BB8410-300

October 22, 2021

[5]:
```
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:2021-10-22 14:28:13,386: Credentials are
already in use. The existing account in the session will be replaced.
```

[2]:
```
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):'))
################################################################################
print('Enter  1 or 0 bit stream ')
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)
print('Enter  1 for H-bases and 0 for Z-bases ')
##############################################################################
g=int(input())
def random_bases(p):
    key1 = ""
    for i in range(p):
        temp = str(random.randint(g,g))
        key1 += temp
    return(key1)
##############################################################################
Alice_bases= random_bases(data)
Bob_bases= random_bases(data)
print('List of ALICE bases where 1 represent H-bases and 0 represent Z-bases ')
alice = [int(x) for x in str(Alice_bases)]
print(alice)
print('\n')
print('List of BOB bases where 1 represent H-bases and 0 represent Z-bases ')
bob = [int(x) for x in str(Bob_bases)]
print(bob)
##############################################################################
def check_bases(b1,b2):
    check = ''
    matches = 0
    for i in range(len(b1)):
        if b1[i] == b2[i]:
            check += "Y"
            matches += 1
        else:
            check += "X"
    return [check,matches]
print('\n')
print('ALICE and BOB bases matching where X= not matched, Y= matched \n')
check_bases(Alice_bases,Bob_bases)
%matplotlib inline
##############################################################################
bob_bits=[]
#r1 = int(input('Enter the length of Identity Gate Between Quantum Channel: '))
#r1 = int(input('Enter the starting Range of Identity Gate Between Quantum␣
 ↪Channel: '))
#r2 = int(input('Enter the Last range of Identity Gate Between Quantum Channel:␣
 ↪'))
##############################################################################
#def createList(r1, r2):
#        return list(range(r1, r2+15,15))
```

```
# Driver Code
#r1, r2 = 10, 300
#print(createList(r1, r2))
#identitylist=createList(r1, r2)
#print(identitylist)
################################################################################
from random import choice
m=0
for i in range(n):
    m=m+10
    print("No of identity Gate:",m)
    if digits[i] == 1:
        qc= QuantumCircuit(1,1)
        qc.x(0)
        qc.barrier()

        if alice[i]==1:
            qc.h(0)
        if alice[i]==0:
            qc.z(0)
        qc.barrier()
        for j in range(m):
            qc.id(0)
            qc.barrier()
        if bob[i]==1:
            qc.h(0)
        if bob[i]==0:
            qc.z(0)
        qc.barrier()
        qc.measure(list(range(1)),list(range(1)))
        from qiskit.tools.monitor import backend_monitor
        IBMQ.load_account()
        provider = IBMQ.get_provider(hub='ibm-q')
        device = provider.get_backend('ibmq_armonk')       # Enable for Real␣
↪Quantum Device
        #device = Aer.get_backend('qasm_simulator')          # Enable for␣
↪simulator
        job = execute(qc, backend=device, shots=1000)
        print(job.job_id())
        from qiskit.tools.monitor import job_monitor
        job_monitor(job)
        device_result = job.result()
        plot_histogram(device_result.get_counts(qc))
        bits = (device_result.get_counts(qc))
        itemMaxValue = max(bits.items(), key=lambda x : x[1])
        print(itemMaxValue)
        # Iterate over all the items in dictionary to find keys with max value
```

```
        for key, value in bits.items():
            if value == itemMaxValue[1]:
                bob_bits.append(key)

    if digits[i] == 0:
        qc = QuantumCircuit(1,1)
        qc.barrier()
        if alice[i]==1:
            qc.h(0)
        if alice[i]==0:
            qc.z(0)
        qc.barrier()
        for j in range(m):
            qc.id(0)
            qc.barrier()
        if bob[i]==1:
            qc.h(0)
        if bob[i]==0:
            qc.z(0)
        qc.barrier()
        qc.measure(list(range(1)),list(range(1)))
        from qiskit.tools.monitor import backend_monitor
        #IBMQ.load_account()
        provider = IBMQ.get_provider(hub='ibm-q')
        device = provider.get_backend('ibmq_armonk')       # Enable for Real␣
↪Quantum Device
        #device = Aer.get_backend('qasm_simulator')          # Enable for␣
↪simulator
        job = execute(qc, backend=device, shots=1000)
        print(job.job_id())
        from qiskit.tools.monitor import job_monitor
        job_monitor(job)
        device_result = job.result()
        #plot_histogram(device_result.get_counts(qc))
        bits =(device_result.get_counts(qc))
        itemMaxValue = max(bits.items(), key=lambda x : x[1])
        print(itemMaxValue)
        # Iterate over all the items in dictionary to find keys with max value
        for key, value in bits.items():
            if value == itemMaxValue[1]:
                bob_bits.append(key)
    #print(qc)
   #%matplotlib inline
    #qc.draw(output='mpl')
    print("\n")
##############################################################################
def check_bits(b1,b2,bck):
```

```
    check = ''
    for i in range(len(b1)):
        if b1[i] == b2[i] and bck[i] == 'Y':
            check += 'Y'
        elif b1[i] == b2[i] and bck[i] != 'Y':
            check += 'R'
        elif b1[i] != b2[i] and bck[i] == 'Y':
            check += '!'
        elif b1[i] != b2[i] and bck[i] != 'Y':
            check += '-'
    return check
##########################################################################
def transferredbits(b1,b2,bck):
    check = ''
    for i in range(len(b1)):
        if b1[i] == b2[i] and bck[i] == 'Y':
            check += 'Y'
        elif b1[i] != b2[i] and bck[i] == 'Y':
            check += '!'
    return check
##########################################################################
def bob_measurement(qc,b):
    backend = Aer.get_backend('qasm_simulator')
    l = len(b)
    for i in range(l):
        if b[i] == '1':
            qc.h(i)
    qc.measure(list(range(l)),list(range(l)))
    result = execute(qc,backend,shots=1).result()
    bits = list(result.get_counts().keys())[0]

    bits = ''.join(list(reversed(bits)))
    qc.barrier()
    return [qc,bits]
```

```
ENTER LENGTH OF BIT STREAM (example 5 For 10110): 30

Enter  1 or 0 bit stream

 1

List of Bit Stream to transfer over Quantum Channel
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1]


Enter  1 for H-bases and 0 for Z-bases

 1
```

```
List of ALICE bases where 1 represent H-bases and 0 represent Z-bases
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1]


List of BOB bases where 1 represent H-bases and 0 represent Z-bases
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1]


ALICE and BOB bases matching where X= not matched, Y= matched

No of identity Gate: 10

ibmqfactory.load_account:WARNING:2021-10-01 14:34:19,887: Credentials are
already in use. The existing account in the session will be replaced.

61571c6ea8477ce2a57c2ac3
Job Status: job has successfully run
('1', 895)


No of identity Gate: 20

ibmqfactory.load_account:WARNING:2021-10-01 14:43:13,579: Credentials are
already in use. The existing account in the session will be replaced.

61571e84b4ae0bc3ebfb1c41
Job Status: job has successfully run
('1', 888)


No of identity Gate: 30

ibmqfactory.load_account:WARNING:2021-10-01 14:51:45,009: Credentials are
already in use. The existing account in the session will be replaced.

61572083565f94bfe489dd3c
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:00:25,977: Credentials are
already in use. The existing account in the session will be replaced.

('1', 911)


No of identity Gate: 40
6157228c4b16c060a5140f7d
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:02:54,012: Credentials are
already in use. The existing account in the session will be replaced.
```

```
('1', 883)


No of identity Gate: 50
6157232049770e59ae13e26e
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:03:21,347: Credentials are
already in use. The existing account in the session will be replaced.

('1', 909)


No of identity Gate: 60
6157233b73064c72cf6170f2
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:04:15,352: Credentials are
already in use. The existing account in the session will be replaced.

('1', 925)


No of identity Gate: 70
61572372a8477c26147c2ae7
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:04:43,726: Credentials are
already in use. The existing account in the session will be replaced.

('1', 898)


No of identity Gate: 80
6157238eb4ae0bd565fb1c62
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:05:11,571: Credentials are
already in use. The existing account in the session will be replaced.

('1', 905)


No of identity Gate: 90
615723aa565f944cf789dd52
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:05:39,208: Credentials are
already in use. The existing account in the session will be replaced.

('1', 913)
```

```
No of identity Gate: 100
615723c5565f9434ae89dd53
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:06:07,699: Credentials are
already in use. The existing account in the session will be replaced.

('1', 897)


No of identity Gate: 110
615723e273064c4e416170f5
Job Status: job has successfully run
('1', 915)


No of identity Gate: 120

ibmqfactory.load_account:WARNING:2021-10-01 15:19:50,079: Credentials are
already in use. The existing account in the session will be replaced.

61572718a8477c82417c2afe
Job Status: job has successfully run
('1', 899)


No of identity Gate: 130

ibmqfactory.load_account:WARNING:2021-10-01 15:34:18,874: Credentials are
already in use. The existing account in the session will be replaced.

61572a7d565f94af5b89dd83
Job Status: job has successfully run
('1', 911)


No of identity Gate: 140

ibmqfactory.load_account:WARNING:2021-10-01 15:46:47,320: Credentials are
already in use. The existing account in the session will be replaced.

61572d694b16c0b1d4140fda
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:47:16,365: Credentials are
already in use. The existing account in the session will be replaced.

('1', 897)


No of identity Gate: 150
61572d8773064cf537617156
Job Status: job has successfully run
```

```
('1', 888)


No of identity Gate: 160

ibmqfactory.load_account:WARNING:2021-10-01 15:47:44,379: Credentials are
already in use. The existing account in the session will be replaced.

61572da3a8477c21d37c2b47
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:48:12,550: Credentials are
already in use. The existing account in the session will be replaced.

('1', 901)


No of identity Gate: 170
61572dbf8657ae6be955e1ac
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:48:54,040: Credentials are
already in use. The existing account in the session will be replaced.

('1', 882)


No of identity Gate: 180
61572de8059e1b8159a63acc
Job Status: job has successfully run
('1', 910)


No of identity Gate: 190

ibmqfactory.load_account:WARNING:2021-10-01 15:49:27,475: Credentials are
already in use. The existing account in the session will be replaced.

61572e0a8657ae847255e1b3
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:49:55,804: Credentials are
already in use. The existing account in the session will be replaced.

('1', 880)


No of identity Gate: 200
61572e264b16c0684c140fe2
Job Status: job has successfully run
('1', 916)
```

```
No of identity Gate: 210

ibmqfactory.load_account:WARNING:2021-10-01 15:50:21,556: Credentials are
already in use. The existing account in the session will be replaced.

61572e40ceec1b7f466f83f5
Job Status: job has successfully run
('1', 884)


No of identity Gate: 220

ibmqfactory.load_account:WARNING:2021-10-01 15:50:56,291: Credentials are
already in use. The existing account in the session will be replaced.

61572e62b4ae0b744cfb1cc1
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:51:23,776: Credentials are
already in use. The existing account in the session will be replaced.

('1', 897)


No of identity Gate: 230
61572e7e73064cbd77617168
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:51:51,486: Credentials are
already in use. The existing account in the session will be replaced.

('1', 896)


No of identity Gate: 240
61572e9aa8477c11b57c2b50
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:52:31,127: Credentials are
already in use. The existing account in the session will be replaced.

('1', 889)


No of identity Gate: 250
61572ec18657ae7e1f55e1b8
Job Status: job has successfully run
('1', 876)


No of identity Gate: 260
```

```
ibmqfactory.load_account:WARNING:2021-10-01 15:52:59,677: Credentials are
already in use. The existing account in the session will be replaced.

61572edeceec1baa766f83f9
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:53:25,465: Credentials are
already in use. The existing account in the session will be replaced.

