Student Information

Name: Kaan Karaçanta

ID: 2448546

Part 1

a)

To show whether the gate U_f is invertible or not, we consider the properties of quantum gates. A gate U_f acts on two qubits as follows:

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

where $\oplus$ denotes addition modulo 2 (XOR operation). For U_f to be invertible, there must exist a U_f^{-1} such that:

$$U_f^{-1}U_f|x\rangle|y\rangle = |x\rangle|y\rangle$$

Since $a \oplus b \oplus b = a$ for binary variables a and b, applying U_f twice will return the original state:

$$U_fU_f|x\rangle|y\rangle = U_f|x\rangle|y \oplus f(x)\rangle = |x\rangle|y \oplus f(x) \oplus f(x)\rangle = |x\rangle|y\rangle$$

Thus, U_f is invertible.

b)

To compute the result of $U_f(H \otimes I)(|0\rangle \otimes |1\rangle)$ for $f(x) = \neg x$, apply each gate in sequence: First, apply the Hadamard gate H to $|0\rangle$ and the identity I to $|1\rangle$:

$$(H \otimes I)(|0\rangle \otimes |1\rangle) = (H|0\rangle) \otimes (I|1\rangle)$$

The Hadamard gate H transforms $|0\rangle$ to a superposition:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

Thus:

$$(H \otimes I)(|0\rangle \otimes |1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |1\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle)$$

Now apply U_f with $f(x) = \neg x$:

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus \neg x\rangle$$

For $|01\rangle$:

$$U_f|01\rangle = |0\rangle|1 \oplus \neg 0\rangle = |0\rangle|1 \oplus 1\rangle = |00\rangle$$

For $|11\rangle$:

$$U_f|11\rangle = |1\rangle|1 \oplus \neg 1\rangle = |1\rangle|1 \oplus 0\rangle = |11\rangle$$

The final state is:

$$U_f\left(\frac{1}{\sqrt{2}}(|01\rangle + |11\rangle)\right) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

This is the result after applying U_f to the superposition created by H and I.

c)

This is a balanced function and here is the implementation of the circuit and measurement outcome:

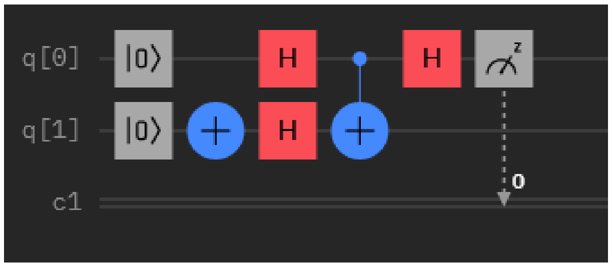

Figure 1: The circuit

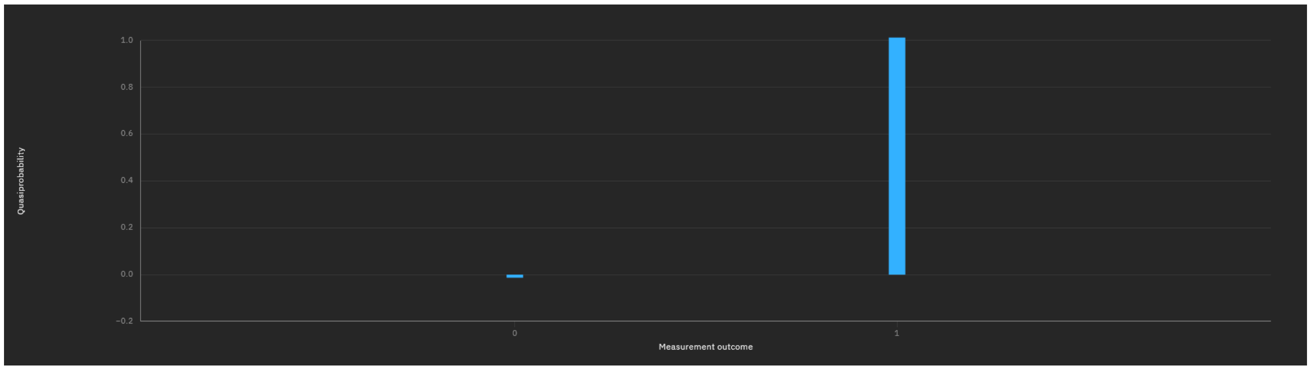

Figure 2: The results of the measurement of the circuit after 1024 shots.

d)

This is a constant function and here is the implementation of the circuit and measurement outcome:

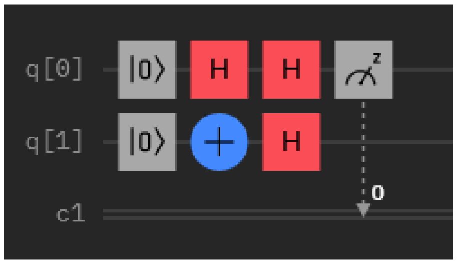

Figure 3: The circuit

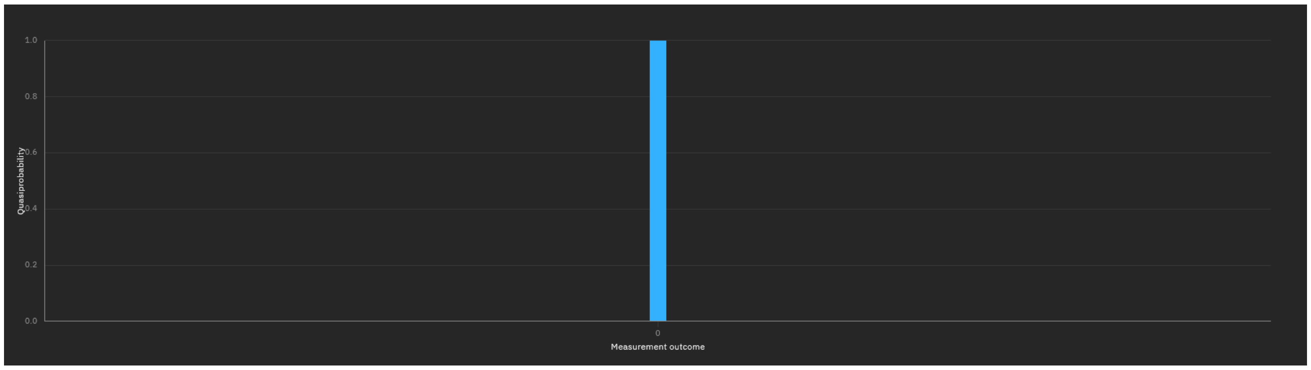

Figure 4: The results of the measurement of the circuit after 1024 shots (on simulator).

Part 2

a)

This is a balanced function and here is the implementation of the circuit and measurement outcome:

