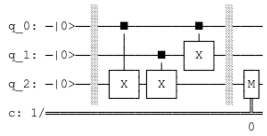
NAND with inputs 1 1 gives output 0



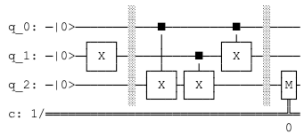
- OR

```
def OR(inp1,inp2):
    """An OR gate.

    Parameters:
        inpt1 (str): Input 1, encoded in qubit 0.
        inpt2 (str): Input 2, encoded in qubit 1.

    Returns:
        QuantumCircuit: Output XOR circuit.
        str: Output value measured from qubit 2.
    """

    qc = QuantumCircuit(3, 1)
    qc.reset(range(3))

    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)

    qc.barrier()

    # this is where your program for quantum OR gate goes
    # Do an x to rotate the helper qubit from 0 to 1 if either qubit is 1
    qc.x(0)
    qc.x(1)
    qc.ccx(0, 1, 2)
    qc.x(2)

    qc.barrier()
    qc.measure(2, 0) # output from qubit 2 is measured

    # We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    # Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc,shots=1,memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

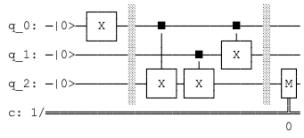
OR with inputs 0 0 gives output 0



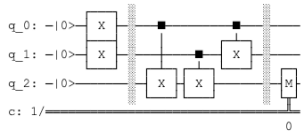
OR with inputs 0 1 gives output 1



OR with inputs 1 0 gives output 1



OR with inputs 1 1 gives output 0

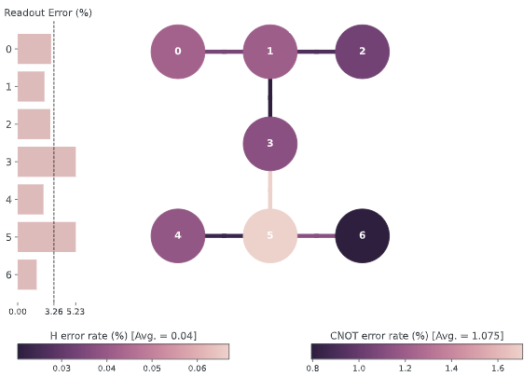


- In Part 2:
    - In Step 2, show your three-qubit initial layout and describe the reason for your choice.

The three-qubit initial layout I used is layout = [2, 3, 1].

The following image is the error map for the system ‘ibm_nairobi’ . In this image, we can observe the error distribution between the qubits. Since most of the gates in the transpiled AND gate are CNOT, we should decide on the three-qubit initial layout based on the CNOT error rate between the qubits. We can find out that the “physical” qubit Q1 is the qubit that has a low CNOT error rate with its neighbor qubits Q2 and Q3, so Q1 can be mapped to the target qubit, and Q2 and Q3 can be mapped to the controlled qubit in the quantum circuit.

Therefore, I put 2 and 3 as the first two elements which represent the “physical” qubits that the two controlled qubits (inp1 and inp2) will map to, and put 1 as the third element which represents the “physical” qubit that the target qubit will map to.



last_update_date: 2023-11-03 15:47:30+00:00

	Type	Gate error
cx6_5	cx	0.01129
cx5_6	cx	0.01129
cx5_4	cx	0.00849
cx4_5	cx	0.00849

	Type	Gate error
cx5_3	cx	0.01707
cx3_5	cx	0.01707
cx1_3	cx	0.00794
cx3_1	cx	0.00794

	Type	Gate error
cx2_1	cx	0.00912
cx1_2	cx	0.00912
cx0_1	cx	0.0106
cx1_0	cx	0.0106

Output of Step 2:

```
cn4xkxqvayrg008ep9ng
Job Status: job has successfully run

Probability of correct answer for inputs 0 0
0.97
---------------------------------
cn4zrvb3r3vg008faqw0
Job Status: job has successfully run

