Compilation and transpilation fundamentals

Compile a logical circuit for a target and verify its meaning after routing and layout changes.

Undergraduate 22 min lesson Dr. Charith Jayasekara

By the end, you'll be able to

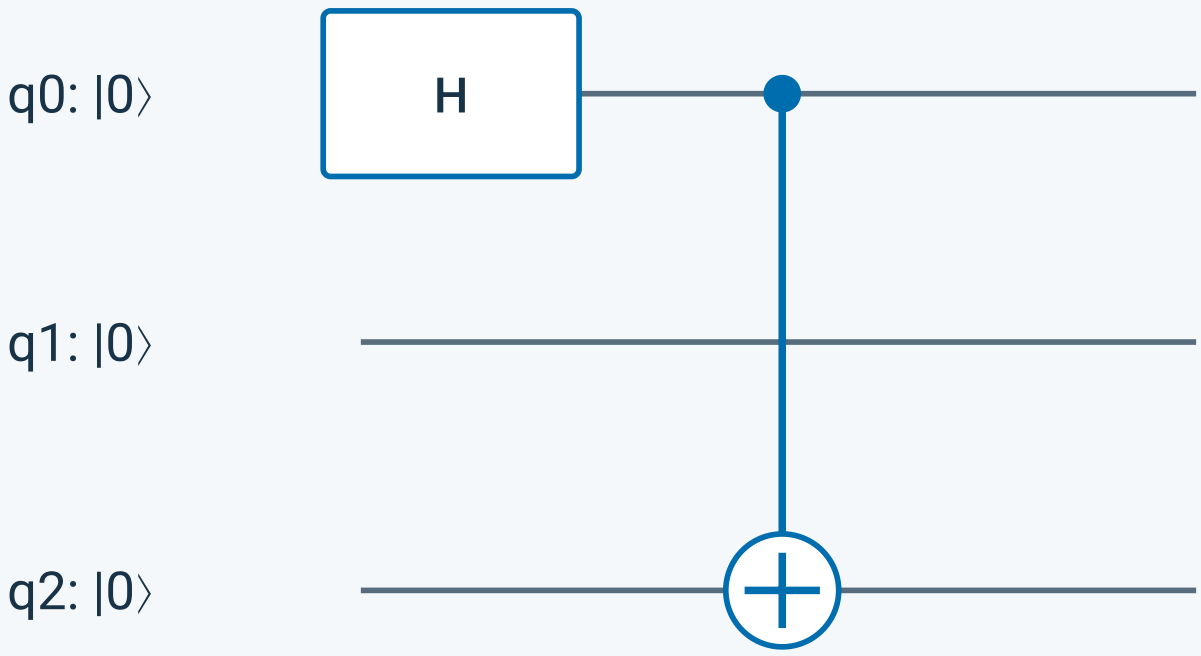
- Explain the role of target gates and connectivity.
- Compile at several optimisation levels.
- Compare a routed circuit with layout-aware equivalence.

Begin with a clear question

A processor accepts particular instructions on particular qubits. Transpilation translates a logical circuit into that target's instructions, adding routing where needed and simplifying where possible. The useful question is whether the compiled program preserves the intended behaviour within the approximation settings you allowed.

Build the reasoning

A CX between distant logical wires may require movement through intermediate physical qubits. The final wire labels can therefore differ from the original logical layout. Comparing raw matrices without accounting for that permutation can report a false failure. Conversely, ignoring layout when processing counts can turn a correct physical run into an incorrect classical answer.



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

Worked example

Create a three-qubit line target, then request an interaction between the ends. Compile at four optimisation levels with a fixed transpiler seed. Operator.from_circuit uses the compiled circuit layout for this comparison. The depth and operation counts measure the compiled small example; they are not a hardware timing benchmark.

$$U_{\text{logical}} \sim U_{\text{compiled}} \quad \text{after the documented input/output layout maps.}$$

```
from qiskit import QuantumCircuit
from qiskit.quantum_info import Operator
from qiskit.providers.fake_provider import GenericBackendV2
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager

backend = GenericBackendV2(3, coupling_map=[[0,1],[1,0],[1,2],[2,1]], seed=42)
logical = QuantumCircuit(3)
logical.h(0); logical.cx(0, 2)
reference = Operator.from_circuit(logical)
for level in range(4):
    pm = generate_preset_pass_manager(backend=backend, optimization_level=level,
seed_transpiler=42)
    compiled = pm.run(logical)
    assert Operator.from_circuit(compiled).equiv(reference)
    print(level, compiled.depth(), dict(compiled.count_ops()))
```

Practical lab

Inspect native instruction support and coupling constraints as notebook 03 does. Save the target description, optimisation level, transpiler seed and final layout with your results. Dense unitary checks are small-instance tools; larger programs need appropriate equivalence methods and task-specific checks. Calibration-weighted choices may change when the target changes, so recompile against the target you actually execute.

Run the corresponding notebook from the software practice collection:

- Download Transpilation: change the circuit, preserve its meaning
- Download A tested local workflow and an explicit hardware handoff

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

- Compile for an explicit target.
- Include layout when verifying and decoding.
- Record resource counts and execution settings.

Exercises

1. Two compiled circuits have equal depth but different two-qubit gate counts. Is their expected reliability necessarily equal?
2. Change one part of the worked program to create a plausible fault. State which assertion should fail and why.

Worked answers

1. No. Gate types, locations, calibrations, idle time, routing and correlations matter. Depth alone is an incomplete resource or noise model.
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

Why might a direct raw-matrix comparison reject a correct routed circuit?

- [ ] The logical-to-physical layout permutation was omitted
- [ ] Routing always changes the intended answer
- [ ] Every optimisation level must produce identical gate text

Answer: The logical-to-physical layout permutation was omitted. Recover the intended logical ordering before comparing behaviour.

Sources

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