Name: Sanjay Kumar Keshava

QFT Simulation and real quantum hardware

In this notebook, you will run both the QFT and its inverse against real and simulated quantum hardware. You will be using some of your answers from HW10.2. The code around QFT, inverse QFT, executing simulations, and jobs has been provided for you. Your job will be to figure out how to manipulate the `QuantumCircuit` object to construct the desired circuits.

Note: Qiskit's least significant bit has the lowest index (0), thus the circuit will be mirrored through the horizontal. See section 5 from the class notebook for more information.

The following is a bunch of helper code, do **not** change!

In [1]:

```
### START DO NOT CHANGE ###
import numpy as np
from numpy import pi
from qiskit import QuantumCircuit, transpile, assemble, Aer, IBMQ
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
from qiskit.visualization import plot_histogram, plot_bloch_multivector

# Common variables
nqubits = 3
shots = 2048

def swap_registers(circuit, n):
    for qubit in range(n//2):
        circuit.swap(qubit, n-qubit-1)
    return circuit

def qft_rotations(circuit, n):
    """Performs qft on the first n qubits in circuit (without swaps)"""
    if n == 0:
        return circuit
    n -= 1
    circuit.h(n)
    for qubit in range(n):
        circuit.cp(pi/2**(n-qubit), qubit, n)
    # At the end of our function, we call the same function again on
    # the next qubits
    # NOTE: we reduced n by one earlier in the function
    qft_rotations(circuit, n)

def qft(circuit, n):
    """QFT on the first n qubits in circuit"""
    qft_rotations(circuit, n)
    swap_registers(circuit, n)
    return circuit.decompose() # .decompose() allows us to see the individual gates

def inverse_qft(circuit, n):
    """Does the inverse QFT on the first n qubits in circuit"""
    # First we create a QFT circuit of the correct size:
    qft_circ = qft(QuantumCircuit(n), n)
    # Then we take the inverse of this circuit
    invqft_circ = qft_circ.inverse()
    # And add it to the first n qubits in our existing circuit
    circuit.append(invqft_circ, circuit.qubits[:n])
    return circuit.decompose() # .decompose() allows us to see the individual gates

def load_IBMQ():
    IBMQ.save_account('57ea5b774ba3ee6147f3fe472cf668743a6f248a4daea41769ebf98c286eff10c9c0ea88430fd2ac4680bee4f52df40b3c373d214390f8e77f2870479f6867d8', overwrite=True)
    IBMQ.load_account()
    provider = IBMQ.get_provider(hub='ibm-q')
    backend = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits >= nqubits
                                           and not x.configuration().simulator
                                           and x.status().operational==True))

    return provider, backend


def simulate(circuit):
    qc = circuit.copy()
    qobj = assemble(qc, shots=shots)
    job = sim.run(qobj)
    job_monitor(job)
    return job.result().get_counts()


def run_job(circuit):
    qc = circuit.copy()
    transpiled_qc = transpile(qc, backend, optimization_level=3)
    job = backend.run(transpiled_qc, shots=shots)
    job_monitor(job)

    return job.result().get_counts()

provider, backend = load_IBMQ()
sim = Aer.get_backend("aer_simulator")

### END DO NOT CHANGE ###
```

1: Apply the QFT to 3-qubit state $|2\rangle$

Your task here is to generate the circuit that constructs the computationl basis state $|2\rangle$.

Determine what $|2\rangle$ is a separable 3-qubit state.

|2> = |010> or (00100000)^T

Fill in the `prepare_state_2` method below by adding the required gate(s) to the `circuit` object.

In [2]:

```
def prepare_state_2(circuit):
# YOUR CODE START
    circuit.x(1)
    return circuit
# YOUR CODE END
```

Once this method has been correctly created, we will construct the circuit for the full state preparation and QFT

In [3]:

```
### BEGIN DO NOT CHANGE ###
qc = QuantumCircuit(3)
qc = prepare_state_2(qc)
qc = qft(qc, 3)
qc.measure_all()
### END DO NOT CHANGE ###
```

Run this circuit against both the Aer simulator and on the actual quantum system

In [4]:

```
# BEGIN your code

simulator = Aer.get_backend('aer_simulator')
qc = transpile(qc, simulator)
result = simulator.run(qc).result()

print('sim')
aerresult = simulate(qc)
print('job')
jobresult = run_job(qc)


# END your code
```

```
sim
```

```
Job Status: job has successfully run
job
Job Status: job has successfully run
```

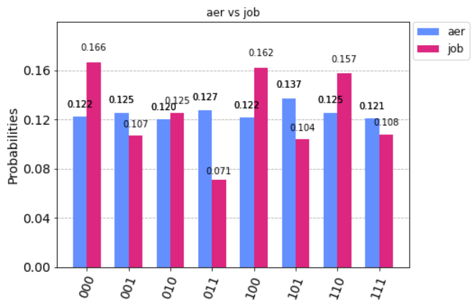
Plot your results

In [5]:
```
# BEGIN your code

plot_histogram([aerresult, jobresult], title='aer vs job', legend=['aer' , 'job'])

# END your code
```

Out[5]:



Discuss how this histogram relates to your answer from 2b

In 2(b) we found QFT of |2> and found that every element of the transformed vector has the same magnitude of 0.125 and this is similar to what we see above. Also we can see the aer simulator is pretty similar to the hardware computation

2: Apply the inverse QFT to state $|\tilde{7}\rangle$

In the class exercise, we covered preparing and running the inverse QFT on the state $|\tilde{5}\rangle$. Your task is to run the inverse QFT on $|\tilde{7}\rangle$.

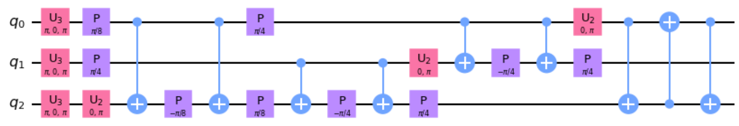
Using your answer from 10.2d, fill out the method `prepare_fstate_7` below.

In [6]:
```
def prepare_fstate_7(circuit):
# YOUR CODE START
    circuit.x(0)
    circuit.x(1)
    circuit.x(2)
    circuit = qft(circuit,3)
# YOUR CODE END
    return circuit
```

If your method is correct, the following code will print out the correct circuit for $|\tilde{7}\rangle$.

In [7]:
```
### BEGIN DO NOT CHANGE ###
qc = QuantumCircuit(3)
qc = prepare_fstate_7(qc)
qc.draw('mpl')
### END DO NOT CHANGE ###
```

Out[7]:



Run this circuit against both the Aer simulator and on the actual quantum system

In [8]:
```
### BEGIN DO NOT CHANGE ###
qc = inverse_qft(qc, nqubits)
qc.measure_all()
### END DO NOT CHANGE ###

print('sim')
aerresult7 = simulate(qc)
print('job')
jobresult7 = run_job(qc)
# END your code
```

```
sim
Job Status: job has successfully run
job
Job Status: job has successfully run
```

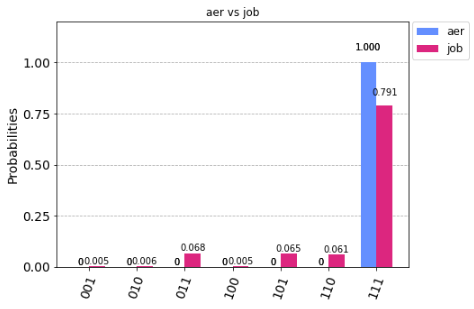
Plot your results

In [9]:
```
# BEGIN your code

plot_histogram([aerresult7, jobresult7], title='aer vs job', legend=['aer' , 'job'])

# END your code
```

Out[9]:



What state has the highest probability and why? Is this 100%? Why or why not?

We are finding the inverse Fourier transform of |~7> which is just |7> so these results make sense. We expect high probability for measuring a 1 on th 8th bit and a low probability for the rest.

References

1. M. Nielsen and I. Chuang, Quantum Computation and Quantum Information, Cambridge Series on Information and the Natural Sciences (Cambridge University Press, Cambridge, 2000).
2. QFT Qiskit tutorital

In [10]:
```
import qiskit.tools.jupyter
%qiskit_version_table
```

Version Information

Qiskit Software	Version
qiskit-terra	0.20.2
qiskit-aer	0.10.4
qiskit-ignis	0.7.1
qiskit-ibmq-provider	0.19.1
qiskit	0.36.2
System information	
Python version	3.9.7
Python compiler	Clang 10.0.0
Python build	default, Sep 16 2021 08:50:36
OS	Darwin
CPUs	8
Memory (Gb)	8.0
	Sat Sep 17 20:35:19 2022 MDT