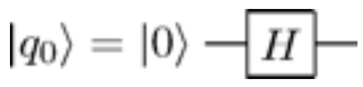
QUANTUM COMPUTER SIMULATION
LAB 1

In this Lab, we use the QX Quantum Computer Simulator available at http://quantum-studio.net

Exercise 1 (Basic Gates)

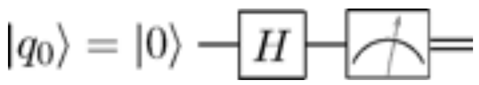
Write the following circuits:

1. The Hadamard gate:

Quantum Code	Quantum Circuit
qubits 1 *# define 1 qubit* display *# initial state* **H** q0 **display** *# result*	

A. Execute this circuit and display the quantum state. What does the circuit do ?

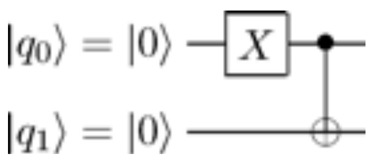
We add a measurement of the qubit “q0” after the Hadamard :

Quantum Code	Quantum Circuit
qubits 1 *# define 1 qubit* display *# initial state* **H** q0 **measure** q0 **display** *# result*	

B. Execute this circuit several times and observe the measurement outcome.
What do you observe ? Why ?

2. The CNOT gate:

Execute the following circuit using the QX simulator with and without the X gate and observe the outcome of the CNOT gate.

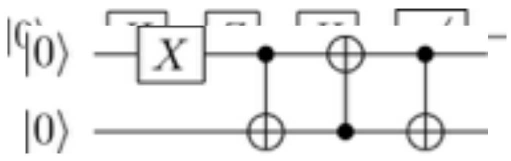
Quantum Code	Quantum Circuit
qubits 2 display **X** q0 **display** **CNOT** q0,q1 **display**	

- What’s the outcome state of this circuit ?

3) Equivalent gates:

A. Write the following quantum circuit.
Replace the {H,Z,H} gate sequence with a single gate and observe the result.

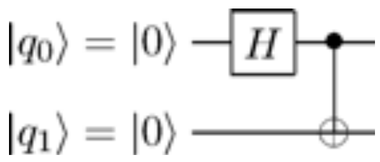
B. Write the following quantum circuit.



Replace the 3 CNOT gates sequence by a single equivalent gate and observe the result. Insert a “display” command at the end of the circuit.

Exercise 2 (The Bell Pair)

Write the following circuit which creates a Bell Pair

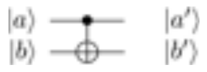
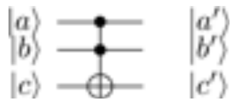


Add measurement on first qubit “q0” and display the outcome state.
What is the output of this circuit ?
Why ?

Exercise 3 (Arithmetics: Quantum Plain Adder)

We want to construct a quantum circuit which implements the classical addition between 2-bit operators.

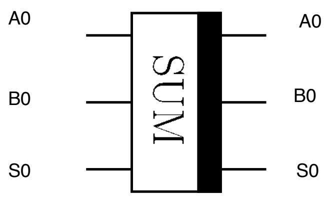
The quantum circuit can be implemented as a network of “Sum” and “Carry” blocks. These two blocks can be built using CNOT and Toffoli gates. We provide the truth tables of the CNOT and the Toffoli gates in the following tables:

CNOT	TOFFOLI
	

a b	a' b'
0 0	0 0
0 1	0 1
1 0	**1 1**
1 1	**1 0**

a b c	a'b'c'
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**

1. The "Sum" Circuit



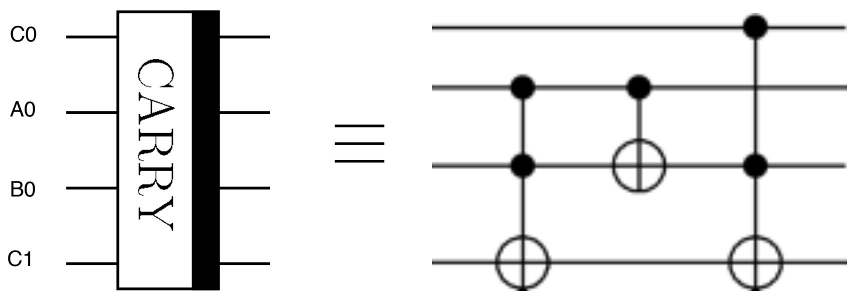
Note: refer to the QX User's Manual to find out how to rename qubits.

Use CNOT gates to implement the "Sum" block which sums two bits A0 and B0 and stores the result in a third bit S0.

Write the circuit and save it in a file named "sum.qc". The sum circuit should include 2 sub-circuits: ".init" and ".sum". We note that the ".init" circuit initialises A0 to l0> and B0 to l1>, the ".sum" circuit performs the operation and stores the result in S0.

2. The "Carry" Circuit

The "Carry" block uses 3 inputs which are : a carry-bit C0 from the previous stage and 2 operator bits (A0 and B0). The output is stored into a fourth bit: a carry-bit C1 which can then be used on the next stage. The circuit is given in the following diagram.



A. Use “CNOT” and “Toffoli” gates to implement the carry block. Write the circuit in a file named “carry.qc”. The circuit is composed of 2 sub-circuits “.init” and “.carry”, each sub-circuit ends with a “display” command. We note that the initialisation circuit initialises the qubits C0, A0, B0 and C1 to the respective values I0>,I1>,I1>,I0>.
B. Copy the previous circuit into a new file named “rcarry.qc”. We modify the new circuit as following:
   - Modify the “.init” sub-circuit to initialise the qubits C0,A0,B0,C1 to the state I0>,I1>,I0>,I1>.
   - Rename the sub-circuit “.carry” to “.rcarry” and reverse the order of the gates of the “.rcarry” gates.
C. Execute the circuit “rcarry.qc” and observe its output state. What’s does the “rcarry.qc” circuit implement ?

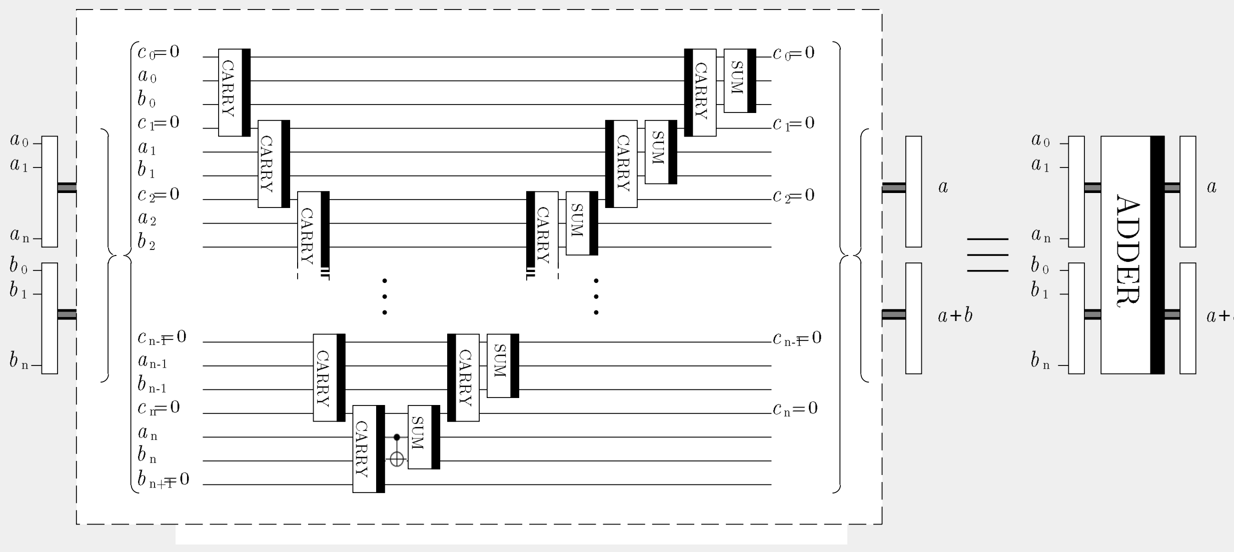
3. Plain 2-bits Adder Circuit

Now we will implement a 2 qubit adder. To this purpose, we use the two blocks implemented in section 2 based on the following generic n-bits adder diagram shown in the next Figure. The circuit starts with a first sub-circuit named “.init” and containing the gate sequence which initialises the bits of the adder’s operators: A and B. We initialise A to the binary value “10” and we initialise B to “01”.

The “sum” and “carry” blocks should be implemented as sub-circuits and named “.carry_1”, “.carry_2”,”.sum_1”, “.sum_2” … etc

Notes:
- Add one “display” command at the end of the initialisation circuit, and add another “display” command at the end of the circuit.
- Notice that both the carry/sum and **<u>their reverse</u>** are used in the plain adder. Finally, note that the CNOT gate between the bottom Carry and Sum blocks is required.

N-bits Adder Circuit Diagram



Exercise 4 (Deutsch Problem Simulation)

Deutsch's algorithm creates four functions which map a set of classical bits to itself.
The four mapping functions are the following:

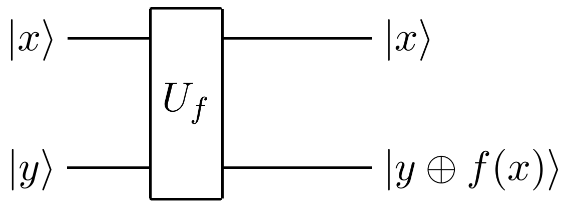
U_{f1} : { 0 —> 0 ; 1 —> 1 }
U_{f2} : { 0 —> 1 ; 1 —> 0 }
U_{f3} : { 0 —> 0 ; 1 —> 0 }
U_{f4} : { 0 —> 1 ; 1 —> 1 }

You can notice that these four functions cover every possible mapping between {0,1} I—> {0,1} .
We define a constant function as f(0) = f(1) and we define a balanced function as f(0) =/= f(1).
Therefore U_{f1} and U_{f2} are balanced while U_{f3} and U_{f4} are constant.

Without knowing which of these functions are used, Deutsch's algorithm can detect whether the used function U_f is constant or balanced. While a classical algorithm needs to evaluate f(0) and then f(1) to decide whether they are equal or different, Deutsch's quantum algorithm can evaluate them simultaneously using the quantum superposition of the basis states.

We want to write and simulate the quantum circuit implementing Deutsh's algorithm.

5.1. The U_f Functions



A. Write four circuits which implement the four U_f functions (U_{f1} , U_{f2} , U_{f3} and U_{f4}) in 4 files named respectively "uf1.qc" , "uf2.qc", "uf3.qc" and "uf3.qc".

To make the circuits more readable, we name the qubits "q0" and "q1" respectively "qy" and "qx" using the QX's "**map**" command (refer to the QX User's Manual).

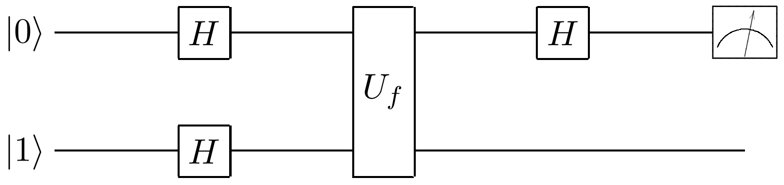
Each of the 4 circuits contains 2 sub-circuits:

1. The first sub-circuit is named ".init" and initialises 2 qubits in the state I01> ("qx" to I0> and "qy" to I1>)
2. The second "sub-circuit" implements the U_f function and is called ".uf" .

Add a "*display*" instruction at the end of each of the 2 sub-circuits.

B. Simulate the execution of the circuits using the QX simulator to check if your circuit implements the function correctly.

5.1. Deutsch's Algorithm Implementation



A. Write the four circuits implementing Deutsch's algorithm and using each of the implemented four U_f functions. The circuit files are named respectively "deutsch_uf1.qc", "deutsch_uf2.qc", "deutsch_uf3.qc" and "deutsch_uf4.qc".

Each of the 4 circuits contains 4 sub-circuits:

1. The first sub-circuit is named ".init" and initialises 2 qubits in the state I01> (q0 to I1> and q1to I0>)
2. The second sub-circuit puts the qubits into a superposition and should be named ".superposition"
3. The third sub-circuit implements the U_f function.
4. The last sub-circuit is called ".result" and applies an H gate the qubit 1 and measures it.

Make sure a "display" instruction is added at the end of each sub-circuit to display all the intermediate and the final quantum states.

B. Simulate the execution of the four circuits and verify that the outcome of the qubit 1 measurement is "1" when a balanced function is used and "0" when a constant one is used.