BB84 Depolarization Noise

August 30, 2021

[14]:
```
import numpy as np

# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *
from qiskit.providers.aer import QasmSimulator

# Loading your IBM Quantum account(s)
provider = IBMQ.load_account()
```

```
ibmqfactory.load_account:WARNING:2021-08-30 18:58:26,904: Credentials are
already in use. The existing account in the session will be replaced.
```

[4]:
```
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram
import qiskit.providers.aer.noise as noise

# Error probabilities
res = []
r1=0.001
r2=0.01
while(r1 < r2+0.001 ):
    res.append(r1)
    r1 += 0.001

print(res)

for i in range(len(res)):
    prob_1 = prob_2 = float(res[i])
    print(prob_1)
    # Depolarizing quantum errors
    error_1 = noise.depolarizing_error(prob_1, 1)
    error_2 = noise.depolarizing_error(prob_2, 2)

    # Add errors to noise model
    noise_model = noise.NoiseModel()
```

```
noise_model.add_all_qubit_quantum_error(error_1, ['u1', 'u2', 'u3'])
noise_model.add_all_qubit_quantum_error(error_2, ['cx'])
print(noise_model)
# Get basis gates from noise model
basis_gates = noise_model.basis_gates

from qiskit.tools.monitor import backend_monitor
from qiskit import *
from qiskit.visualization import plot_histogram
from random import randrange, seed, sample
from sys import argv, exit
import random

#data = int(input('ENTER LENGTH OF BIT STREAM (example 5 For 10110):'))
data=1
################################################################################
print('|00>')
h=0
#h=int(input())
def bit_stream(p):
    key1 = ""
    for i in range(p):
        temp = str(random.randint(h,h))
        key1 += temp
    return(key1)

bitstream= bit_stream(data)
digits = [int(x) for x in str(bitstream)]
################################################################################
#print('List of Bit Stream to transfer over Quantum Channel')
#print(digits)
print('\n')
#n = len(digits)
n=1
################################################################################

bob_bits=[]
from random import choice
#m=0
for i in range(n):
    #m=m+10
    #print("No of identity Gate:",m)
    if digits[i] == 0:
        q = QuantumRegister(1, 'q')
        c = ClassicalRegister(1, 'c')
        qc = QuantumCircuit(q, c)
        qc.barrier()
```

```
            qc.h(0)
            qc.barrier()
            for j in range(10):
                qc.id(0)
                qc.barrier()
            qc.h(0)
            qc.barrier()
            qc.measure(q[0], c[0])
            #print(qc)

                    # Perform a noise simulation
            result = execute(qc, Aer.get_backend('qasm_simulator'),
                             basis_gates=basis_gates,
                             noise_model=noise_model,shots=1000).result()
            counts = result.get_counts(qc)
            plot_histogram(counts)
            #print(qc)
            bits = (result.get_counts(qc))
            print(bits)
            #print(bits)
            #itemMaxValue = max(bits.items(), key=lambda x : x[1])
            #print(itemMaxValue)
            #print(bits.get('0'))
            #print('\n')
            #print("============================================================")
```

```
[0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009000000000000001,
0.010000000000000002]
0.001
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'0': 1000}
0.002
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'1': 1, '0': 999}
0.003
```

```
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'0': 1000}
0.004
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'1': 4, '0': 996}
0.005
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'1': 4, '0': 996}
0.006
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'1': 1, '0': 999}
0.007
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'1': 6, '0': 994}
0.008
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
```

```
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'1': 12, '0': 988}
0.009000000000000001
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'1': 5, '0': 995}
0.010000000000000002
NoiseModel:
  Basis gates: ['cx', 'id', 'u1', 'u2', 'u3']
  Instructions with noise: ['u2', 'cx', 'u3', 'u1']
  All-qubits errors: ['u1', 'u2', 'u3', 'cx']
|00>


{'1': 16, '0': 984}
```

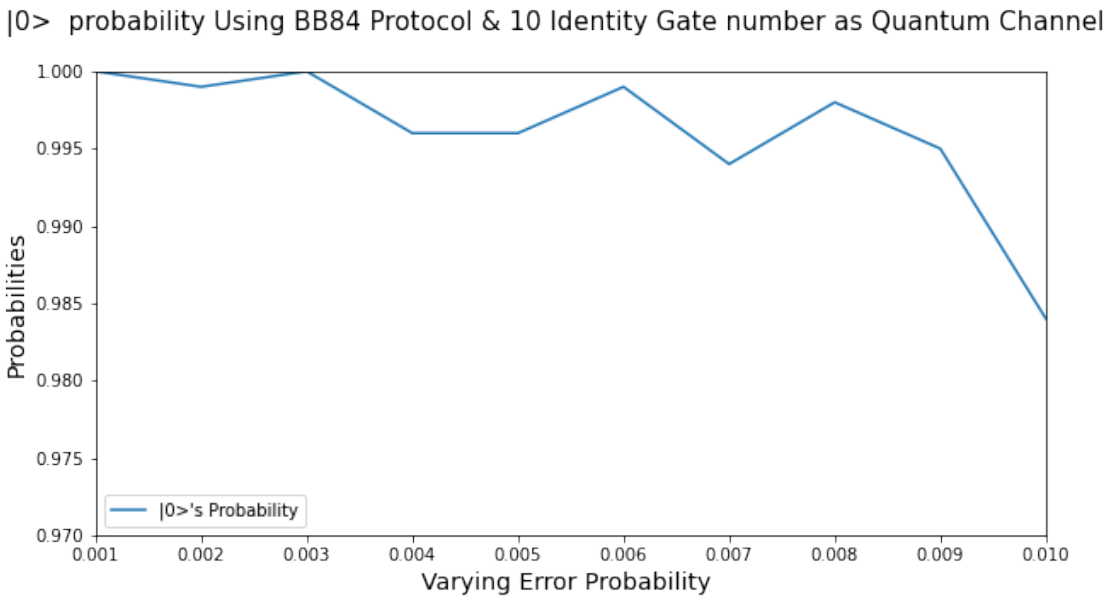
[13]:
```
import matplotlib.pyplot as plt
from matplotlib.ticker import (AutoMinorLocator, MultipleLocator)

fig, ax = plt.subplots(figsize=(10, 5))
fig.suptitle('|0>  probability Using BB84 Protocol & 10 Identity Gate number as␣
 ↪Quantum Channel',fontsize=15)

# naming the x axis
plt.xlabel('Varying Error Probability ',fontsize=14)
# naming the y axis
plt.ylabel('Probabilities',fontsize=14)
# giving a title to my graph
# Set axis ranges; by default this will put major ticks every 25.
#ax.set_xlim(0, 300)
#ax.set_ylim(0, 1)
ax.set_xlim(0.001, 0.01)
ax.set_ylim(0.97, 1)
fig = plt.figure(figsize=(8,5))
# line 2 points
y2 = [1.000,0.999,1.000,0.996,0.996,0.999,0.994,0.998,0.995,0.984]
x2 = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01]
# plotting the line 2 points
ax.plot(x2, y2, label = "|0>'s Probability")
```

```
#ax.axes.xaxis.set_ticks([])
# show a legend on the plot
ax.legend()
```

[13]: <matplotlib.legend.Legend at 0x7f7d205530d0>



<Figure size 576x360 with 0 Axes>

[ ]: