QuantumTeleportation_qiskit

December 30, 2020

[85]:
```
%matplotlib inline
# Importing standard Qiskit libraries
import numpy as np
from qiskit import *
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.extensions import Initialize
from qiskit.visualization import *
from iqx import *
from qiskit_textbook.tools import random_state, array_to_latex
#from numpy import random_state
# Loading your IBM Q account(s)
provider = IBMQ.load_account()
```

```
/opt/conda/lib/python3.7/site-packages/qiskit/providers/ibmq/ibmqfactory.py:192:
UserWarning: Timestamps in IBMQ backend properties, jobs, and job results are
all now in local time instead of UTC.
  warnings.warn('Timestamps in IBMQ backend properties, jobs, and job results '
ibmqfactory.load_account:WARNING:2020-12-30 06:51:51,589: Credentials are
already in use. The existing account in the session will be replaced.
```

1 Defining the Circuit

We will first define the circuit in the following block of code

1.1 Make a Random State

In the following bit of code, we will initialize q_0 with a random state and then transport that onto q_2

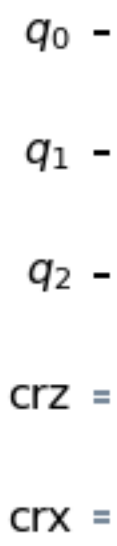
[86]:
```
# Create random 1-qubit state
psi = random_state(1)

# Display it nicely
array_to_latex(psi, pretext="|\\psi\\rangle =")
# Show it on a Bloch sphere
```

```
plot_bloch_multivector(psi)
init_gate = Initialize(psi)
init_gate.label = "init"
```

$$|\psi\rangle = \begin{bmatrix} 0.55533 + 0.20147i \\ 0.6003 - 0.53913i \end{bmatrix}$$

[87]:
```
quantReg = QuantumRegister(3, name = 'q')
crz = ClassicalRegister(1, name = 'crz')
crx = ClassicalRegister(1, name = 'crx')
qCirc = QuantumCircuit(quantReg, crz, crx)
qCirc.draw()
```

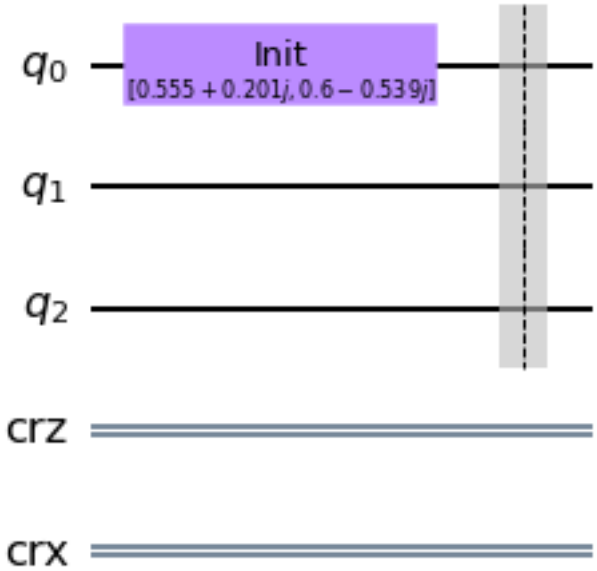
[87]:



Here, we will set the state of $ q_0 $ to 1 and then teleport that bit to $ q_2 $

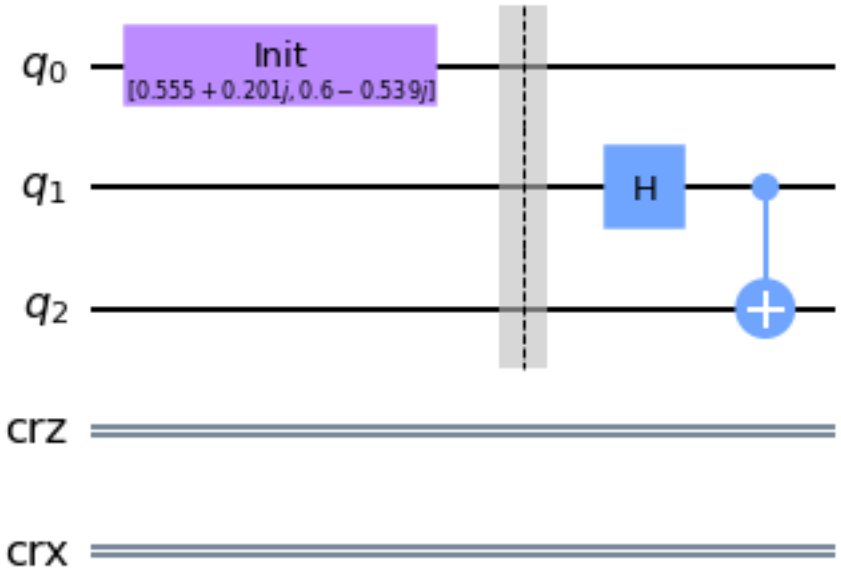
[88]:
```
qCirc.append(init_gate, [0])
qCirc.barrier()
qCirc.draw()
```

