QUANTUM COMPUTING WITH QISKIT

Presentation · December 2020

DOI: 10.13140/RG.2.2.29107.35360

1 author:

Aniruddh Bharath
University of Washington

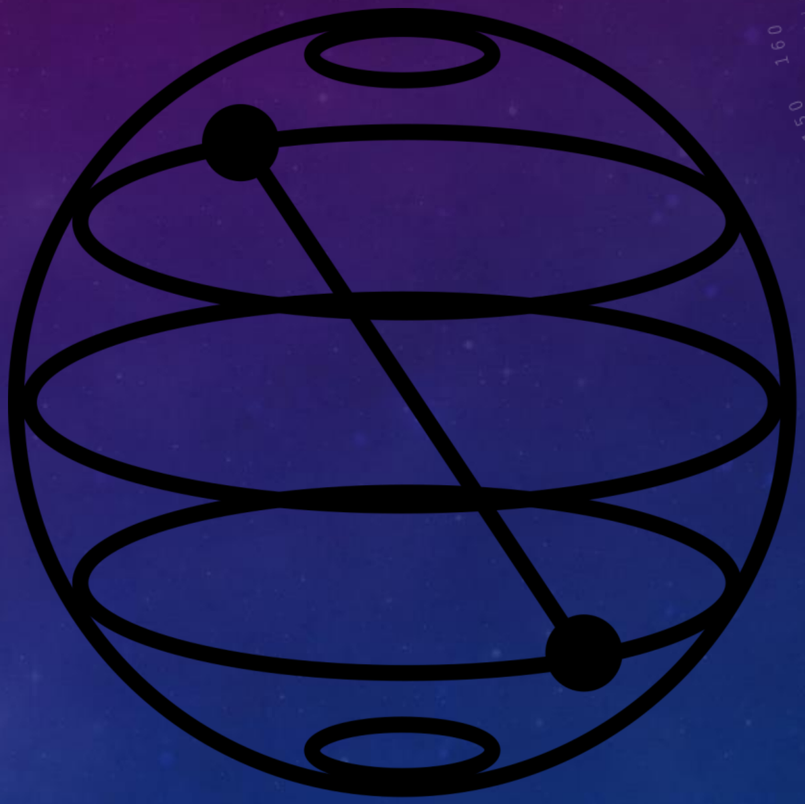
QUANTUM COMPUTING WITH QISKIT

ANIRUDDH BHARATH

MOTIVATION

OVERVIEW

- Qiskit's basic elements and ideas
- Algorithms and theory
- Quantum computers
- Applications in experiment



QISKIT? WHAT'S THAT?

Programming language from IBM based on python.

Made to simulate and compute solutions for quantum devices

Much more efficient than classical computers

THE BUILDING BLOCHS

Qubits and states

Gates

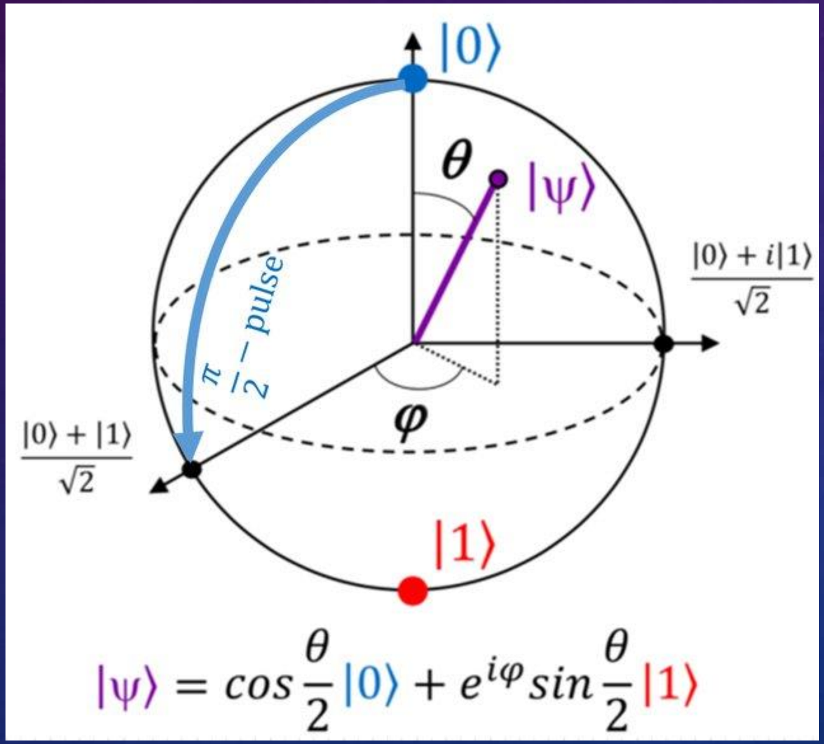
Visualizing states

QUBITS AND EXPRESSING THEM

- Quantum bits or qubits are bits in superposition
- Allow for computing on many states simultaneously
- Collapses to one state upon measuring
- Qubits are defined and expressed in kets (Dirac notation): $|0\rangle$, $|1\rangle$

THE BLOCH SPHERE AND Q SPHERE

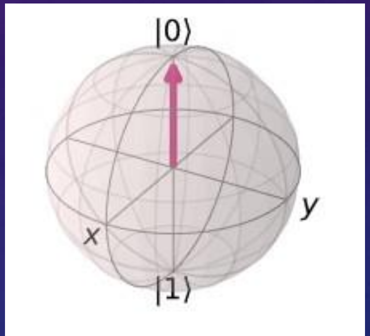
Pure state expression: $|q\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$

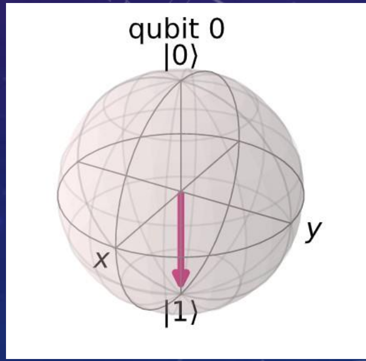


- Twice the angles of Hilbert space
- Expresses all normalized states
- Computations cause changes in coordinates

GATES: OPERATING ON QUBITS

- Operators represented as elementary computations
- Pauli gates: X, Y, Z are rotations of pi
- flipping bit states: CNOT as the X-gate
- Flipping phases: Z gate
- Both flipping bits and phases: Y gate





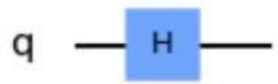
HADAMARD GATE

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

- Important for algorithms
- Create superposition when applied
- Initialized at all 0 to superposition of all possible states
- Used to change between bases X, Y, Z

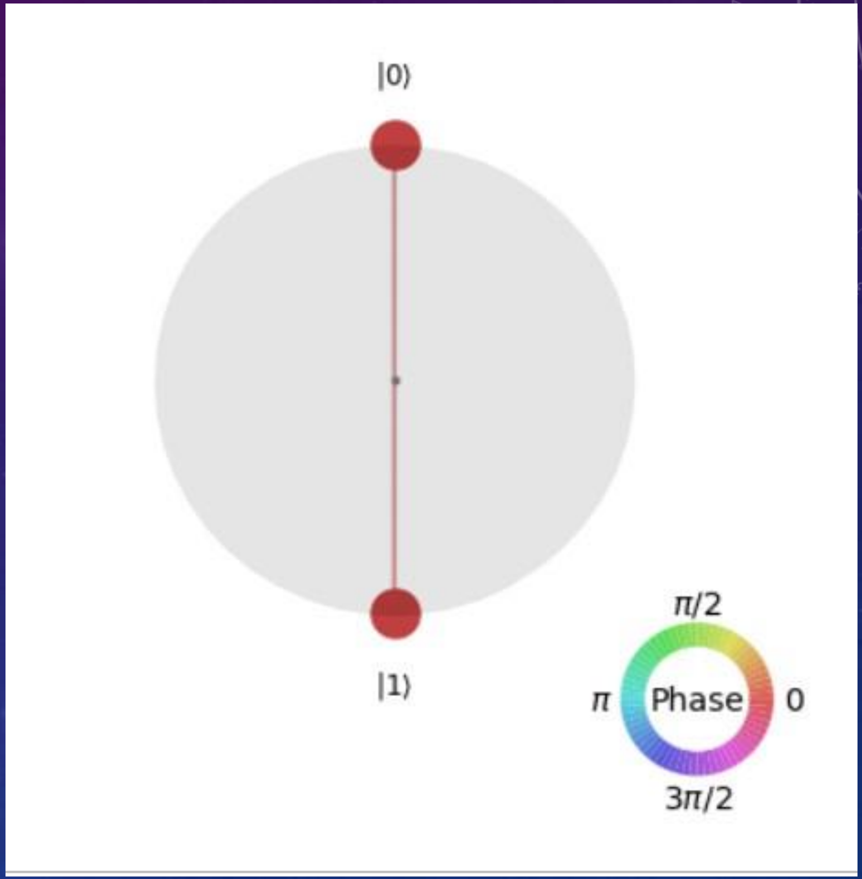
Lab demo

```python
sv = Statevector.from_label('0')
mycircuit = QuantumCircuit(1)
mycircuit.h(0)
mycircuit.draw('mpl')
```



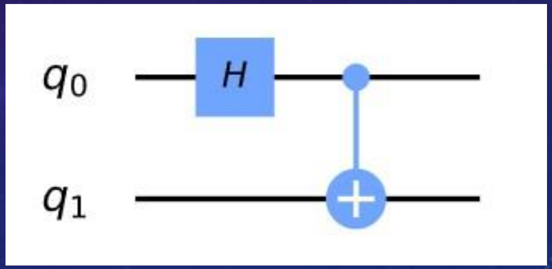

```python
new_sv = sv.evolve(mycircuit)
print(new_sv)
plot_state_qsphere(new_sv.data)
```

```
Statevector([0.70710678+0.j, 0.70710678+0.j],
            dims=(2,))
```



ENTANGLED STATES AND BELL STATES

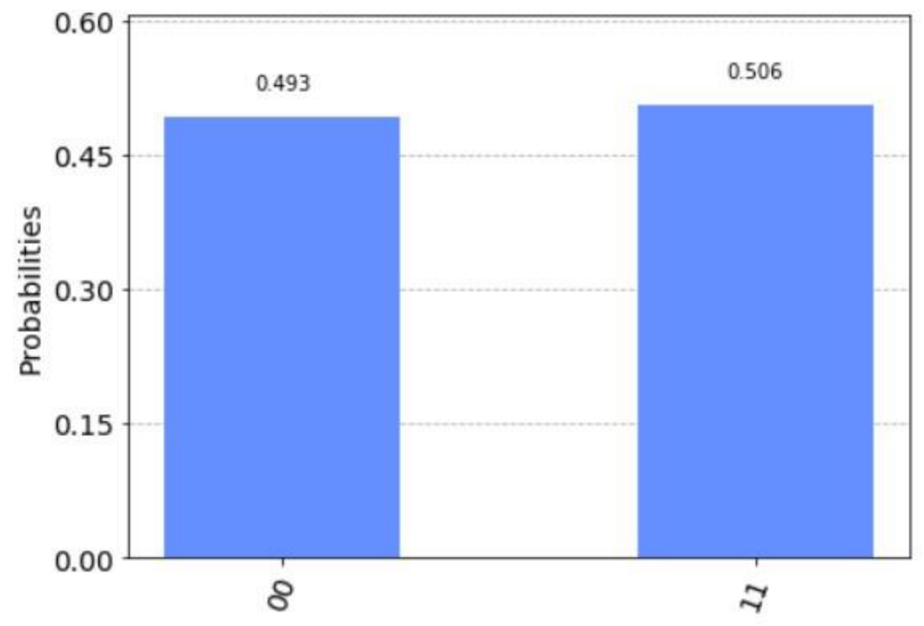
- Combined qubit states that cannot be expressed separately
- Maximally entangled into 4 states: bell state
- Measuring one tells you information about the other



```python
# making the quantum circuit
bell = QuantumCircuit(2, 2)
bell.h(0)
bell.cx(0, 1)
# measuring
meas = QuantumCircuit(2, 2)
meas.measure([0,1], [0,1])

# executing the quantum circuit
backend = BasicAer.get_backend('qasm_simulator') # the device to run on
state = bell + meas
result = execute(state, backend, shots=2000).result()
counts  = result.get_counts(state)
print(counts)
```

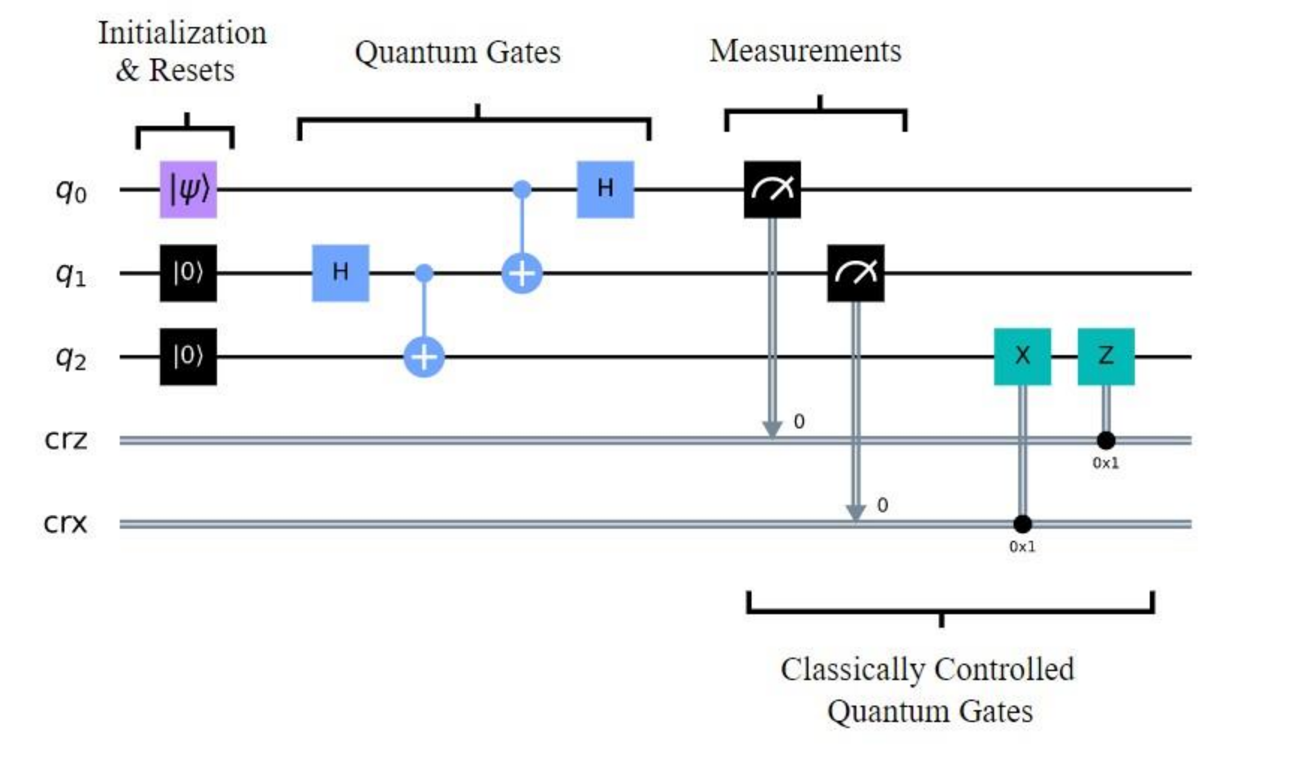
```
{'11': 1013, '00': 987}
```

Out[6]:



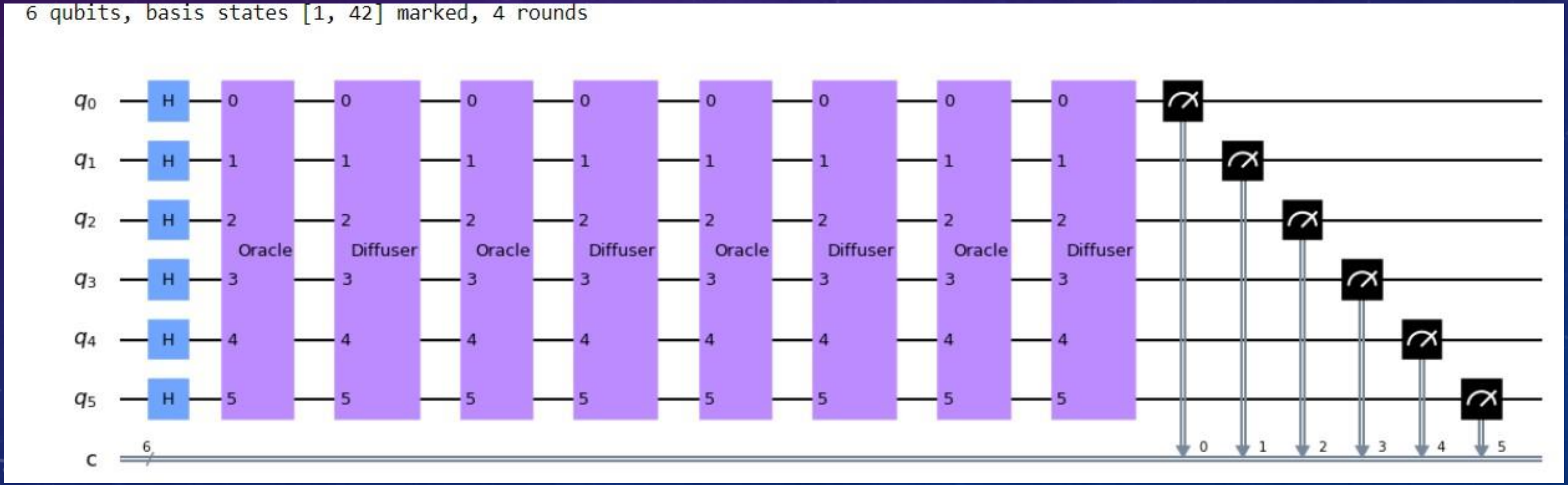
ALGORITHM IMPLEMENTATION AND REVERSIBILITY

- Protocols and quantum circuits



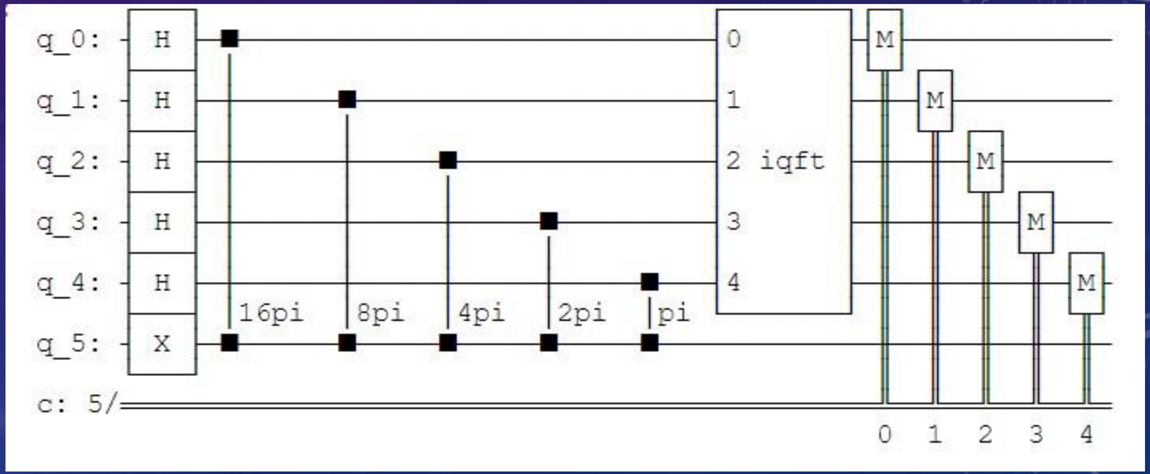
QUANTUM ALGORITHMS

- Grover's Algorithm: Searching key terms with amplitude amplification



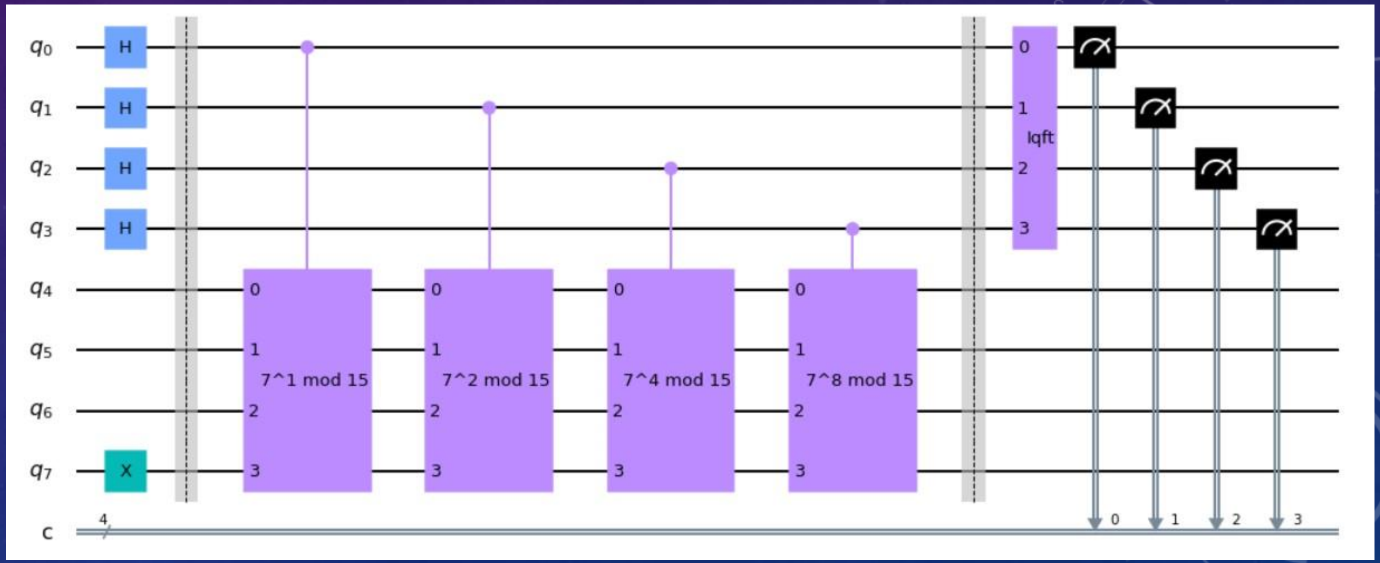
- Shor's Algorithm: Unitary properties with quantum phase estimation (Finding the period of a given periodic function)

```python
def qpe_program(n, theta):

    # Create a quantum circuit on n+1 qubits (n measurement, 1 target)
    qc = QuantumCircuit(n+1, n)

    # Initialize the qubits
    initialize_qubits(qc, range(n), n)

    # Apply the controlled unitary operators in sequence
    for x in range(n):
        exponent = 2**(n-x-1)
        unitary_operator_exponent(qc, x, n, theta, exponent)

    # Apply the inverse quantum Fourier transform
    apply_iqft(qc, range(n), n)

    # Measure all qubits
    qc.measure(range(n), range(n))

    return qc

n = 5; theta = 0.5
mycircuit = qpe_program(n, theta)
mycircuit.draw(output='text')
```



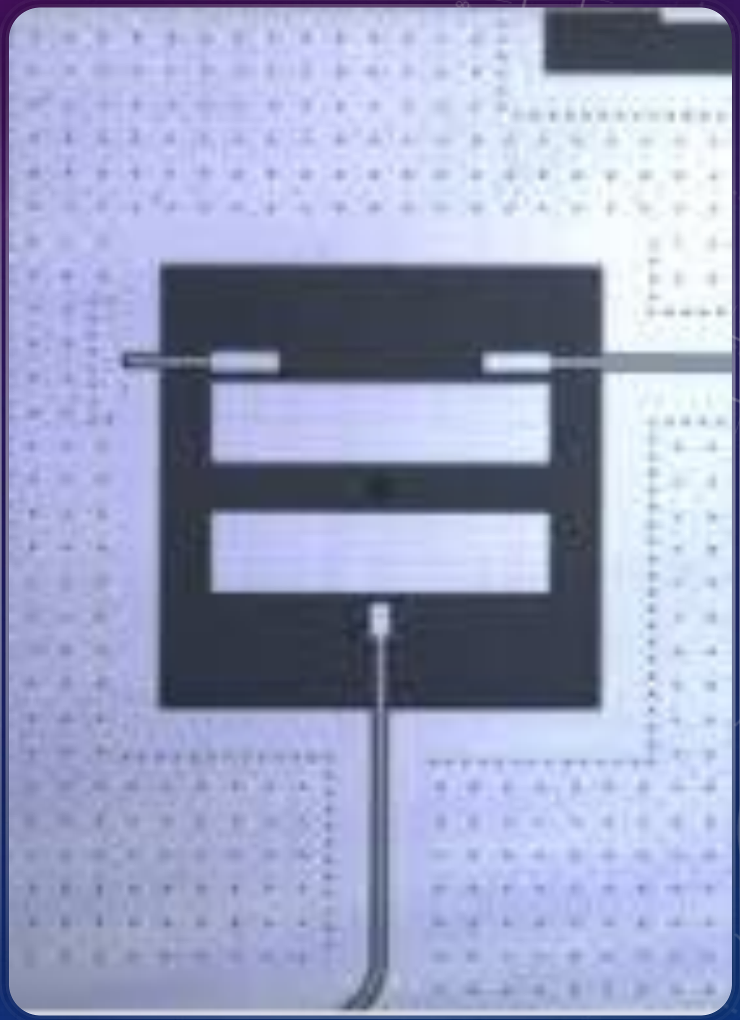
- quantum fourier transforms and phase estimation for factoring prime numbers from N (N = p * q)

```python
def shor_program(n, m, a):

    # set up quantum circuit
    shor = QuantumCircuit(n+m, n)

    # initialize the qubits
    initialize_qubits(shor, n, m)
    shor.barrier()

    # apply modular exponentiation
    modular_exponentiation(shor, n, m, a)
    shor.barrier()

    # apply inverse QFT
    apply_iqft(shor, range(n))

    # measure the first n qubits
    shor.measure(range(n), range(n))

    return shor

n = 4; m = 4; a = 7
mycircuit = shor_program(n, m, a)
mycircuit.draw()
```

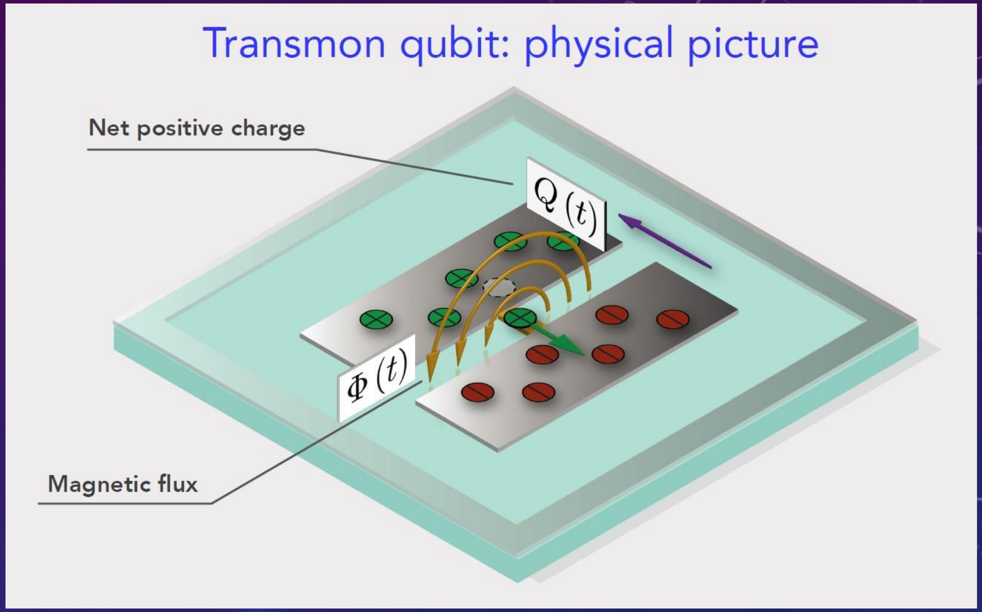


THEORY TO REALITY : THE TRANSMON QUBIT

- Two charged plates with a junction bridge in between
- Mimics anharmonic oscillators
- Fed in microwaves corresponding to known energies that excites the system from ground state |0> to |1>



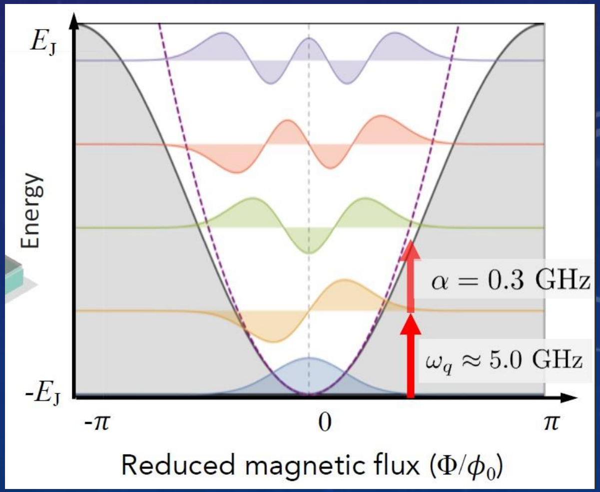
TRANSMON BEHAVIOR



- Current through junction bridge causes magnetic flux

- Behavior is similar to an anharmonic oscillator of known energies

- Can feed in signals of these energies based on the Hamiltonian approximation



REFERENCES

- https://qiskit.org/learn/