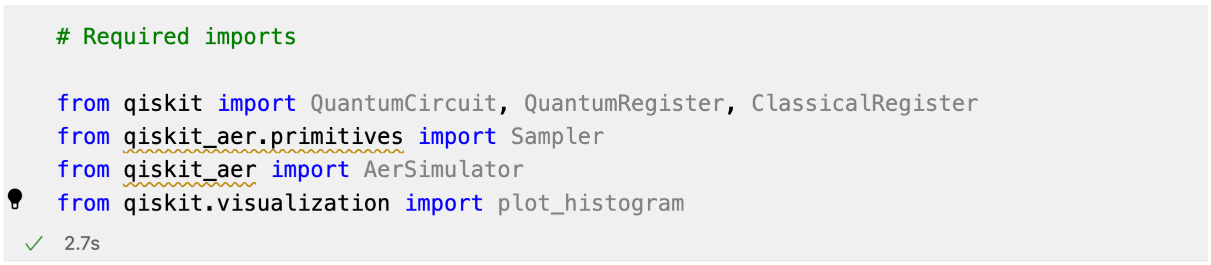
Implementation of superdense coding

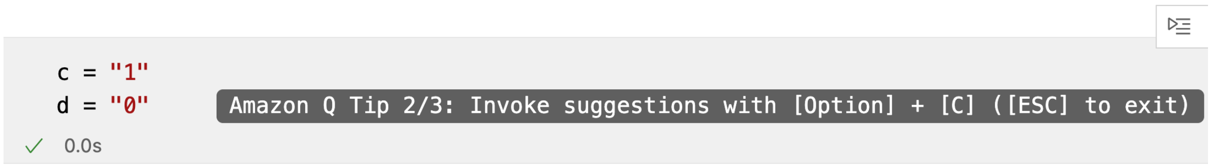
Here is a simple implementation of superdense coding where we specify the circuit itself depending on the bits to be transmitted. First let's specify the bits to be transmitted. (Try changing the bits to see that it works correctly.)



```
# Required imports

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit_aer.primitives import Sampler
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
```

✓ 2.7s



```
c = "1"
d = "0"
```

Amazon Q Tip 2/3: Invoke suggestions with [Option] + [C] ([ESC] to exit)

✓ 0.0s

Now we'll build the circuit accordingly. Here we'll just allow Qiskit to use the default names for the qubits: q0 for the top qubit and q1 for the bottom one.

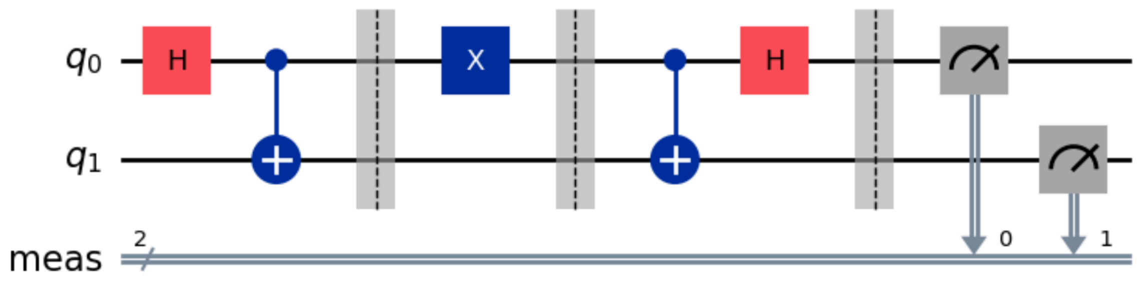
```python
protocol = QuantumCircuit(2)

# Prepare ebit used for superdense coding
protocol.h(0)
protocol.cx(0, 1)
protocol.barrier()

# Alice's operations
if d == "1":
    protocol.z(0)
if c == "1":
    protocol.x(0)
protocol.barrier()

# Bob's actions
protocol.cx(0, 1)
protocol.h(0)
protocol.measure_all()

display(protocol.draw(output='mpl'))
```
✓ 2.4s



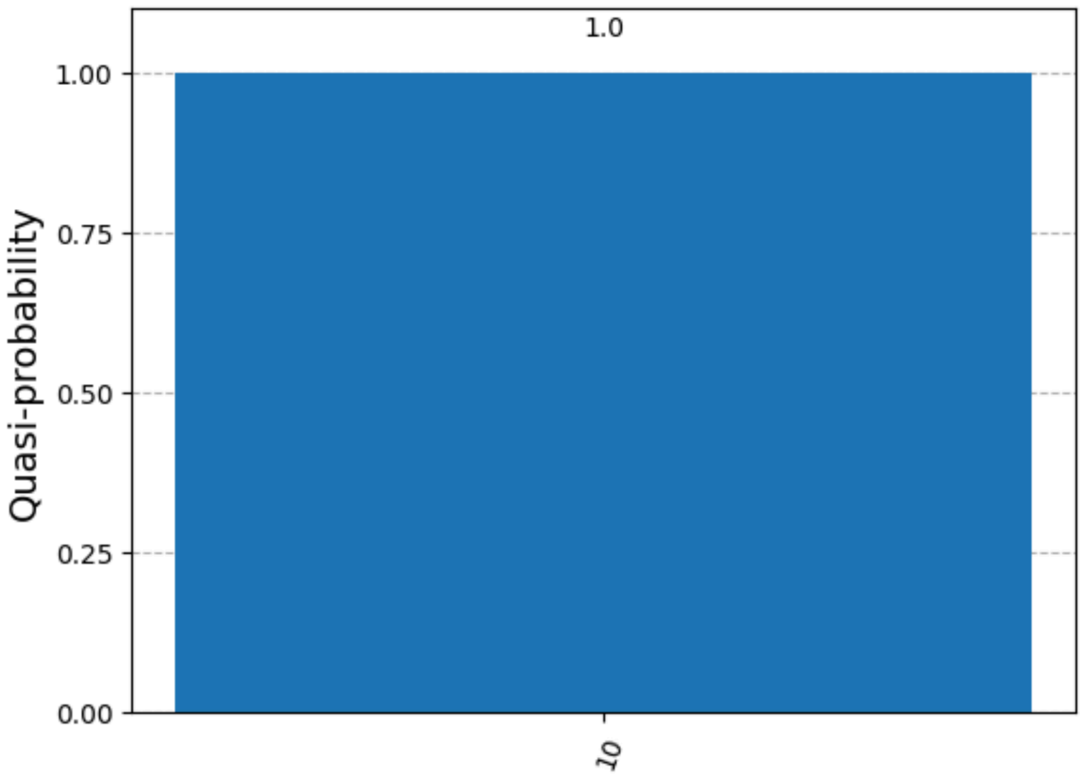
Not much is new here, except the measure_all function, which measures all of the qubits and puts the results into a single classical register (therefore having two bits in this case).
Running the Aer simulator produces the expected output.

```
result = Sampler().run(protocol).result()
statistics = result.quasi_dists[0].binary_probabilities()

for outcome, frequency in statistics.items():
    print(f"Measured {outcome} with frequency {frequency}")

display(plot_histogram(statistics))
```
✓ 6.5s

```
Measured 10 with frequency 1.0
```



Just for fun, we can use an additional qubit as a random bit generator to randomly choose c and d, then run the superdense coding protocol to see that these bits are transmitted correctly.

```python
rbg = QuantumRegister(1, "randomizer")
ebit0 = QuantumRegister(1, "A")
ebit1 = QuantumRegister(1, "B")

Alice_c = ClassicalRegister(1, "Alice c")
Alice_d = ClassicalRegister(1, "Alice d")

test = QuantumCircuit(rbg, ebit0, ebit1, Alice_d, Alice_c)

# Initialize the ebit
test.h(ebit0)
test.cx(ebit0, ebit1)
test.barrier()

# Use the 'randomizer' qubit twice to generate Alice's bits c and d.
test.h(rbg)
test.measure(rbg, Alice_c)
test.h(rbg)
test.measure(rbg, Alice_d)
test.barrier()
```

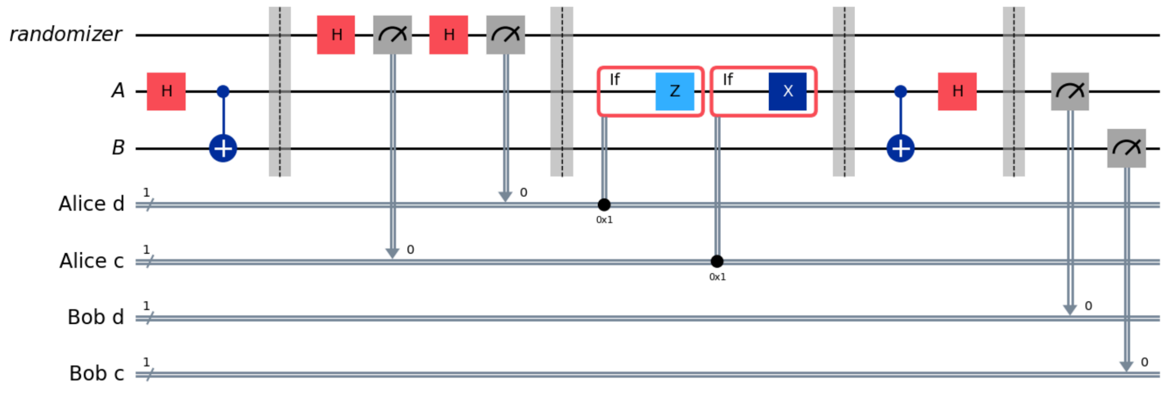
```python
# Now the protocol runs, starting with Alice's actions, which depend
# on her bits.
with test.if_test((Alice_d, 1), label="Z"):
    test.z(ebit0)
with test.if_test((Alice_c, 1), label="X"):
    test.x(ebit0)
test.barrier()

# Bob's actions
test.cx(ebit0, ebit1)
test.h(ebit0)
test.barrier()

Bob_c = ClassicalRegister(1, "Bob c")
Bob_d = ClassicalRegister(1, "Bob d")
test.add_register(Bob_d)
test.add_register(Bob_c)
test.measure(ebit0, Bob_d)
test.measure(ebit1, Bob_c)

display(test.draw(output='mpl'))
```

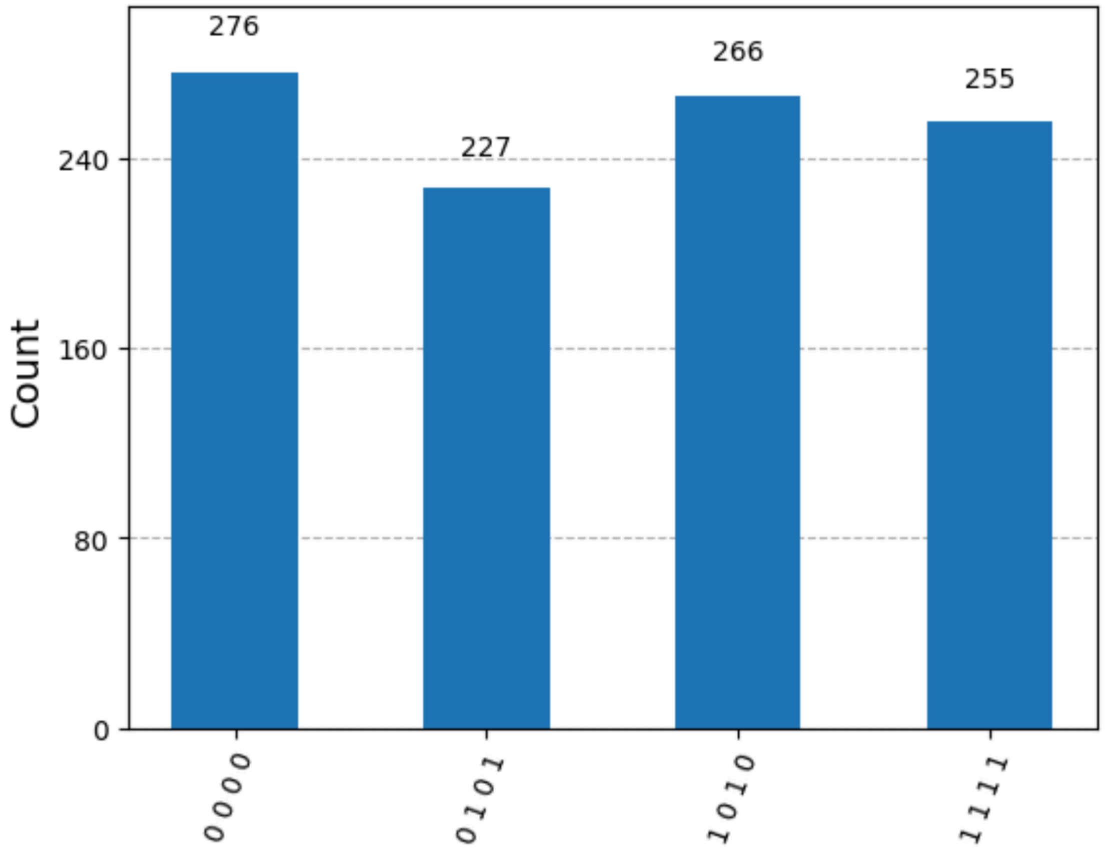
✓ 0.7s



Running the Aer simulator shows the results: Alice and Bob's classical bits always agree.

```
result = AerSimulator().run(test).result()
statistics = result.get_counts()
display(plot_histogram(statistics))
```

✓ 0.4s


Count