Probability of correct answer for inputs 0 1
0.95
---------------------------------
cn4zyb1p1am0008qcs30
Job Status: job has successfully run

Probability of correct answer for inputs 1 0
0.94
---------------------------------
cn4zzpe3r3vg008farag
Job Status: job has successfully run

Probability of correct answer for inputs 1 1
0.85
---------------------------------


The highest of these probabilities was 0.97
The lowest of these probabilities was 0.85
```

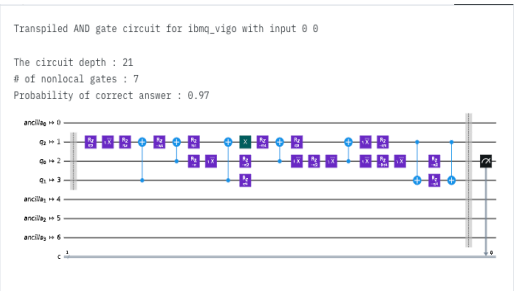
- In Step 3, explain the reasons for the dissimilarity of the circuits. Describe the relations between the property of the circuit and the accuracy of the outcomes.

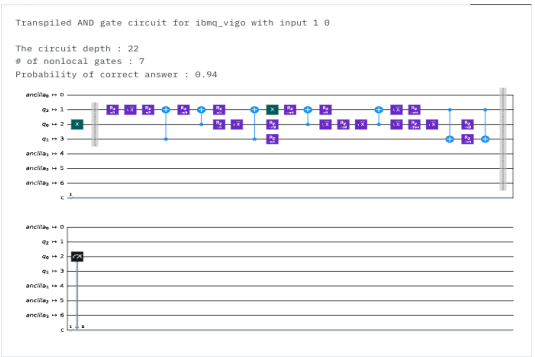
Dissimilarity: From these four circuits, we can see that the difference between them lies in their input, the number of nonlocal gates in the circuit, and the circuit depth. The dissimilarity is because each circuit addresses a unique input, resulting in a different arrangement of the gates. The arrangement of gates can significantly impact the circuit depth and the number of nonlocal gates. For example, in the case of input 01, its circuit has a unique input 01, so it has a different gate arrangement than other inputs' and it also makes the number of nonlocal gates more than other inputs'.

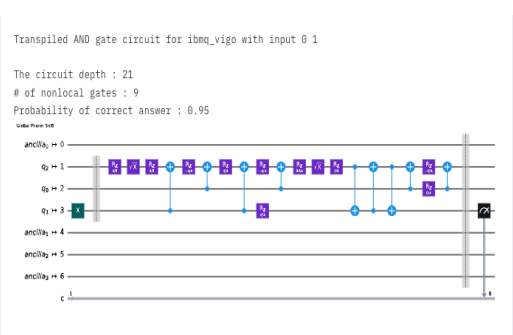
Relations between the property of the circuit and the accuracy of the outcomes:
The properties of the circuit are "the circuit depth" and "# of nonlocal gates". The circuit depth is proportional to the number of gates in a circuit and corresponds to the runtime of the circuit on hardware, so it doesn't make much impact on the accuracy of the outcomes. Another property is the number of nonlocal gates. In this case, the only nonlocal gate is the CNOT gate. CNOT gates are the most expensive gates to perform and have different gate error rates depending on the controlled qubit and the target qubit, so it will affect the accuracy of the outcomes.

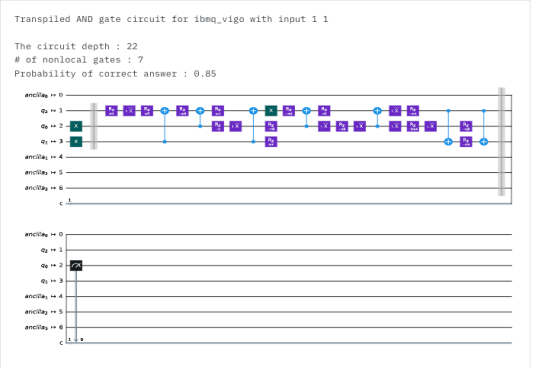
Examples: The CNOT gate will rotate the target qubit only if the controlled qubit is 0. In the case of input 00, the controlled qubit is 0 and it won't rotate the target qubit, so the error generated by the CNOT gate won't have a significant impact on the accuracy of the outcomes of this circuit. However, in the case of input 11, the controlled qubit is 1 and it will rotate the target qubit, so the CNOT gate is literally working, and the probability of error generated by the CNOT gate would be much larger. By comparing the result between input 00 and input 11, we can observe that the probability of correct answer for input 11 is much lower than the probability for input 00 which proves my point.

Output of Step 3:









- Additional task:
    - Do Part 2 again, but this time for the logical expression

(x AND y) OR z

where x,y,z are Boolean variables that you will represent as three qubits. In Step 2, show your qubit initial layout for all the qubits you use and describe the reason for your choice. In Step 3, list the accuracy of the outcome for each circuit, and list the maximum difference across those accuracies.

Step 2:

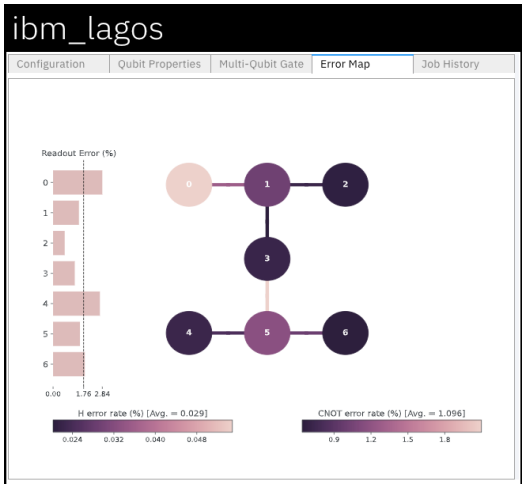
Code for the logical expression:

```
def TASK(inp1, inp2, inp3, backend, layout):

    qc = QuantumCircuit(5, 1)
    qc.reset(range(5))

    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)
    if inp3=='1':
        qc.x(2)

    qc.barrier()
    # x AND y
    qc.ccx(0, 1, 3)

    # (x AND y) OR z
    qc.x(2)
    qc.x(3)
    qc.ccx(2, 3, 4)
    qc.x(4)

    qc.barrier()
    qc.measure(4, 0)

    qc_trans = transpile(qc, backend, initial_layout=layout, optimization_level=3)
    job = backend.run(qc_trans, shots=8192)
    print(job.job_id())
    job_monitor(job)

    output = job.result().get_counts()

    return qc_trans, output

layout = []
```

```
output_all = []
qc_trans_all = []
prob_all = []

worst = 1
best = 0
for input1 in ['0','1']:
    for input2 in ['0','1']:
        for input3 in ['0', '1']:
            qc_trans, output = TASK(input1, input2, input3, backend, layout)

            output_all.append(output)
            qc_trans_all.append(qc_trans)

            prob = output[str(int( input1=='1' and input2=='1' and input3=='1'))]/8192
            prob_all.append(prob)

            print('\nProbability of correct answer for inputs',input1,input2,input3)
            print('{:.2f}'.format(prob) )
            print('--------------------------------')

            worst = min(worst,prob)
            best = max(best, prob)

print('')
print('\nThe highest of these probabilities was {:.2f}'.format(best))
print('The lowest of these probabilities was {:.2f}'.format(worst))
```

The five-qubit initial layout I used is layout = [4, 6, 1, 5, 3].

The following image is the error map for the system ‘ibm_lagos’ . Since most of the gates we will use for the logical expression are also CNOT, we should decide on the five-qubit initial layout based on the CNOT error rate between the qubits.



ibm_lagos

Configuration | Qubit Properties | Multi-Qubit Gate | Error Map | Job History

last_update_date: 2023-11-07 17:03:03+00:00

	Type	Gate error
cx5_4	cx	0.00828
cx4_5	cx	0.00828
cx3_1	cx	0.0064
cx1_3	cx	0.0064

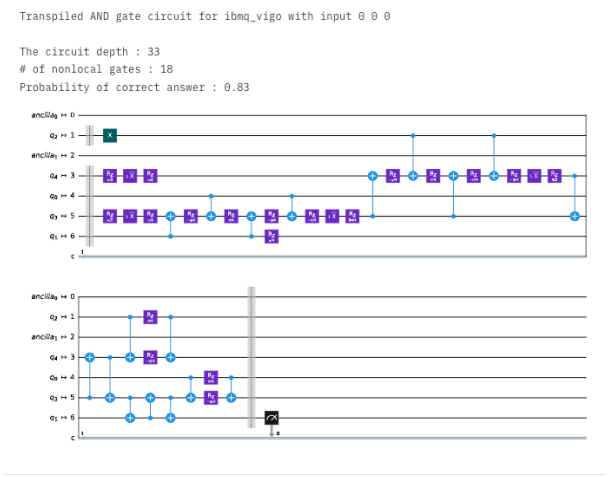
	Type	Gate error
cx5_6	cx	0.00996
cx6_5	cx	0.00996
cx3_5	cx	0.02097
cx5_3	cx	0.02097

	Type	Gate error
cx2_1	cx	0.00727
cx1_2	cx	0.00727
cx0_1	cx	0.01286
cx1_0	cx	0.01286

First Toffoli gate: we can find out that the “physical” qubit Q5 is the qubit that has a low CNOT error rate with its neighbor qubits Q4 and Q6, so Q5 can be mapped to the target qubit, and Q4 and Q6 can be mapped to the controlled qubits for the first CCNOT gate. Therefore, I put 4 and 6 as the first two elements which represent the “physical” qubits that the two controlled qubits (inp1 and inp2) will map to, and put 5 as the fourth element which represents the “physical” qubit that the target qubit will map to.

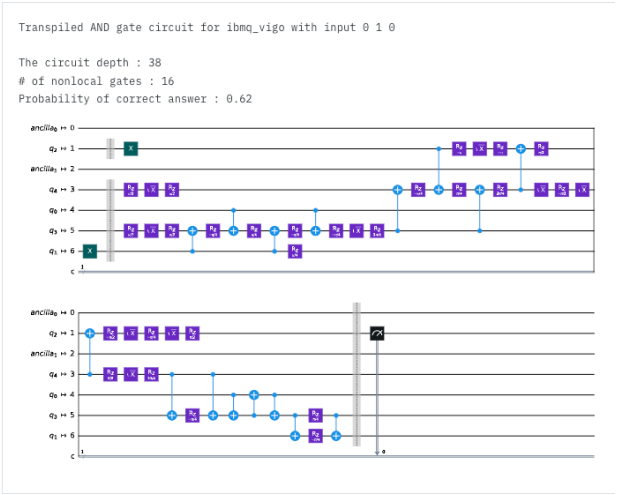
Second Toffoli gate: Since Q5 must be one of the controlled qubits, Q3 must be the target qubit which is the only qubit with the neighbor Q5 besides Q4 and Q6. Q1 is another neighbor of Q3, so it should be another controlled qubit. Therefore, I put 1 as the third element which represents the “physical” qubit (inp3) that one of the controlled qubits will map to, and put 3 as the last element which represents the “physical” qubit that the target qubit will map to and also the measured output.

Step 3:
The accuracy of the outcome for each circuit:
000: 0.83



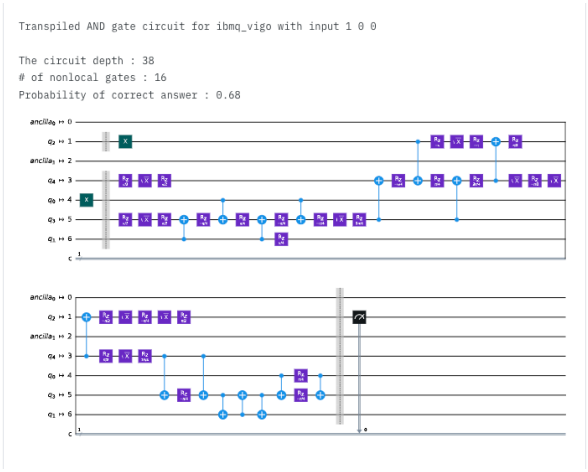
001: 0.13

010: 0.62

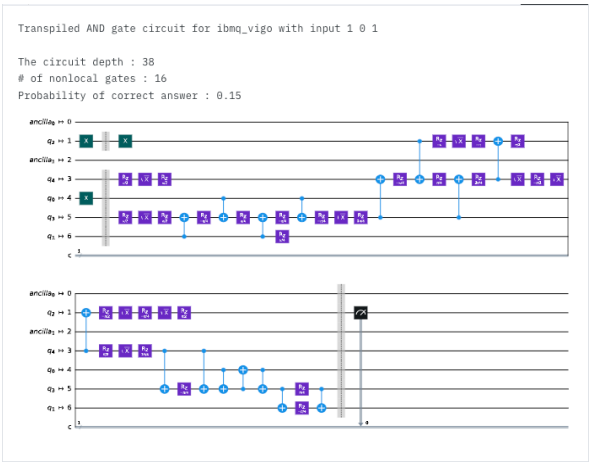


011: 0.19

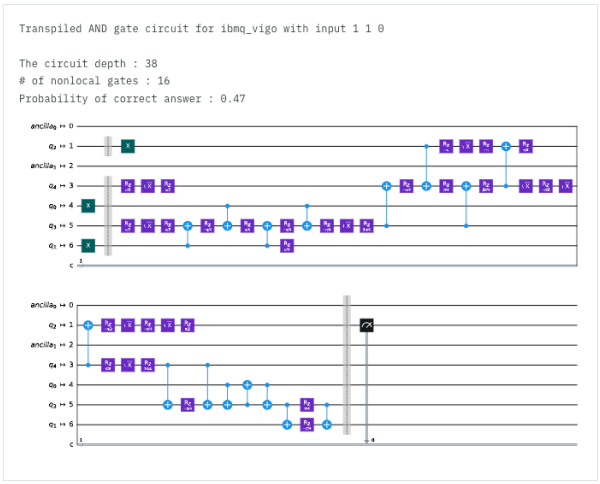
100: 0.68



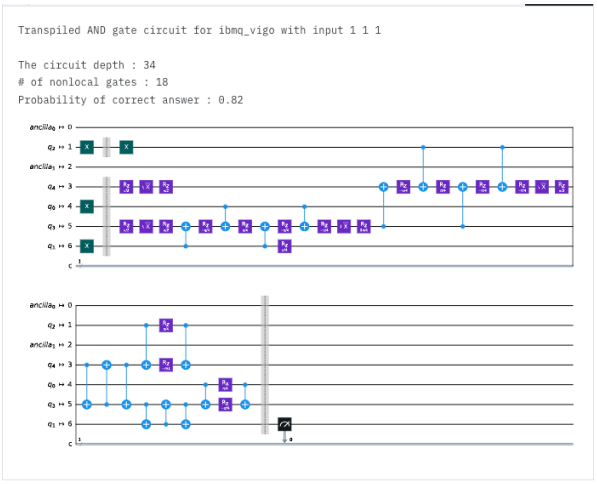
101: 0.15



110: 0.47



111: 0.82



The maximum difference across those accuracies:

Max = 0.83, Min = 0.13

The maximum difference across those accuracies = Max – Min = 0.83 – 0.13 = 0.7