Task_2_Victor_Onofre

February 20, 2021

1 Task 2

1.0.1 Victor Onofre

[1]:
```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute,␣
 ↪Aer, IBMQ, BasicAer
from qiskit.visualization import plot_bloch_multivector,plot_bloch_vector,␣
 ↪plot_histogram, plot_state_qsphere
from qiskit.quantum_info import Statevector, partial_trace
import random
import numpy as np
import matplotlib
```

[2]:
```
backend_statevector = BasicAer.get_backend('statevector_simulator')# This␣
 ↪backend executes a single shot of a Qiskit QuantumCircuit and returns the␣
 ↪final quantum statevector of the simulation.
style = {'backgroundcolor': 'lightyellow'} # Style of the circuits
```

[3]:
```
backend_qasm = BasicAer.get_backend('qasm_simulator')# This backend is designed␣
 ↪to mimic an actual device.
                                                            #It executes a Qiskit␣
 ↪QuantumCircuit and returns a count dictionary containing the final values of␣
 ↪any classical registers in the circuit.
shots = 1024 #Number of individual shots
```

2 Part 1: Build the circuit to prepare the Bell state

Build the following simple circuit to prepare the Bell state:

The initial state in the circuit is:

$$|\Psi_0>=|0>_0|0>_1$$

Then the hadamard gate is apply in the qubit $|0>_0$

$$|\Psi_1>=(H\otimes I)|\Psi_0>=H|0>_0 I|0>_1=\left[\frac{1}{\sqrt{2}}\Big(|0>_0+|1>_0\Big)\right]|0>_1=\frac{1}{\sqrt{2}}\Big(|0>_0|0>_1+|1>_0|0>_1\Big)$$

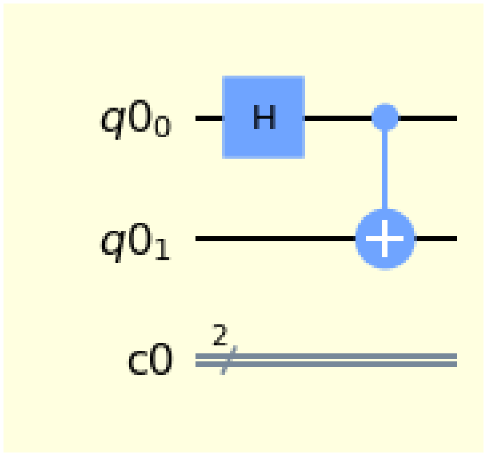
The CNOT gate is apply to the state $|\Psi_1>$

$$|\Psi_2>= CNOT_{01}|\Psi_1>= \frac{1}{\sqrt{2}}\Big(CNOT_{01}|0>_0|0>_1 + CNOT_{01}|1>_0|0>_1\Big) = \frac{1}{\sqrt{2}}\Big(|0>_0|0>_1 + |1>_0|1>_1\Big)$$

The result is a entangled state

[4]:
```
qreg = QuantumRegister(2)
registerBell = ClassicalRegister(2)

qBell = QuantumCircuit(qreg,registerBell )

qBell.h(0)
qBell.cx(0,1)
qBell.draw(output='mpl', style=style)
```

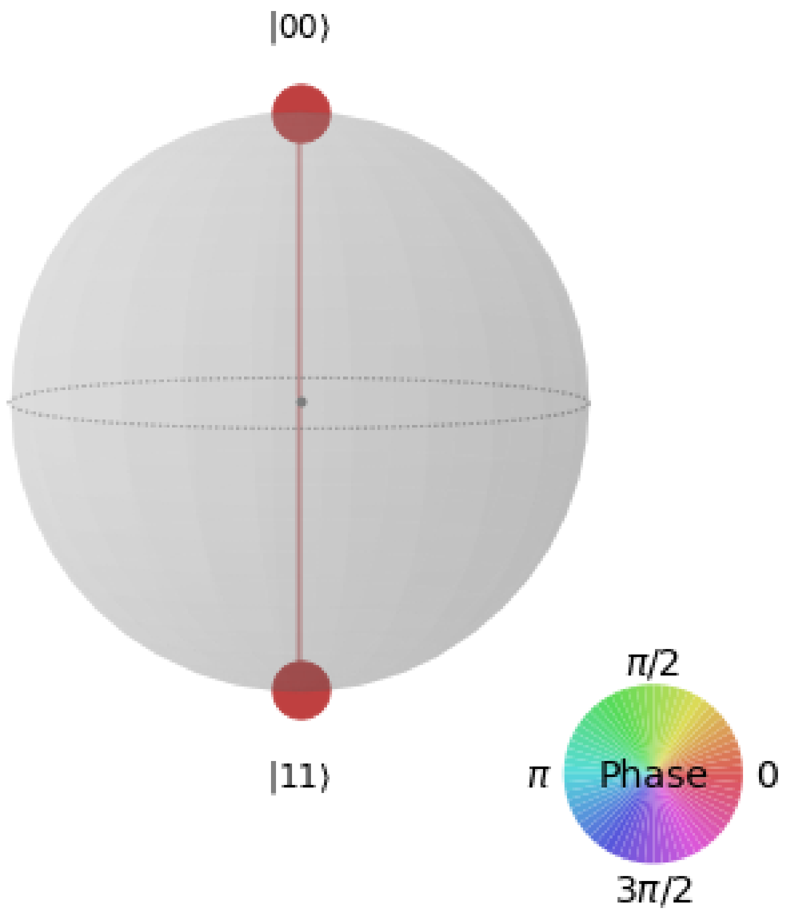
[4]:



```
We can see the result of the circuit in the qsphere
```

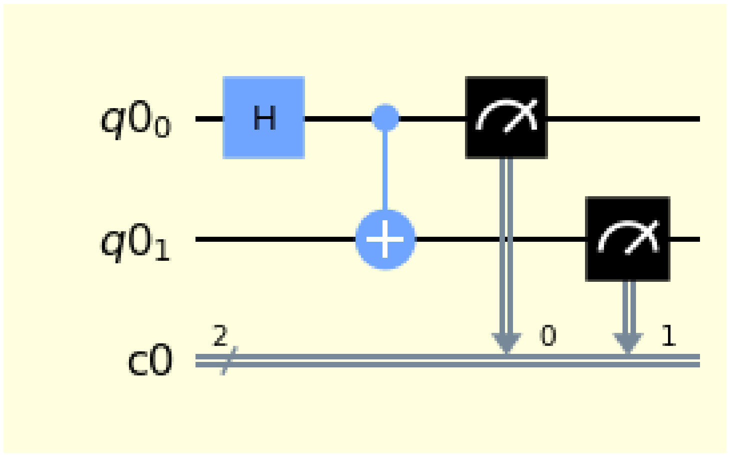
[5]:
```
out_vectorBell = execute(qBell, backend_statevector).result().get_statevector()
plot_state_qsphere(out_vectorBell)
```

[5]:



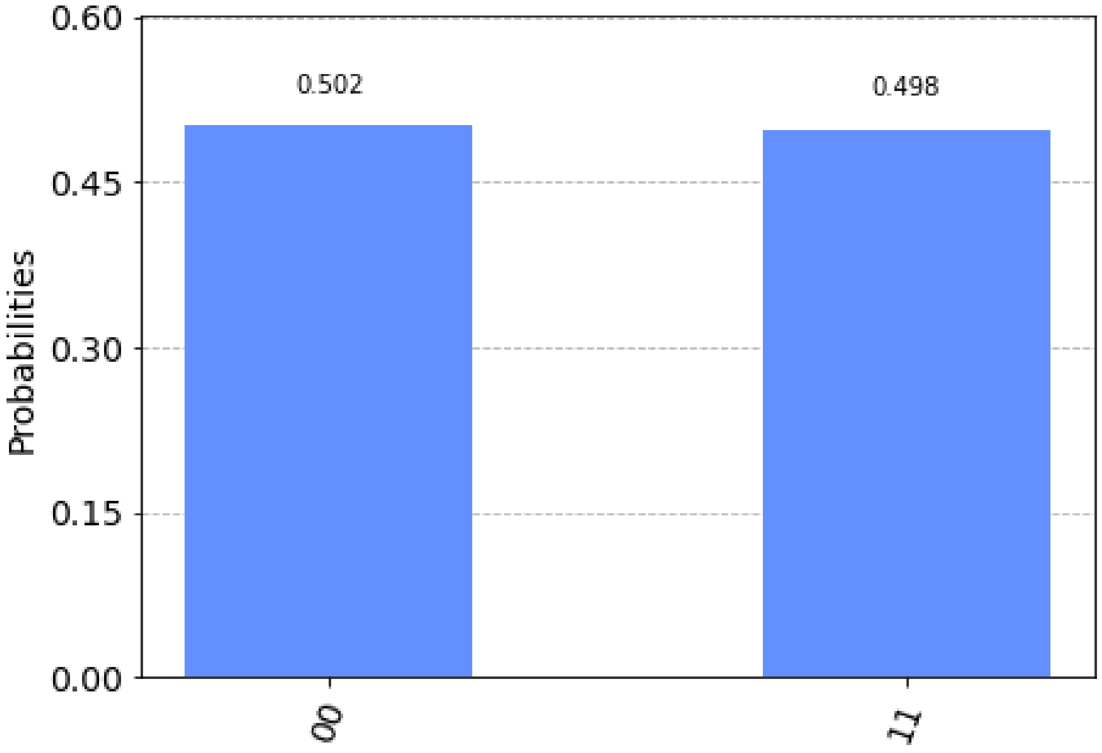
```
[6]: qBell.measure(qreg[0],registerBell [0])
     qBell.measure(qreg[1],registerBell [1])
     qBell.draw(output='mpl', style=style)
```

```
[6]:
```



[7]:
```
results = execute(qBell, backend=backend_qasm, shots=shots).result()
answer = results.get_counts()

plot_histogram(answer)
```

[7]:



[ ]:
```
```

```
[ ]:
```

3 Part 2: Add an arbitrary "error gate" to the circuit

Now add, right before the CNOT gate and for each of the two qubits, an arbitrary ``error gate''. By error gate we mean that with a certain probability (that you can decide but must be non-zero for all the choices) you have a 1 qubit unitary which can be either the identity, or the X gate (bit-flip error) or the Z gate (sign-flip error).

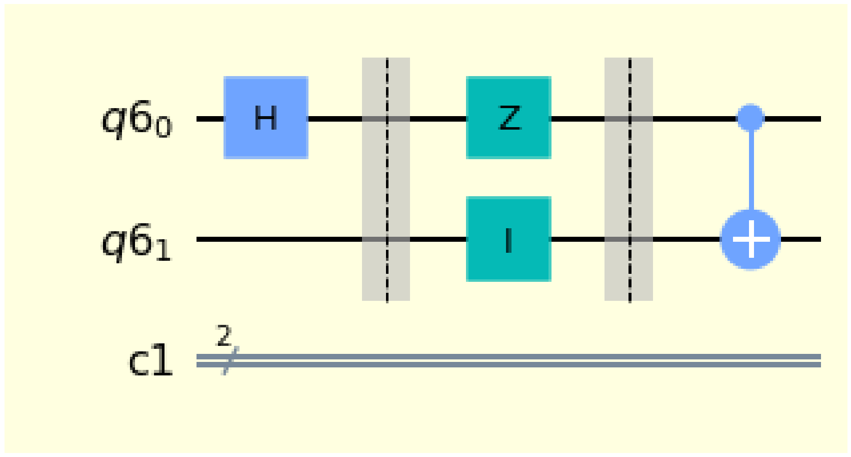
```
[8]: def gate_errorX():
         ############################################################
         # This function will return ['X'] with a probability 0.2   #
         # ['Z'] with a probability 0.2                             #
         # and ['I'] with a probability 0.6                         #
         ##########################################################
         random_gate = random.choices(population= ['Z', 'X', 'I'],weights=[0.2,0.2,␣
      ↪0.6],k=1)

         return(random_gate)
```

```
[ ]:
```

```
[9]: def noise(qcirc):

         if gate_errorX() == ['X']:
         ### Bit-flip in the qubit 0 #####
             qcirc.x(0)
         elif gate_errorX() == ['Z']:
         ### phase-flip in the qubit 0 #####
             qcirc.z(0)
         else:
             qcirc.i(0)

         if gate_errorX() == ['X']:
         ### Bit-flip in the qubit 1 #####
             qcirc.x(1)
         elif gate_errorX() == ['Z']:
         ### phase-flip in the qubit 0 #####
             qcirc.z(1)
         else:
             qcirc.i(1)

         return qcirc
```

[10]:
```
qregNoise = QuantumRegister(2)
registerBellNoise = ClassicalRegister(2)
qBellNoise = QuantumCircuit(qregNoise,registerBellNoise )

qBellNoise.h(0)
qBellNoise.barrier()
noise(qBellNoise)
#qBellNoise.z(0)
qBellNoise.barrier()
qBellNoise.cx(0,1)

qBellNoise.draw(output='mpl', style=style)
```

[10]:



We can see the effect of noise in the state, in this case a z gate is apply to the qubit 0:

$$|\Psi_1>=\frac{1}{\sqrt{2}}\Big(|0>_0|0>_1+|1>_0|0>_1\Big)$$

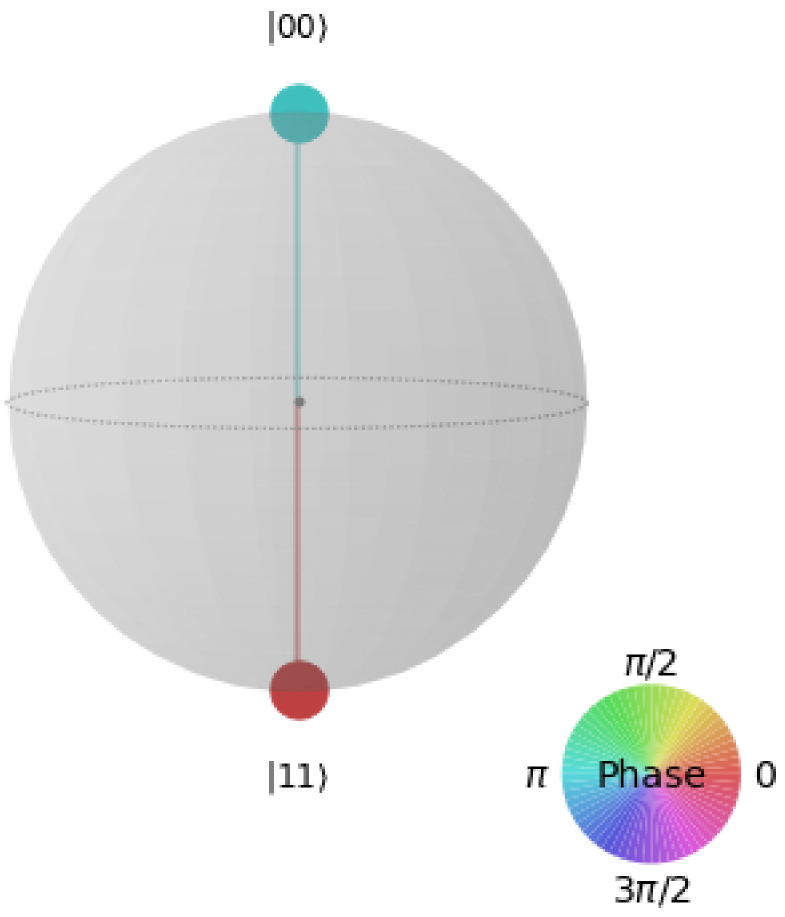
$$|\Psi_1>_{Noise}=(Z\otimes I)|\Psi_1>=\frac{1}{\sqrt{2}}\Big(Z|0>_0 I|0>_1+Z|1>_0 I|0>_1\Big)=\frac{1}{\sqrt{2}}\Big(|0>_0|0>_1-|1>_0|0>_1\Big)$$

The sign (phase) has been change in the state!

Again, we can see the result of the circuit in the qsphere

[11]:
```
out_vectorBellNoise = execute(qBellNoise, backend_statevector).result().
 ↪get_statevector()
plot_state_qsphere(out_vectorBellNoise)
```

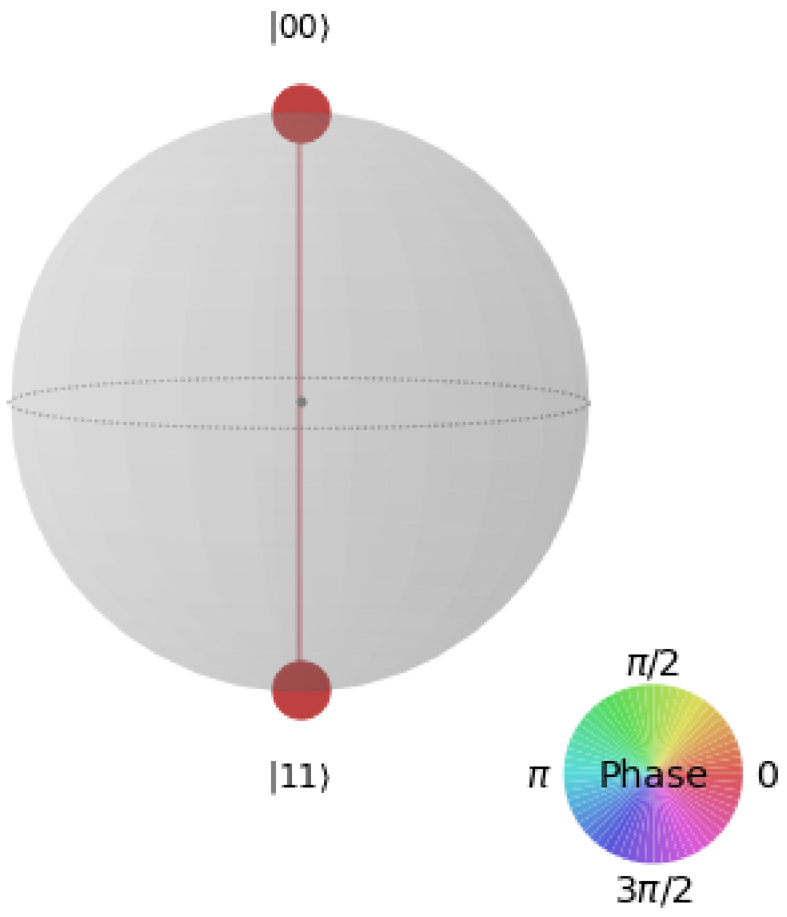
[11]:



Comparing with the state without noise

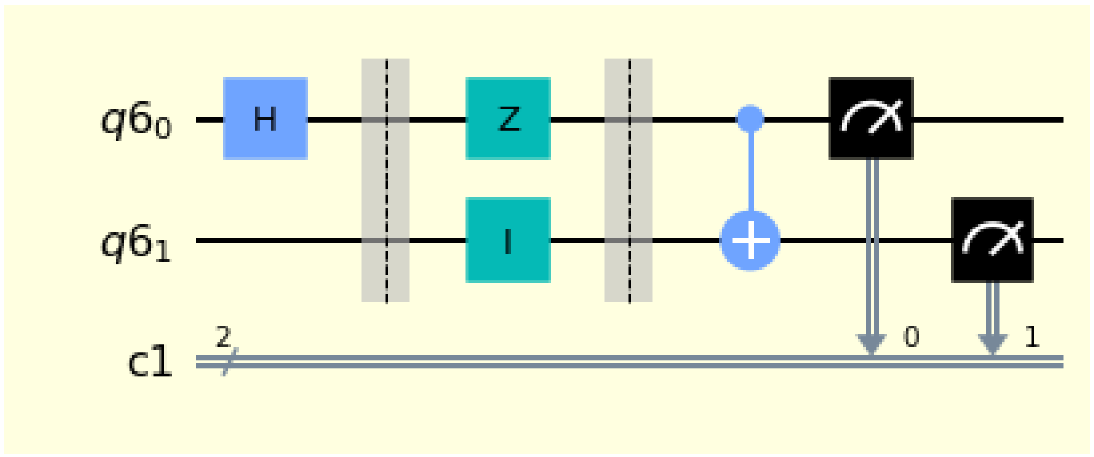
```
[12]: plot_state_qsphere(out_vectorBell)
```

```
[12]:
```



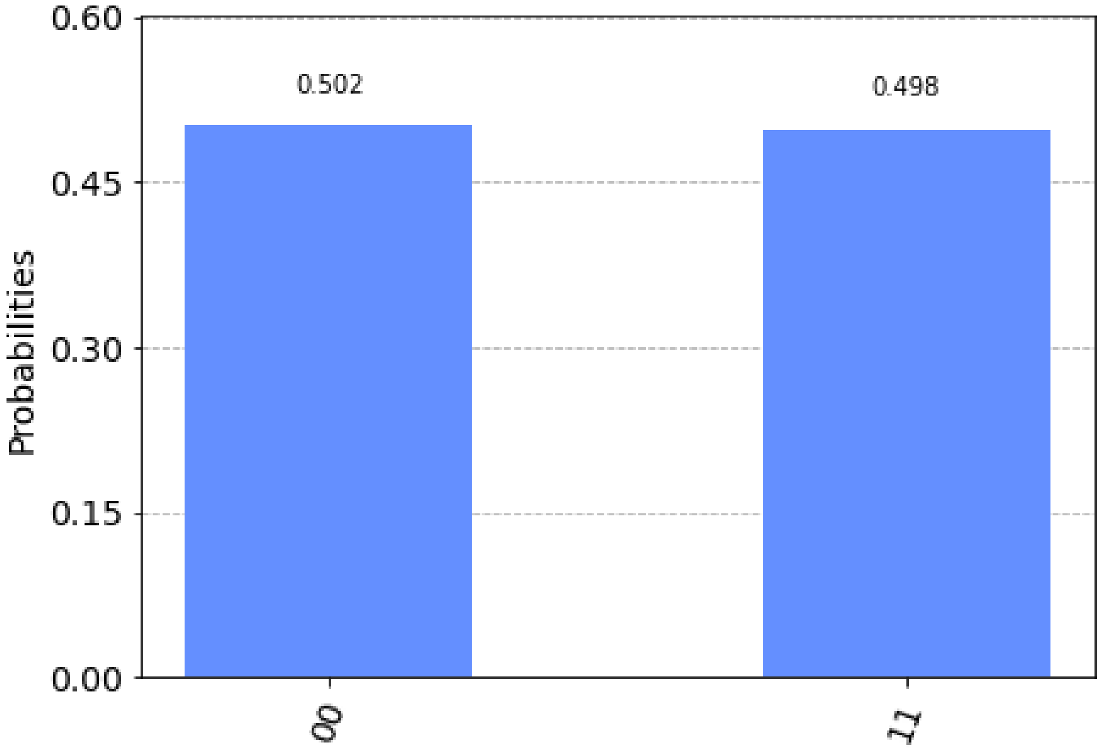
```
[13]: qBellNoise.measure(qregNoise[0],registerBellNoise[0])
      qBellNoise.measure(qregNoise[1],registerBellNoise[1])
      qBellNoise.draw(output='mpl', style=style)
```

```
[13]:
```



[14]:
```
resultsNoise = execute(qBellNoise, backend=backend_qasm, shots=shots).result()
answerNoise = resultsNoise.get_counts()

plot_histogram(answerNoise)
```

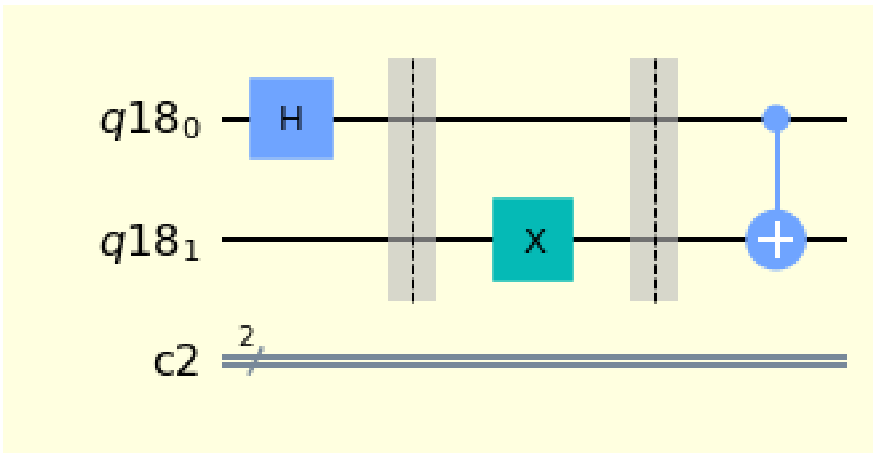
[14]:



3.1 Noise in the qubit 1

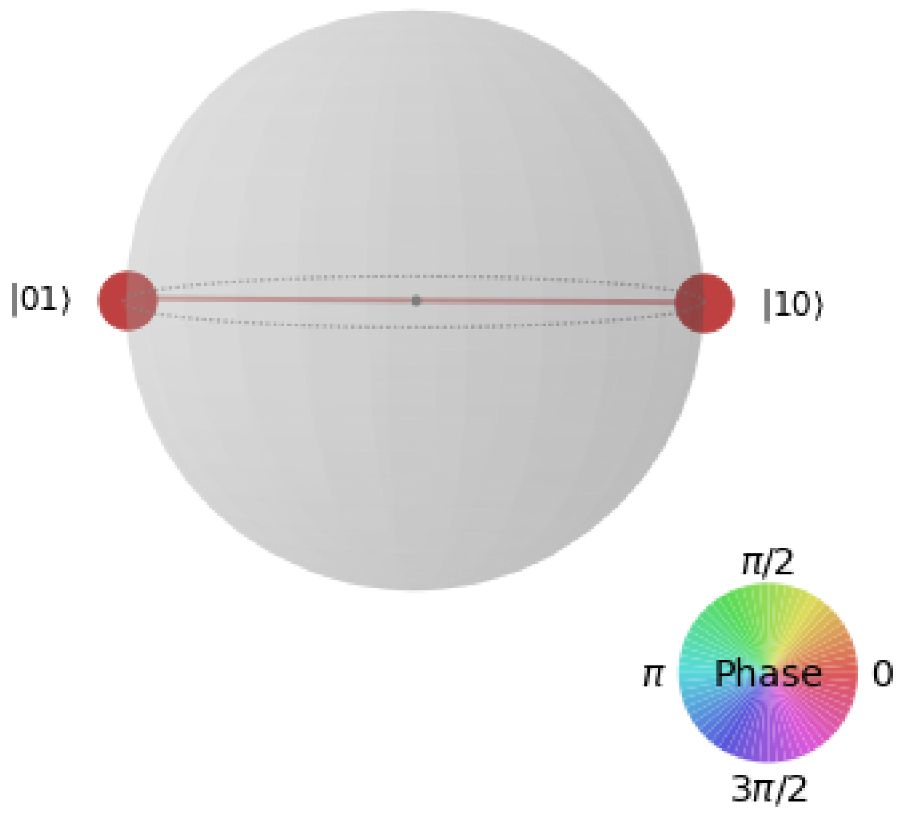
[15]:
```
qreg_X1_Noise = QuantumRegister(2)
registerBell_X1_Noise = ClassicalRegister(2)
qBell_X1_Noise = QuantumCircuit(qreg_X1_Noise,registerBell_X1_Noise )

qBell_X1_Noise.h(0)
qBell_X1_Noise.barrier()
#noise(qBellNoise)
qBell_X1_Noise.x(1)
qBell_X1_Noise.barrier()
qBell_X1_Noise.cx(0,1)

qBell_X1_Noise.draw(output='mpl', style=style)
```

[15]:



[16]:
```
out_vectorBell_X1_Noise = execute(qBell_X1_Noise, backend_statevector).result().
 ↪get_statevector()
plot_state_qsphere(out_vectorBell_X1_Noise)
```

[16]:



```
[ ]:
```

4 Part 3: Encode each of the two qubits with a sign-flip or a bit-flip code

Encode each of the two qubits with a sign-flip or a bit-flip code, in such a way that all the possible choices for the error gates described in 2), occurring on the logical qubits, can be detected and fixed. Motivate your choice. This is the most non-trivial part of the problem, so do it with a lot of care!

5 Part 3.1: The bit-flip code with the Toffoli gate

[17]:
```
Qreg1 = QuantumRegister(1)
Qreg2 = QuantumRegister(2)

qc_bit_flip = QuantumCircuit(Qreg1,Qreg2)
```

[18]:
```
state_to_protect = [np.sqrt(0.2),np.sqrt(0.8)]
```

Define the state that needs to be protected against bit-flips

$$|\psi >= \sqrt{0.2}|0 > +\sqrt{0.8}|1 >$$

[19]:
```
qc_bit_flip.initialize(state_to_protect,0) # Apply initialisation operation to␣
 ↪the 0th qubit
qc_bit_flip.barrier()
```

[19]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e69399490>
```

5.0.1 Showing the state $|\psi >$ ("qubit 0") in the Bloch sphere.

``qubit 1'' and ``qubit 2'' will be used to encode the state $|\psi >$.

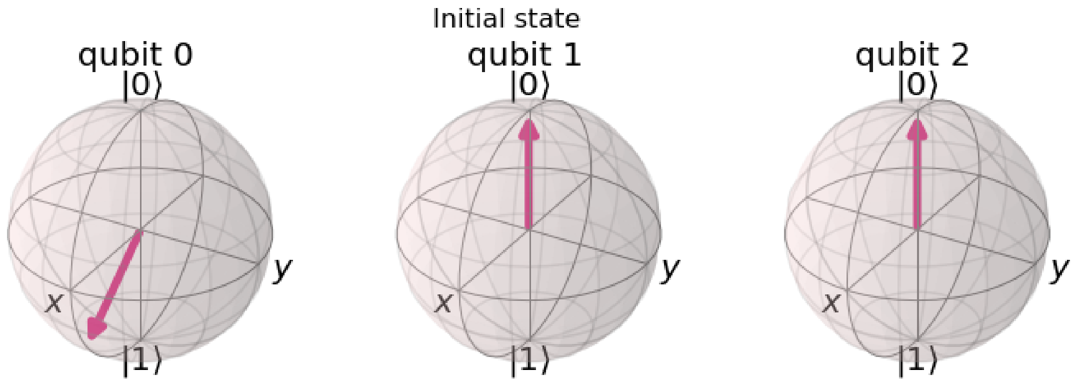
The initial state, $|\Psi_1 >$, in the circuit is:

$$|\Psi_1 >= |\psi > |0 >_1 |0 >_2 = \Big(\sqrt{0.2}|0 > +\sqrt{0.8}|1 > \Big)|0 >_1 |0 >_2$$

$$|\Psi_1 >= \sqrt{0.2}|0 >_0 |0 >_1 |0 >_2 + \sqrt{0.8}|1 >_0 |0 >_1 |0 >_2$$

[20]:
```
Initial_state = Statevector.from_instruction(qc_bit_flip)
plot_bloch_multivector(Initial_state, title='Initial state')
```

[20]:



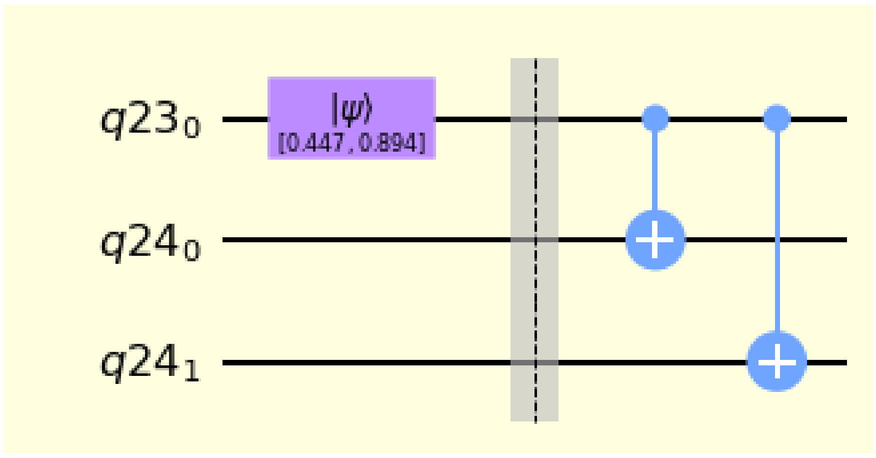
5.0.2 Performing the encoding

In this case we want to perform this encoding:

$$\alpha|0> +\beta|1> \rightarrow \alpha|000> +\beta|111>$$

[21]:
```python
qc_bit_flip.cx(0,1)
qc_bit_flip.cx(0,2)
qc_bit_flip.draw(output='mpl', style=style)
```

[21]:



$$|\Psi_2 >= CNOT_{02}CNOT_{01}|\Psi_1 >= \sqrt{0.2}CNOT_{02}CNOT_{01}|0>_0 |0>_1 |0>_2 +\sqrt{0.8}CNOT_{02}CNOT_{01}|1>_0 |0>_1 |0$$

$$|\Psi_2 >= \sqrt{0.2}|0>_0 |0>_1 |0>_2 +\sqrt{0.8}|1>_0 |1>_1 |1>_2$$

5.0.3 Apply some noise

In this case the x gate in the first qubit

[22]:
```python
qc_bit_flip.barrier()
qc_bit_flip.x(0)
```

[22]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e68b58370>
```

$$|\Psi_3 >= (X \otimes I \otimes I)|\Psi_2 >= \sqrt{0.2}X|0>_0 I|0>_1 I|0>_2 +\sqrt{0.8}X|1>_0 I|1>_1 I|1>_2$$

$$|\Psi_3 >= \sqrt{0.2}|1>_0 |0>_1 |0>_2 +\sqrt{0.8}|0>_0 |1>_1 |1>_2$$

Aplying the inverse of the encoding procedure we recover the state $|\psi>$

[23]:
```
qc_bit_flip.barrier()
qc_bit_flip.cx(0,2)
qc_bit_flip.cx(0,1)
```

[23]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e694565b0>
```

$$|\Psi_4 >= CNOT_{01}CNOT_{02}|\Psi_3 >= \sqrt{0.2}CNOT_{01}CNOT_{02}|1 >_0 |0 >_1 |0 >_2 + \sqrt{0.8}CNOT_{01}CNOT_{02}|0 >_0 |1 >_1 |1$$

$$|\Psi_4 >= \sqrt{0.2}|1 >_0 |1 >_1 |1 >_2 + \sqrt{0.8}|0 >_0 |1 >_1 |1 >_2$$

[24]:
```
qc_bit_flip.barrier()
qc_bit_flip.ccx(2,1,0)
```

[24]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5ea8725340>
```

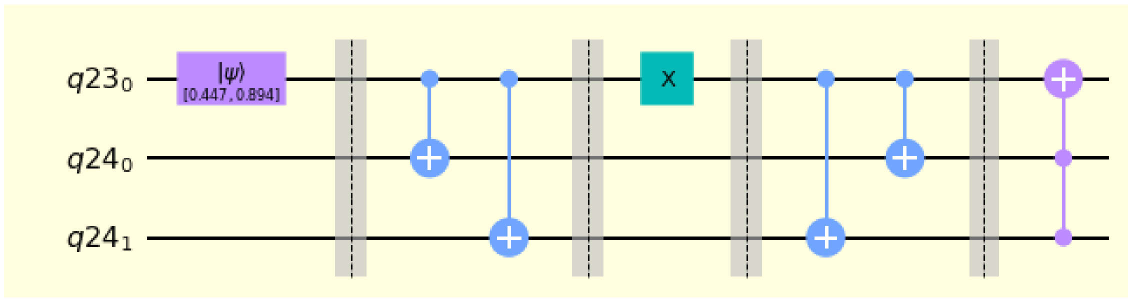
We apply the Toffoli gate.It has 3-bit inputs and outputs if the first two bits are both set to 1, it inverts the third bit, otherwise all bits stay the same

$$|\Psi_5 >= CCNOT_{012}|\Psi_4 >= \sqrt{0.2}CCNOT_{012}|1 >_0 |1 >_1 |1 >_2 + \sqrt{0.8}CCNOT_{012}|0 >_0 |1 >_1 |1 >_2$$

$$|\Psi_5 >= \sqrt{0.2}|0 >_0 |1 >_1 |1 >_2 + \sqrt{0.8}|1 >_0 |1 >_1 |1 >_2$$

[25]:
```
qc_bit_flip.draw(output='mpl', style=style)
```

[25]:



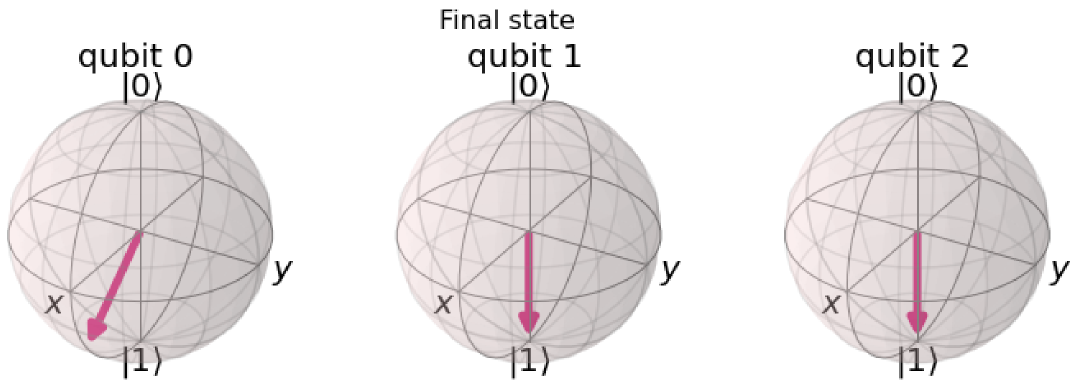
5.0.4 Showing the final state in the Bloch sphere.

We see that ``qubit 1'' and ``qubit 2'' has been modified but the $|\psi\quad>$ (``qubit 0'') has been protected from the noise (X gate).

[26]:
```
```

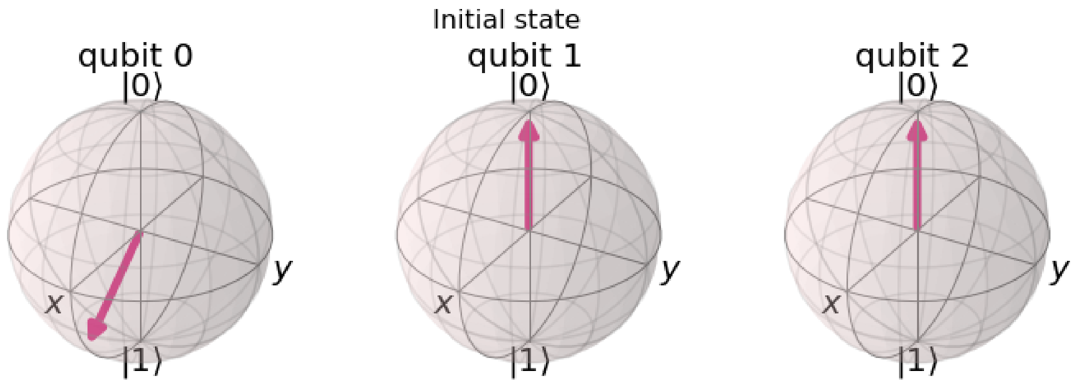
```
out_vector = execute(qc_bit_flip, backend_statevector).result().
 ↪get_statevector()
plot_bloch_multivector(out_vector, title='Final state')
```

[26]:



```
[27]: plot_bloch_multivector(Initial_state, title='Initial state')
```

[27]:



```
[ ]:
```

6 Part 3.2: The bit-flip code with measurements

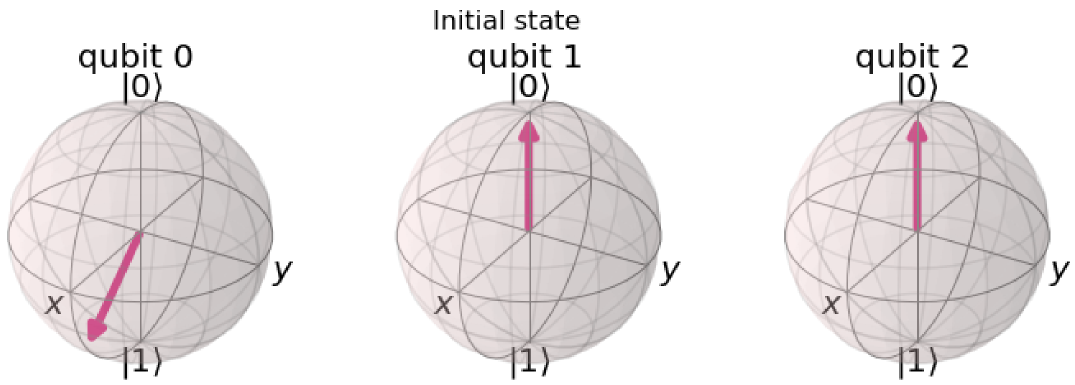
```
[28]: qregister1 = QuantumRegister(1)
      qregister2 = QuantumRegister(2)
      cregister = ClassicalRegister(2, 'c')

      qc_bit_flip_measurements = QuantumCircuit(qregister1,qregister2,cregister)
```

[29]:
```
qc_bit_flip_measurements.initialize(state_to_protect,0) # Apply initialisation␣
↪operation to the 0th qubit
```

[29]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e5a065d90>
```

[30]:
```
Initial_state1 = Statevector.from_instruction(qc_bit_flip_measurements)
plot_bloch_multivector(Initial_state1, title='Initial state')
```

[30]:



[31]:
```
### Perfmoing the encoding #####
qc_bit_flip_measurements.barrier()
qc_bit_flip_measurements.cx(0,1)
qc_bit_flip_measurements.cx(0,2)
```

[31]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e5a0d4490>
```

[32]:
```
### Apply some noise #####
qc_bit_flip_measurements.barrier()
#noise(qc)
qc_bit_flip_measurements.x(0)
```

[32]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e5a065220>
```

[33]:
```
### Aplying the inverse of the encoding procedure ###
qc_bit_flip_measurements.barrier()
qc_bit_flip_measurements.cx(0,2)
qc_bit_flip_measurements.cx(0,1)
```

[33]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e59e526d0>
```

Only in the case where the error occurred on the first qubit do we need to do anything: and this is the case where the measurement outcomes are both $|1\quad>$ and so we use the Toffoli to correct this error.This suggest that a different way to implement this error correcting circuit and that is to measure the second and third qubits and only apply the X gate if the result is 11

[34]:
```
qc_bit_flip_measurements.barrier()
qc_bit_flip_measurements.measure(qregister2,cregister)

qc_bit_flip_measurements.x(0).c_if(cregister, 3)
```

[34]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e59e52280>
```

$$|\Psi_4>=\sqrt{0.2}|1>_0|1>_1|1>_2+\sqrt{0.8}|0>_0|1>_1|1>_2=\sqrt{0.2}\Big(|1>_0+\sqrt{0.8}|0>_0\Big)|1>_1|1>_2$$

We apply the X gate to the ``qubit 0'' if the measurement in the qubit 1 and 2 is $|1>_1|1>_2$

In this case the measurement in the qubit 1 and 2 gives $\rightarrow|1>_1|1>_2$

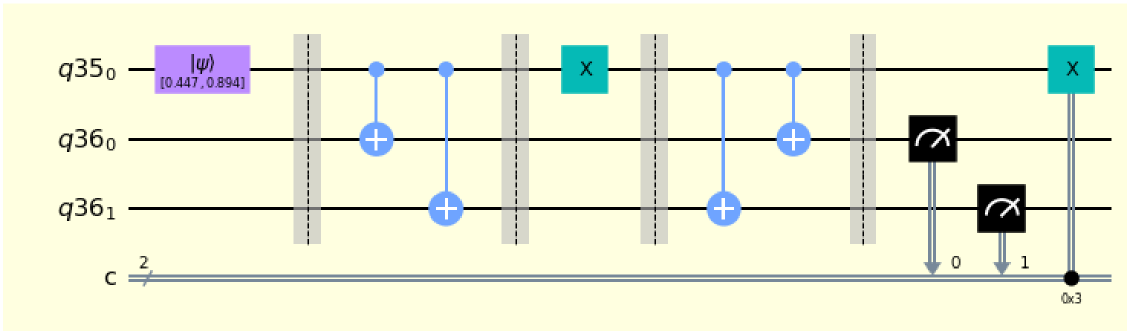
Appling the X gate to the qubit 0

$$|\Psi_4>=(X\otimes I\otimes I)|\Psi_4>=\sqrt{0.2}\Big(X|1>_0+\sqrt{0.8}X|0>_0\Big)I|1>_1 I|1>_2$$

$$|\Psi_4>=\sqrt{0.2}\Big(|0>_0+\sqrt{0.8}|1>_0\Big)|1>_1|1>_2$$

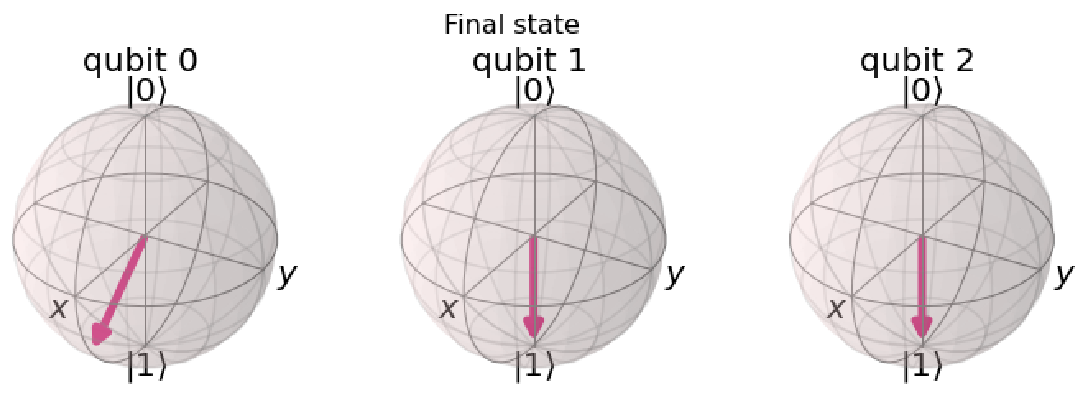
[35]:
```
qc_bit_flip_measurements.draw(output='mpl', style=style)
```

[35]:



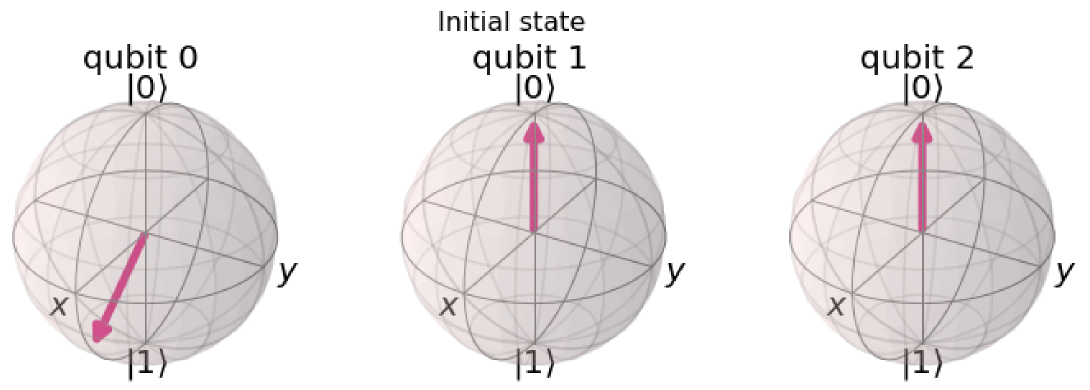
[36]:
```
out_vector1 = execute(qc_bit_flip_measurements, backend_statevector).result().
↪get_statevector()
plot_bloch_multivector(out_vector1, title='Final state')
```

[36]:



[37]:
```
plot_bloch_multivector(Initial_state1, title='Initial state')
```

[37]:



7 Part 3.3: The bit-flip code with ancillas and measurements

[38]:
```
qEncoding = QuantumRegister(3)
qAncillas = QuantumRegister(2, 'ancilla')
AncillasRegister = ClassicalRegister(2, 'c')

qc_bit_flip_ancillas_measuremnts =␣
↪QuantumCircuit(qEncoding,qAncillas,AncillasRegister)
```

[39]:
```
qc_bit_flip_ancillas_measuremnts.initialize(state_to_protect,0) # Apply␣
↪initialisation operation to the 0th qubit
```

[39]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e598a3ca0>
```

7.0.1 Showing the state $|\psi>$ ("qubit 0") in the Bloch sphere.

``qubit 1'' and ``qubit 2'' will be used to encode the state $|\Psi\quad>$. ``qubit 3'' and ``qubit 4'' are the ancillas

$$|\psi>=\sqrt{0.2}|0>+\sqrt{0.8}|1>$$

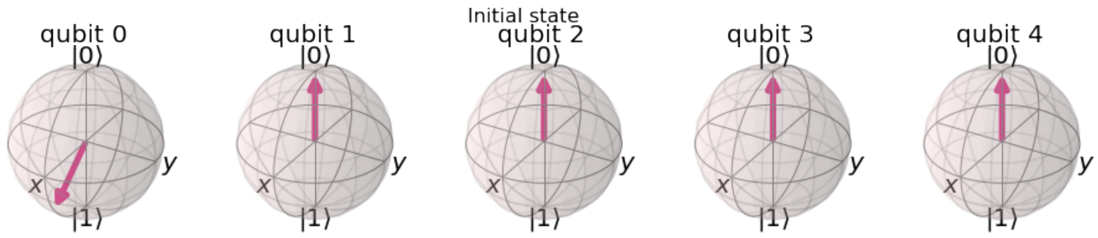
The initial state, $|\Psi_1>$, in the circuit is:

$$|\Psi_1>=|\psi>|0>_1|0>_2\left(|0>_{a_0}|0>_{a_1}\right)=\left(\sqrt{0.2}|0>+\sqrt{0.8}|1>\right)|0>_1|0>_2\left(|0>_{a_0}|0>_{a_1}\right)$$

$$|\Psi_1>=\sqrt{0.2}|0>_0|0>_1|0>_2\left(|0>_{a_0}|0>_{a_1}\right)+\sqrt{0.8}|1>_0|0>_1|0>_2\left(|0>_{a_0}|0>_{a_1}\right)$$

[40]:
```
Initial_state2 = Statevector.from_instruction(qc_bit_flip_ancillas_measuremnts)
plot_bloch_multivector(Initial_state2, title='Initial state')
```

[40]:



[41]:
```
### Perfmoing the encoding #####
qc_bit_flip_ancillas_measuremnts.barrier()
qc_bit_flip_ancillas_measuremnts.cx(0,1)
qc_bit_flip_ancillas_measuremnts.cx(0,2)
```

[41]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e587e8eb0>
```

$$|\Psi_2>=CNOT_{02}CNOT_{01}|\Psi_1>=\sqrt{0.2}CNOT_{02}CNOT_{01}|0>_0|0>_1|0>_2\left(|0>_{a_0}|0>_{a_1}\right)+\sqrt{0.8}CNOT_{02}CNOT$$

$$|\Psi_2>=\sqrt{0.2}|0>_0|0>_1|0>_2\left(|0>_{a_0}|0>_{a_1}\right)+\sqrt{0.8}|1>_0|1>_1|1>_2\left(|0>_{a_0}|0>_{a_1}\right)$$

[42]:
```
### Apply some noise #####
qc_bit_flip_ancillas_measuremnts.barrier()
#noise(qc)
qc_bit_flip_ancillas_measuremnts.x(0)
```

```
[42]: <qiskit.circuit.instructionset.InstructionSet at 0x7f5e598a3880>
```

$$|\Psi_3>= \Big(X\otimes I\otimes I\otimes I\otimes I\Big)|\Psi_2>= \sqrt{0.2}\Big(X\otimes I\otimes I\otimes I\otimes I\Big)|0>_0\ |0>_1\ |0>_2\ \Big(|0>_{a_0}\ |0>_{a_1}\ \Big)+\sqrt{0.8}\Big(X\otimes I\otimes I\otimes I\otimes I\Big)|1$$

$$|\Psi_3>= \sqrt{0.2}|1>_0\ |0>_1\ |0>_2\ \Big(|0>_{a_0}\ |0>_{a_1}\ \Big) + \sqrt{0.8}|0>_0\ |1>_1\ |1>_2\ \Big(|0>_{a_0}\ |0>_{a_1}\ \Big)$$

```
[43]: qc_bit_flip_ancillas_measuremnts.barrier()
      qc_bit_flip_ancillas_measuremnts.cx(0,4)
      qc_bit_flip_ancillas_measuremnts.cx(1,4)
```

```
[43]: <qiskit.circuit.instructionset.InstructionSet at 0x7f5e5881c490>
```

$$|\Psi_4>= CNOT_{1,a_1}CNOT_{0,a_1}|\Psi_3>= \sqrt{0.2}CNOT_{1,a_1}CNOT_{0,a_1}|1>_0\ |0>_1\ |0>_2\ \Big(|0>_{a_0}\ |0>_{a_1}\ \Big)+\sqrt{0.8}CNOT_{1,a_1}$$

$$|\Psi_4>= \sqrt{0.2}|1>_0\ |0>_1\ |0>_2\ \Big(|0>_{a_0}\ |1>_{a_1}\ \Big) + \sqrt{0.8}|0>_0\ |1>_1\ |1>_2\ \Big(|0>_{a_0}\ |1>_{a_1}\ \Big)$$

```
[44]: qc_bit_flip_ancillas_measuremnts.barrier()
      qc_bit_flip_ancillas_measuremnts.cx(1,3)
      qc_bit_flip_ancillas_measuremnts.cx(2,3)
```

```
[44]: <qiskit.circuit.instructionset.InstructionSet at 0x7f5e5881c7f0>
```

$$|\Psi_5>= CNOT_{2,a_0}CNOT_{1,a_0}|\Psi_4>= \sqrt{0.2}CNOT_{2,a_0}CNOT_{1,a_0}|1>_0\ |0>_1\ |0>_2\ \Big(|0>_{a_0}\ |1>_{a_1}\ \Big)+\sqrt{0.8}CNOT_{2,a_0}$$

$$|\Psi_5>= \sqrt{0.2}|1>_0\ |0>_1\ |0>_2\ \Big(|0>_{a_0}\ |1>_{a_1}\ \Big) + \sqrt{0.8}|0>_0\ |1>_1\ |1>_2\ \Big(|0>_{a_0}\ |1>_{a_1}\ \Big)$$

```
[45]: qc_bit_flip_ancillas_measuremnts.barrier()
      qc_bit_flip_ancillas_measuremnts.measure(qAncillas,AncillasRegister)
```

```
[45]: <qiskit.circuit.instructionset.InstructionSet at 0x7f5e5881cf10>
```

$$|\Psi_5>= \Big(\sqrt{0.2}|1>_0\ |0>_1\ |0>_2 +\sqrt{0.8}|0>_0\ |1>_1\ |1>_2\Big)\Big(|0>_{a_0}\ |1>_{a_1}\ \Big)$$

`Measurement in the ancillas qubits gives` $\rightarrow\ |0>_{a_0}\ |1>_{a_1}$

Ancilla Measurement	Collapsed State	Consequence
00	$\alpha\|000>+\beta\|111>$	No error
01	$\alpha\|100>+\beta\|011>$	An X gate is applied on qubit 0
10	$\alpha\|001>+\beta\|110>$	An X gate is appliedon qubit 2
11	$\alpha\|010>+\beta\|101>$	An X gate is appliedon qubit 1

If the ancillas measurement gives$\rightarrow$ $|0>_{a_0}|1>_{a_1}$

Apply the x gate to the ``qubit 0''

[46]:
```
qc_bit_flip_ancillas_measuremnts.x(0).c_if(AncillasRegister, 2)
qc_bit_flip_ancillas_measuremnts.x(1).c_if(AncillasRegister, 3)
qc_bit_flip_ancillas_measuremnts.x(2).c_if(AncillasRegister, 1)
```

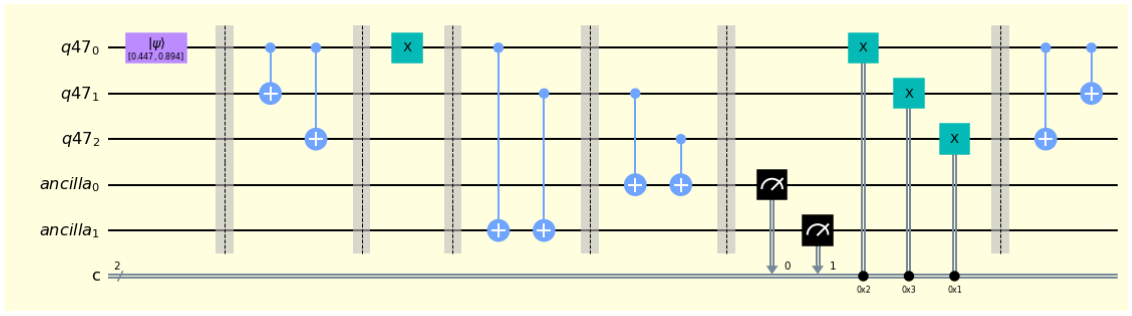
[46]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e5881c1f0>
```

[47]:
```
qc_bit_flip_ancillas_measuremnts.barrier()
qc_bit_flip_ancillas_measuremnts.cx(0,2)
qc_bit_flip_ancillas_measuremnts.cx(0,1)
```

[47]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e5881cc70>
```

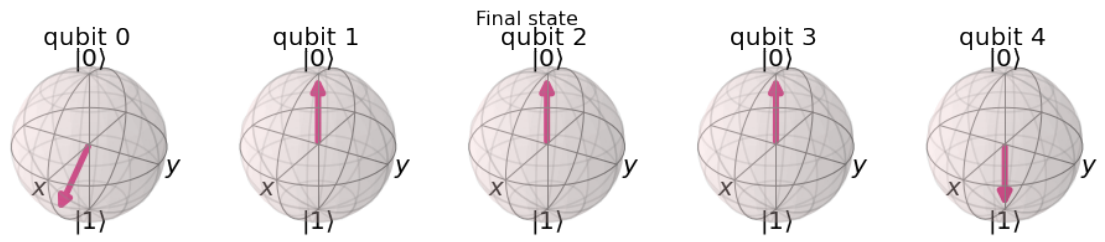
[48]:
```
qc_bit_flip_ancillas_measuremnts.draw(output='mpl', style=style)
```

[48]:



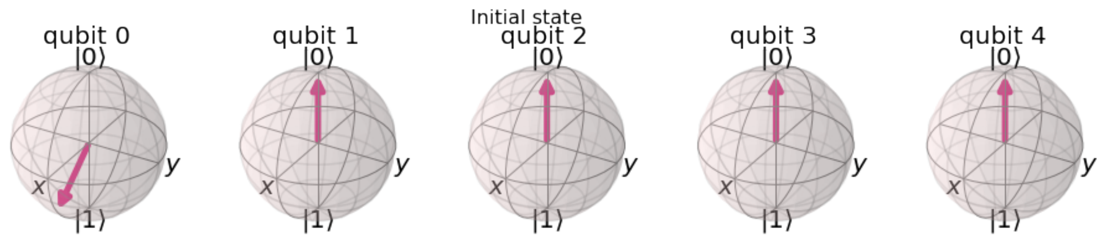
[49]:
```
out_vector2 = execute(qc_bit_flip_ancillas_measuremnts,backend_statevector).
 ↪result().get_statevector()
plot_bloch_multivector(out_vector2, title='Final state')
```

[49]:



[50]:
```
plot_bloch_multivector(Initial_state2, title='Initial state')
```

[50]:



8 Part 3.4: The sign-flip (phase-flip) code

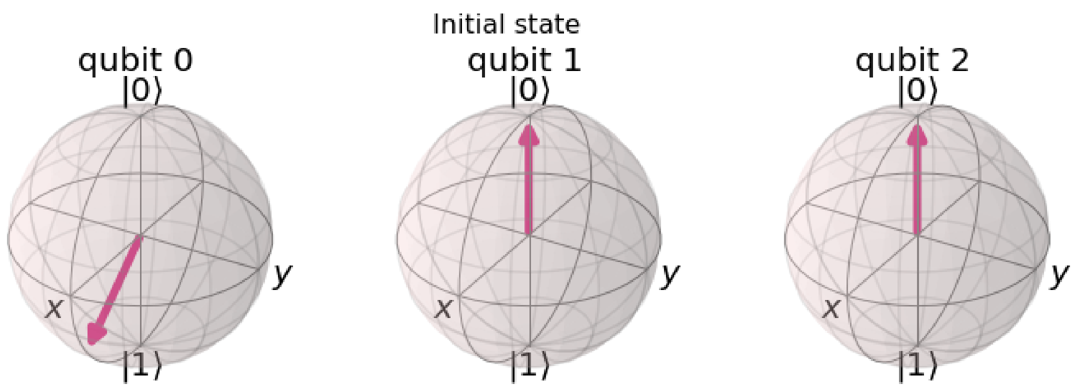
[51]:
```
qregister_to_protect = QuantumRegister(1)
qregister = QuantumRegister(2)
creg = ClassicalRegister(2, 'c')

qc_phase_flip = QuantumCircuit(qregister_to_protect,qregister,creg)
```

[52]:
```
qc_phase_flip.initialize(state_to_protect,0) # Apply initialisation operation␣
 ↪to the 0th qubit
```

[52]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4fdcf1f0>
```

[53]:
```
Initial_state3 = Statevector.from_instruction(qc_phase_flip)
plot_bloch_multivector(Initial_state3, title='Initial state')
```

[53]:



[54]:
```
### Perfmoing the encoding #####
qc_phase_flip.barrier()
qc_phase_flip.cx(0,1)
qc_phase_flip.cx(0,2)
```

[54]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4fdcf580>
```

We known that $HZH = X$. Just before to sending our information through the quantum channel and just after receiving the quantum information we should apply Hadamard gates

[55]:
```
### Apply thje hadamard gates #####
qc_phase_flip.barrier()
qc_phase_flip.h(0)
qc_phase_flip.h(1)
qc_phase_flip.h(2)
```

[55]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4fbd7400>
```

[56]:
```
### Apply some noise #####
qc_phase_flip.barrier()
#noise(qc)
qc_phase_flip.z(0)

qc_phase_flip.barrier()
```

[56]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4fdcf7c0>
```

[57]:
```
### Apply thje hadamard gates #####
qc_phase_flip.h(0)
qc_phase_flip.h(1)
qc_phase_flip.h(2)
```

[57]: `<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4fb11130>`

[58]:
```
### Aplying the inverse of the encoding procedure ###
qc_phase_flip.barrier()

qc_phase_flip.cx(0,2)
qc_phase_flip.cx(0,1)
```

[58]: `<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4fb11670>`

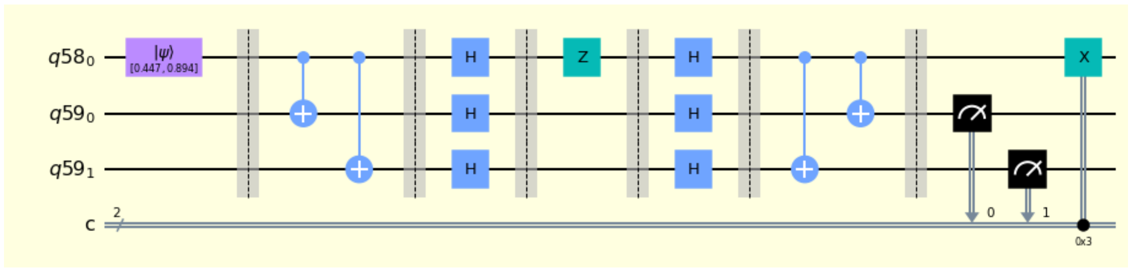
We apply the X gate to the ``qubit 0'' if the measurement in the qubit 1 and 2 is $|1>_1 |1>_2$

[59]:
```
qc_phase_flip.barrier()
qc_phase_flip.measure(qregister,creg)

qc_phase_flip.x(0).c_if(cregister, 3)
```

[59]: `<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4fb11550>`

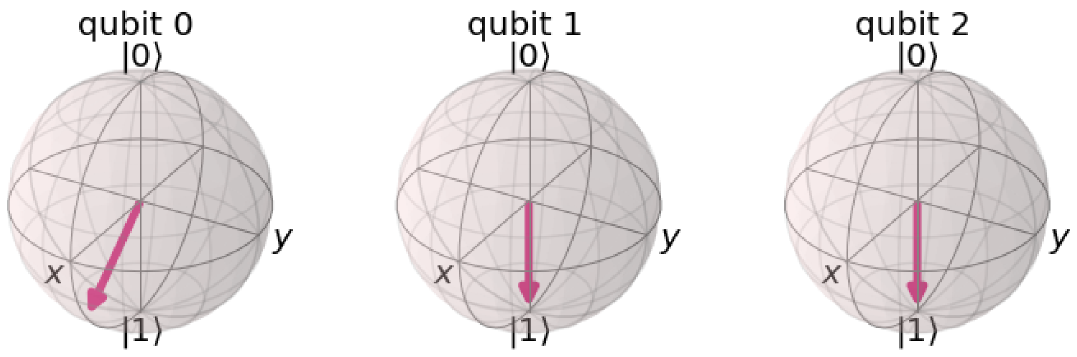
[60]:
```
qc_phase_flip.draw(output='mpl', style=style)
```

[60]:



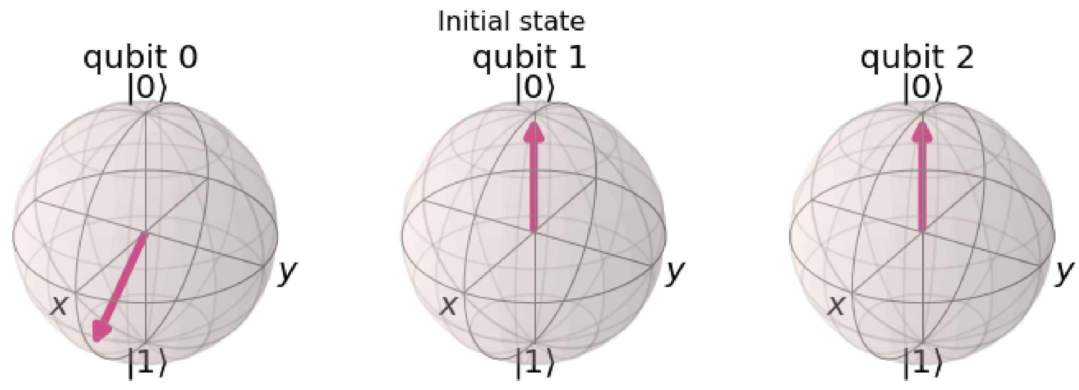
[61]:
```
out_vector4 = execute(qc_phase_flip, backend_statevector).result().
 ↪get_statevector()
plot_bloch_multivector(out_vector4)
```

[61]:



[62]:
```
plot_bloch_multivector(Initial_state3, title='Initial state')
```

[62]:



9 Part 3.5: The 9-qubit error correcting code

The nine qubit error correcting code or the Shor code is able to correct a logical qubit from one bit-flip, one phase-flip or one of each.

[63]:
```
qregA = QuantumRegister(1)
qr1 = QuantumRegister(2)
qregB = QuantumRegister(1)
qr2 = QuantumRegister(2)
qregC = QuantumRegister(1)
qr3 = QuantumRegister(2)
r1 = ClassicalRegister(2, 'c1')
r2 = ClassicalRegister(2, 'c2')
r3 = ClassicalRegister(2, 'c3')
registerBC = ClassicalRegister(2, 'c')

qc_shor_code = QuantumCircuit(qregA, qr1,qregB,␣
 ↪qr2,qregC,qr3,r1,r2,r3,registerBC)
```

[64]:
```
qc_shor_code.initialize(state_to_protect,0) # Apply initialisation operation to␣
 ↪the 0th qubit
qc_shor_code.barrier()
```

[64]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e5820e730>
```

The initial state, $|\Psi_1>$, in the circuit is:

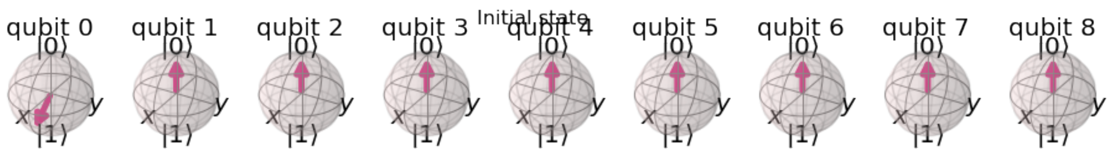
$$|\Psi_1>=|\psi>|0>_1|0>_2|0>_3|0>_4|0>_5|0>_6|0>_7|0>_8=\left(\sqrt{0.2}|0>_0+\sqrt{0.8}|1>_0\right)||0>_1|0>_2|0>_3|0>_4|0$$

$|\Psi_1>= \sqrt{0.2}|0>_0 |0>_1 |0>_2 |0>_3 |0>_4 |0>_5 |0>_6 |0>_7 |0>_8 + \sqrt{0.8}|1>_0 |0>_1 |0>_2 |0>_3 |0>_4 |0>_5 |0>_6 |$

The state $|\psi>$ is ``qubit 0''

[65]:
```
Initial_state4 = Statevector.from_instruction(qc_shor_code)
plot_bloch_multivector(Initial_state4, title='Initial state')
```

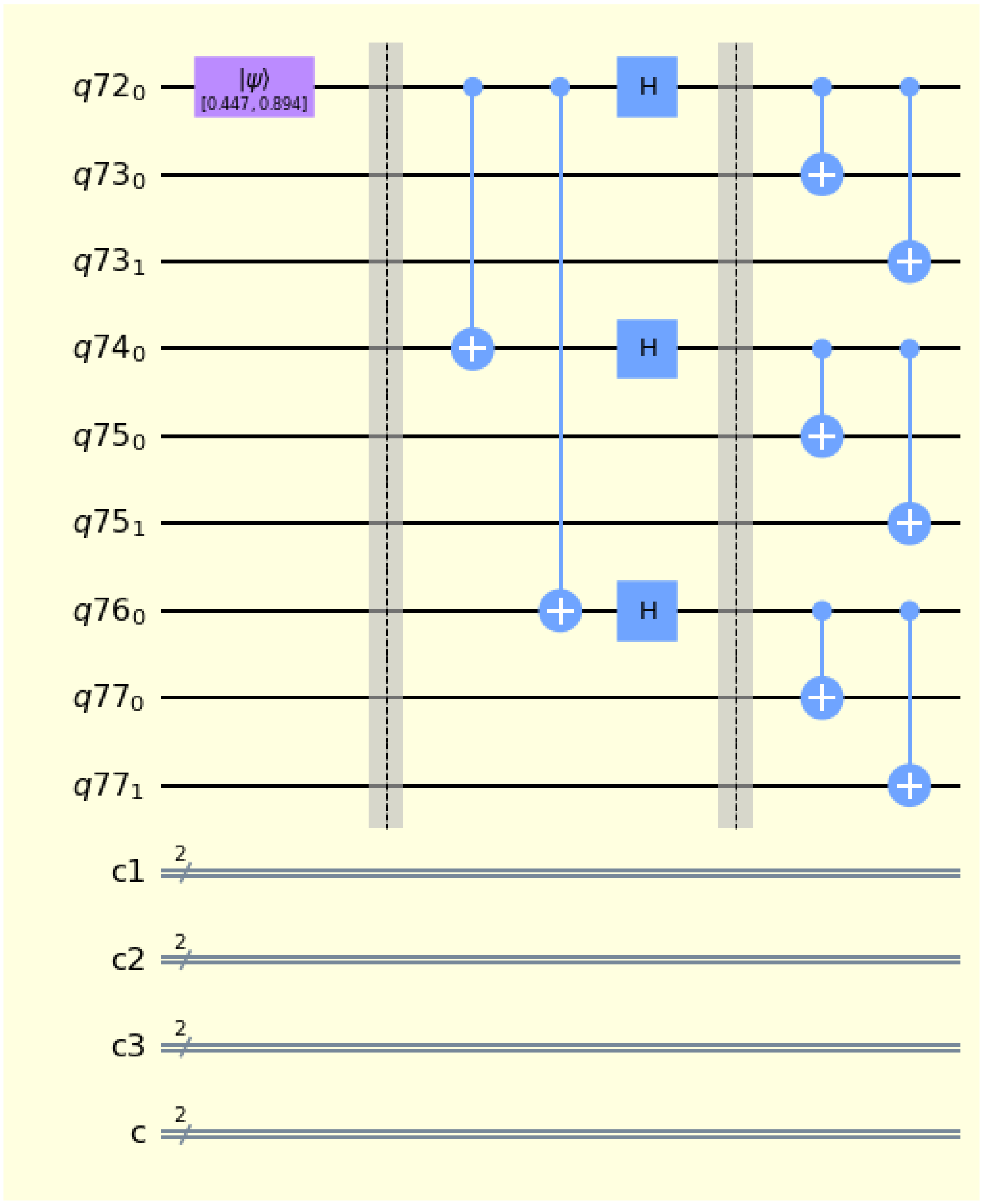
[65]:



9.0.1 Performing the encoding

[66]:
```
qc_shor_code.cx(0,3)
qc_shor_code.cx(0,6)
qc_shor_code.h(0)
qc_shor_code.h(3)
qc_shor_code.h(6)

qc_shor_code.barrier()

qc_shor_code.cx(0,1)
qc_shor_code.cx(0,2)

qc_shor_code.cx(3,4)
qc_shor_code.cx(3,5)

qc_shor_code.cx(6,7)
qc_shor_code.cx(6,8)

qc_shor_code.draw(output='mpl', style=style)
```

[66]:



$$|\Psi_2> = \Big(CNOT_{68}CNOT_{67}\Big)\Big(CNOT_{35}CNOT_{34}\Big)\Big(CNOT_{02}CNOT_{01}\Big)\Big(H_6H_3H_0\Big)\Big(CNOT_{06}CNOT_{03}\Big)|\Psi_1>$$

$$H_0 = H \otimes I \otimes I \otimes I \otimes I \otimes I \otimes I \otimes I \otimes I$$

$$H_3 = I \otimes I \otimes I \otimes H \otimes I \otimes I \otimes I \otimes I \otimes I$$
$$H_6 = I \otimes I \otimes I \otimes I \otimes I \otimes I \otimes H \otimes I \otimes I$$

$$|\Psi_2 >= \sqrt{0.2}\frac{\Big(|000>+|111>\Big)\Big(|000>+|111>\Big)\Big(|000>+|111>\Big)}{2\sqrt{2}}+\sqrt{0.8}\frac{\Big(|000>-|111>\Big)\Big(|000>-|111>}{2\sqrt{2}}$$

[67]:
```
### Apply some gate error, in this case X y Z gate ###
qc_shor_code.barrier()
qc_shor_code.x(0)
qc_shor_code.z(6)
```

[67]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4eea2550>
```

[68]:
```
## This is the saame as the bit-flip code with measurements (Part 3.2) repeated␣
 ↪ 3 times ##
qc_shor_code.barrier()

qc_shor_code.cx(0,2)
qc_shor_code.cx(0,1)

qc_shor_code.cx(3,5)
qc_shor_code.cx(3,4)

qc_shor_code.cx(6,8)
qc_shor_code.cx(6,7)

qc_shor_code.barrier()
qc_shor_code.measure(qr1,r1)
qc_shor_code.measure(qr2,r2)
qc_shor_code.measure(qr3,r3)
```

[68]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4f0a7eb0>
```

We apply the X gate to the ``qubit 0'' if the measurement in the qubits 1 and 2 is $|1>_1$ $|1>_2$, or the measurement in the qubits if 4 and 5 is $|1>_4$ $|1>_5$, or the measurement in the qubits 7 and 8 is $|1>_7$ $|1>_8$

[69]:
```
qc_shor_code.x(qregA).c_if(r1, 3)
qc_shor_code.x(qregB).c_if(r2, 3)
qc_shor_code.x(qregC).c_if(r3, 3)
```

[69]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e685d3dc0>
```

[70]:
```
### Aplying the inverse of the encoding procedure ###
```

```
qc_shor_code.barrier()
qc_shor_code.h(0)
qc_shor_code.h(3)
qc_shor_code.h(6)

qc_shor_code.cx(0,6)
qc_shor_code.cx(0,3)
```

[70]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e69498190>
```

[71]:
```
qc_shor_code.measure(qregB,registerBC[0])
qc_shor_code.measure(qregC,registerBC[1])
```

[71]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e5bf5e820>
```

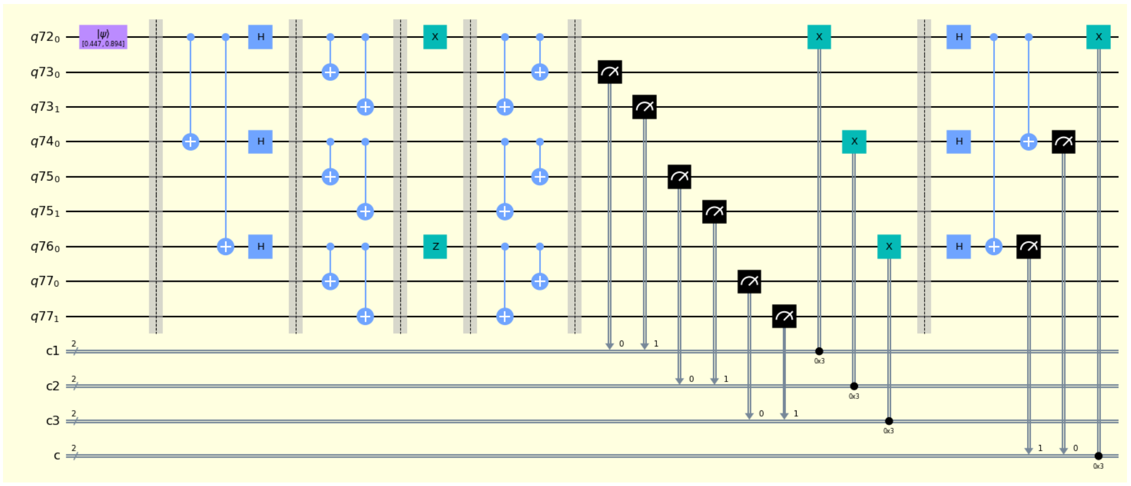
We apply the X gate to the ``qubit 0'' if the measurement in the qubit 3 and 6 is $|1>_3 |1>_6$

[72]:
```
qc_shor_code.x(qregA).c_if(registerBC, 3)
```

[72]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e69345070>
```

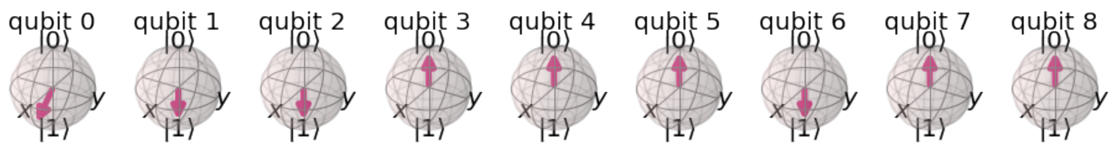
[73]:
```
qc_shor_code.draw(output='mpl', style=style, fold= 30)
```

[73]:



[74]:
```
out_vector5 = execute(qc_shor_code,backend_statevector).result().
↪get_statevector()
plot_bloch_multivector(out_vector5)
```

[74]:

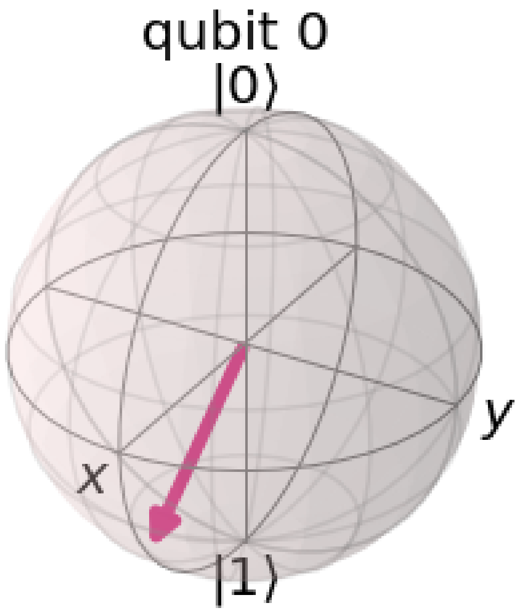


```
[75]: reduced_state_shor_code = partial_trace(out_vector5 , [1,2,3,4,5,6,7,8])
```

We see that the state has been protected of the noise

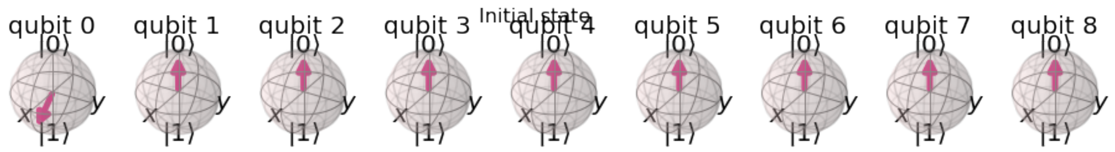
```
[76]: plot_bloch_multivector(reduced_state_shor_code)
```

[76]:



```
[77]: plot_bloch_multivector(Initial_state4, title='Initial state')
```

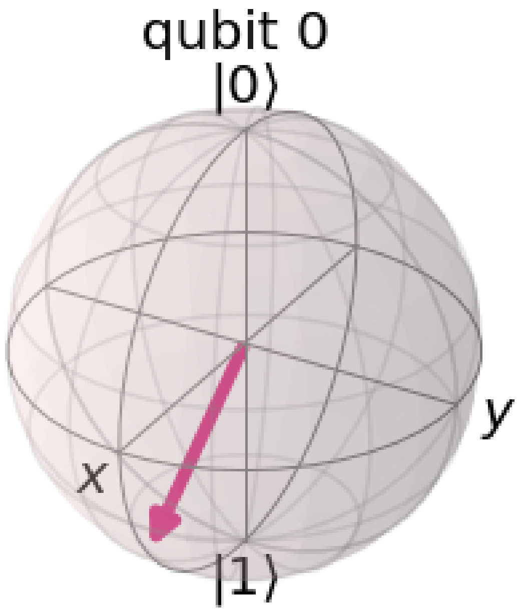
[77]:



```
[78]: reduced_intial_state_shor_code = partial_trace(Initial_state4 ,␣
       ↪[1,2,3,4,5,6,7,8])
```

[79]:
```
plot_bloch_multivector(reduced_intial_state_shor_code)
```

[79]:



10 Part 4: Test your solution by making many measurements over the final state and testing that the results are in line with the expectations.

[80]:
```
def shor_encoding(qc_shor_encode):
    ## Encoding for the shor code ##
    ################################
    qc_shor_encode.cx(0,3)
    qc_shor_encode.cx(0,6)
    qc_shor_encode.h(0)
    qc_shor_encode.h(3)
    qc_shor_encode.h(6)

    qc_shor_encode.barrier()

    qc_shor_encode.cx(0,1)
    qc_shor_encode.cx(0,2)

    qc_shor_encode.cx(3,4)
    qc_shor_encode.cx(3,5)

    qc_shor_encode.cx(6,7)
    qc_shor_encode.cx(6,8)

    qc_shor_encode.barrier()

    return(qc_shor_encode)
```

[81]:
```
def shor_correct(qc_shor_code_correct,qCorrect_regA, qCorrect_r1,qCorrect_regB,␣
↪qCorrect_r2,qCorrect_regC,qCorrect_r3,rCorrect1,rCorrect2,rCorrect3,registerCorrectBC␣
↪):
    ## Correction of the error in the shor code ##
    ##############################################
    qc_shor_code_correct.barrier()
    qc_shor_code_correct.cx(0,2)
    qc_shor_code_correct.cx(0,1)

    qc_shor_code_correct.cx(3,5)
    qc_shor_code_correct.cx(3,4)

    qc_shor_code_correct.cx(6,8)
    qc_shor_code_correct.cx(6,7)

    qc_shor_code_correct.barrier()
    qc_shor_code_correct.measure(qCorrect_r1,rCorrect1)
    qc_shor_code_correct.measure(qCorrect_r2,rCorrect2)
    qc_shor_code_correct.measure(qCorrect_r3,rCorrect3)

    qc_shor_code_correct.x(qCorrect_regA).c_if(rCorrect1, 3)
    qc_shor_code_correct.x(qCorrect_regB).c_if(rCorrect2, 3)
    qc_shor_code_correct.x(qCorrect_regC).c_if(rCorrect3, 3)

    qc_shor_code_correct.barrier()
    qc_shor_code_correct.h(0)
    qc_shor_code_correct.h(3)
    qc_shor_code_correct.h(6)

    qc_shor_code_correct.cx(0,6)
    qc_shor_code_correct.cx(0,3)

    qc_shor_code_correct.measure(qCorrect_regB,registerCorrectBC[0])
    qc_shor_code_correct.measure(qCorrect_regC,registerCorrectBC[1])

    qc_shor_code_correct.x(qCorrect_regA).c_if(registerCorrectBC, 3)

    return(qc_shor_code_correct)
```

[82]:
```
### Ciruit to encode the state for the bit-flip code ####
q_Encoding_bit_flip_reg1 = QuantumRegister(1)
```

```
q_Encoding_bit_flip_reg2 = QuantumRegister(2)

qc_Encoding_bit_flip = QuantumCircuit(Qreg1,Qreg2)

qc_Encoding_bit_flip.cx(0,1)
qc_Encoding_bit_flip.cx(0,2)
```

```
[82]: <qiskit.circuit.instructionset.InstructionSet at 0x7f5e4cbb6430>
```

```
[83]: ### Ciruit to correct the state for the bit-flip code ####
qBit_flip_correct_register1 = QuantumRegister(1)
qBit_flip_correct_register2 = QuantumRegister(2)
cBit_flip_correct_register = ClassicalRegister(2, 'c')

qcBit_flip_correct =␣
 ↪QuantumCircuit(qBit_flip_correct_register1,qBit_flip_correct_register2,cBit_flip_correct_re

qcBit_flip_correct.barrier()
qcBit_flip_correct.cx(0,2)
qcBit_flip_correct.cx(0,1)
qcBit_flip_correct.barrier()
qcBit_flip_correct.
 ↪measure(qBit_flip_correct_register2,cBit_flip_correct_register)
qcBit_flip_correct.x(0).c_if(cBit_flip_correct_register, 3)
qcBit_flip_correct.barrier()
```

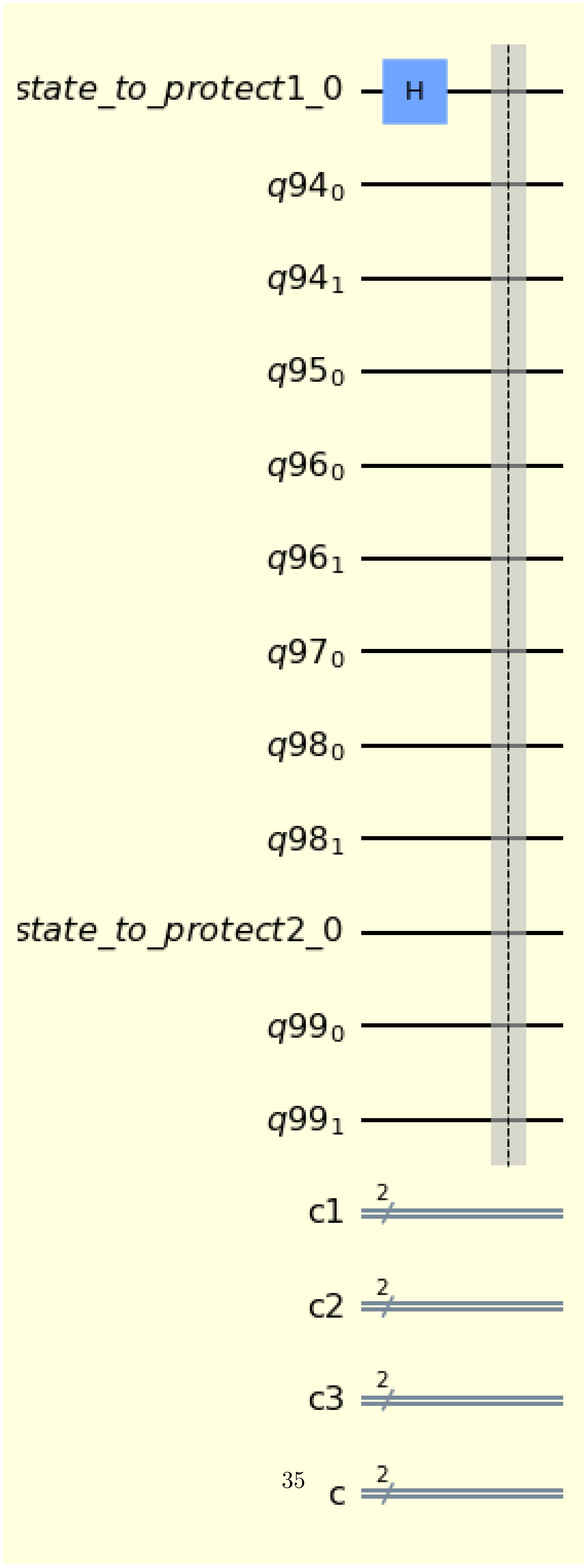
```
[83]: <qiskit.circuit.instructionset.InstructionSet at 0x7f5e4cb9f730>
```

10.0.1 Prepare the circuit to protect from noise

```
[84]: q_To_Protect1 = QuantumRegister(1, name = 'state_to_protect1')
qCorrect_r1 = QuantumRegister(2)
qCorrect_regB = QuantumRegister(1)
qCorrect_r2 = QuantumRegister(2)
qCorrect_regC = QuantumRegister(1)
qCorrect_r3 = QuantumRegister(2)
rCorrect1 = ClassicalRegister(2, 'c1')
rCorrect2 = ClassicalRegister(2, 'c2')
rCorrect3 = ClassicalRegister(2, 'c3')
registerCorrectBC = ClassicalRegister(2, 'c')
q_To_Protect2 =  QuantumRegister(1, name = 'state_to_protect2')
qregister_test2 = QuantumRegister(2)

qbell_test = QuantumCircuit(q_To_Protect1, qCorrect_r1,qCorrect_regB,␣
 ↪qCorrect_r2,qCorrect_regC,qCorrect_r3,rCorrect1,rCorrect2,rCorrect3,registerCorrectBC,␣
 ↪q_To_Protect2, qregister_test2)
```

```
qbell_test.h(0)
qbell_test.barrier()
qbell_test.draw(output='mpl', style=style)
```

[84]:


state_to_protect1_0
H
$q94_0$
$q94_1$
$q95_0$
$q96_0$
$q96_1$
$q97_0$
$q98_0$
$q98_1$
state_to_protect2_0
$q99_0$
$q99_1$
c1
2
c2
2
c3
2
c
2

state_to_protect1_0 is the first qubit of our circuit that will generate the bell state, we apply the hadamard gate to it. The state_to_protect2_0 is the second qubit that is needed to create the bell state. These two qubits are the ones that need to be protected from noise. In the case of the first qubit we need to use the shor code to protect the state from noise like the Z, X gate. In the case of the second one (state_to_protect2_0) the state is $|0>$ the Z gate will not affect the state so we only need to protect against the bit-flips.

10.0.2 Encode the states to protect

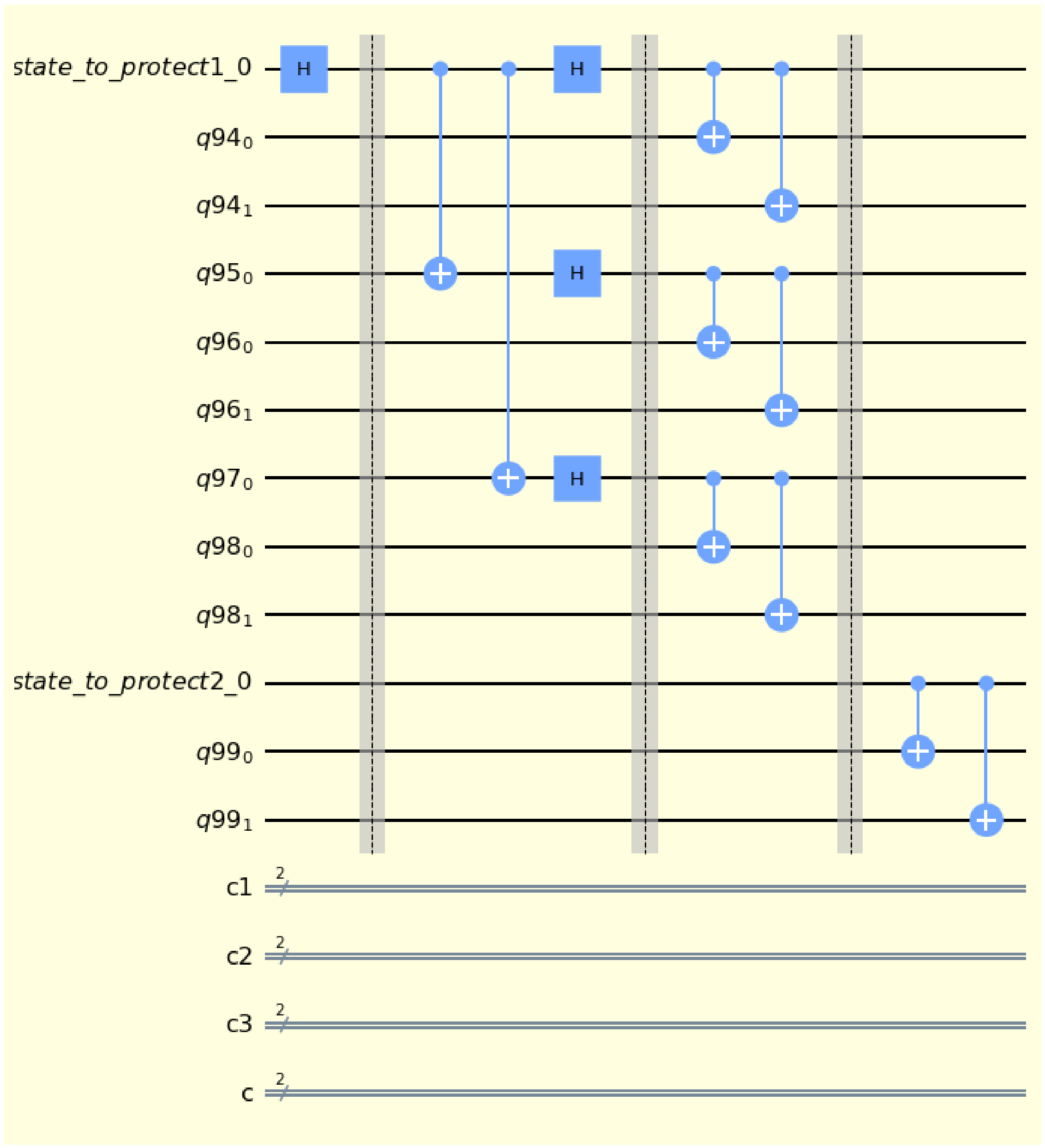
[85]:
```
shor_encoding(qbell_test)
```

[85]:
```
<qiskit.circuit.quantumcircuit.QuantumCircuit at 0x7f5e4cb9a820>
```

[86]:
```
new_qc1 = qbell_test.compose(qc_Encoding_bit_flip,[9,10,11])
```

[87]:
```
new_qc1.draw(output='mpl', style=style)
```

[87]:



10.0.3 Apply error gates, in this case a X gate to the first qubit and a Z gate to the second qubit

[88]:
```
new_qc1.barrier()
new_qc1.x(0)
new_qc1.z(9)
```

[88]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4c7d2370>
```

10.0.4 Apply the shor code to the first qubit and the bit flip code to the second qubit

[89]:
```
new_qc2 = new_qc1.compose(qcBit_flip_correct,[9,10,11])
```

[90]:
```
shor_correct(new_qc2,q_To_Protect1, qCorrect_r1,qCorrect_regB,␣
↪qCorrect_r2,qCorrect_regC,qCorrect_r3,rCorrect1,rCorrect2,rCorrect3,registerCorrectBC)
```

[90]:
```
<qiskit.circuit.quantumcircuit.QuantumCircuit at 0x7f5e4cb95ee0>
```

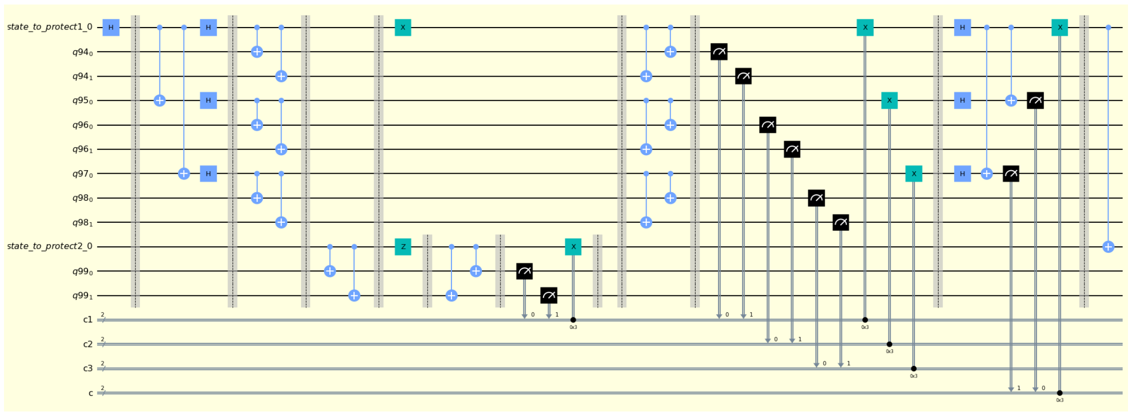
10.0.5 Apply the CNOT gate to create the bell state

[91]:
```
new_qc2.barrier()
new_qc2.cx(0,9)
```

[91]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4c765520>
```

[92]:
```
new_qc2.draw(output='mpl', style=style, fold= 42)
```

[92]:



We can use the partial trace to obtain the state_to_protect1_0 and state_to_protect2_0

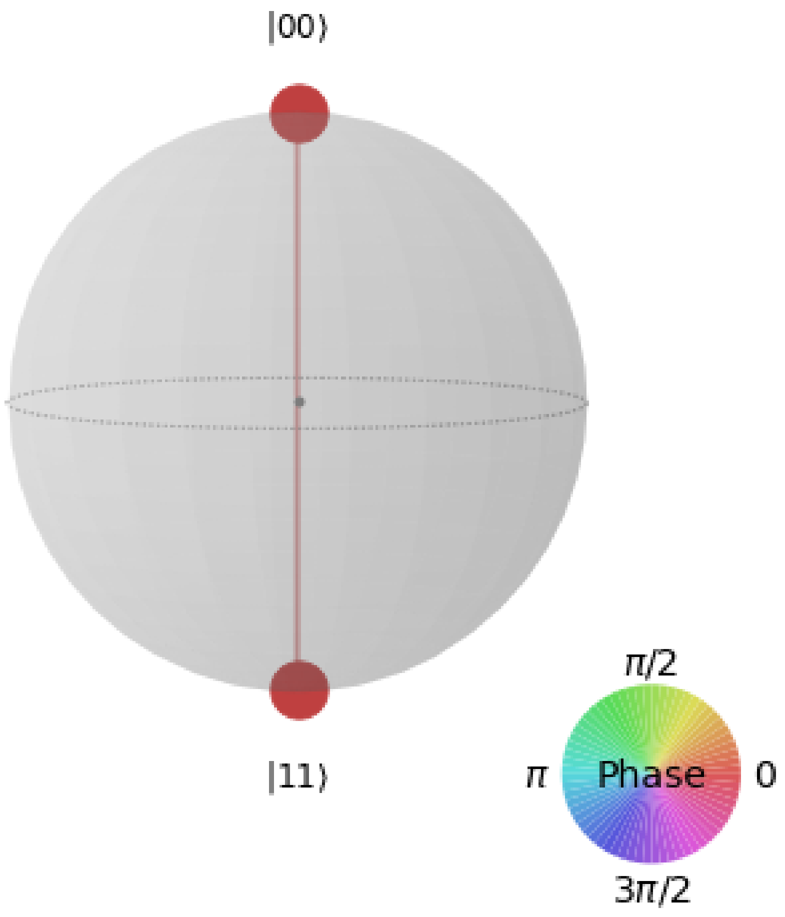
$$\rho^A = tr_B[\rho^{AB}]$$

In this case the subsystem A is the state_to_protect1_0 and state_to_protect2_0 (qubits 0 and 9) and subsystem B is the qubist 1,2,3,4,5,6,7,8,10 and 11

[93]:
```
out_vector8 = execute(new_qc2,backend_statevector).result().get_statevector()
reduced_state1 = partial_trace(out_vector8 , [1,2,3,4,5,6,7,8,10,11])
```

[94]:
```
plot_state_qsphere(reduced_state1)
```

[94]:



At the end of the ciruit has created the expected bell state as shown in the qsphere

11 Error of type ZX

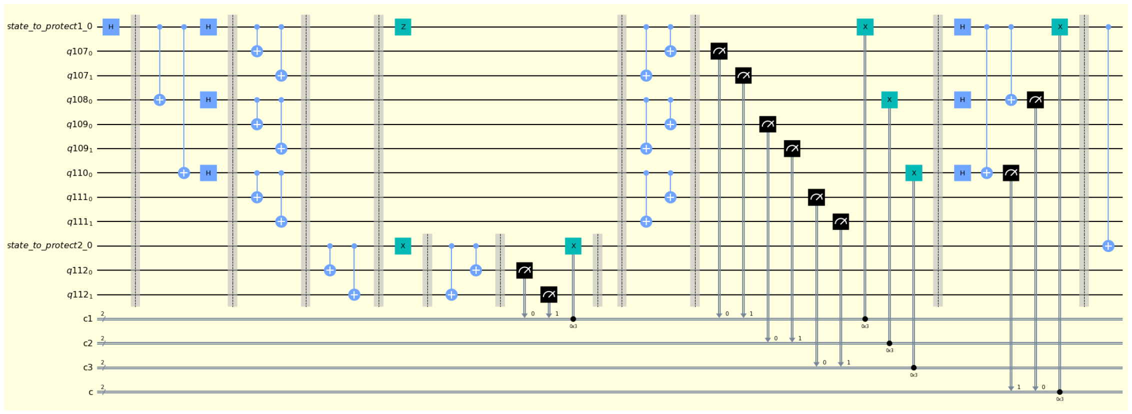
```
[95]: q_To_Protect1_ZX = QuantumRegister(1, name = 'state_to_protect1')
qCorrect_r1_ZX = QuantumRegister(2)
qCorrect_regB_ZX = QuantumRegister(1)
qCorrect_r2_ZX = QuantumRegister(2)
qCorrect_regC_ZX = QuantumRegister(1)
qCorrect_r3_ZX = QuantumRegister(2)
rCorrect1_ZX = ClassicalRegister(2, 'c1')
rCorrect2_ZX = ClassicalRegister(2, 'c2')
rCorrect3_ZX = ClassicalRegister(2, 'c3')
```

```
registerCorrectBC_ZX = ClassicalRegister(2, 'c')
q_To_Protect2_ZX =  QuantumRegister(1, name = 'state_to_protect2')
qregister_test2_ZX = QuantumRegister(2)

qbell_test_ZX = QuantumCircuit(q_To_Protect1_ZX,␣
 ↪qCorrect_r1_ZX,qCorrect_regB_ZX,␣
 ↪qCorrect_r2_ZX,qCorrect_regC_ZX,qCorrect_r3_ZX,rCorrect1_ZX,rCorrect2_ZX,rCorrect3_ZX,regis
 ↪q_To_Protect2_ZX, qregister_test2_ZX)

qbell_test_ZX.h(0)
qbell_test_ZX.barrier()

shor_encoding(qbell_test_ZX)

new_qc1_ZX = qbell_test_ZX.compose(qc_Encoding_bit_flip,[9,10,11])

#### Error gates ######
new_qc1_ZX.barrier()
new_qc1_ZX.z(0)
new_qc1_ZX.x(9)

new_qc2_ZX = new_qc1_ZX.compose(qcBit_flip_correct,[9,10,11])

shor_correct(new_qc2_ZX,q_To_Protect1_ZX , qCorrect_r1_ZX ,qCorrect_regB_ZX,␣
 ↪qCorrect_r2_ZX,qCorrect_regC_ZX,qCorrect_r3_ZX,rCorrect1_ZX,rCorrect2_ZX,rCorrect3_ZX,regis

new_qc2_ZX.barrier()
new_qc2_ZX.cx(0,9)
```

[95]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4aedb640>
```

[96]:
```
new_qc2_ZX.draw(output='mpl', style=style, fold= 42)
```

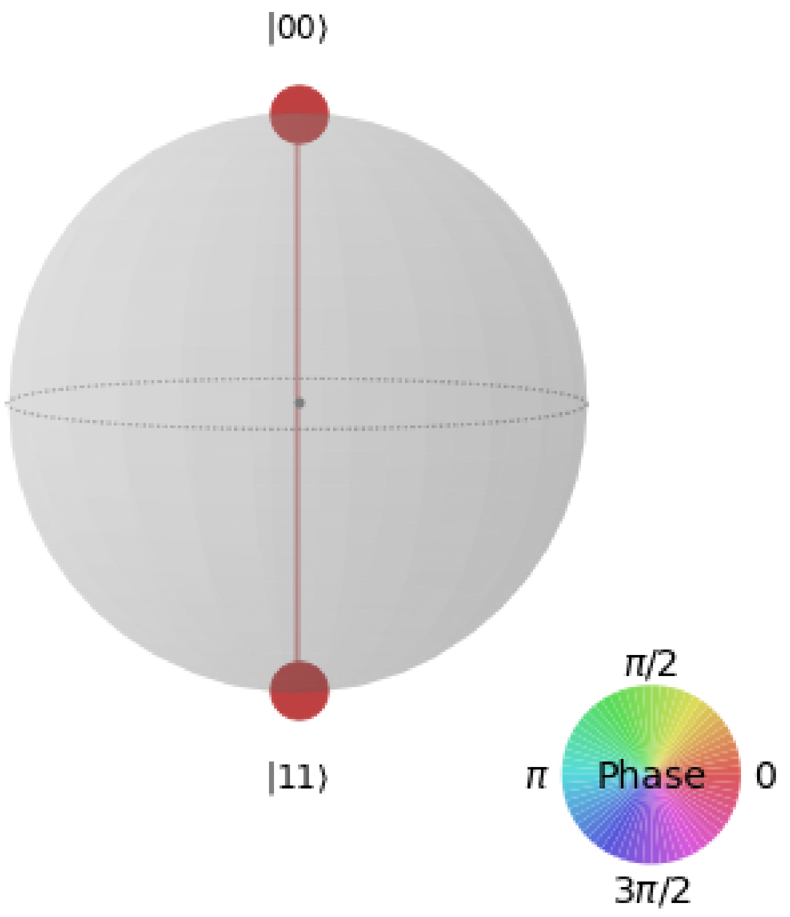
[96]:



[97]:
```
out_vector8 = execute(new_qc2,backend_statevector).result().get_statevector()
reduced_state1 = partial_trace(out_vector8 , [1,2,3,4,5,6,7,8,10,11])
```

[98]:
```
plot_state_qsphere(reduced_state1)
```

[98]:



```
The ciruit has created the expected bell state as shown in the qsphere. The YX
error were corrected
```

12 Error of type YX

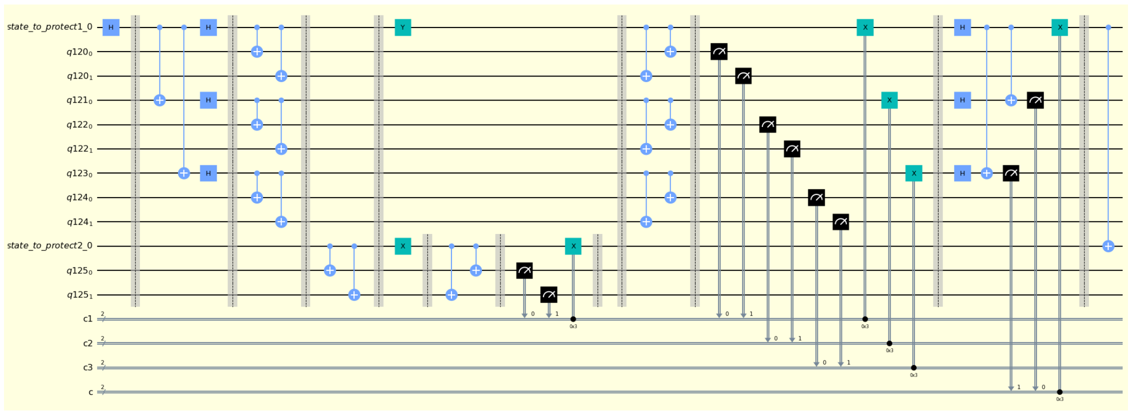
[99]:
```
q_To_Protect1_YX = QuantumRegister(1, name = 'state_to_protect1')
qCorrect_r1_YX = QuantumRegister(2)
qCorrect_regB_YX = QuantumRegister(1)
qCorrect_r2_YX = QuantumRegister(2)
```

```
qCorrect_regC_YX = QuantumRegister(1)
qCorrect_r3_YX = QuantumRegister(2)
rCorrect1_YX = ClassicalRegister(2, 'c1')
rCorrect2_YX = ClassicalRegister(2, 'c2')
rCorrect3_YX = ClassicalRegister(2, 'c3')
registerCorrectBC_YX = ClassicalRegister(2, 'c')
q_To_Protect2_YX =  QuantumRegister(1, name = 'state_to_protect2')
qregister_test2_YX = QuantumRegister(2)

qbell_test_YX = QuantumCircuit(q_To_Protect1_YX,␣
 ↪qCorrect_r1_YX,qCorrect_regB_YX,␣
 ↪qCorrect_r2_YX,qCorrect_regC_YX,qCorrect_r3_YX,rCorrect1_YX,rCorrect2_YX,rCorrect3_YX,regis
 ↪q_To_Protect2_YX, qregister_test2_YX)

qbell_test_YX.h(0)
qbell_test_YX.barrier()
qbell_test_YX.draw(output='mpl', style=style)

shor_encoding(qbell_test_YX)

new_qc1_YX = qbell_test_YX.compose(qc_Encoding_bit_flip,[9,10,11])

#### Error gates ######
new_qc1_YX.barrier()
new_qc1_YX.y(0)
new_qc1_YX.x(9)

new_qc2_YX = new_qc1_YX.compose(qcBit_flip_correct,[9,10,11])

shor_correct(new_qc2_YX,q_To_Protect1_YX , qCorrect_r1_YX ,qCorrect_regB_YX,␣
 ↪qCorrect_r2_YX,qCorrect_regC_YX,qCorrect_r3_YX,rCorrect1_YX,rCorrect2_YX,rCorrect3_YX,regis

new_qc2_YX.barrier()
new_qc2_YX.cx(0,9)
```

[99]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4bcc2310>
```

[100]:
```
new_qc2_YX.draw(output='mpl', style=style, fold= 42)
```

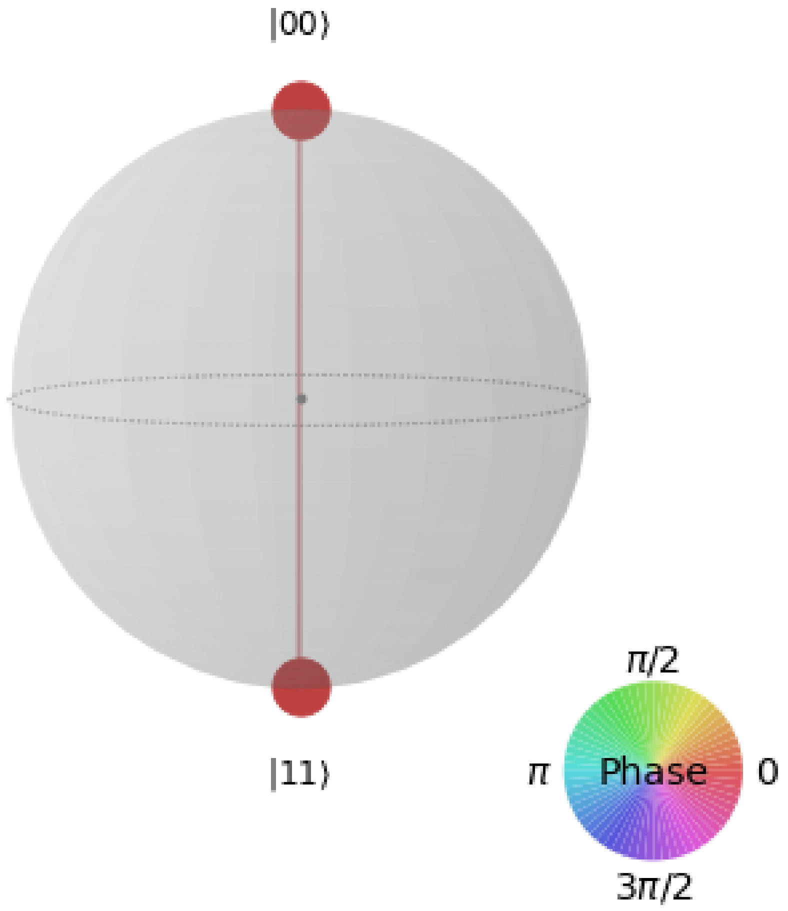
[100]:



[101]:
```
out_vector9 = execute(new_qc2_YX,backend_statevector).result().get_statevector()
reduced_state2 = partial_trace(out_vector9 , [1,2,3,4,5,6,7,8,10,11])
```

[102]:
```
plot_state_qsphere(reduced_state2)
```

[102]:



13 Random error gate and their corrections

[103]:
```
def random_error_gate(qcirc):

    if gate_errorX() == ['X']:
    ### Bit-flip in the qubit 0 #####
        qcirc.x(0)
    elif gate_errorX() == ['Z']:
    ### phase-flip in the qubit 0 #####
        qcirc.z(0)
    else:
        qcirc.i(0)

    if gate_errorX() == ['X']:
```

```
    ### Bit-flip in the qubit 9 #####
        qcirc.x(9)
    elif gate_errorX() == ['Z']:
    ### phase-flip in the qubit 9 #####
        qcirc.z(9)
    else:
        qcirc.i(9)

    return qcirc
```

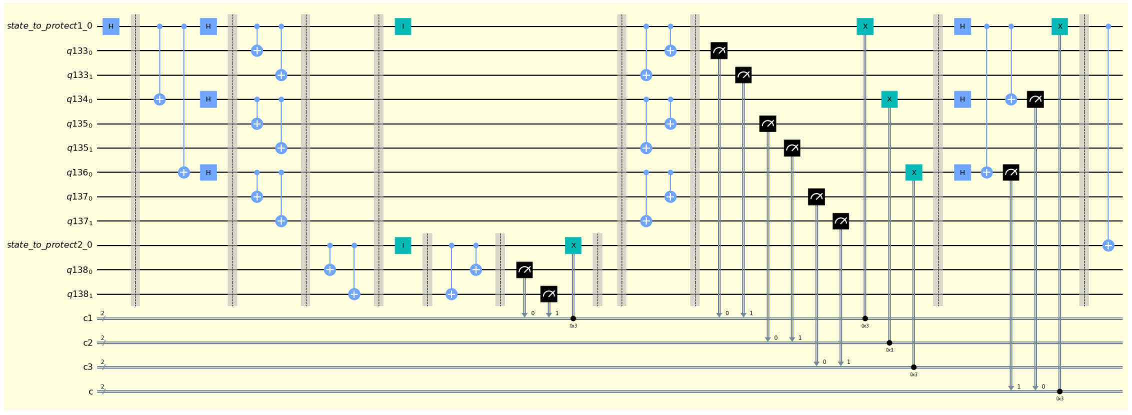
[104]:
```
q_To_Protect1_Random = QuantumRegister(1, name = 'state_to_protect1')
qCorrect_r1_Random = QuantumRegister(2)
qCorrect_regB_Random = QuantumRegister(1)
qCorrect_r2_Random = QuantumRegister(2)
qCorrect_regC_Random = QuantumRegister(1)
qCorrect_r3_Random = QuantumRegister(2)
rCorrect1_Random = ClassicalRegister(2, 'c1')
rCorrect2_Random = ClassicalRegister(2, 'c2')
rCorrect3_Random = ClassicalRegister(2, 'c3')
registerCorrectBC_Random = ClassicalRegister(2, 'c')
q_To_Protect2_Random =  QuantumRegister(1, name = 'state_to_protect2')
qregister_test2_Random = QuantumRegister(2)

qbell_test_Random = QuantumCircuit(q_To_Protect1_Random,␣
 ↪qCorrect_r1_Random,qCorrect_regB_Random,␣
 ↪qCorrect_r2_Random,qCorrect_regC_Random,qCorrect_r3_Random,rCorrect1_Random,rCorrect2_Rando
 ↪q_To_Protect2_Random, qregister_test2_Random)

qbell_test_Random.h(0)
qbell_test_Random.barrier()

shor_encoding(qbell_test_Random)

new_qc1_Random = qbell_test_Random.compose(qc_Encoding_bit_flip,[9,10,11])

#### Error gates ######
new_qc1_Random.barrier()
random_error_gate(new_qc1_Random)
################################

new_qc2_Random = new_qc1_Random.compose(qcBit_flip_correct,[9,10,11])

shor_correct(new_qc2_Random,q_To_Protect1_Random , qCorrect_r1_Random␣
 ↪,qCorrect_regB_Random,␣
 ↪qCorrect_r2_Random,qCorrect_regC_Random,qCorrect_r3_Random,rCorrect1_Random,rCorrect2_Rando

new_qc2_Random.barrier()
```

```
new_qc2_Random.cx(0,9)
```

[104]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f5e4aa9b8e0>
```

[105]:
```
new_qc2_Random.draw(output='mpl', style=style, fold= 42)
```

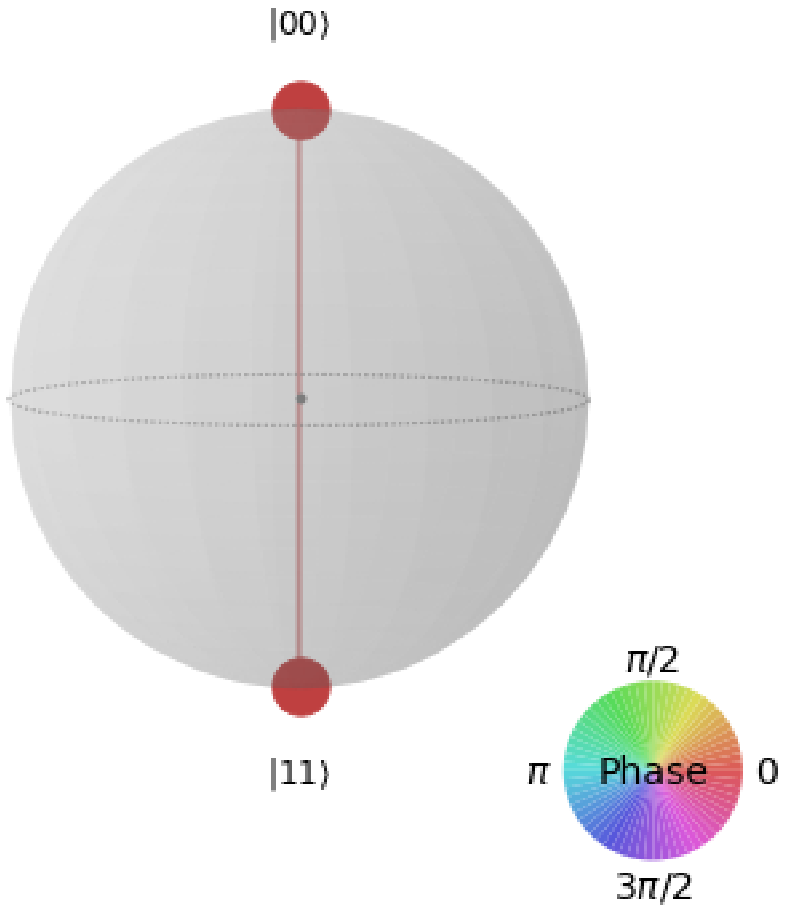
[105]:



[106]:
```
out_vector10 = execute(new_qc2_Random,backend_statevector).result().
 ↪get_statevector()
reduced_state3 = partial_trace(out_vector10 , [1,2,3,4,5,6,7,8,10,11])
```

[107]:
```
plot_state_qsphere(reduced_state3)
```

[107]:



13.0.1 References:

[1] Mermin N.D., Quantum Computer Science: An Introduction, Cambridge University Press, 2007

[2] D. Bacon, http://courses.cs.washington.edu/courses/cse599d/06wi/lecturenotes16.pdf [20. february 2021]

[3] John Watrous, https://cs.uwaterloo.ca/~watrous/QC-notes/ [20. february 2021]

[4] Devitt, Simon J and Munro, William J and Nemoto, Kae. ``Quantum error correction for beginners''. Reports on Progress in Physics, IOP Publishing, 2013. url: http://dx.doi.org/10.1088/0034-4885/76/7/076001

```
[ ]:
```