Group 22

June 6, 2021

[26]:
```
#initialization
import matplotlib.pyplot as plt
import numpy as np

# importing Qiskit
from qiskit import execute, transpile
from qiskit import IBMQ, Aer, assemble, transpile
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit.providers.ibmq import least_busy

# import basic plot tools
from qiskit.visualization import plot_histogram
```

0.1 1. Algorithm

[27]:
```
w = 22 % 8
print(w)
```

```
6
```

[28]:
```
wb = bin(w)[2:]

print(wb)
```

```
110
```

[29]:
```
x= len(wb)

print('number of qubits: ', x)
```

```
number of qubits:  3
```

[30]:
```
def initialize_s(qc, qubits):
    """Apply a H-gate to 'qubits' in qc"""
    for q in qubits:
        qc.h(q)
    return qc
```

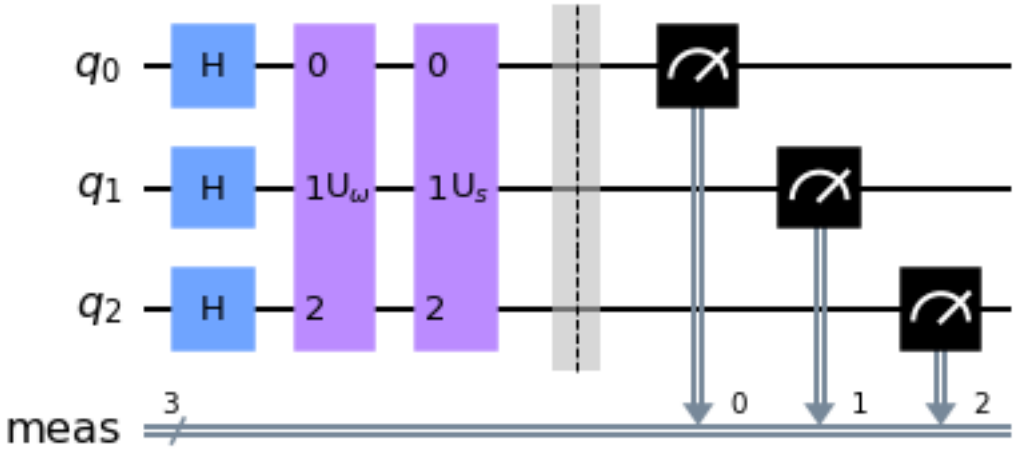
[31]:
```
def diffuser(nqubits):
    qc = QuantumCircuit(nqubits)
    # Apply transformation |s> -> |00..0> (H-gates)
    for qubit in range(nqubits):
        qc.h(qubit)
    # Apply transformation |00..0> -> |11..1> (X-gates)
    for qubit in range(nqubits):
        qc.x(qubit)
    # Do multi-controlled-Z gate
    qc.h(nqubits-1)
    qc.mct(list(range(nqubits-1)), nqubits-1)  # multi-controlled-toffoli
    qc.h(nqubits-1)
    # Apply transformation |11..1> -> |00..0>
    for qubit in range(nqubits):
        qc.x(qubit)
    # Apply transformation |00..0> -> |s>
    for qubit in range(nqubits):
        qc.h(qubit)
    # We will return the diffuser as a gate
    U_s = qc.to_gate()
    U_s.name = "U$_s$"
    return U_s
```

[32]:
```
qc = QuantumCircuit(3)
#change in phase
qc.h(2)
qc.ccx(0, 1, 2)
qc.h(2)

qc.x(0)
oracle_ex3 = qc.to_gate()
oracle_ex3.name = "U$_\omega$"
```

[33]:
```
n = 3
grover_circuit = QuantumCircuit(n)
grover_circuit = initialize_s(grover_circuit, [0,1,2])
grover_circuit.append(oracle_ex3, [0,1,2])
grover_circuit.append(diffuser(n), [0,1,2])
grover_circuit.measure_all()
grover_circuit.draw(output='mpl')
```

[33]:



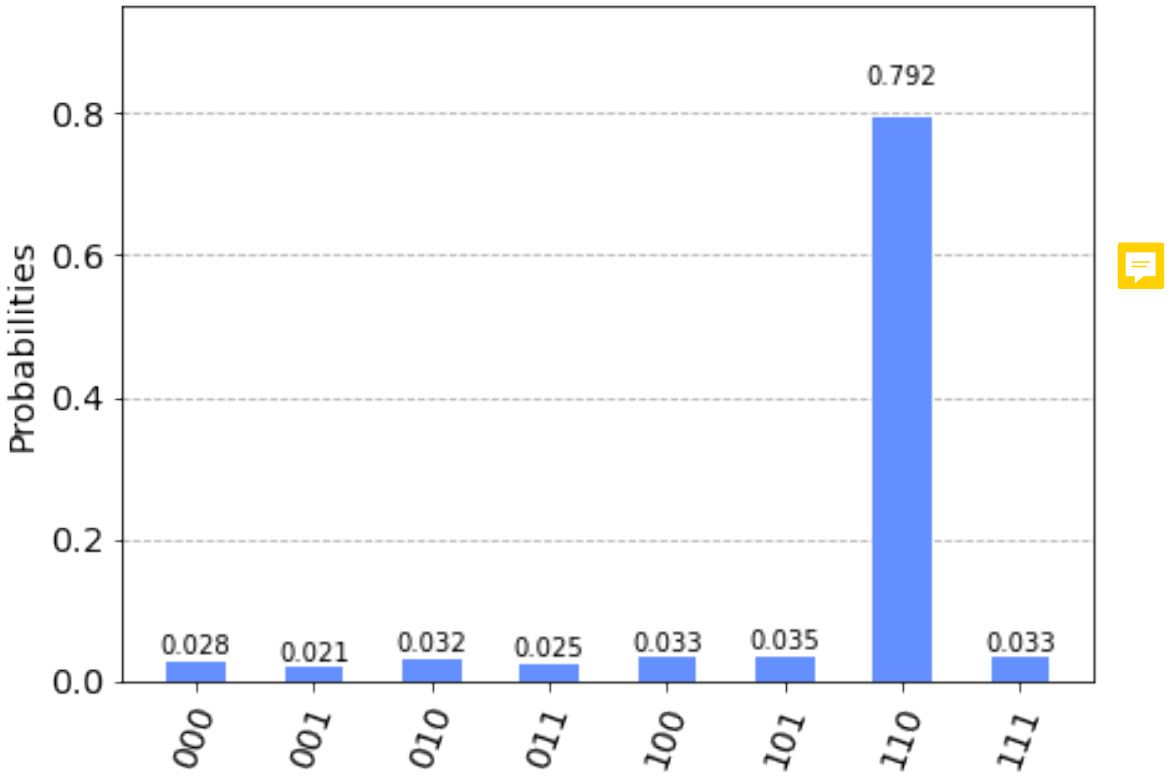
[34]:
```
import math as m

times= round(m.sqrt(2**x))
print(times)
```

3

[74]:
```
backend = Aer.get_backend("qasm_simulator")
shots=1024
result = execute(grover_circuit, backend, shots=shots).result()
counts_sim = result.get_counts(grover_circuit)
plot_histogram(counts_sim)
```

[74]:



[73]:
```
grover_circuit.depth()
```

[73]: 4

0.2 2. Simulation with noise

[55]:
```
%qiskit_backend_overview
```

```
VBox(children=(HTML(value="<h2 style ='color:#ffffff; background-color:#000000;
↪padding-top: 1%; padding-bottom...
```

[208]:
```
my_provider_ibmq = IBMQ.get_provider(hub='ibm-q', group='open', project='main')

# Define backend
backend_device = my_provider_ibmq.get_backend('ibmq_manila')

# See backend information
backend_device
```

```
VBox(children=(HTML(value="<h1 style='color:#ffffff;background-color:#000000;
↪padding-top: 1%;padding-bottom: 1...
```

[208]:
```
<IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open',
project='main')>
```

In these devices the connections between the qubits are not arbitrary.

Trying to connect qubits that are not directly linked in the devices arquitecture, increases the circuit size, and consequently, increase the probability of errors.

[190]:
```
coupling_map = backend_device.configuration().coupling_map
```

[191]:
```
from qiskit.providers.aer.noise import NoiseModel
```

[192]:
```
# Construct the noise model from backend properties
noise_model = NoiseModel.from_backend(backend_device)
print(noise_model)
```

```
NoiseModel:
  Basis gates: ['cx', 'id', 'rz', 'sx', 'x']
  Instructions with noise: ['x', 'sx', 'cx', 'measure', 'id']
  Qubits with noise: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
  Specific qubit errors: [('id', [0]), ('id', [1]), ('id', [2]), ('id', [3]),
('id', [4]), ('id', [5]), ('id', [6]), ('id', [7]), ('id', [8]), ('id', [9]),
('id', [10]), ('id', [11]), ('id', [12]), ('id', [13]), ('id', [14]), ('sx',
[0]), ('sx', [1]), ('sx', [2]), ('sx', [3]), ('sx', [4]), ('sx', [5]), ('sx',
[6]), ('sx', [7]), ('sx', [8]), ('sx', [9]), ('sx', [10]), ('sx', [11]), ('sx',
[12]), ('sx', [13]), ('sx', [14]), ('x', [0]), ('x', [1]), ('x', [2]), ('x',
[3]), ('x', [4]), ('x', [5]), ('x', [6]), ('x', [7]), ('x', [8]), ('x', [9]),
('x', [10]), ('x', [11]), ('x', [12]), ('x', [13]), ('x', [14]), ('cx', [14,
0]), ('cx', [0, 14]), ('cx', [14, 13]), ('cx', [13, 14]), ('cx', [6, 8]), ('cx',
[8, 6]), ('cx', [5, 9]), ('cx', [9, 5]), ('cx', [4, 10]), ('cx', [10, 4]),
('cx', [11, 3]), ('cx', [3, 11]), ('cx', [12, 2]), ('cx', [2, 12]), ('cx', [13,
1]), ('cx', [1, 13]), ('cx', [13, 12]), ('cx', [12, 13]), ('cx', [11, 12]),
('cx', [12, 11]), ('cx', [10, 11]), ('cx', [11, 10]), ('cx', [9, 10]), ('cx',
[10, 9]), ('cx', [9, 8]), ('cx', [8, 9]), ('cx', [7, 8]), ('cx', [8, 7]), ('cx',
[5, 6]), ('cx', [6, 5]), ('cx', [5, 4]), ('cx', [4, 5]), ('cx', [4, 3]), ('cx',
[3, 4]), ('cx', [2, 3]), ('cx', [3, 2]), ('cx', [1, 2]), ('cx', [2, 1]), ('cx',
[1, 0]), ('cx', [0, 1]), ('measure', [0]), ('measure', [1]), ('measure', [2]),
('measure', [3]), ('measure', [4]), ('measure', [5]), ('measure', [6]),
('measure', [7]), ('measure', [8]), ('measure', [9]), ('measure', [10]),
('measure', [11]), ('measure', [12]), ('measure', [13]), ('measure', [14])]
```

[193]:
```
# Get the basis gates for the noise model
basis_gates = noise_model.basis_gates
print(basis_gates)
```

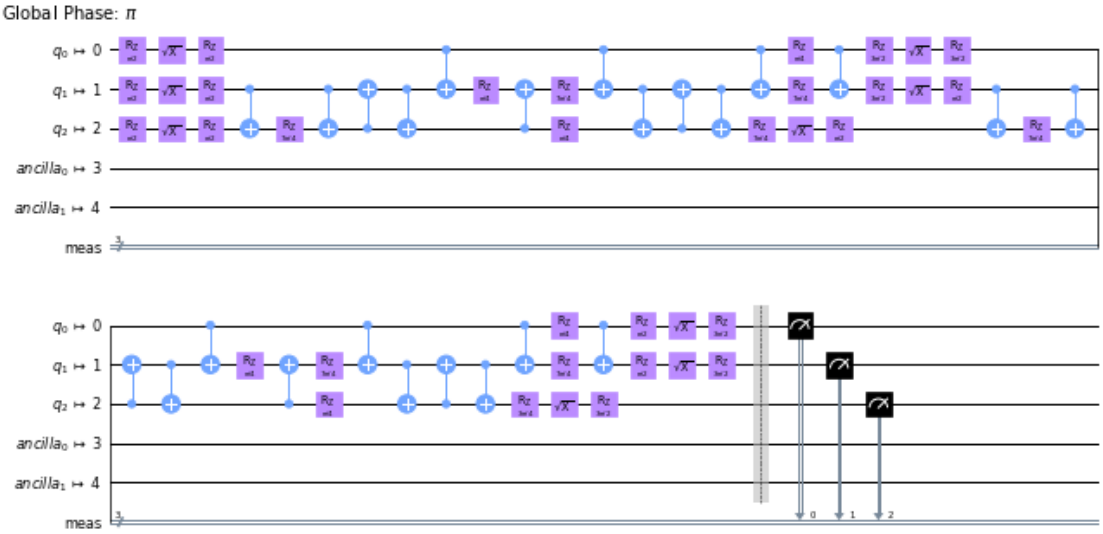
```
['cx', 'id', 'rz', 'sx', 'x']
```

[314]:
```
from qiskit.compiler import transpile
```

```
qc_t_real = transpile(grover_circuit, backend=backend_device)

#qc_t_real.draw(output='mpl', scale=0.5)
```

[314]:



[229]:
```
grover_circuit.depth()
```

[229]: 4

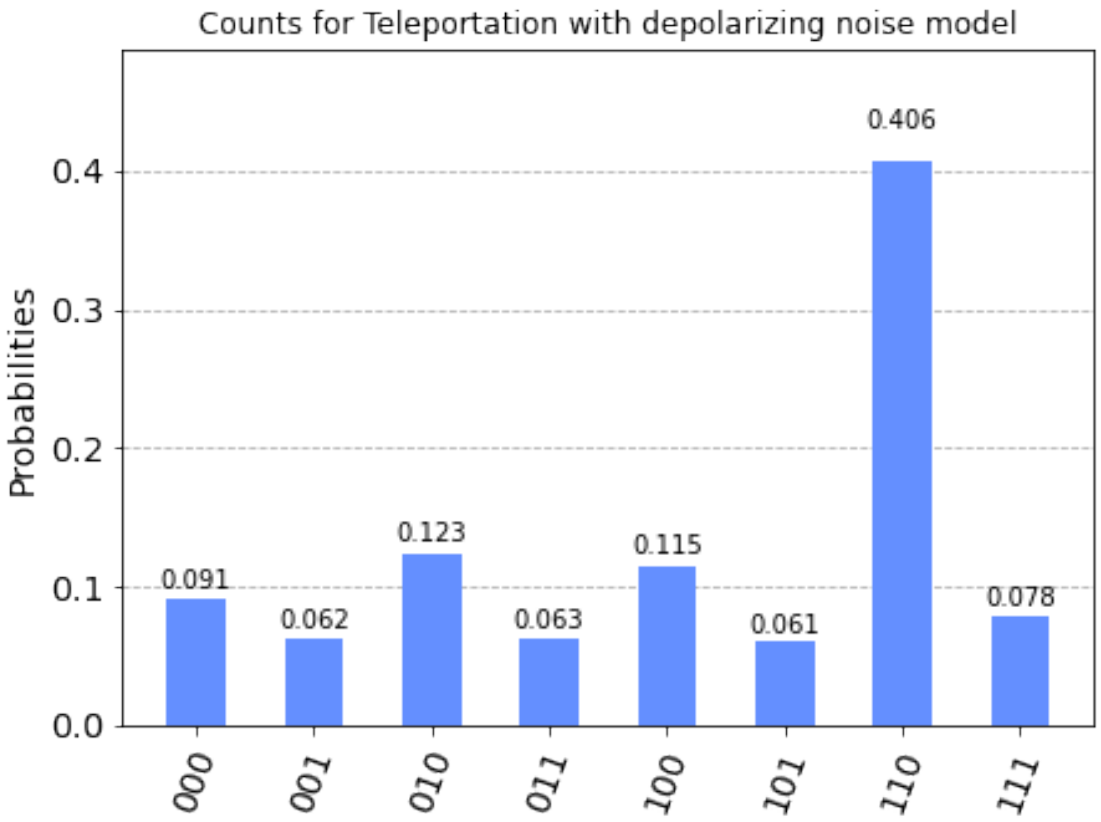
[230]:
```
qc_t_real.depth()
```

[230]: 42

[307]:
```
# Execute noisy simulation and get counts
result_noise = execute(qc_t_real, backend,
                       noise_model=noise_model,
                       coupling_map=coupling_map,
                       basis_gates=basis_gates).result()

counts_noise = result_noise.get_counts(qc_t_real)
plot_histogram(counts_noise, title="Counts for Teleportation with depolarizing␣
 ↪noise model")
```

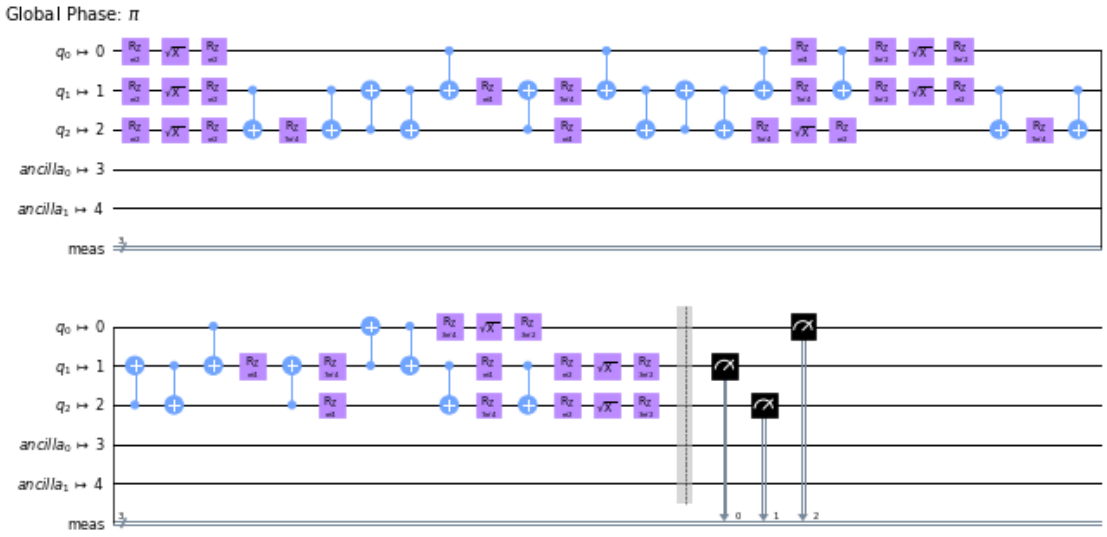
[307]:



0.2.1 Level 1

[313]:
```
qc_optimized = transpile(grover_circuit, backend=backend_device,␣
↪optimization_level=1)
#qc_optimized.draw(output='mpl', scale=0.5)
```

[313]:



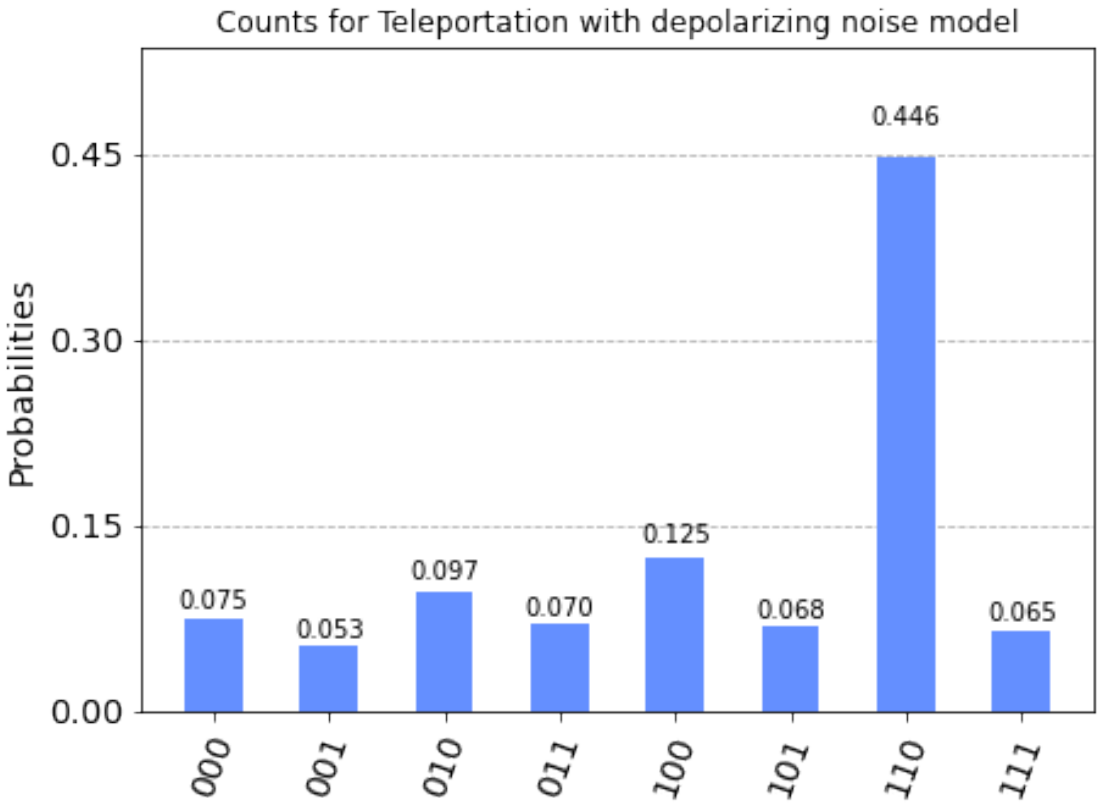
[305]:
```
qc_optimized.depth()
```

[305]: 40

[306]:
```
# Execute noisy simulation and get counts
result_noise = execute(qc_optimized, backend,
                       noise_model=noise_model,
                       coupling_map=coupling_map,
                       basis_gates=basis_gates).result()

counts_noise = result_noise.get_counts(qc_optimized)
plot_histogram(counts_noise, title="Counts for Teleportation with depolarizing␣
 ↪noise model")
```

[306]:



0.2.2 Level 2

[280]:
```
qc_optimized = transpile(grover_circuit, backend=backend_device,␣
 ↪optimization_level=2)
#qc_optimized.draw(output='mpl', scale=0.5)
```

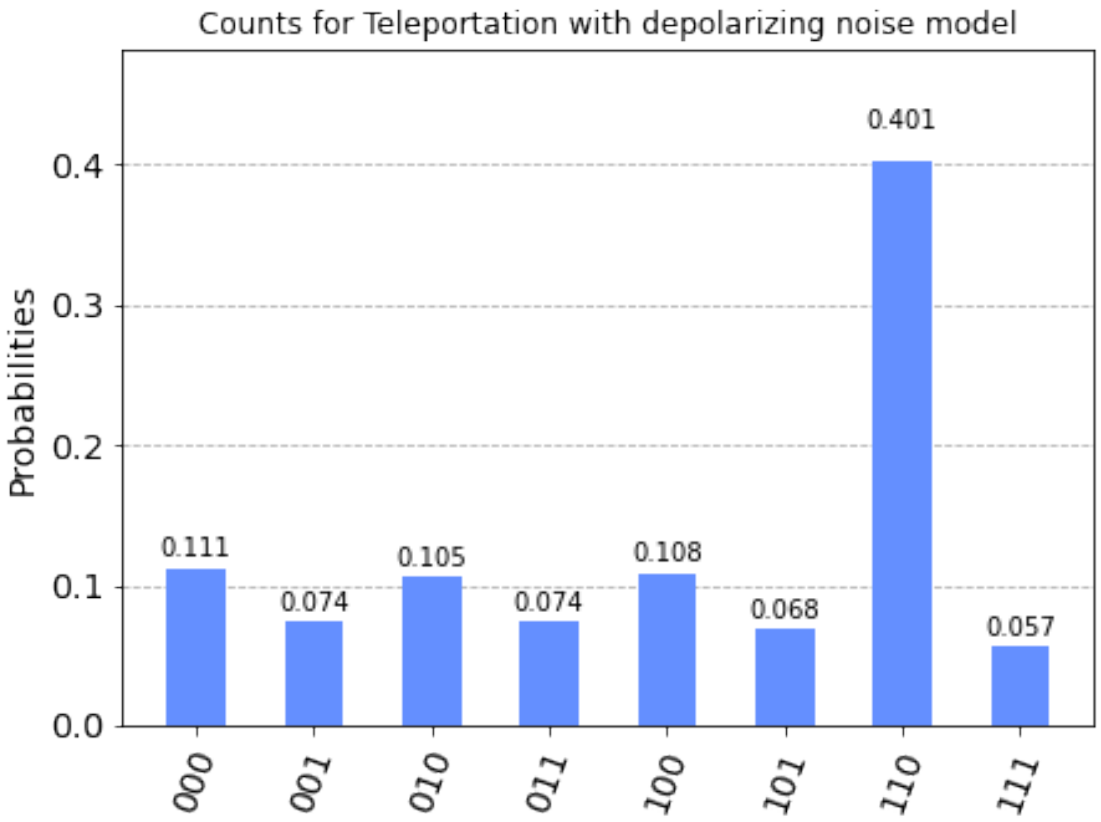
[281]:
```
qc_optimized.depth()
```

[281]: 45

[284]:
```
# Execute noisy simulation and get counts
result_noise = execute(qc_optimized, backend,
                       noise_model=noise_model,
                       coupling_map=coupling_map,
                       basis_gates=basis_gates).result()

counts_noise = result_noise.get_counts(qc_optimized)
plot_histogram(counts_noise, title="Counts for Teleportation with depolarizing␣
 ↪noise model")
```

[284]:



0.2.3 Level 3

[285]:
```
qc_optimized = transpile(grover_circuit, backend=backend_device,␣
 ↪optimization_level=3)
#qc_optimized.draw(output='mpl', scale=0.5)
```

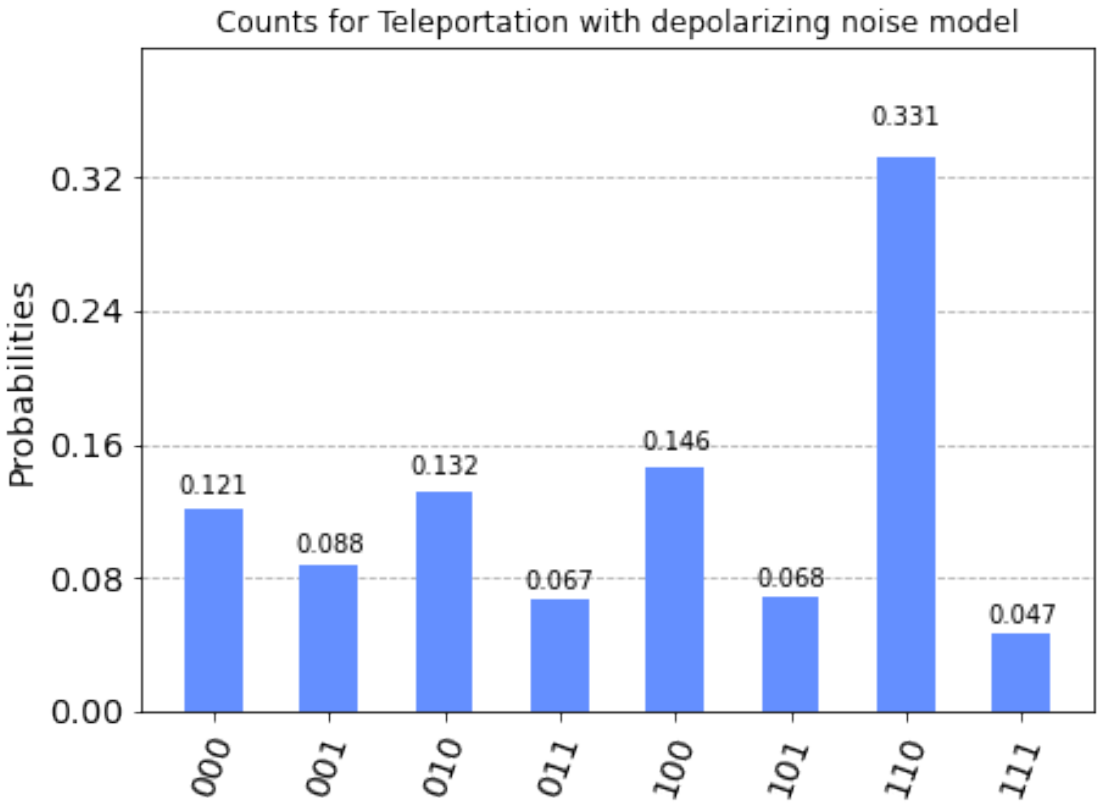
[286]:
```
qc_optimized.depth()
```

[286]: 55

[287]:
```
# Execute noisy simulation and get counts
result_noise = execute(qc_optimized, backend,
                       noise_model=noise_model,
                       coupling_map=coupling_map,
                       basis_gates=basis_gates).result()

counts_noise = result_noise.get_counts(qc_optimized)
plot_histogram(counts_noise, title="Counts for Teleportation with depolarizing␣
 ↪noise model")
```

[287]:



Optimization level 1 seems to be the best one in this machine

0.3 3. Run in real machine

[37]:
```
provider = IBMQ.load_account()
provider.backends()
```

```
ibmqfactory.load_account:WARNING:2021-06-06 14:58:33,333: Credentials are
already in use. The existing account in the session will be replaced.
```

[37]:
```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_armonk') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_athens') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_santiago') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_lima') from IBMQ(hub='ibm-q', group='open',
```

```
project='main')>,
 <IBMQBackend('ibmq_belem') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_quito') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQSimulator('simulator_statevector') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQSimulator('simulator_mps') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQSimulator('simulator_extended_stabilizer') from IBMQ(hub='ibm-q',
group='open', project='main')>,
 <IBMQSimulator('simulator_stabilizer') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open',
project='main')>]
```

[38]:
```
# Backend overview
import qiskit.tools.jupyter

%qiskit_backend_overview
```

```
VBox(children=(HTML(value="<h2 style ='color:#ffffff; background-color:#000000;
 ↪padding-top: 1%; padding-bottom...
```

[39]:
```
from qiskit.tools.monitor import backend_overview, backend_monitor

backend_overview()
```

```
ibmq_manila
-----------
Num. Qubits:  5
Pending Jobs: 12
Least busy:   False
Operational:  True
Avg. T1:      163.8
Avg. T2:      68.2

ibmq_quito
----------
Num. Qubits:  5
Pending Jobs: 17
Least busy:   False
Operational:  True
Avg. T1:      80.3
Avg. T2:      71.2

ibmq_belem
----------
Num. Qubits:  5
Pending Jobs: 9
Least busy:   False
Operational:  True
Avg. T1:      75.9
Avg. T2:      75.6


ibmq_lima
---------
Num. Qubits:  5
Pending Jobs: 0
Least busy:   True
Operational:  True
Avg. T1:      62.4
Avg. T2:      50.6

ibmq_santiago
-------------
Num. Qubits:  5
Pending Jobs: 8
Least busy:   False
Operational:  True
Avg. T1:      110.7
Avg. T2:      100.7

ibmq_athens
-----------
Num. Qubits:  5
Pending Jobs: 2
Least busy:   False
Operational:  True
Avg. T1:      96.0
Avg. T2:      104.1
```

```
ibmq_armonk                ibmq_16_melbourne          ibmqx2
-----------                -----------------          ------
Num. Qubits:  1            Num. Qubits:  15           Num. Qubits:  5
Pending Jobs: 1            Pending Jobs: 18           Pending Jobs: 6
Least busy:   False        Least busy:   False        Least busy:   False
Operational:  True         Operational:  True         Operational:  True
Avg. T1:      169.7        Avg. T1:      54.6         Avg. T1:      58.6
Avg. T2:      261.0        Avg. T2:      53.1         Avg. T2:      35.8
```

[308]:
```
print("Running on: ", backend_device)
```

```
Running on:  ibmq_manila
```

[309]:
```
# See backend information
backend_device
```

```
VBox(children=(HTML(value="<h1 style='color:#ffffff;background-color:#000000;
 ↪padding-top: 1%;padding-bottom: 1...
```

[309]:
```
<IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open',
project='main')>
```

[310]:
```
backend_monitor(backend_device)
```

```
ibmq_manila
===========
Configuration
-------------
    n_qubits: 5
    operational: True
    status_msg: active
    pending_jobs: 11
    backend_version: 1.0.1
    basis_gates: ['id', 'rz', 'sx', 'x', 'cx', 'reset']
    local: False
    simulator: False
    allow_object_storage: True
    description: 5 qubit device
    credits_required: True
    rep_times: [0.001]
    multi_meas_enabled: True
    meas_map: [[0, 1, 2, 3, 4]]
    memory: True
    hamiltonian: {'description': 'Qubits are modeled as Duffing oscillators. In
```

this case, the system includes higher energy states, i.e. not just |0> and |1>. The Pauli operators are generalized via the following set of transformations:\n\n$(\\mathbb{I}-\\sigma_{i}^z)/2 \\rightarrow O_i \\equiv b^\\dagger_{i} b_{i}$,\n\n$\\sigma_{+} \\rightarrow b^\\dagger$,\n\n$\\sigma_{-} \\rightarrow b$,\n\n$\\sigma_{i}^X \\rightarrow b^\\dagger_{i} + b_{i}$.\n\nQubits are coupled through resonator buses. The provided Hamiltonian has been projected into the zero excitation subspace of the resonator buses leading to an effective qubit-qubit flip-flop interaction. The qubit resonance frequencies in the Hamiltonian are the cavity dressed frequencies and not exactly what is returned by the backend defaults, which also includes the dressing due to the qubit-qubit interactions.\n\nQuantities are returned in angular frequencies, with units 2*pi*GHz.\n\nWARNING: Currently not all system Hamiltonian information is available to the public, missing values have been replaced with 0.\n', 'h_latex': '\\begin{align} \\mathcal{H}/\\hbar = & \\sum_{i=0}^{4}\\left(\\frac{\\omega_{q,i}}{2}(\\mathbb{I}-\\sigma_i^{z})+\\frac{\\Delta_{i}}{2}(O_i^2-O_i)+\\Omega_{d,i}D_i(t)\\sigma_i^{X}\\right) \\\\ & + J_{0,1}(\\sigma_{0}^{+}\\sigma_{1}^{-}+\\sigma_{0}^{-}\\sigma_{1}^{+}) + J_{1,2}(\\sigma_{1}^{+}\\sigma_{2}^{-}+\\sigma_{1}^{-}\\sigma_{2}^{+}) + J_{2,3}(\\sigma_{2}^{+}\\sigma_{3}^{-}+\\sigma_{2}^{-}\\sigma_{3}^{+}) + J_{3,4}(\\sigma_{3}^{+}\\sigma_{4}^{-}+\\sigma_{3}^{-}\\sigma_{4}^{+}) \\\\ & + \\Omega_{d,0}(U_{0}^{(0,1)}(t))\\sigma_{0}^{X} + \\Omega_{d,1}(U_{1}^{(1,0)}(t)+U_{2}^{(1,2)}(t))\\sigma_{1}^{X} \\\\ & + \\Omega_{d,2}(U_{3}^{(2,1)}(t)+U_{4}^{(2,3)}(t))\\sigma_{2}^{X} + \\Omega_{d,3}(U_{6}^{(3,4)}(t)+U_{5}^{(3,2)}(t))\\sigma_{3}^{X} \\\\ & + \\Omega_{d,4}(U_{7}^{(4,3)}(t))\\sigma_{4}^{X} \\\\ \\end{align}', 'h_str': ['_SUM[i,0,4,wq{i}/2*(I{i}-Z{i})]', '_SUM[i,0,4,delta{i}/2*O{i}*O{i}]', '_SUM[i,0,4,-delta{i}/2*O{i}]', '_SUM[i,0,4,omegad{i}*X{i}||D{i}]', 'jq0q1*Sp0*Sm1', 'jq0q1*Sm0*Sp1', 'jq1q2*Sp1*Sm2', 'jq1q2*Sm1*Sp2', 'jq2q3*Sp2*Sm3', 'jq2q3*Sm2*Sp3', 'jq3q4*Sp3*Sm4', 'jq3q4*Sm3*Sp4', 'omegad1*X0||U0', 'omegad0*X1||U1', 'omegad2*X1||U2', 'omegad1*X2||U3', 'omegad3*X2||U4', 'omegad4*X3||U6', 'omegad2*X3||U5', 'omegad3*X4||U7'], 'osc': {}, 'qub': {'0': 3, '1': 3, '2': 3, '3': 3, '4': 3}, 'vars': {'delta0': -2.1573187977651487, 'delta1': -2.1753119475601674, 'delta2': -2.159281266514359, 'delta3': -2.158603148482815, 'delta4': -2.1495256907311115, 'jq0q1': 0.011836082919670043, 'jq1q2': 0.01196783968906386, 'jq2q3': 0.012402113956012368, 'jq3q4': 0.012186910370408229, 'omegad0': 0.9505887781812132, 'omegad1': 0.9796550909047368, 'omegad2': 0.9467296599666566, 'omegad3': 0.9626890363901895, 'omegad4': 0.9751038859120009, 'wq0': 31.18216600722354, 'wq1': 30.400590237333425, 'wq2': 31.647957025882164, 'wq3': 31.10986463403737, 'wq4': 31.832794679300704}}
    rep_delay_range: [0.0, 500.0]
    n_registers: 1
    online_date: 2021-04-28 04:00:00+00:00
    acquisition_latency: []
    meas_lo_range: [[6.663214088e+18, 7.663214088e+18], [6.783322155e+18, 7.783322155e+18], [6.718928102e+18, 7.718928102e+18], [6.610142342e+18, 7.610142342e+18], [6.846997692e+18, 7.846997692e+18]]
    url: None

```
    pulse_num_qubits: 3
    meas_kernels: ['hw_qmfk']
    uchannels_enabled: True
    parametric_pulses: ['gaussian', 'gaussian_square', 'drag', 'constant']
    u_channel_lo: [[{'q': 1, 'scale': (1+0j)}], [{'q': 0, 'scale': (1+0j)}],
[{'q': 2, 'scale': (1+0j)}], [{'q': 1, 'scale': (1+0j)}], [{'q': 3, 'scale':
(1+0j)}], [{'q': 2, 'scale': (1+0j)}], [{'q': 4, 'scale': (1+0j)}], [{'q': 3,
'scale': (1+0j)}]]
    quantum_volume: 32
    max_experiments: 75
    dt: 0.2222222222222222
    qubit_channel_mapping: [['u1', 'u0', 'd0', 'm0'], ['u0', 'u3', 'u1', 'u2',
'd1', 'm1'], ['u5', 'u3', 'u4', 'd2', 'u2', 'm2'], ['m3', 'u5', 'u6', 'd3',
'u4', 'u7'], ['m4', 'd4', 'u7', 'u6']]
    conditional: False
    backend_name: ibmq_manila
    open_pulse: False
    processor_type: {'family': 'Falcon', 'revision': '5.11', 'segment': 'L'}
    n_uchannels: 8
    meas_levels: [1, 2]
    coupling_map: [[0, 1], [1, 0], [1, 2], [2, 1], [2, 3], [3, 2], [3, 4], [4,
3]]
    max_shots: 8192
    supported_instructions: ['setf', 'id', 'cx', 'reset', 'u3', 'sx', 'play',
'acquire', 'delay', 'u1', 'shiftf', 'x', 'u2', 'rz', 'measure']
    qubit_lo_range: [[4.462795856361695e+18, 5.462795856361695e+18],
[4.33840420918283e+18, 5.33840420918283e+18], [4.536928799429025e+18,
5.536928799429025e+18], [4.451288735426785e+18, 5.451288735426784e+18],
[4.5663466256400916e+18, 5.566346625640092e+18]]
    discriminators: ['quadratic_discriminator', 'hw_qmfk',
'linear_discriminator']
    pulse_num_channels: 9
    dynamic_reprate_enabled: True
    sample_name: family: Falcon, revision: 5.11, segment: L
    allow_q_object: True
    channels: {'acquire0': {'operates': {'qubits': [0]}, 'purpose': 'acquire',
'type': 'acquire'}, 'acquire1': {'operates': {'qubits': [1]}, 'purpose':
'acquire', 'type': 'acquire'}, 'acquire2': {'operates': {'qubits': [2]},
'purpose': 'acquire', 'type': 'acquire'}, 'acquire3': {'operates': {'qubits':
[3]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire4': {'operates':
{'qubits': [4]}, 'purpose': 'acquire', 'type': 'acquire'}, 'd0': {'operates':
{'qubits': [0]}, 'purpose': 'drive', 'type': 'drive'}, 'd1': {'operates':
{'qubits': [1]}, 'purpose': 'drive', 'type': 'drive'}, 'd2': {'operates':
{'qubits': [2]}, 'purpose': 'drive', 'type': 'drive'}, 'd3': {'operates':
{'qubits': [3]}, 'purpose': 'drive', 'type': 'drive'}, 'd4': {'operates':
{'qubits': [4]}, 'purpose': 'drive', 'type': 'drive'}, 'm0': {'operates':
{'qubits': [0]}, 'purpose': 'measure', 'type': 'measure'}, 'm1': {'operates':
{'qubits': [1]}, 'purpose': 'measure', 'type': 'measure'}, 'm2': {'operates':
```

```
{'qubits': [2]}, 'purpose': 'measure', 'type': 'measure'}, 'm3': {'operates':
{'qubits': [3]}, 'purpose': 'measure', 'type': 'measure'}, 'm4': {'operates':
{'qubits': [4]}, 'purpose': 'measure', 'type': 'measure'}, 'u0': {'operates':
{'qubits': [0, 1]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u1':
{'operates': {'qubits': [1, 0]}, 'purpose': 'cross-resonance', 'type':
'control'}, 'u2': {'operates': {'qubits': [1, 2]}, 'purpose': 'cross-resonance',
'type': 'control'}, 'u3': {'operates': {'qubits': [2, 1]}, 'purpose': 'cross-
resonance', 'type': 'control'}, 'u4': {'operates': {'qubits': [2, 3]},
'purpose': 'cross-resonance', 'type': 'control'}, 'u5': {'operates': {'qubits':
[3, 2]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u6': {'operates':
{'qubits': [3, 4]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u7':
{'operates': {'qubits': [4, 3]}, 'purpose': 'cross-resonance', 'type':
'control'}}
    conditional_latency: []
    dtm: 0.2222222222222222
    input_allowed: ['job']
    default_rep_delay: 250.0

Qubits [Name / Freq / T1 / T2 / RZ err / SX err / X err / Readout err]
----------------------------------------------------------------------
    Q0 / 4.96280 GHz / 123.82827 us / 98.24464 us / 0.00000 / 0.00034 / 0.00034
/ 0.04810
    Q1 / 4.83840 GHz / 240.78977 us / 96.55333 us / 0.00000 / 0.00019 / 0.00019
/ 0.02280
    Q2 / 5.03693 GHz / 128.87275 us / 24.10595 us / 0.00000 / 0.00026 / 0.00026
/ 0.02790
    Q3 / 4.95129 GHz / 164.31096 us / 75.18325 us / 0.00000 / 0.00025 / 0.00025
/ 0.02760
    Q4 / 5.06635 GHz / 161.18781 us / 46.67074 us / 0.00000 / 0.00032 / 0.00032
/ 0.02270

Multi-Qubit Gates [Name / Type / Gate Error]
--------------------------------------------
    cx4_3 / cx / 0.00960
    cx3_4 / cx / 0.00960
    cx2_3 / cx / 0.00617
    cx3_2 / cx / 0.00617
    cx1_2 / cx / 0.00882
    cx2_1 / cx / 0.00882
    cx0_1 / cx / 0.00633
    cx1_0 / cx / 0.00633
```

[311]:
```
%qiskit_job_watcher
```

```
Accordion(children=(VBox(layout=Layout(max_width='710px', min_width='710px')),),␣
 ↪layout=Layout(max_height='500...
```

```
<IPython.core.display.Javascript object>
```

[330]:
```
job_r = execute(grover_circuit, backend_device, shots=1024)

jobID_r = job_r.job_id()

print('JOB ID: {}'.format(jobID_r))
```

```
JOB ID: 60bd0c00b454d04decaa76de
```

[331]:
```
job_get=backend_device.retrieve_job("60bd0c00b454d04decaa76de")

result_r = job_get.result()
counts_run = result_r.get_counts(grover_circuit)
```

[312]:
```
job_r = execute(qc_optimized, backend_device, shots=1024)

jobID_r = job_r.job_id()

print('JOB ID: {}'.format(jobID_r))
```

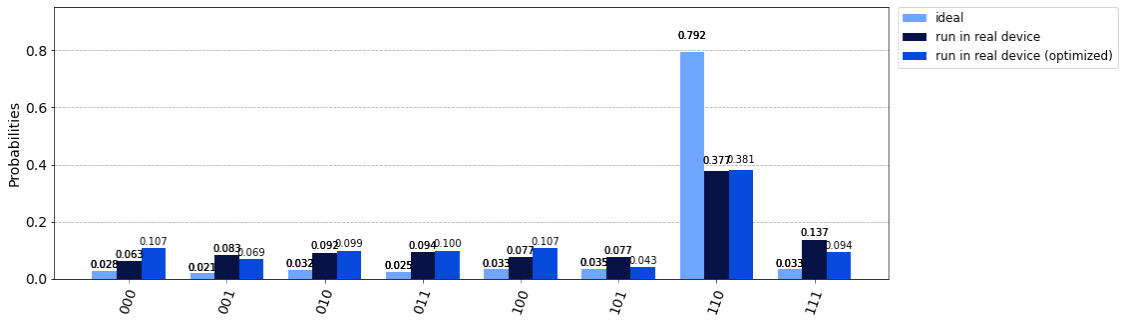
```
JOB ID: 60bcf001917aa094019b7bc1
```

[316]:
```
job_get=backend_device.retrieve_job("60bcf001917aa094019b7bc1")

result_r = job_get.result()
counts_opt = result_r.get_counts(grover_circuit)
```

[332]:
```
legend = ['ideal', 'run in real device', 'run in real device (optimized)']
color = ['#6ea6ff','#051243','#054ada']
plot_histogram([counts_sim, counts_run, counts_opt ], legend=legend,␣
 ↪color=color, figsize=(15,5))
```

[332]:



0.4 4. Ignis

[315]:
```
# Import measurement calibration functions
from qiskit.ignis.mitigation.measurement import (complete_meas_cal,␣
 ↪tensored_meas_cal,
                                                  CompleteMeasFitter,␣
 ↪TensoredMeasFitter)
```

[318]:
```
# Generate the calibration circuits
qr = QuantumRegister(x)

# meas_calibs:
# list of quantum circuit objects containing the calibration circuits
# state_labels:
# calibration state labels
meas_calibs, state_labels = complete_meas_cal(qubit_list=[0,1,2], qr=qr,␣
 ↪circlabel='mcal')
```

[319]:
```
state_labels
```

[319]:
```
['000', '001', '010', '011', '100', '101', '110', '111']
```

[320]:
```
job_ignis = execute(meas_calibs, backend=backend_device, shots=shots)

jobID_run_ignis = job_ignis.job_id()

print('JOB ID: {}'.format(jobID_run_ignis))
```

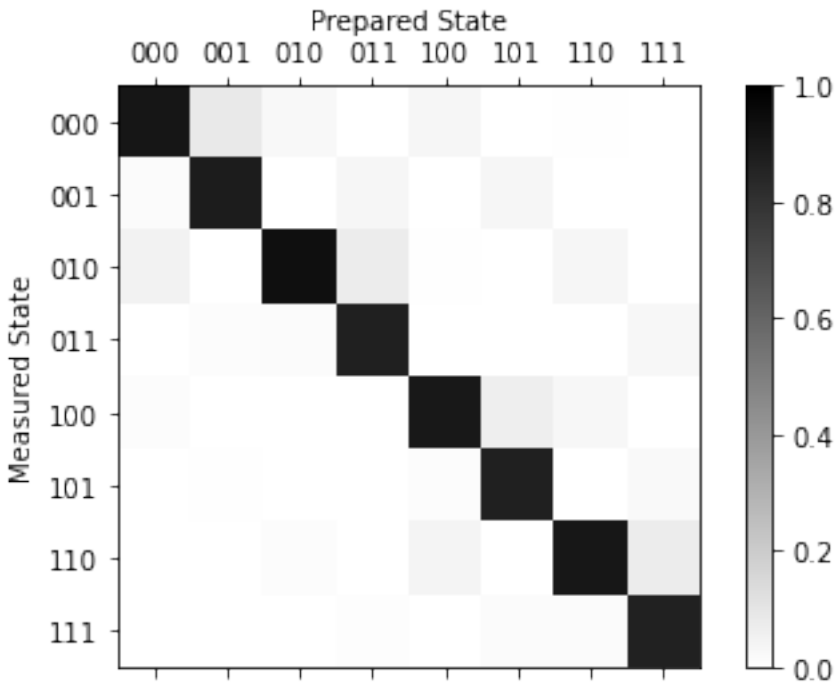
```
JOB ID: 60bcf3fbb454d0dd7caa75ae
```

[321]:
```
job_get=backend_device.retrieve_job("60bcf3fbb454d0dd7caa75ae")

cal_results = job_get.result()
```

[322]:
```
meas_fitter = CompleteMeasFitter(cal_results, state_labels, circlabel='mcal')

# Plot the calibration matrix
meas_fitter.plot_calibration()
```



[323]:
```
# What is the measurement fidelity?
print("Average Measurement Fidelity: %f" % meas_fitter.readout_fidelity())
```

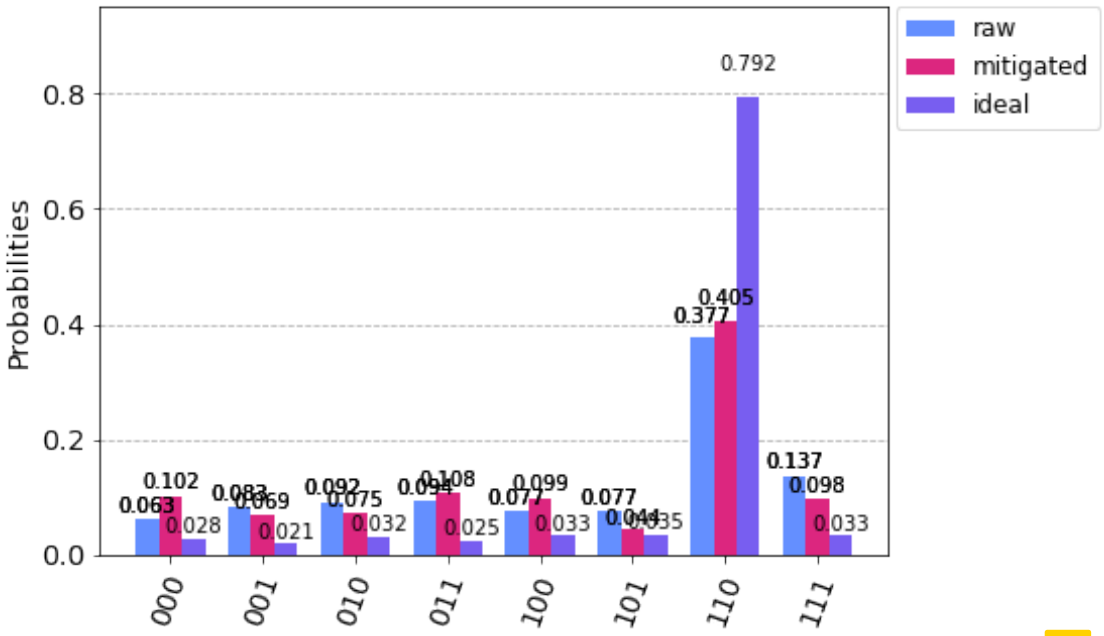
```
Average Measurement Fidelity: 0.894531
```

[324]:
```
# Get the filter object
meas_filter = meas_fitter.filter

# Results with mitigation
mitigated_results = meas_filter.apply(result_r)
mitigated_counts = mitigated_results.get_counts()
```

[333]:
```
plot_histogram([counts_run, mitigated_counts, counts_sim], legend=['raw',␣
↪'mitigated', 'ideal'])
```

[333]:


raw
mitigated
ideal
Probabilities
0.0
0.2
0.4
0.6
0.8
000
001
010
011
100
101
110
111
0.063
0.102
0.028
0.083
0.069
0.021
0.092
0.075
0.032
0.094
0.108
0.025
0.077
0.099
0.033
0.077
0.044
0.035
0.377
0.405
0.792
0.137
0.098
0.033

[ ]: