In [1]: #entanglement of extreme qubits 0 and 3 on the ibmqx2 backend starting from a Bell pair of 0 and 1.
   #we first entangle qbits 0 and 1.
   #then we swap 1 with 2 and swap 2 with 3.
   #many other possibilities can be explored.
   #we use \text{SWAP}(i,j) = \text{CNOT}(i-j) \text{CNOT}(j-i) \text{CNOT}(i-j).
   #when there are directionality constraints use \text{CNOT}(i-j) = H_i H_j \text{CNOT}(j-i) H_i H_j.

from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit import IBMQ
from qiskit.tools.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

# if you dont have an account you must create one, get an API token, and enable
# or save the account.
# once the account is saved you just need to use load below.
# see Qiskit terra for instructions.

#IBMQ.save_account('my API token')

IBMQ.load_accounts()

# currently existing backends
#backend = IBMQ.get_backend('ibmq_16_melbourne')
#backend = IBMQ.get_backend('ibmqx4')
#backend = IBMQ.get_backend('ibmqx2')
#backend = IBMQ.get_backend('ibmq_qasm_simulator')

In [2]: # definition helpers
   # registers with n qbits and n cbits
   def init(n):
      q = QuantumRegister(n)
      c = ClassicalRegister(n)
      qc = QuantumCircuit(q, c)
      return q, c, qc

   # swap operation as \text{swap}(x,y) = \text{CNOT}(x-y)H(x)H(y)\text{CNOT}(x-y)H(x)H(y)\text{CNOT}(x-y)
   # here only \text{CNOT}(x-y) with x controlling y is available
   def swap(circuit, qregister, x: int, y: int):
      circuit.cx(qregister[x], qregister[y])
      circuit.h(qregister[x])
      circuit.h(qregister[y])
      circuit.cx(qregister[x], qregister[y])
      circuit.h(qregister[x])
      circuit.h(qregister[y])
      circuit.cx(qregister[x], qregister[y])
In [3]:
# circuit initialization
q, c, qc = init(5)

#initial entangled pair of qbits 0 and 1
qc.h(q[0])  
qc.cx(q[0], q[1])  
qc.barrier(q)

#swapping operations
swap(qc, q[1], 2)  
swap(qc, q[3], 2)
qc.barrier(q)

#now 0 and 3 are entangled (in theory)

#measurements of 0 and 7 only. easier to interpret histograms
qc.measure(q[0], c[0])  
qc.measure(q[3], c[3])

# drawing circuit
qc.draw()
```python
In [4]:
print("Available backends:")
IBMQ.backends()

Available backends:

Out[4]:
[<IBMQBackend('ibmqx4') from IBMQ()>,
 <IBMQBackend('ibmqx2') from IBMQ()>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ()>,
 <IBMQBackend('ibmq_qasm_simulator') from IBMQ()>]

In [5]:
#simulation
backend = Aer.get_backend('qasm_simulator')
job_sim = execute(qc, backend)
sim_result = job_sim.result()

print(sim_result.get_counts(qc))
plot_histogram(sim_result.get_counts(qc))

{'00000': 512, '01001': 512}

/out\anaconda3/lib/python3.7/site-packages/marshmallow/schema.py:364: ChangedInMarshmallow3Warning: strict=False is not recommended. In marshmallow 3.0, schemas will always be strict. See https://marshmallow.readthedocs.io/en/latest/upgrading.html#schemas-are-always-strict
ChangedInMarshmallow3Warning

Out[5]:

In [6]:
#experiment
#you can also check for availability and current parameters of the backend before calling it.
#see instructions in Quiskit terra.

backend = IBMQ.get_backend('ibmqx2')  #circuit above respects constraints of melbourne device.
shots = 1024  # Number of shots to run the program (experiment); maximum is 8192 shots.
max_credits = 3  # Maximum number of credits to spend on executions.

job_exp = execute(qc, backend=backend, shots=shots, max_credits=max_credits)
job_monitor(job_exp)
```
In [7]:
#results of experiment
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(qc)

print(result_exp.get_counts(qc))
plot_histogram([counts_exp])

{'00000': 478, '00001': 109, '01001': 319, '01000': 118}
#entanglement of 0 and 4 by first entangling 01 and then swapping through path 12, 23, 34.

#circuit initialization
q, c, qc = init(5)

#initial entangled pair of qbits 0 and 1
qc.h(q[0])
qc.cx(q[0], q[1])
qc.barrier(q)

#swapping operations
swap(qc, q, 1, 2)
swap(qc, q, 3, 2)
swap(qc, q, 3, 4)
qc.barrier(q)

#now 0 and 3 are entangled (in theory)

#measurements of 0 and 7 only. easier to interpret histograms
qc.measure(q[0], c[0])
qc.measure(q[4], c[4])

#drawing circuit
qc.draw()
In [4]:
#simulation

backend = Aer.get_backend('qasm_simulator')
job_sim = execute(qc, backend)
sim_result = job_sim.result()

print(sim_result.get_counts(qc))
plot_histogram(sim_result.get_counts(qc))

{'00000': 484, '10001': 540}

Out[4]:

Out[4]:

In [5]:
#experiment

# you can also check for availability and current parameters of the backend before calling it.
# see instructions in Quiskit terra.

backend = IBMQ.get_backend('ibmqx2')  # circuit above respects constriants of melbourne device.
shots = 1024  # Number of shots to run the program (experiment); maximum is 8192 shots.
max_credits = 3  # Maximum number of credits to spend on executions.

job_exp = execute(qc, backend=backend, shots=shots, max_credits=max_credits)
job_monitor(job_exp)

Got a 500 code response to /api/Jobs/5c9237aec3a2f90052b69340/status: 500 Error: Failed to establish a backside connection
In [6]:
# results of experiment

result_exp = job_exp.result()
counts_exp = result_exp.get_counts(qc)

print(result_exp.get_counts(qc))
plot_histogram([counts_exp])

{'10000': 172, '00000': 438, '10001': 285, '00001': 129}