

M_Q: Complete circuits and Qiskit

Culmination: implement the models in Qiskit, then connect phase estimation to period finding and Shor.

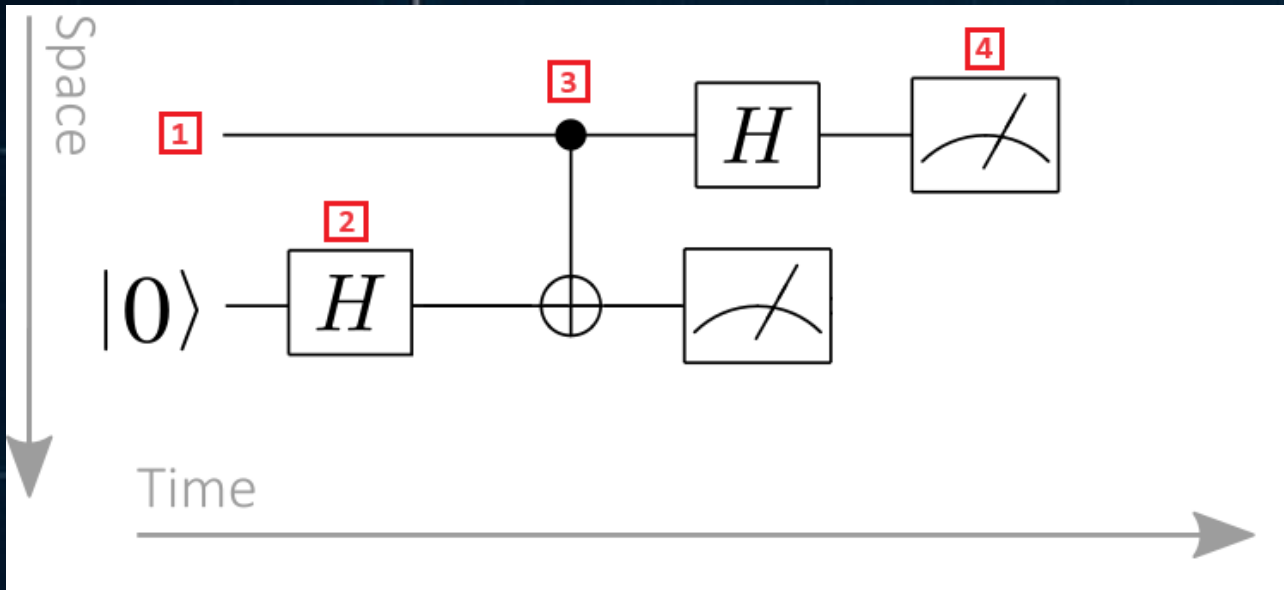
The complete circuit model

Specify state preparation, a universal set of operations, ancillas, and measurement.

The earlier M-models are deliberate restrictions of this framework. We now connect the mathematical operations to executable circuits.

Quantum Circuit Diagrams 101

- **Qubit wires:** horizontal lines (top to bottom = different qubits)
- **Time flows left to right**
- **Boxes** = gates (e.g., H, X, Z, etc.)
- **Dots and lines** = control and target (e.g., CNOT)
- **Measurement** shown with  or [M]



1. **Qubit Registers** – Horizontal lines represent qubits. Top to bottom = Qubit 0, Qubit 1, etc.
2. **Quantum Gate** – Boxes apply quantum operations. The H gate here puts the qubit into superposition.
3. **Controlled Gate** – The CNOT gate uses a control (filled circle) and a target (plus in a circle). It flips the target qubit if the control is $|1\rangle$.
4. **Measurement** – Meter symbols measure qubits, collapsing them to classical bits (0 or 1).

1. Hadamard gate on qubit 0:

$$|0\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

→ Now the state is:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle$$

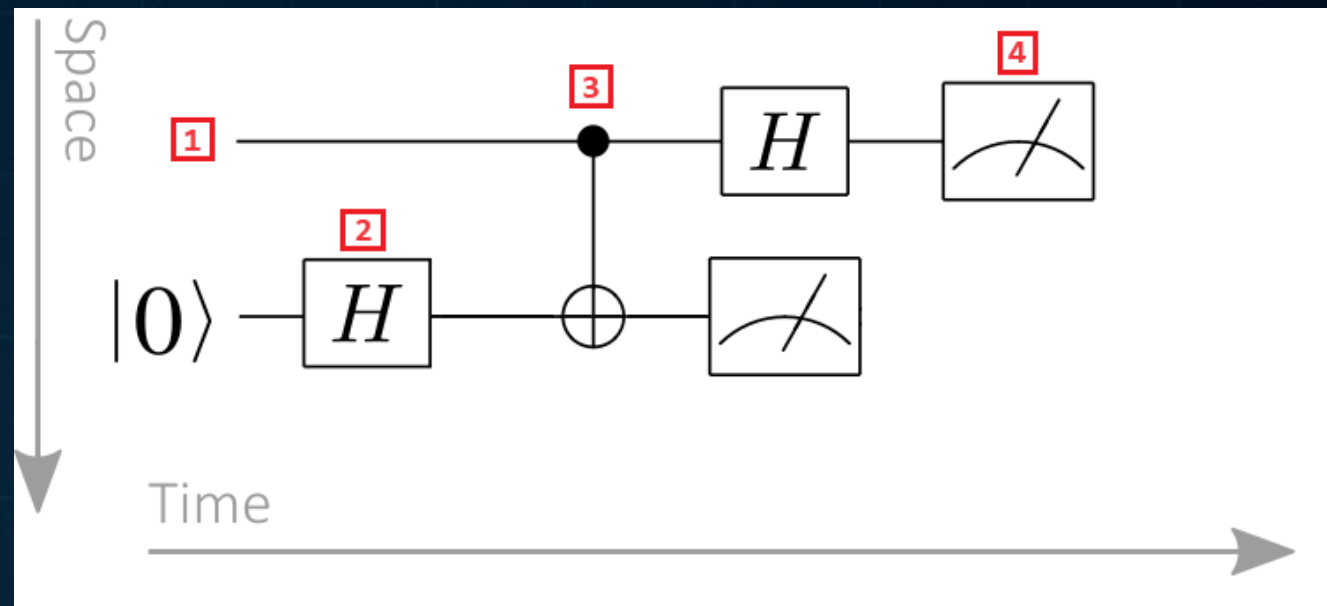
2. CNOT gate, with:

Control: qubit 0

Target: qubit 1

→ This flips the second qubit only when the first is $|1\rangle$

Final state: $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$



Controlled not gate:

WHAT IS A QUANTUM CIRCUIT?

A quantum circuit is a program.

- A quantum state is prepared
 - Gates transform the state
 - Measurement extracts an outcome
- Time flows from left to right.
Each wire represents a qubit.

```
from qiskit import QuantumCircuit
qc = QuantumCircuit(1)
qc.draw('text')
```

OUTPUT: q:

This quantum circuit has a single qubit. The default name for the quantum register containing that qubit is 'q'.

The draw text command displays the circuit using ascii art. In this case the simplest possible circuit.

QISKIT

- **Qiskit** is an open-source quantum computing framework developed by [IBM](#)
- It allows us to:
- define quantum circuits
- apply quantum gates (unitary transformations)
- simulate or run on real quantum hardware

```
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector

qc = QuantumCircuit(1)
qc.h(0)

state = Statevector.from_label('0').evolve(qc)
print(state)
```

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
```

```
qc = QuantumCircuit(1, 1)
```

```
# Start in  $|0\rangle$  (default)
```

```
# Change measurement basis to X
qc.h(0) # Hadamard rotates  $Z \rightarrow X$  basis
```

```
# Measure
qc.measure(0, 0)
```

```
sim = AerSimulator()
result = sim.run(qc, shots=1000).result()
counts = result.get_counts()
```

```
print(counts)
```

QuantumCircuit(1,1) creates a quantum circuit with 1 qubit and 1 classical bit to store the measurement result.

The Hadamard operator, H, changes the measurement basis from Z to X.

Run the code 1000 times and print how many times each state was observed.

```
{ '0' : 499, '1' : 501 }
```

FROM ROTATION TO GATES

- Quantum Gates Rotate the State in Hilbert Space
- Applying a Gate = Matrix multiplication
- Qiskit is performing the same linear algebra we just derived
- In this example it rotates the state about the x axis on the Bloch sphere by $\frac{\pi}{3}$
- A single rotation doesn't compute anything.
- But a *sequence* of carefully chosen rotations can steer the probability of outcomes.

