QOSF Cohort 6- Task 1

Trey Jiron

1 Problem

Task 1 Multiplier To make a multiplier, for this we design the input of two positive integers to a function and this function will process a quantum algorithm that makes the multiplier (see Draper adder) and returns the result in an integer.

you cannot use any implementation already designed by the framework

def multiplier(int:number1, int ,number2):

number1 : integer positive value that is the first parameter to the multiplier function, number2 : integer positive value that is the second parameter to the multiplier function. Return the positive integer value of the multiplication between number1 and number2

use a framework that works with quantum circuits, qiskit, cirq, pennylane, etc.

define a quantum circuit to convert the integer values in qubits, example bases encoding basis encoding: n bits are equals to a state of n qubits, example The integer value 3 convert to a binary string that is 11, the basis encoding is $|11\rangle$

use the state of the art to check the possibles ways to design a multiplier

Return the result of the quantum circuit into an integer value

Example:

A = multiplier(5,6) print(A)

30

2 Thought process

The idea is to base the quantum circuit for multiplication off of a classical circuit. In classical computation a multiplication of two bits can be achieved by an AND gate, an AND gate is a logic gate that has two inputs and one output. In quantum circuit the equivalent to an AND gate is the Toffoli gate. A Toffoli gate is a quantum logic gate that has 3 inputs and 3 outputs. We can compare the output table of an AND gate and of a Toffoli gate, and see that the output of Toffoli gate can match that of an AND gate given the 3rd qubit is initially 0.

With a Toffoli gate we are able to multiply two bits. We can now look at a classical circuit for multiplying two 2-bit numbers. A circuit for this can be seen in fig. 3. Note that this circuit requires an XOR gate another type of classical logic gate. This also has a quantum counter part in a form of a controlled not

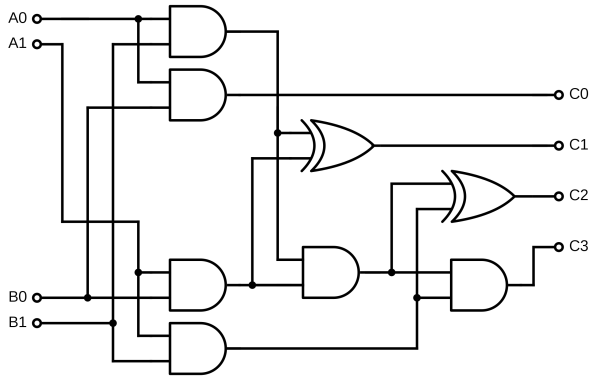
Figure 1: Logic table for classical AND gate. Source: AND gate Wikipedia page.

Input		Output
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2: Logic table for Toffoli gate, note that so long as the 3rd input is 0, it then the 3rd output follows the same logic as a classical AND gate. Source: Toffoli gate Wikipedia page

INPUT			OUTPUT		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Figure 3: Logic circuit for multiply 2 2-bit numbers. multiplies number A with bits A0 and A1 and number B with bits B0 and B1.Source: Binary multiplier Wikipedia page



gate (CNOT gate). A table that compares the logic tables for these two gates are seen in Fig. 4.

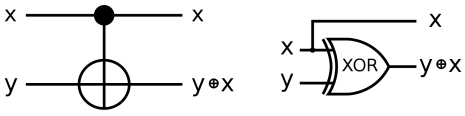
Now that we have a quantum equivalent to both an AND gate and a XOR gate we are able to implement the circuit seen in fig. 3 on a quantum computer.

3 Implementation

The initial problem found with implementing this circuit was figuring out the number of qubits needed for it. While the classical diagram shows only 4 bits, I found that we need additional bits to store the outputs of the gates for later usage. I used IBM's Q experience to design the circuit and test the initial circuits. I ended up using a quantum circuit that uses 11 qubits, to help with the design process I took the diagram in Fig. 3 and put numbers over the gates. These numbers indicated which qubit was being used to store their output, making it easier to trace what outputs had to feed into what input. This is shown in Fig. 5, note that 8 and 4 are used twice. that is because the XOR gate is a CNOT gate that uses the same qubit as a input as the same qubit as an output meaning it can stay on the same qubit. Lastly we use switch gates to switch the qubits to the proper classical registers as seen in Fig.3. The final quantum gate is shown in Fig. 6

We are then able to export this circuit to qiskit code. By writing additional code that takes input as integers, turns them into binary, change the needed qubits and using this circuit gives us an answer in binary. We can then change the binary answer back to an integer to get a final answer. This is a 2x2 bit multiplier implemented on a quantum computer. The code for this is in the same file as this file.

Figure 4: Logic tables for XOR gate and CNOT gate.Source: CNOT gate Wikipedia page



input		output	
x	y	x	y+x
|0⟩	|0⟩	|0⟩	|0⟩
|0⟩	|1⟩	|0⟩	|1⟩
|1⟩	|0⟩	|1⟩	|1⟩
|1⟩	|1⟩	|1⟩	|0⟩

input		output	
x	y	x	y+x
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Figure 5: Classical gate modified to show what qubits are used.

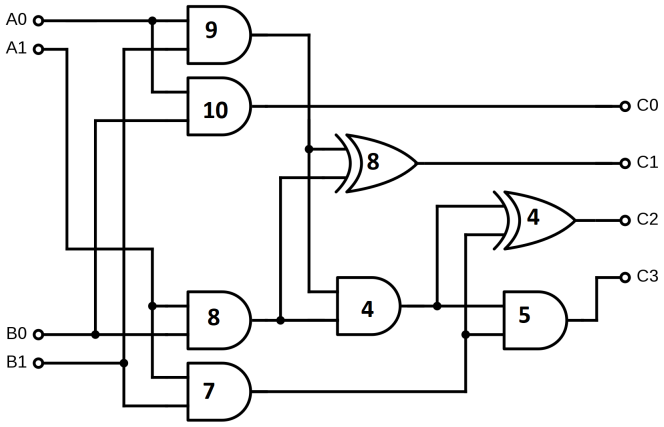
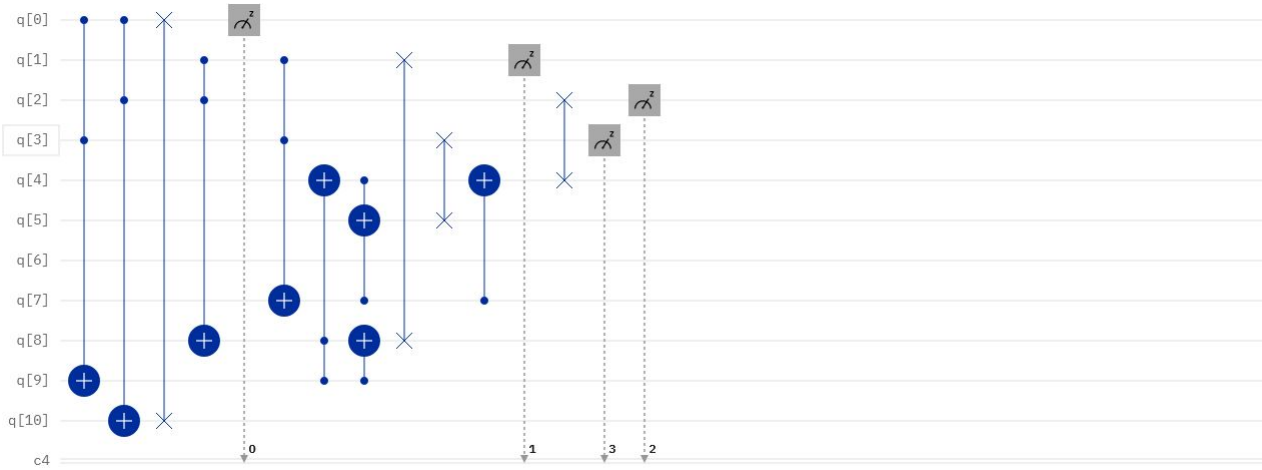


Figure 6: Final quantum circuit used, made in IBM's Q experience.



4 Conclusion

Although this process does produce a 2x2 bit quantum multiplier, it is most likely not the most efficient way to do so. It requires a non-trivial amount of qubits at 11, and this number will only increases as you increase how big you want the numbers that are being multiplied. Not to mention the number of gates would also increase, leading to more error when implemented on a physical machine. As such I am curious how other people tackled this task and how they got over the issues involved in the task.