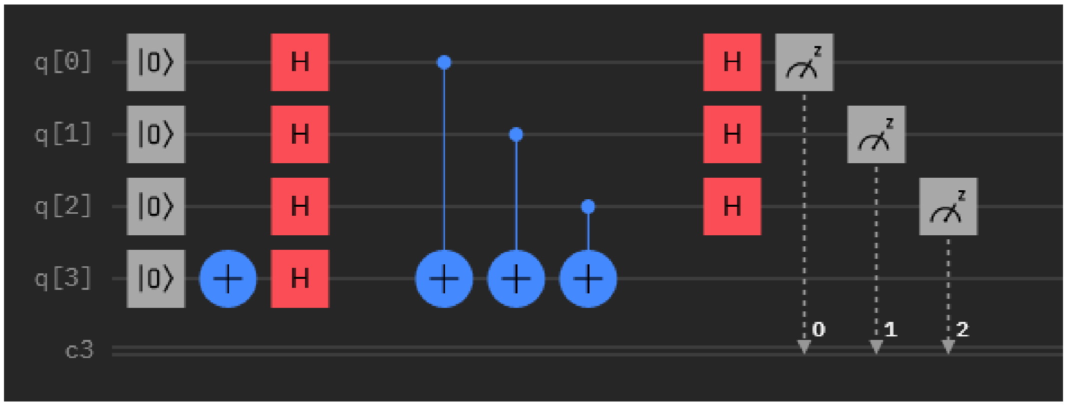

Figure 5: The circuit

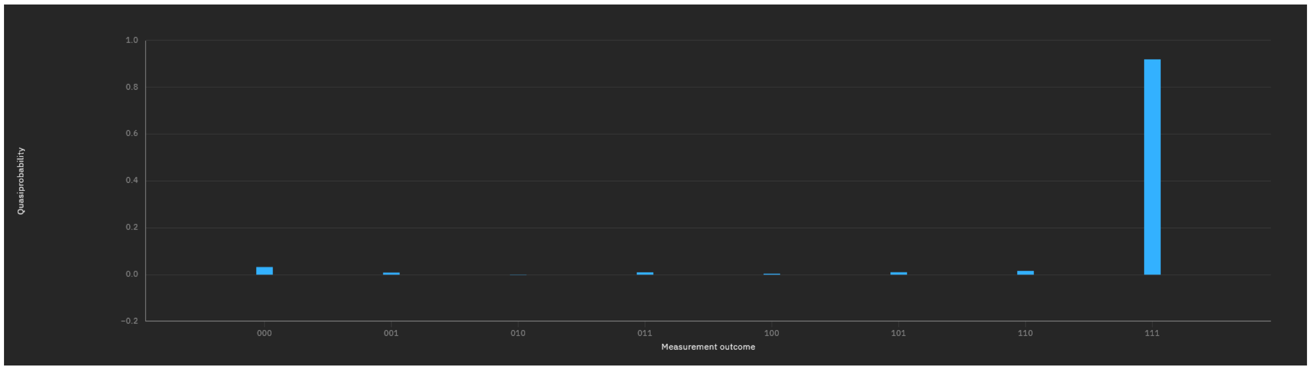

Figure 6: The results of the measurement of the circuit after 1024 shots.

b)

This is a constant function and here is the implementation of the circuit and measurement outcome:

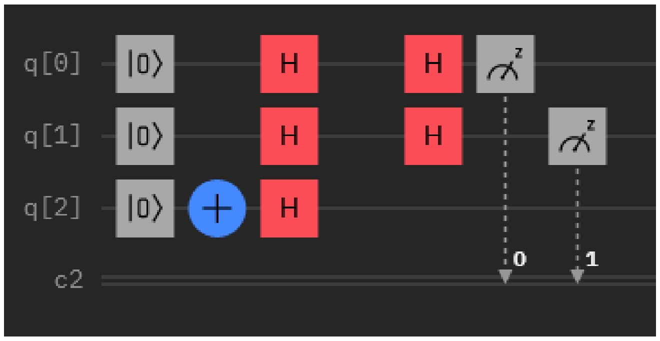

Figure 7: The circuit

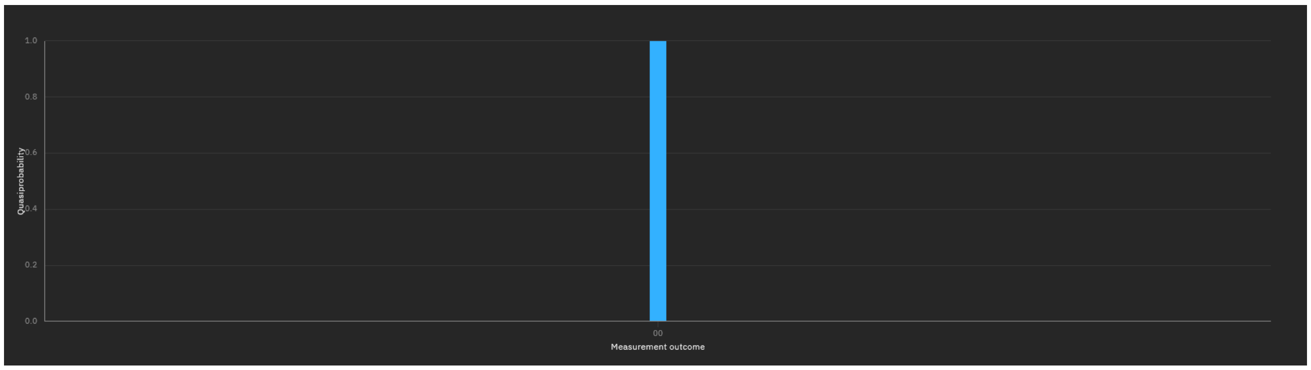

Figure 8: The results of the measurement of the circuit after 1024 shots (on simulator).

c)

This is a balanced function and here is the implementation of the circuit and measurement outcome:

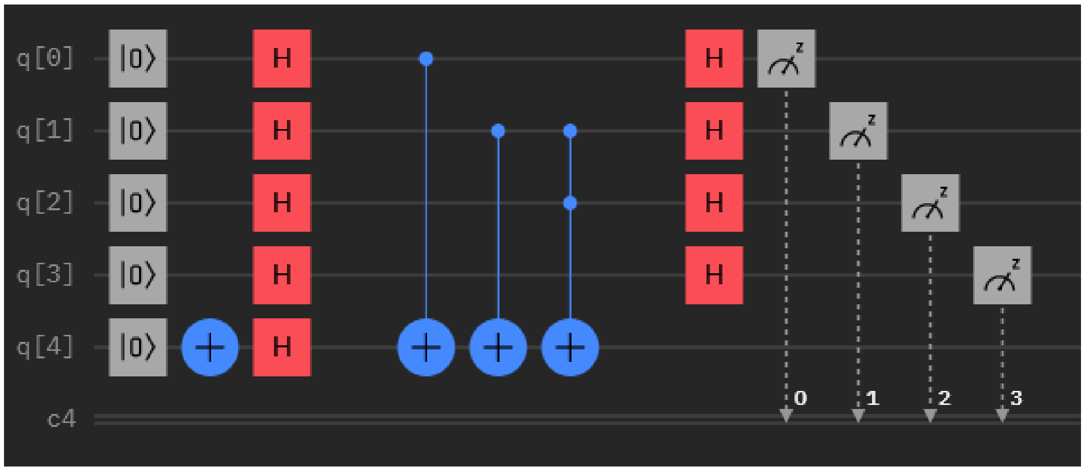

Figure 9: The circuit

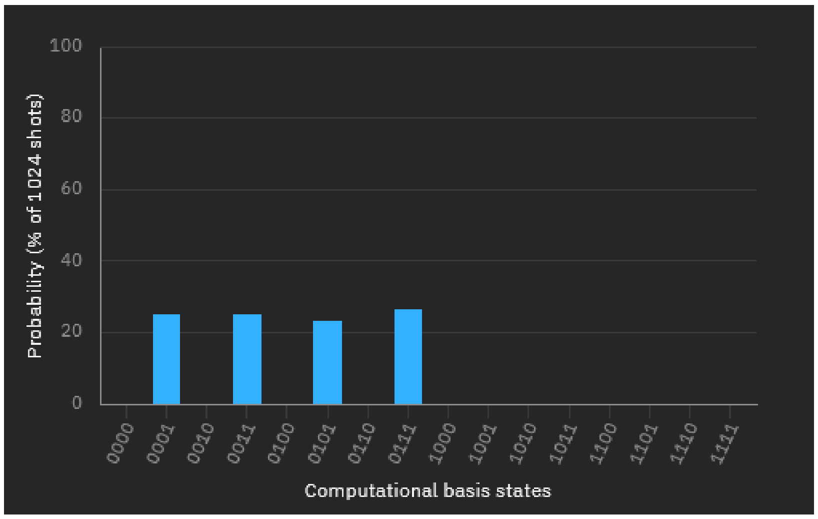

Figure 10: The results of the measurement of the circuit after 1024 shots (on simulator).

Appendix

1-c)

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.reset(qreg_q[0])
circuit.reset(qreg_q[1])
circuit.x(qreg_q[1])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.h(qreg_q[0])
circuit.measure(qreg_q[0], creg_c[0])
# @columns [0,0,1,2,2,3,4,5]
```

1-d)

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.reset(qreg_q[0])
circuit.reset(qreg_q[1])
circuit.h(qreg_q[0])
circuit.x(qreg_q[1])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.measure(qreg_q[0], creg_c[0])
# @columns [0,0,1,1,2,2,3]
```

2-a)

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(4, 'q')
creg_c = ClassicalRegister(3, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.reset(qreg_q[0])
circuit.reset(qreg_q[1])
circuit.reset(qreg_q[2])
circuit.reset(qreg_q[3])
circuit.x(qreg_q[3])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])
circuit.CX(qreg_q[0], qreg_q[3])
circuit.cx(qreg_q[1], qreg_q[3])
circuit.cx(qreg_q[2], qreg_q[3])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[1], creg_c[1])
circuit.measure(qreg_q[2], creg_c[2])
# @columns [0,0,0,0,1,2,2,2,2,4,5,6,8,8,8,9,10,11]
```

2-b)

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(3, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.reset(qreg_q[0])
circuit.reset(qreg_q[1])
circuit.reset(qreg_q[2])
circuit.x(qreg_q[2])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[1], creg_c[1])
# @columns [0,0,0,1,2,2,2,4,4,5,6]
```

2-c)

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(5, 'q')
creg_c = ClassicalRegister(4, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.reset(qreg_q[0])
circuit.reset(qreg_q[1])
circuit.reset(qreg_q[2])
circuit.reset(qreg_q[3])
circuit.reset(qreg_q[4])
circuit.x(qreg_q[4])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])
circuit.h(qreg_q[4])
circuit.cx(qreg_q[0], qreg_q[4])
circuit.cx(qreg_q[1], qreg_q[4])
circuit.ccx(qreg_q[1], qreg_q[2], qreg_q[4])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[1], creg_c[1])
circuit.measure(qreg_q[2], creg_c[2])
circuit.measure(qreg_q[3], creg_c[3])
# @columns [0,0,0,0,0,1,2,2,2,2,2,4,5,6,8,8,8,8,9,10,11,12]
```