('1', 895)


No of identity Gate: 270
61572ef873064c10c961716d
Job Status: job has successfully run
('1', 905)


No of identity Gate: 280

ibmqfactory.load_account:WARNING:2021-10-01 15:53:54,083: Credentials are
already in use. The existing account in the session will be replaced.

61572f14f9a6d9bee898cc52
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:54:21,611: Credentials are
already in use. The existing account in the session will be replaced.

('1', 896)


No of identity Gate: 290
61572f30f9a6d9822398cc55
Job Status: job has successfully run

ibmqfactory.load_account:WARNING:2021-10-01 15:54:49,373: Credentials are
already in use. The existing account in the session will be replaced.

('1', 920)


No of identity Gate: 300
61572f4cceec1bb02f6f83fc
Job Status: job has successfully run
('1', 897)
```

[ ]:
```
0.895, 0.888, 0.900, 0.883, 0.909, 0.925, 0.898, 0.905, 0.903, 0.897, 0.905, 0.
↪899, 0.900, 0.897, 0.888, 0.900, 0.882, 0.900, 0.880, 0.906, 0.884, 0.897, 0.
↪896, 0.889, 0.876, 0.895, 0.905, 0.896, 0.920, 0.897,
```

[9]:
```
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):'))
################################################################################
print('Enter  1 or 0 bit stream ')
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)
print('Enter  1 for H-bases and 0 for Z-bases ')
################################################################################
g=int(input())
def random_bases(p):
    key1 = ""
    for i in range(p):
        temp = str(random.randint(g,g))
        key1 += temp
    return(key1)
################################################################################
Alice_bases= random_bases(data)
Bob_bases= random_bases(data)
print('List of ALICE bases where 1 represent H-bases and 0 represent Z-bases ')
alice = [int(x) for x in str(Alice_bases)]
print(alice)
print('\n')
print('List of BOB bases where 1 represent H-bases and 0 represent Z-bases ')
bob = [int(x) for x in str(Bob_bases)]
print(bob)
################################################################################
def check_bases(b1,b2):
    check = ''
```

```
    matches = 0
    for i in range(len(b1)):
        if b1[i] == b2[i]:
            check += "Y"
            matches += 1
        else:
            check += "X"
    return [check,matches]
print('\n')
print('ALICE and BOB bases matching where X= not matched, Y= matched \n')
check_bases(Alice_bases,Bob_bases)
%matplotlib inline
##############################################################################
bob_bits=[]
#r1 = int(input('Enter the length of Identity Gate Between Quantum Channel: '))
#r1 = int(input('Enter the starting Range of Identity Gate Between Quantum␣
 ↪Channel: '))
#r2 = int(input('Enter the Last range of Identity Gate Between Quantum Channel:␣
 ↪'))
##############################################################################
#def createList(r1, r2):
#        return list(range(r1, r2+15,15))

# Driver Code
#r1, r2 = 10, 300
#print(createList(r1, r2))
#identitylist=createList(r1, r2)
#print(identitylist)
##############################################################################
from random import choice
m=0
for i in range(n):
    m=m+10
    print("No of identity Gate:",m)
    if digits[i] == 1:
