circuit-basics

October 24, 2019

In [23]:
```
import numpy as np
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import BasicAer
from qiskit import execute
from qiskit.quantum_info import Pauli, state_fidelity, basis_state, process_fidelity

#Building circuits

#create quantum + classical registers with 2 qubits
q0=QuantumRegister(2,'q0')
q1=QuantumRegister(2,'q1')
c0=ClassicalRegister(2,'c0')
c1=ClassicalRegister(2,'c1')
#create quantum circuit acting on q0 & q1 register
circ=QuantumCircuit(q0,q1)
# CX on q0
circ.x(q0[1])
# add CX (cnot) gate on Control q0 => target q1, putting the qubits in Bell state
circ.x(q1[0])
# => a total circuit state q1xq0=|0110>

# add measure
meas=QuantumCircuit(q0,q1,c0,c1)
meas.measure(q0,c0)
meas.measure(q1,c1)
qc=circ+meas
#vizulalize the circuit
circ.draw()
#qc.draw()

#now, the statevector of the circuit has size = 16; the 6th element of the statevecto
backend_sim=BasicAer.get_backend('statevector_simulator')
result=execute(circ,backend_sim).result()
state=result.get_statevector(circ)
print('statevector=','\n',state)
#print(int('0110',2))

#we can check the state fidelity with basis_state
```

```
state_fidelity(basis_state('0110',4),state)

#we can make the unitary operator representing the circuit (for no measurements). thi
# I*X*X*I (tensor product- produs vectorial); we check that it is correct with Pauli 
backend_sim=BasicAer.get_backend('unitary_simulator')
result=execute(circ,backend_sim).result()
unitary=result.get_unitary(circ)
#print(unitary)
process_fidelity(Pauli(label='IXXI').to_matrix(),unitary)
#qc.draw()

#we can use the qasm_simulator and check the counts - asta merge pe circuitul cu masu
backend_sim=BasicAer.get_backend('qasm_simulator')
result=execute(qc,backend_sim).result()
counts=result.get_counts(qc)
print('counts=',counts)
```

```
statevector=
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j
 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
counts= {'01 10': 1024}
```

In [24]:
```
#now about circuit resources

#prepare new circuit
q = QuantumRegister(6)
circuit = QuantumCircuit(q)
circuit.h(q[0])
circuit.ccx(q[0], q[1], q[2])
circuit.cx(q[1], q[3])
circuit.x(q)
circuit.h(q[2])
circuit.h(q[3])
circuit.draw()
```

Out[24]: <qiskit.tools.visualization._text.TextDrawing at 0x7f69f5578a20>

In [17]:
```
#the total number of operations in the circuit
circuit.size()
```

Out[17]: 11

In [18]:
```
#the depth of circuit = number of ops on the critical path - de lamurit ???
#pare a fi un timp care nu trebuie sa depaseasca timpul de coerenta al masinii <- gas
#pare a fi dpdv teoretic drumul critic intr-un graf
#cel mai bine am gasit ca inseamna: the longest series of sequential operations in a 
circuit.depth()
```

Out[18]: 5

In [20]:
```
#number of qubits;
circuit.width()
```

Out[20]: 6

In [21]:
```
# operations by type
circuit.count_ops()
```

Out[21]: {'x': 6, 'h': 3, 'ccx': 1, 'cx': 1}

In [22]:
```
#number of unentangled subcircuits; each subcircuit can be executed on a different qu
circuit.num_tensor_factors()
```

Out[22]: 3

In [ ]: