# Quantum Computing Architecture & Electronics Homework 1

Héctor Calero (5844460)

March 2023

## 1 Exercise 1: Bell Pair (0.5 Points)

First we shall write a Bell.qc code in our QX simulator that creates a Bell pair and measures qubit $q_0$:

```
# Bell Pair circuit
version 0.5
qubits 2
h q[0]
cnot q[0],q[1]
display
measure q[0]
display
```

### 1.1 What is the final state (before measurement)? [0.25 points]

The resulting state after applying a Hadamard gate on the first qubit and a CNOT between the first and second qubit is $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. This is indeed one of the Bell states and is in agreement with the results of the simulation (before the measurement):

```
--------------[quantum state]--------------
  [p = +0.5000000]  +0.7071068 + +0.0000000 * i |00> +
  [p = +0.5000000]  +0.7071068 + +0.0000000 * i |11> +
-------------------------------------------
```

### 1.2 What is the state of the system after measuring $q_0$? Why? [0.25 points]

From the previous state it is clear that the measurement of $q_0$ will yield 1 or -1, both with 50% probability. If we measure +1 the system will collapse onto the state $|00\rangle$ and if we measure -1 it will collapse onto $|11\rangle$. The question of why this is the case is rather philosophical but from an intuitive point of view we can argue that since the pre-measurement state is a Bell state (maximally entangled), the state of each of the qubits is completely determined by the other. Thus, measuring $q_0$ allows us to obtain deterministic information about $q_1$ and vice-versa.

## 2 Exercise 2: Teleportation (1.5 Points)

The code for the steps of the Teleportation protocol that we used is the following:

```
# Teleportation circuit
version 0.5
qubits 3

.epr
h q[1]
cnot q[1],q[2]
display

.init
x q[0] #Standard Initialization
#Ry q[0],pi/2  # pi/2 rotation around the Y axis
#Rx q[0],pi/2   #pi/2 rotation around X axis
display

.encode
cnot q[0],q[1]
h q[0]
measure q[0]
measure q[1]
display

.decode
c-x b[1],q[2]
c-z b[0],q[2]
#h q[2]
#measure q[2]
display
```

After running the simulations we observe that the state of $q_2$ always coincides with the state of $q_0$ after the initialization and therefore the teleportation protocol is successful with probability 1.

### 2.1 Repeat the experiment by changing the initialisation to an arbitrary rotation about the Y-axis by $\pi/2$. Display the state received by Bob after teleportation and decoding. [0.6 points]

Now we change the initialization and add a $\pi/2$ rotation around the Y axis instead of an X gate. The outcome of the simulation is:

```
-------------------------------------------
Complex amplitudes with probabilities
011       0.707107 + 0 * i (0.500000)
111       0.707107 + 0 * i (0.500000)
-------------------------------------------
```

This state can be expressed as $|11\rangle_{Alice} \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_{Bob} = |\psi\rangle_{Alice} \otimes |+\rangle_{Bob}$
Alice's state may be different for each simulation but Bob's state is always in the state $|+\rangle$.

### 2.2 Bob measures the qubit. What is the value he gets? What would happen if Alice initialised to an arbitrary rotation about the X-axis by $\pi/2$ instead? [0.6 points]

If we measure the state $|+\rangle$ in the computational basis we have a 50% probability of obtaining a measurement outcome +1 (state $|0\rangle$) and 50% of measuring -1 (state $|1\rangle$).

If we repeated the same procedure but initializing the state to a $\pi/2$ rotation around the X axis, Bob's state would read $|\psi\rangle_{Bob} = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$. Again, the probability of measuring $\pm 1$ is 50%.

**2.3 What do we need to do to distinguish between these two cases? [0.3 points]** As we have seen, we cannot distinguish between these cases if we measure in the computational basis. The solution is to measure in a different basis. For instance, if we recall that the action of the Hadamard gate on the state $|+\rangle$ yields $|0\rangle$, we can guarantee that if we apply an H gate before the measurement on the Z basis, we will always obtain the measurement outcome +1 for this state (note that this is equivalent to measure in the X basis). Nevertheless, if Bob's state is $\frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$, we will again obtain a 50% probability for each possible outcome. Then we can easily distinguish between these two cases.

## 3 Exercise 3: Quantum Plain Adder (4 Points)

### 3.1 The Sum Circuit

**Write the circuit and save it in a file named sum.qc. The sum circuit should include two sub-circuits: .init and .sum. We note that the .init circuit initializes A0 to $|0\rangle$ and B0 to $|1\rangle$, the .sum circuit performs the operation and stores the result in S0. [1.0 point]**

```
# Sum circuit
version 0.5
qubits 3

.init
display
x q[1]

.sum
cnot q[0],q[2]
cnot q[1],q[2]
display
```

The result of running this circuit yields:

```
-------------------------------------------
Complex amplitudes with probabilities
110       1 + 0 * i (1.000000)
```

Which represents A0=0, B0=1 and S0=A0+B0=1, as expected.

### 3.2 The Carry Circuit

**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 two sub-circuits .init and .carry, each sub-circuit ends with a display command. We note that the initialization circuit initializes the qubits C0, A0, B0 and C1 to the respective values $|0\rangle$, $|1\rangle$, $|1\rangle$, $|0\rangle$. [1.0 point]**

The carry.qc code reads:

```
# Carry circuit
version 0.5
qubits 4

.init
x q[1]
x q[2]
display

.carry
Toffoli q[1],q[2],q[3]
cnot q[1],q[2]
Toffoli q[0],q[2],q[3]
display
```

And its output is $|C_1B_0A_0C_0\rangle = |1010\rangle$.

**B. Copy the previous circuit into a new file named rcarry.qc. Modify the new circuit as following: Change the .init sub-circuit to initialize the qubits C0, A0, B0, C1 to the states $|0\rangle$, $|1\rangle$, $|0\rangle$, $|1\rangle$. Rename the subcircuit .carry to .rcarry and reverse the order of the gates of the .rcarry gates.**

After applying the modifications, the new code is the following:

```
# Rcarry circuit
version 0.5
qubits 4

.init
x q[1]
x q[3]
display

.rcarry
Toffoli q[0],q[2],q[3]
cnot q[1],q[2]
Toffoli q[1],q[2],q[3]
display
```

**C. Execute the circuit rcarry.qc and observe its output state. What does the rcarry.qc circuit implement? [1.0 point]**

The output state that we get is 0110.

```
---------------------------------------------
Complex amplitudes with probabilities
0110       1 + 0 * i (1.000000)
```

This state is precisely the input of the carry state for which we have an output of 1010. One can check that in general the rcarry.qc implements precisely the inverse operation of carry.qc.

## 3.3 Plain 2-bits Adder Circuit

**Now we will implement a 2-qubit adder. To this purpose, we will use the two blocks implemented in Section 3 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 contains the gate sequence which initializes the bits of the adder's operators: A and B. We initialize A to the binary value 10 and we initialize 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. [1.0 point].**

For the 2-bits Adder circuit we need 7 qubits $(a_0, a_0, b_0, b_1)$ and three carry bits.

```
# Adder circuit
version 0.5
qubits 7

.init
x q[2]
x q[4] # We initialize A=10, B=01
display

.carry1
Toffoli q[1],q[2],q[3]
cnot q[1],q[2]
Toffoli q[0],q[2],q[3]

.carry2
Toffoli q[4],q[5],q[6]
cnot q[4],q[5]
Toffoli q[3],q[5],q[6]

.cnot
cnot q[4],q[5]

.sum1
```

```
cnot q[3],q[5]
cnot q[4],q[5]

.rcarry1
Toffoli q[0],q[2],q[3]
cnot q[1],q[2]
Toffoli q[1],q[2],q[3]

.sum2
cnot q[0],q[2]
cnot q[1],q[2]
display
```

The result of running this circuit is 0110100:

```
-------------------------------------------
Complex amplitudes with probabilities
0110100       1 + 0 * i  (1.000000)
```

We need to interpret this result in the following way:

```
 0  1  1  0  1  0  0
s2 s1 a1 c1 s0 a0 c0
```

This is, the initialization value of A is stored in the 3rd and 6th digits (from left to right). We can indeed check that it corresponds to A=10. The value of the carry bits is 0 and the value of the sum (S=A+B) is defined by the 1st, 2nd and 5th bits. In our case S=011 which effectively corresponds to 10+01. In fact we can check that our circuit works for all possible combinations of A and B provided that we correctly initialize our carry bits in the state $|0\rangle$.

## 4 Exercise 4: Deutsch Problem Simulation (4 Points)

### 4.1 The $\mathbf{U_f}$ Functions

**A. Write four circuits which implement the four $U_f$ functions ($U_{f1}$, $U_{f2}$, $U_{f3}$ and $U_{f4}$) in four files named uf1.qc, uf2.qc, uf3.qc and uf4.qc, respectively. [4*0.5 points]**

The definition of the $U_f$ functions is trivial from its definition. The first function f(x) is 0 whenever x is 0 and 1 if x is 1. This is equivalent to perform a CNOT between the qx and qy qubits.

```
# Uf1 circuit
version 0.5
qubits 2

map q[0],qx
map q[1],qy

.init
#x qy
display

.uf
cnot qx,qy
display
```

For the second function, it does the exact opposite as the first one, which we can simply introduce by applying an extra X gate on qy.

```
# Uf2 circuit
version 0.5
qubits 2

map q[0],qx
map q[1],qy

.init
#x qy
display

.uf
cnot qx,qy
x qy
display
```

The third function always yields 0, so we must not perform any change on the second qubit register (or performing the identity operator if we wish).

```
# Uf3 circuit
version 0.5
qubits 2

map q[0],qx
map q[1],qy

.init
#x qy
display

.uf
display
```

Finally, for $U_{f4}$ we get that f(x)=1 $\forall x$. This corresponds to implementing an X gate on qy:

```
# Uf4 circuit
version 0.5
qubits 2

map q[0],qx
map q[1],qy

.init
#x qy
display

.uf
x qy
display
```

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

We simulated all four functions for both possible initializations of qy and the results were the expected.

## 4.2 Deutsch's Algorithm Implementation

**A. Write the four circuits implementing Deutsch's algorithm and use each of the implemented four $U_f$ functions. The circuit files are named deutsch1.qc, deutsch_uf2.qc, deutsch_uf3.qc and deutsch_uf4.qc, respectively.** [4*0.5 points]

The four implementations of the Deutsch's algorithm are equivalent, since only the unitary function changes in each case:

```
#Deutschs Algorithm Uf1 circuit
version 0.5
qubits 2

map q[0],qx
map q[1],qy

.init
x qy
display

.superposition
h qx
h qy
display

.uf
cnot qx,qy
display

.result
h qx
measure qx
display
```

```
# Deutschs Algorithm Uf2 circuit
version 0.5
qubits 2

map q[0],qx
map q[1],qy

.init
x qy
display

.superposition
h qx
h qy
display

.uf
cnot qx,qy
x qy
display

.result
h qx
measure qx
display
```

```
# Deutschs Algorithm Uf3 circuit
version 0.5
qubits 2

map q[0],qx
map q[1],qy

.init
x qy
display

.superposition
h qx
h qy
display

.uf
display

.result
h qx
measure qx
display
```

```
# Deutschs Algorithm Uf4 circuit
version 0.5
qubits 2

map q[0],qx
map q[1],qy

.init
x qy
display

.superposition
h qx
h qy
display

.uf
x qy
display

.result
h qx
measure qx
display
```

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

The results that we obtained after running the Deutsch's Algorithm for $U_{f1}$,$U_{f2}$,$U_{f3}$ and $U_{f4}$ were respectively (omitting the unused qubits in the register):

```
--------------------------------------------
[>>] measurement register (Uf1)             : |          0 |          1 |
--------------------------------------------
--------------------------------------------
[>>] measurement register (Uf2)             : |          0 |          1 |
--------------------------------------------
--------------------------------------------
[>>] measurement register (Uf3)             : |          0 |          0 |
--------------------------------------------
--------------------------------------------
[>>] measurement register (Uf4)             : |          0 |          0 |
--------------------------------------------
```

If we look at the second digit (which represents qx after being measured), we can conclude that $U_{f1}$ and $U_{f2}$ are balanced while $U_{f3}$ and $U_{f4}$ are constant.