[88]:



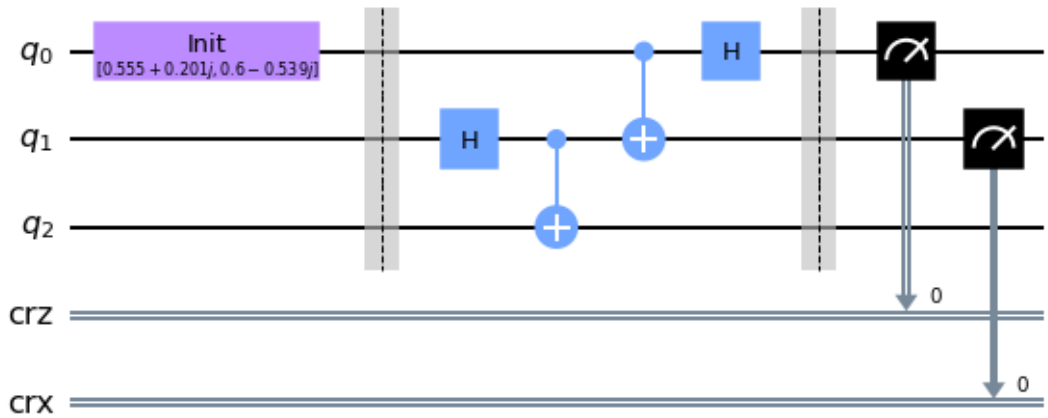
[89]:
```
qCirc.h(1)
qCirc.cx(1,2)
qCirc.draw()
```

[89]:



[90]:
```
qCirc.cx(0,1)
qCirc.h(0)
qCirc.barrier()
qCirc.measure([0,1],[0,1])
qCirc.draw()
```

[90]:



