Quantum circuits as code: qubits, registers and gates

Build a Bell circuit, keep wire order explicit and separate quantum evolution from readout.

Core 22 min lesson Dr. Charith Jayasekara

By the end, you'll be able to

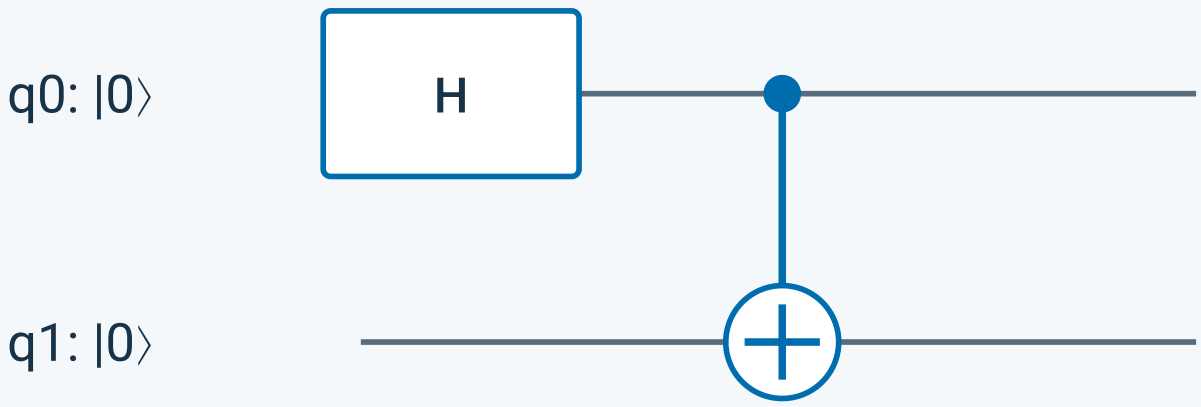
- Construct gates and measurements as separate program stages.
- Predict a Bell state from its circuit.
- Check an asymmetric bit-order example.

Begin with a clear question

A circuit object is a recipe. Adding H or CX records an operation; it does not by itself produce a measurement result. Keep the coherent preparation separate from the measured copy so you can inspect the ideal state and later collect finite-shot data from the same preparation.

Build the reasoning

Start with two qubits in zero. H on q0 makes two branches. CX uses q0 as control and flips q1 only in the branch where the control is one. This gives equal amplitudes on 00 and 11. Qiskit circuit drawings put q0 on the top wire, while its statevector strings put q0 on the right. The browser labs use q0 on the left. Label this conversion wherever you compare the two displays.



Follow time from left to right. Qiskit statevector labels put q0 on the right; diagrams put q0 on the top wire.

Worked example

The Bell state is symmetric under bit reversal, so it is a poor test of your display conversion. Add a second circuit that prepares only q0 in one. Its native Qiskit label must be 01. This small asymmetric case detects an error that a beautiful Bell histogram would miss.

$$|00\rangle \xrightarrow{H_0} \frac{|00\rangle + |01\rangle}{\sqrt{2}} \xrightarrow{\mathrm{CX}_{0\to 1}} \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad (\text{Qiskit labels: } q_1 q_0).$$

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

bell = QuantumCircuit(2)
bell.h(0)
bell.cx(0, 1)
state = Statevector.from_instruction(bell)
assert np.allclose(state.data, np.array([1, 0, 0, 1]) / np.sqrt(2))
probe = QuantumCircuit(2)
probe.x(0)
assert np.isclose(Statevector.from_instruction(probe).probabilities_dict()["01"], 1)
measured = bell.copy()
measured.measure_all()
print(measured.draw(output="text"))
```

Practical lab

Notebook 02 adds sampling and confidence intervals. Make the prediction from the unmeasured circuit first, then execute its measured copy. Name classical registers deliberately and inspect the result field belonging to the register you measured. Reversing a displayed bitstring is a boundary conversion, not a reason to reverse the control and target in your quantum program.

Run the corresponding notebook from the software practice collection:

- Download Circuits, amplitudes and the phase you cannot see in a histogram
- Download Bell circuits, finite shots and honest uncertainty

Keep a short execution record: predicted result, observed result, environment, inputs and the check that would reject a plausible defect. Explain a failure before changing the expected value.

Chapter summary

- Separate the coherent preparation from its measured copy.
- Trace CX using the control wire.
- Check bit order with an asymmetric state.

Exercises

1. Prepare q1 in one and q0 in zero. Give the native Qiskit string and the book's q0-first display.
2. Change one part of the worked program to create a plausible fault. State which assertion should fail and why.

> Worked answers
>
> 1. The native Qiskit string is 10. The q0-first book display is 01. The physical preparation is the same.
> 2. A useful answer identifies a specific changed gate, parameter, mapping or model and links it to an independent expected behaviour. Keep the intended specification fixed while changing the implementation; save the failing case as a regression test.

Check your understanding

PAUSE & THINK

Which test best catches a reversed two-bit display?

- [ ] Prepare only q0 in one
- [ ] Use only the symmetric Bell state
- [ ] Increase the number of shots on the Bell state

Answer: Prepare only q0 in one. An asymmetric preparation makes the convention visible.

Sources

IBM's bit-ordering guide, Statevector API and Operator API describe the SDK conventions and equivalence tools. The notebook bundle records its pinned environment and local execution evidence.