

In [1]:

```
import qiskit as qk
import numpy as np
from qiskit.tools.monitor import job_monitor
```

```
/Users/palyi/anaconda/envs/Qiskitenv/lib/python3.7/site-packages/m
arshmallow/schema.py:364: ChangedInMarshmallow3Warning: strict=Fal
se is not recommended. In marshmallow 3.0, schemas will always be
strict. See https://marshmallow.readthedocs.io/en/latest/upgrading
.html#schemas-are-always-strict
  ChangedInMarshmallow3Warning
```

In [2]:

```
# connect to IBMQ
APIToken = '0250efe316b67d89659b77d868d9a8be89a47fd01f6a6e8de7c943ea63ce24b84c
d91b9f42090a9c0ea182c710c56e30946f9662e0ef7d1c149027f2a5c3313d'
qk.IBMQ.enable_account(APIToken)
```

In [3]:

```
# parameters we can change
backend = qk.IBMQ.get_backend('ibmq_16_melbourne') # the device to run on
shots = 1024 # how many times each circuit gets run
gates_per_step = 16 # increase in the number of identity gates between circuit
s
steps = 60 # total number of different circuits
# max number of id gates: gates_per_step * steps = 960
qubit = 0 # set the qubit we want to measure
```

In [4]:

```
# parameters of the qcomp and the measurement
gate_length = 110 # nanoseconds
time_per_step = gates_per_step * gate_length
```

In [5]:

```
# create the quantum registers
q = qk.QuantumRegister(1)
c = qk.ClassicalRegister(1)
```

In [6]:

```
# set which virtual qubit corresponds to which physical qubit on the qcomp
def make_layout():
    layout = {}
    for i in range(1):
        layout[(q, i)] = i # the i-th virtual qubit of quantum register q should correspond to qubit #i of the qcomp
    return layout
# create a layout dictionary we can use in qk.execute()
initial_layout = qk.mapper.Layout(make_layout())
```

In [7]:

```
# function for including identity gates in the circuit
def pad_QId(circuit, N, qr):
    for ii in range(N):
        circuit.barrier(qr)
        circuit.iden(qr)
    return circuit
```

In [8]:

```
circuits = [] # the quantum circuits are created as a list
for step in range(steps):
    circuits.append(qk.QuantumCircuit(q, c))
    circuits[step].x(q[qubit])
    circuits[step] = pad_QId(circuits[step], gates_per_step * step, q[qubit])
    circuits[step].barrier(q[qubit])
    circuits[step].measure(q[qubit], c[qubit])
```

In []:

```
# executing the circuits might take a long time depending on the parameters
job = qk.execute(circuits, backend = backend, shots = shots, initial_layout =
initial_layout)
job_monitor(job)
```

In [11]:

```
result = job.result()
result.get_counts(circuits[0])
enumerate(circuits)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\marshmallow\schema.py:3
64: ChangedInMarshmallow3Warning: strict=False is not recommended.
In marshmallow 3.0, schemas will always be strict. See https://mar
shmallow.readthedocs.io/en/latest/upgrading.html#schemas-are-alway
s-strict
    ChangedInMarshmallow3Warning
```

Out[11]:

```
<enumerate at 0x2443428c5a0>
```

In [12]:

```
# keys of the results
# keys[0] corresponds to the excited state, keys[1] corresponds to the ground state
keys = list(result.get_counts(circuits[0]).keys())

# prepare arrays for data storage
data = np.zeros(len(circuits))
sigma_data = np.zeros(len(circuits))
time = np.zeros(len(circuits))
#

# getting the data
for i, circuit in enumerate(circuits): # i is the number of the circuit in the list, circuit is the circuit object itself
    data[i] = float(result.get_counts(circuit)[keys[1]]) / shots # percentage of qubits in the ground state
    sigma_data[i] = np.sqrt(data[i] * (1 - data[i])) / np.sqrt(shots) # stddev of the data
    time[i] = time_per_step * i / 1000
#
```

In [13]:

```
# save the data in a csv file
import csv

with open("Q%s_test.csv" % (str(qubit)), "w") as f:
    writer = csv.writer(f, delimiter='\t')
    writer.writerows(zip(data, sigma_data, time))
quit()
```