[91]:
```
qCirc.barrier()
```

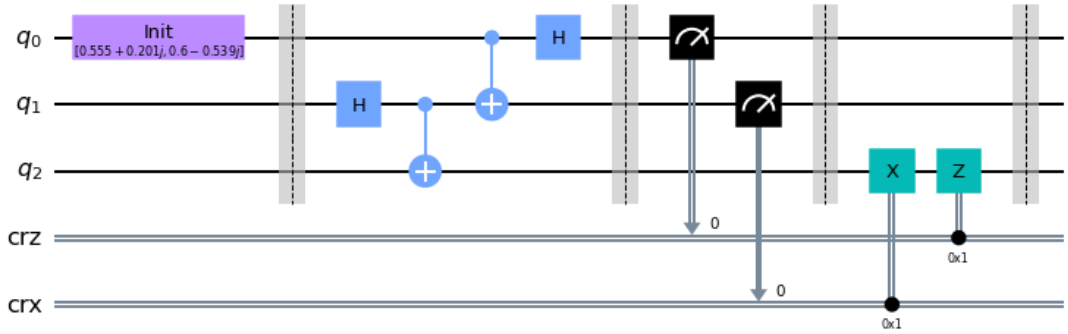
[91]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f176f235050>
```

This function is a funtion on the `intermediate' $ q_1 $ qubit so that we can transport the state to $ q_2 $.

[92]:
```
def measurementAction(qCirc, qubit, crz, crx) :
    qCirc.x(qubit).c_if(crx, 1)
    qCirc.z(qubit).c_if(crz, 1)
```

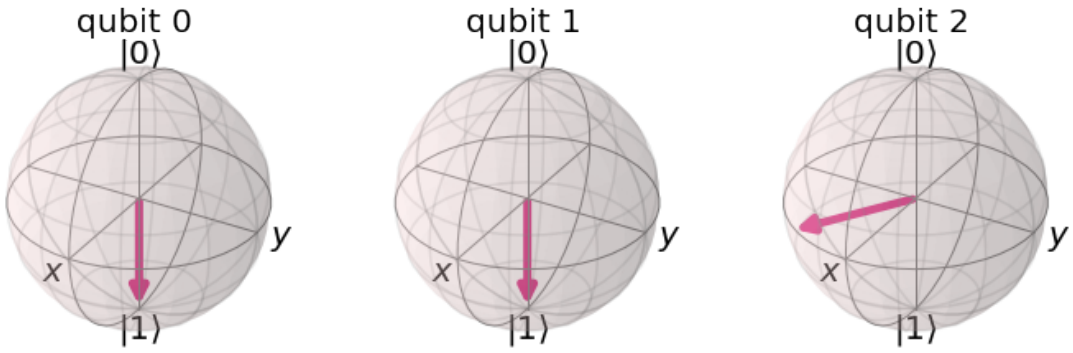
[93]:
```
measurementAction(qCirc, quantReg[2], crz, crx)
qCirc.barrier()
qCirc.draw()
```

[93]:



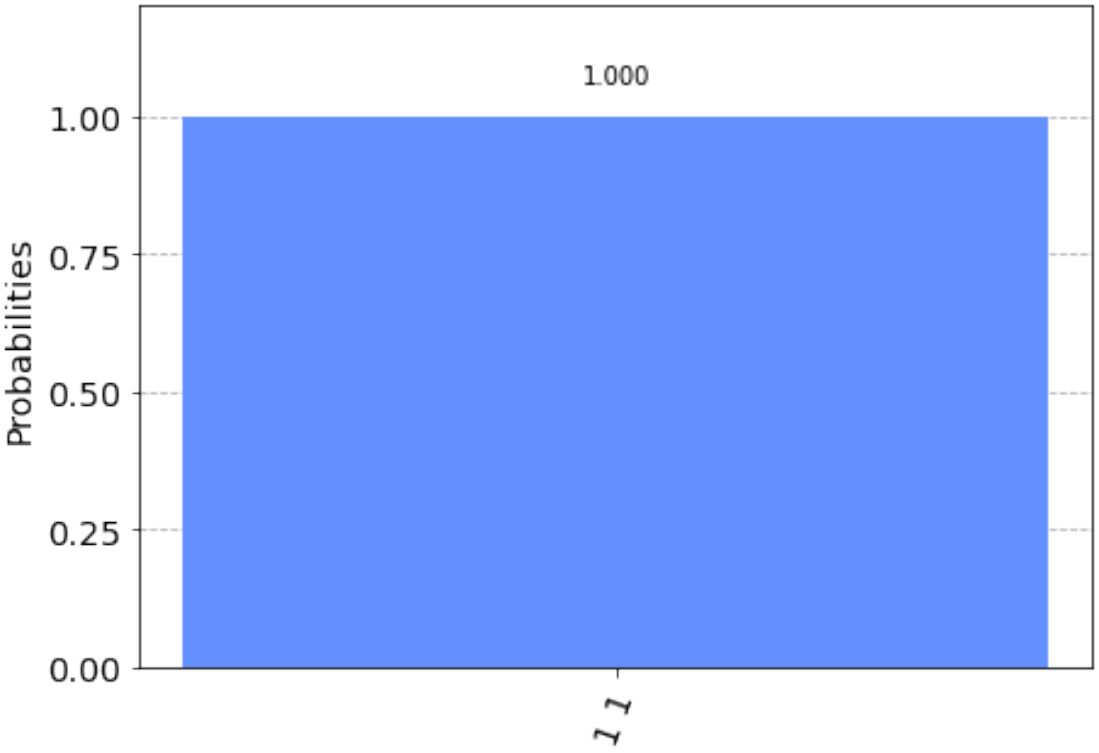
[94]:
```
backend = Aer.get_backend('statevector_simulator')
result = execute(qCirc, backend).result()
counts = result.get_counts()
stateVector = result.get_statevector()
plot_bloch_multivector(stateVector)
```

[94]:



[95]:
```
plot_histogram(counts)
```

[95]:


1.000
1.00
0.75
0.50
0.25
0.00
Probabilities
1 1