```
from qiskit import QuantumCircuit
from qiskit.quantum_info import
Statevector
import numpy as np

state = Statevector.from_label('0')

qc = QuantumCircuit(1)

qc.rx(np.pi/3, 0)

new_state = state.evolve(qc)
print(new_state)
```

From Theory to Programming

- **Quantum Mechanics provides the rules**

Schrödinger equation, unitary evolution, measurement

- **Quantum Circuits encode those rules**

Gates represent unitary operators; circuits evolve qubit states

Quantum Programming Languages

Qiskit (IBM)

Cirq (Google)

Braket (Amazon)

Ocean (D-Wave, for annealing)

Qubits, Gates, and Circuits in Code (qiskit)

Now we will entangle three qubits with qiskit and then measure their correlation.

```
import numpy as np
from qiskit import QuantumCircuit

# 1. A quantum circuit for preparing the quantum state  $|000\rangle + i |111\rangle / \sqrt{2}$ 
qc = QuantumCircuit(3)
qc.h(0)           # generate superposition
qc.p(np.pi / 2, 0) # add quantum phase
qc.cx(0, 1)       # 0th-qubit-Controlled-NOT gate on 1st qubit
qc.cx(0, 2)       # 0th-qubit-Controlled-NOT gate on 2nd qubit
```

<qiskit.circuit.instructionset.InstructionSet at 0x147a396db910>

Qubits, Gates, and Circuits in Code (qiskit)

2. Add the classical output in the form of measurement of all qubits

```
qc_measured = qc.measure_all(inplace=False)
```

3. Execute using the Sampler primitive

```
from qiskit.primitives import StatevectorSampler
```

```
sampler = StatevectorSampler()
```

```
job = sampler.run([qc_measured], shots=1000)
```

```
result = job.result()
```

```
print(f"> Counts: {result[0].data['meas'].get_counts()}")
```

```
> Counts: {'111': 508, '000': 492}
```

The **StatevectorEstimator** computes properties **analytically** from the full quantum state, not just from sampling.

PLACEHOLDER FOR MATTHEW

A tested Qiskit lab

Specify supported package versions and test the examples in the actual teaching environment.

Explain qubit order, statevectors, shots, and counts. Reproduce the earlier H/Z and Bell-state calculations before running an algorithm.

Computability and efficiency

An ordinary quantum circuit can be simulated classically, potentially at large cost. Quantum algorithms change resource requirements for some problems.

Exponential state-vector size does not prove that every classical simulation needs exponential time. Factoring is not known to be NP-complete.

THE CENTRAL QUESTION

How does modular exponentiation turn factoring into a period problem?

For suitable a , the repetition in $a^x \bmod N$ contains information that can reveal factors of N .

Prediction: state what should change in the first experiment and what must remain invariant.

KEY TERMS

modular exponentiation
order
reversibility
greatest common divisor

MODEL

$f(x) = a^x \bmod N$; find the least r with $a^r = 1 \bmod N$

FIRST WORKED EXPERIMENT

Start with one case small enough to check by hand.

1. Write the input and prediction
2. Apply the operation
3. Compare the observed evidence

Evidence to keep: period tables, reversibility checks, and gcd factor recovery. Label basis order, shots, and simulator assumptions.

EXPERIMENT

Enumerate small modular sequences and identify successful and unsuccessful choices of a .

THE CENTRAL QUESTION

How do a quantum period estimate and classical post-processing produce factors?

The quantum subroutine estimates an order; continued fractions and gcd computations finish the job.

Prediction: state what should change in the first experiment and what must remain invariant.

KEY TERMS

order finding
phase estimation
continued fractions
factor recovery

MODEL

if r is even, $\gcd(a^{(r/2)} \pm 1, N)$ may reveal factors

FIRST WORKED EXPERIMENT

Start with one case small enough to check by hand.

1. Write the input and prediction
2. Apply the operation
3. Compare the observed evidence

Evidence to keep: measured phase samples,
recovered order, and explicit failure modes.
Label basis order, shots, and simulator
assumptions.

EXPERIMENT

Trace one small semiprime through base choice,
period peaks, and factor recovery.

Period finding and Shor capstone

Trace a small modular-exponentiation example through order finding and classical factor recovery.

Explain failed base choices, finite precision, and the difference between a demonstration circuit and a scalable implementation.