1. AND GATE

code:

from qiskit import \*

def And(inp1, inp2):

qc = QuantumCircuit(3,1)

if(inp1 == '1'):

qc.x(0)

if(inp2 == '1'):

qc.x(1)

qc.barrier()

qc.ccx(0,1,2)

qc.measure(2,0)

qc.draw()

backend = Aer.get\_backend('qasm\_simulator')

job = execute(qc, backend, shots = 1, memory= True)

output = job.result().get\_memory()[0]

return qc, output

for inp1 in ['0', '1']:

for inp2 in ['0','1']:

qc, out = And(inp1, inp2)

print('And gate input is', inp1, inp2, 'output is ', out)

display(qc.draw())

print('/n')

And gate input is 0 0 output is 0

░

q\_0: ─░───■─────

░ │

q\_1: ─░───■─────

░ ┌─┴─┐┌─┐

q\_2: ─░─┤ X ├┤M├

░ └───┘└╥┘

c: 1/═════════╩═

0

/n

And gate input is 0 1 output is 0

░

q\_0: ──────░───■─────

┌───┐ ░ │

q\_1: ┤ X ├─░───■─────

└───┘ ░ ┌─┴─┐┌─┐

q\_2: ──────░─┤ X ├┤M├

░ └───┘└╥┘

c: 1/══════════════╩═

0

/n

And gate input is 1 0 output is 0

┌───┐ ░

q\_0: ┤ X ├─░───■─────

└───┘ ░ │

q\_1: ──────░───■─────

░ ┌─┴─┐┌─┐

q\_2: ──────░─┤ X ├┤M├

░ └───┘└╥┘

c: 1/══════════════╩═

0

/n

And gate input is 1 1 output is 1

┌───┐ ░

q\_0: ┤ X ├─░───■─────

├───┤ ░ │

q\_1: ┤ X ├─░───■─────

└───┘ ░ ┌─┴─┐┌─┐

q\_2: ──────░─┤ X ├┤M├

░ └───┘└╥┘

c: 1/══════════════╩═

0

/n

probability of the state getting the expected output value by IMBQ-belem is

6078128326eae171236f90f9

Job Status: job has successfully run

Probability of correct answer for inputs 0 0

0.87

---------------------------------

607812c14638592368b63466

Job Status: job has successfully run

Probability of correct answer for inputs 0 1

0.88

---------------------------------

6078131626eae1c88b6f9100

Job Status: job has successfully run

Probability of correct answer for inputs 1 0

0.90

---------------------------------

607813d465fc992c5cd7f998

Job Status: job has successfully run

Probability of correct answer for inputs 1 1

0.64

---------------------------------

The highest of these probabilities was 0.90

The lowest of these probabilities was 0.64

Transpiled AND gate circuit for ibmq\_athens with input 0 0

The circuit depth : 28

# of nonlocal gates : 19

Probability of correct answer : 0.87

2. NOT GATE

from qiskit import \*

from qiskit.visualization import plot\_histogram

import numpy as np

def Not(inp):

qc = QuantumCircuit(1,1)

if(inp == '1'):

qc.x(0)

qc.barrier()

qc.x(0)

qc.barrier()

qc.measure(0,0)

qc.draw()

backend = Aer.get\_backend('qasm\_simulator')

job = execute(qc, backend, shot = 1, memory= True)

output = job.result().get\_memory()[0]

return qc, output

for inp in ['0','1']:

qc, out = Not(inp)

print('not input is ',inp , 'output is', out)

display(qc.draw())

print('/n')

not input is 0 output is 1

░ ┌───┐ ░ ┌─┐

q\_0: ─░─┤ X ├─░─┤M├

░ └───┘ ░ └╥┘

c: 1/════════════╩═

0

/n

not input is 1 output is 0

┌───┐ ░ ┌───┐ ░ ┌─┐

q\_0: ┤ X ├─░─┤ X ├─░─┤M├

└───┘ ░ └───┘ ░ └╥┘

c: 1/═════════════════╩═

0

/n

Similarly I done or gate , nand gate, nor gate, xor gates. Here are the some files for it

1 Bit Logic gates

from qiskit import \*

def Not(inp):

print('Not gate')

qc = QuantumCircuit(1,1)

if(inp == '1'):

qc.x(0)

qc.barrier()

qc.x(0)

qc.barrier()

qc.measure(0,0)

qc.draw()

backend = Aer.get\_backend('qasm\_simulator')

job = execute(qc, backend, shot = 1, memory= True)

output = job.result().get\_memory()[0]

return qc, output

def And(inp1, inp2):

print('And gate')

qc = QuantumCircuit(3,1)

if(inp1 == '1'):

qc.x(0)

if(inp2 == '1'):

qc.x(1)

qc.barrier()

qc.ccx(0,1,2)

qc.measure(2,0)

qc.draw()

backend = Aer.get\_backend('qasm\_simulator')

job = execute(qc, backend, shots = 1, memory= True)

output = job.result().get\_memory()[0]

return qc, output

def Or(inp1, inp2):

print('Or gate')

qc = QuantumCircuit(3, 1)

qc.reset(range(3))

if inp1 == '1':

qc.x(0)

if inp2 == '1':

qc.x(1)

qc.barrier()

qc.x(1)

qc.cswap(0,1,2)

qc.x(1)

qc.measure(1, 0)

backend = Aer.get\_backend('qasm\_simulator')

job = execute(qc, backend, shots = 1, memory = True)

output = job.result().get\_memory()[0]

return qc, output

def Nor(inp1, inp2):

print('Nor gate')

qc = QuantumCircuit(3, 1)

qc.reset(range(3))

if inp1 == '1':

qc.x(0)

if inp2 == '1':

qc.x(1)

qc.barrier()

qc.x(1)

qc.cswap(0,1,2)

qc.measure(1, 0)

backend = Aer.get\_backend('qasm\_simulator')

job = execute(qc, backend, shots = 1, memory = True)

output = job.result().get\_memory()[0]

return qc, output

def Nand(inp1, inp2):

print('Nand gate')

qc = QuantumCircuit(3,1)

if(inp1 == '1'):

qc.x(0)

if(inp2 == '1'):

qc.x(1)

qc.barrier()

qc.ccx(0,1,2)

qc.x(2)

qc.measure(2,0)

qc.draw()

backend = Aer.get\_backend('qasm\_simulator')

job = execute(qc, backend, shots = 1, memory= True)

output = job.result().get\_memory()[0]

return qc, output

def Xor(inp1, inp2):

print('And gate')

qc = QuantumCircuit(2,1)

if(inp1 == '1'):

qc.x(0)

if(inp2 == '1'):

qc.x(1)

qc.barrier()

qc.cx(0,1)

qc.measure(1,0)

qc.draw()

backend = Aer.get\_backend('qasm\_simulator')

job = execute(qc, backend, shots = 1, memory= True)

output = job.result().get\_memory()[0]

return qc, output

def All(num, inp1, inp2):

if(num == 0):

return Not(inp1)

elif(num == 1):

return And(inp1, inp2)

elif(num == 2):

return Or(inp1, inp2)

elif(num == 3):

return Nor(inp1, inp2)

elif(num == 4):

return Nand(inp1, inp2)

elif(num == 5):

return Xor(inp1, inp2)

else:

return print('invalid format')

import numpy as np

sim = Aer.get\_backend('qasm\_simulator')

value = int(input('Enter a number'))

shots = 2\*\*14

A = 1.47e-6

E\_sim = []

for num in [value]:

for inp1 in ['0', '1']:

for inp2 in ['0', '1']:

qc, output = All(num, inp1, inp2)

print('input to gate is ', inp1, inp2, 'output is', output)

display(qc.draw())

Energy\_meas = []

counts = execute(qc, sim, shots = shots).result().get\_counts()

probs = {}

for output in ['0','1']:

if output in counts:

probs[output] = counts[output]/shots

else:

probs[output] = 0

Energy\_meas.append(probs['0'] - probs['1'])

print('/n')

E\_sim.append(A \* np.sum(np.array(Energy\_meas)))

print('Energy expection value of the state 00 : {:.3e} eV'.format(E\_sim[0]))

print('Energy expection value of the state 01 : {:.3e} eV'.format(E\_sim[1]))

print('Energy expection value of the state 10 : {:.3e} eV'.format(E\_sim[2]))

print('Energy expection value of the state 11 : {:.3e} eV'.format(E\_sim[3]))

hbar, c = 4.1357e-15, 3e10

# energy difference between the triplets and singlet

E\_del = abs(E\_sim[0] - E\_sim[3])

# frequency associated with the energy difference

f = E\_del/hbar

# convert frequency to wavelength in (cm)

wavelength = c/f

print('The wavelength of the radiation from the transition\

in the hyperfine structure is : {:.1f} cm'.format(wavelength))

output:

Enter a number 2

Or gate

input to gate is 0 0 output is 0

░

q\_0: ─|0>──░───────■─────────

░ ┌───┐ │ ┌───┐┌─┐

q\_1: ─|0>──░─┤ X ├─X─┤ X ├┤M├

░ └───┘ │ └───┘└╥┘

q\_2: ─|0>──░───────X───────╫─

░ ║

c: 1/══════════════════════╩═

0

/n

Or gate

input to gate is 0 1 output is 1

░

q\_0: ─|0>───────░───────■─────────

┌───┐ ░ ┌───┐ │ ┌───┐┌─┐

q\_1: ─|0>─┤ X ├─░─┤ X ├─X─┤ X ├┤M├

└───┘ ░ └───┘ │ └───┘└╥┘

q\_2: ─|0>───────░───────X───────╫─

░ ║

c: 1/═══════════════════════════╩═

0

/n

Or gate

input to gate is 1 0 output is 1

┌───┐ ░

q\_0: ─|0>─┤ X ├─░───────■─────────

└───┘ ░ ┌───┐ │ ┌───┐┌─┐

q\_1: ─|0>───────░─┤ X ├─X─┤ X ├┤M├

░ └───┘ │ └───┘└╥┘

q\_2: ─|0>───────░───────X───────╫─

░ ║

c: 1/═══════════════════════════╩═

0

/n

Or gate

input to gate is 1 1 output is 1

┌───┐ ░

q\_0: ─|0>─┤ X ├─░───────■─────────

├───┤ ░ ┌───┐ │ ┌───┐┌─┐

q\_1: ─|0>─┤ X ├─░─┤ X ├─X─┤ X ├┤M├

└───┘ ░ └───┘ │ └───┘└╥┘

q\_2: ─|0>───────░───────X───────╫─

░ ║

c: 1/═══════════════════════════╩═

0

/n

Energy expection value of the state 00 : 1.470e-06 eV

Energy expection value of the state 01 : -1.470e-06 eV

Energy expection value of the state 10 : -1.470e-06 eV

Energy expection value of the state 11 : -1.470e-06 eV

The wavelength of the radiation from the transition in the hyperfine structure is : 42.2 cm

Energy expectation is the amount of energy being disspated while outputting the states