        qc= QuantumCircuit(1,1)
        qc.x(0)
        qc.barrier()
        if alice[i]==1:
            qc.h(0)
        if alice[i]==0:
            qc.z(0)
        qc.barrier()
        for j in range(m):
            qc.id(0)
            qc.barrier()
        if bob[i]==1:
```

```
            qc.h(0)
        if bob[i]==0:
            qc.z(0)
        qc.barrier()
        qc.measure(list(range(1)),list(range(1)))
        from qiskit.tools.monitor import backend_monitor
        IBMQ.load_account()
        provider = IBMQ.get_provider(hub='ibm-q')
        device = provider.get_backend('ibmq_armonk')       # Enable for Real␣
↪Quantum Device
        #device = Aer.get_backend('qasm_simulator')          # Enable for␣
↪simulator
        job = execute(qc, backend=device, shots=1000)
        print(job.job_id())
        from qiskit.tools.monitor import job_monitor
        job_monitor(job)
        device_result = job.result()
        plot_histogram(device_result.get_counts(qc))
        bits = (device_result.get_counts(qc))
        itemMaxValue = max(bits.items(), key=lambda x : x[1])
        print(itemMaxValue)
        # Iterate over all the items in dictionary to find keys with max value
        for key, value in bits.items():
            if value == itemMaxValue[1]:
                bob_bits.append(key)

    if digits[i] == 0:
        qc = QuantumCircuit(1,1)
        qc.barrier()
        if alice[i]==1:
            qc.h(0)
        if alice[i]==0:
            qc.z(0)
        qc.barrier()
        for j in range(m):
            qc.id(0)
            qc.barrier()
        if bob[i]==1:
            qc.h(0)
        if bob[i]==0:
            qc.z(0)
        qc.barrier()
        qc.measure(list(range(1)),list(range(1)))
        from qiskit.tools.monitor import backend_monitor
        #IBMQ.load_account()
        provider = IBMQ.get_provider(hub='ibm-q')
```

```
        device = provider.get_backend('ibmq_armonk')        # Enable for Real␣
↪Quantum Device
        #device = Aer.get_backend('qasm_simulator')          # Enable for␣
↪simulator
        job = execute(qc, backend=device, shots=1000)
        print(job.job_id())
        from qiskit.tools.monitor import job_monitor
        job_monitor(job)
        device_result = job.result()
        #plot_histogram(device_result.get_counts(qc))
        bits =(device_result.get_counts(qc))
        itemMaxValue = max(bits.items(), key=lambda x : x[1])
        print(itemMaxValue)
        # Iterate over all the items in dictionary to find keys with max value
        for key, value in bits.items():
            if value == itemMaxValue[1]:
                bob_bits.append(key)
    #print(qc)
   #%matplotlib inline
    #qc.draw(output='mpl')
    print("\n")
##############################################################################
def check_bits(b1,b2,bck):
    check = ''
    for i in range(len(b1)):
        if b1[i] == b2[i] and bck[i] == 'Y':
            check += 'Y'
        elif b1[i] == b2[i] and bck[i] != 'Y':
            check += 'R'
        elif b1[i] != b2[i] and bck[i] == 'Y':
            check += '!'
        elif b1[i] != b2[i] and bck[i] != 'Y':
            check += '-'
    return check
##############################################################################
def transferredbits(b1,b2,bck):
    check = ''
    for i in range(len(b1)):
        if b1[i] == b2[i] and bck[i] == 'Y':
            check += 'Y'
        elif b1[i] != b2[i] and bck[i] == 'Y':
            check += '!'
    return check
##############################################################################
def bob_measurement(qc,b):
    backend = Aer.get_backend('qasm_simulator')
    l = len(b)
```

```
    for i in range(l):
        if b[i] == '1':
            qc.h(i)
    qc.measure(list(range(l)),list(range(l)))
    result = execute(qc,backend,shots=1).result()
    bits = list(result.get_counts().keys())[0]

    bits = ''.join(list(reversed(bits)))
    qc.barrier()
    return [qc,bits]
```

```
ENTER LENGTH OF BIT STREAM (example 5 For 10110): 30

Enter  1 or 0 bit stream

 0

List of Bit Stream to transfer over Quantum Channel
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0]


Enter  1 for H-bases and 0 for Z-bases

 1

List of ALICE bases where 1 represent H-bases and 0 represent Z-bases
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1]


List of BOB bases where 1 represent H-bases and 0 represent Z-bases
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1]


ALICE and BOB bases matching where X= not matched, Y= matched

No of identity Gate: 10
6172d62ebf409e54aba4d984
Job Status: job has successfully run
('0', 896)


No of identity Gate: 20
6172d656d347eafd525b6f30
Job Status: job has successfully run
('0', 914)
```

```
No of identity Gate: 30
6172d689b02bdc5879d840a7
Job Status: job has successfully run
('0', 921)


No of identity Gate: 40
6172d6cab02bdc9d6bd840aa
Job Status: job has successfully run
('0', 924)


No of identity Gate: 50
6172d6fe706f425371432a60
Job Status: job has successfully run
('0', 911)


No of identity Gate: 60
6172d732976ff9c75b81c47d
Job Status: job has successfully run
('0', 922)


No of identity Gate: 70
6172d767731fe68d91bce0ae
Job Status: job has successfully run
('0', 923)


No of identity Gate: 80
6172d79bd347ea64935b6f40
Job Status: job has successfully run
('0', 905)


No of identity Gate: 90
6172d7cf9e07dda059578801
Job Status: job has successfully run
('0', 906)


No of identity Gate: 100
6172d809bf409e39d3a4d9a5
Job Status: job has successfully run
('0', 912)
```

```
No of identity Gate: 110
6172d83fd347eac9a95b6f4d
Job Status: job has successfully run
('0', 925)


No of identity Gate: 120
6172d8c8b02bdc1e58d840ca
Job Status: job has successfully run
('0', 907)


No of identity Gate: 130
6172d8fc4829a096011614d0
Job Status: job has successfully run
('0', 910)


No of identity Gate: 140
6172d96d4829a0cd461614d8
Job Status: job has successfully run
('0', 899)


No of identity Gate: 150
6172d9fd731fe6001dbce0d0
Job Status: job has successfully run
('0', 899)


No of identity Gate: 160
6172daca706f425dba432a96
Job Status: job has successfully run
('0', 922)


No of identity Gate: 170
6172db5a6ab9409fc510dccb
Job Status: job has successfully run
('0', 910)


No of identity Gate: 180
6172dbcc6ab940317510dcce
Job Status: job has successfully run
('0', 904)
```

```
No of identity Gate: 190
6172dc3fd347ea870b5b6f7d
Job Status: job has successfully run
('0', 935)


No of identity Gate: 200
6172dcc7731fe66794bce0ec
Job Status: job has successfully run
('0', 927)


No of identity Gate: 210
6172dd52976ff96f8a81c4d6
Job Status: job has successfully run
('0', 908)


No of identity Gate: 220
6172de2d6ab940e17110dce7
Job Status: job has successfully run
('0', 911)


No of identity Gate: 230
6172defa6ab940b62010dced
Job Status: job has successfully run
('0', 927)


No of identity Gate: 240
6172dfcc976ff9c26181c4ec
Job Status: job has successfully run
('0', 911)


No of identity Gate: 250
6172e0bd4829a03686161517
Job Status: job has successfully run
('0', 916)


No of identity Gate: 260
6172e1fab02bdc0fb4d84127
Job Status: job has successfully run
('0', 928)
```

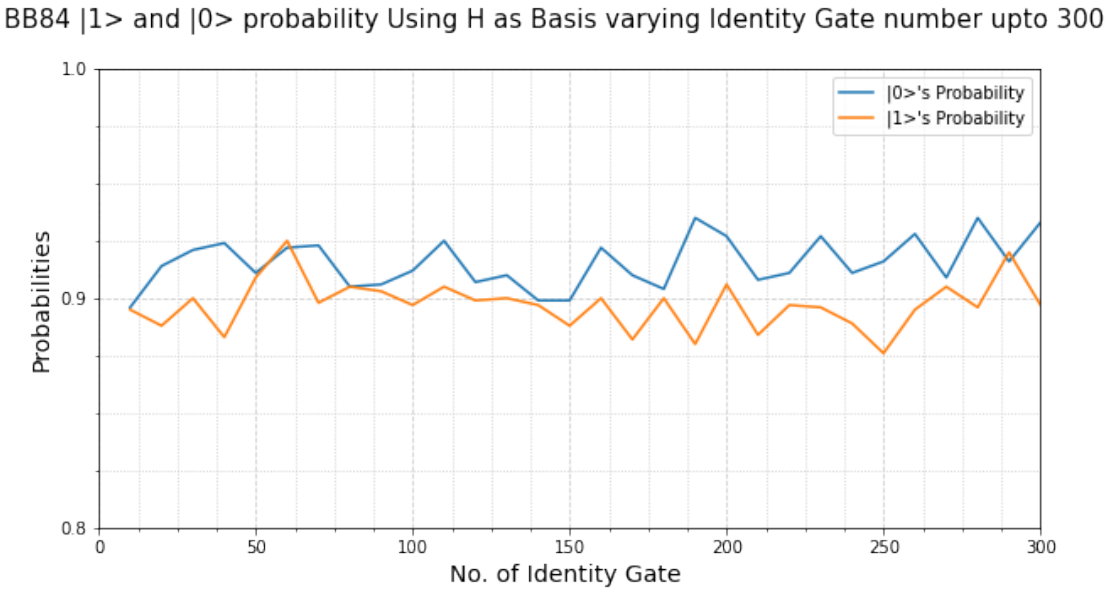
```
No of identity Gate: 270
6172e31f706f427165432ae1
Job Status: job has successfully run
('0', 909)


No of identity Gate: 280
6172e3ebbf409e1a81a4da12
Job Status: job has successfully run
('0', 935)


No of identity Gate: 290
6172e4259e07dd7ecc57886f
Job Status: job has successfully run
('0', 916)


No of identity Gate: 300
6172e462bf409e17b4a4da16
Job Status: job has successfully run
('0', 933)
```

[ ]:
```
[0.896, 0.914, 0.921, 0.924, 0.911, 0.922, 0.923, 0.905, 0.906, 0.912, 0.925, 0.
↪907, 0.910, 0.899, 0.899, 0.922, 0.910, 0.904, 0.935, 0.927, 0.908, 0.911, 0.
↪927, 0.911, 0.916, 0.928, 0.909, 0.935, 0.916, 0.933]
```

[10]:
```
import matplotlib.pyplot as plt
from matplotlib.ticker import (AutoMinorLocator, MultipleLocator)

fig, ax = plt.subplots(figsize=(10, 5))
fig.suptitle('BB84 |1> and |0> probability Using H as Basis varying Identity␣
↪Gate number upto 300',fontsize=15)

# naming the x axis
plt.xlabel('No. of Identity Gate ',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, 300)
ax.set_ylim(0.8, 1)
# Change major ticks to show every 20.
```

```
ax.xaxis.set_major_locator(MultipleLocator(50))
ax.yaxis.set_major_locator(MultipleLocator(0.1))

# Change minor ticks to show every 5. (20/4 = 5)
ax.xaxis.set_minor_locator(AutoMinorLocator(4))
ax.yaxis.set_minor_locator(AutoMinorLocator(4))

# Turn grid on for both major and minor ticks and style minor slightly
# differently.
ax.grid(which='major', color='#CCCCCC', linestyle='--')
ax.grid(which='minor', color='#CCCCCC', linestyle=':')

fig = plt.figure(figsize=(8,5))
# line 1  for one probability points
y1 = [0.896, 0.914, 0.921, 0.924, 0.911, 0.922, 0.923, 0.905, 0.906, 0.912, 0.
 ↪925, 0.907, 0.910, 0.899, 0.899, 0.922, 0.910, 0.904, 0.935, 0.927, 0.908, 0.
 ↪911, 0.927, 0.911, 0.916, 0.928, 0.909, 0.935, 0.916, 0.933]
x1 =␣
 ↪[10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250
ax.plot(x1, y1, label = "|0>'s Probability")

# plotting the line 1 points
#ax.plot(x1, y1, label = "|1>'s Probability",  color='red')

# line 2 points
y2 = [0.895, 0.888, 0.900, 0.883, 0.909, 0.925, 0.898, 0.905, 0.903, 0.897, 0.
 ↪905, 0.899, 0.900, 0.897, 0.888, 0.900, 0.882, 0.900, 0.880, 0.906, 0.884, 0.
 ↪897, 0.896, 0.889, 0.876, 0.895, 0.905, 0.896, 0.920, 0.897]
x2 = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160,␣
 ↪170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300]
ax.plot(x2, y2, label = "|1>'s Probability")


# show a legend on the plot
ax.legend()
```

[10]: <matplotlib.legend.Legend at 0x7f2b283c7640>




```
<Figure size 576x360 with 0 Axes>
```

[ ]: