In [1]:
```
import qiskit
qiskit.__version__
```

Out[1]: '1.4.2'

In [2]:
```
import qiskit_ibm_runtime
qiskit_ibm_runtime.__version__
```

Out[2]: '0.37.0'

In [3]:
```
import qiskit_aer
qiskit_aer.__version__
```

Out[3]: '0.15.1'

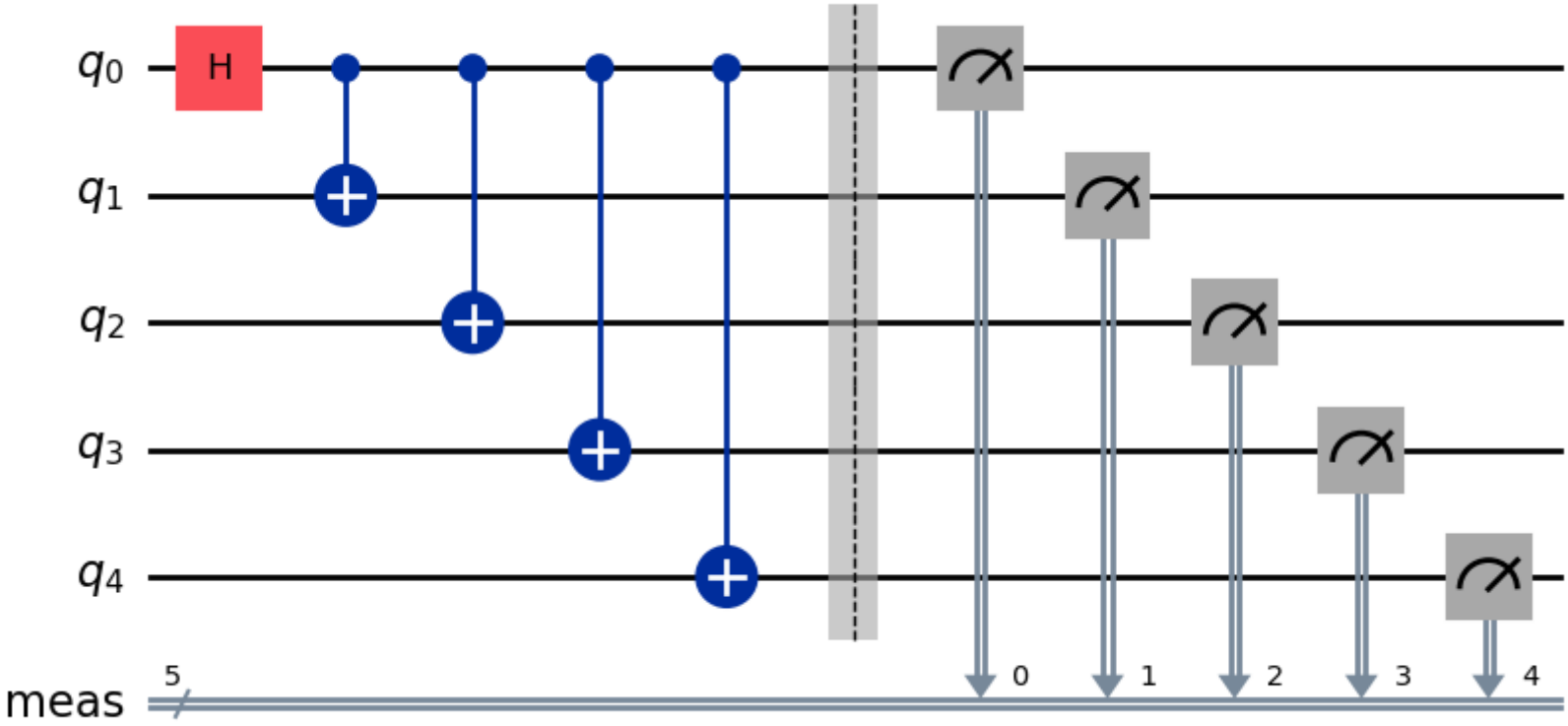
In [3]:
```
from qiskit.circuit import QuantumCircuit
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit.primitives import BackendSamplerV2 as Sampler
```

In [4]:
```
from qiskit_ibm_runtime.fake_provider import FakeKyiv

backend = FakeKyiv()
```

In [5]:

```
num_qubits = 5

ghz_circ = QuantumCircuit(num_qubits)
ghz_circ.h(0)
[ghz_circ.cx(0,i) for i in range(1,num_qubits)]
ghz_circ.measure_all()
ghz_circ.draw('mpl')
```

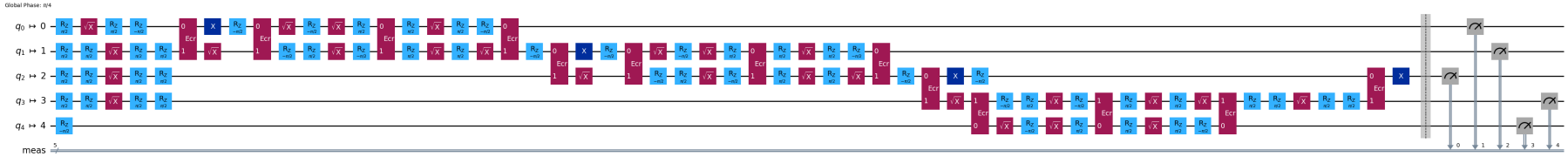
Out[5]:



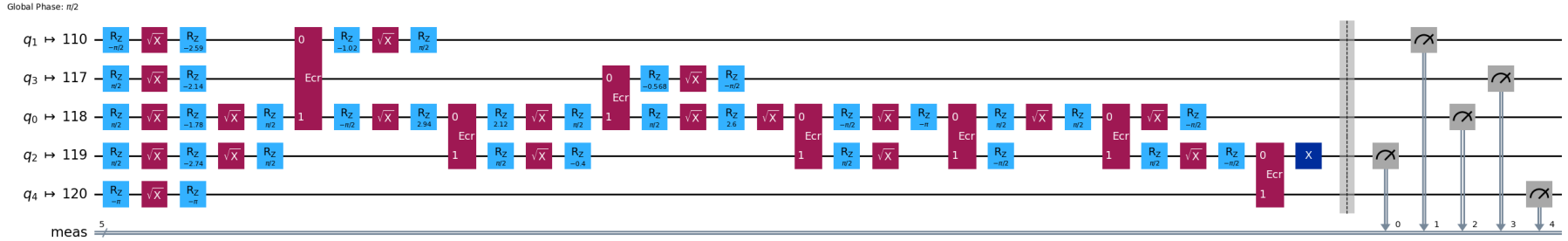
In [6]:

```
pm0 = generate_preset_pass_manager(optimization_level=0, backend=backend, seed_transpiler=777)
pm2 = generate_preset_pass_manager(optimization_level=2, backend=backend, seed_transpiler=777)
circ0 = pm0.run(ghz_circ)
circ2 = pm2.run(ghz_circ)
print("optimization_level=0:"); display(circ0.draw('mpl', idle_wires=False, fold=-1))
print("optimization_level=2:"); display(circ2.draw('mpl', idle_wires=False, fold=-1))
```

optimization_level=0:

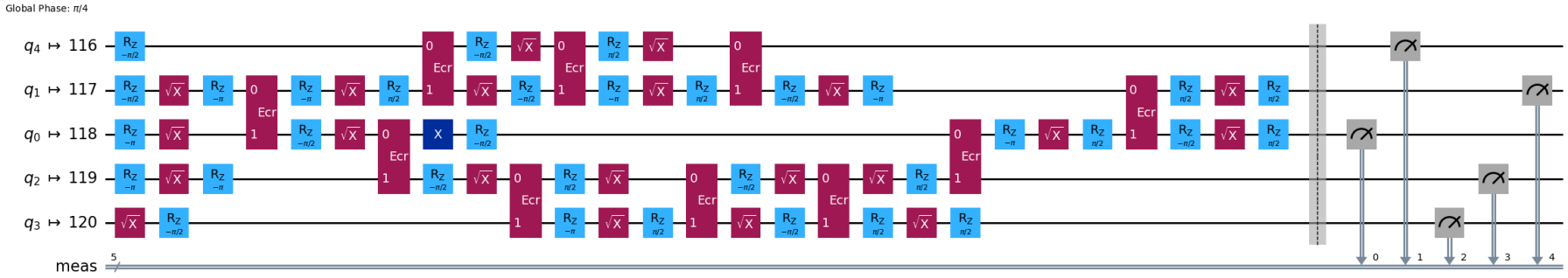


optimization_level=2:



In [7]:
```
pm1 = generate_preset_pass_manager(optimization_level=1, backend=backend, seed_transpiler=777)
circ1 = pm1.run(ghz_circ)
print("optimization_level=1:"); display(circ1.draw('mpl', idle_wires=False, fold=-1))
```

```
optimization_level=1:
```



In [8]:
```
# run the circuits on the fake backend (noisy simulator)
sampler = Sampler(backend=backend)
job = sampler.run([circ0, circ2], shots=10000)
print(f"Job ID: {job.job_id()}")
```

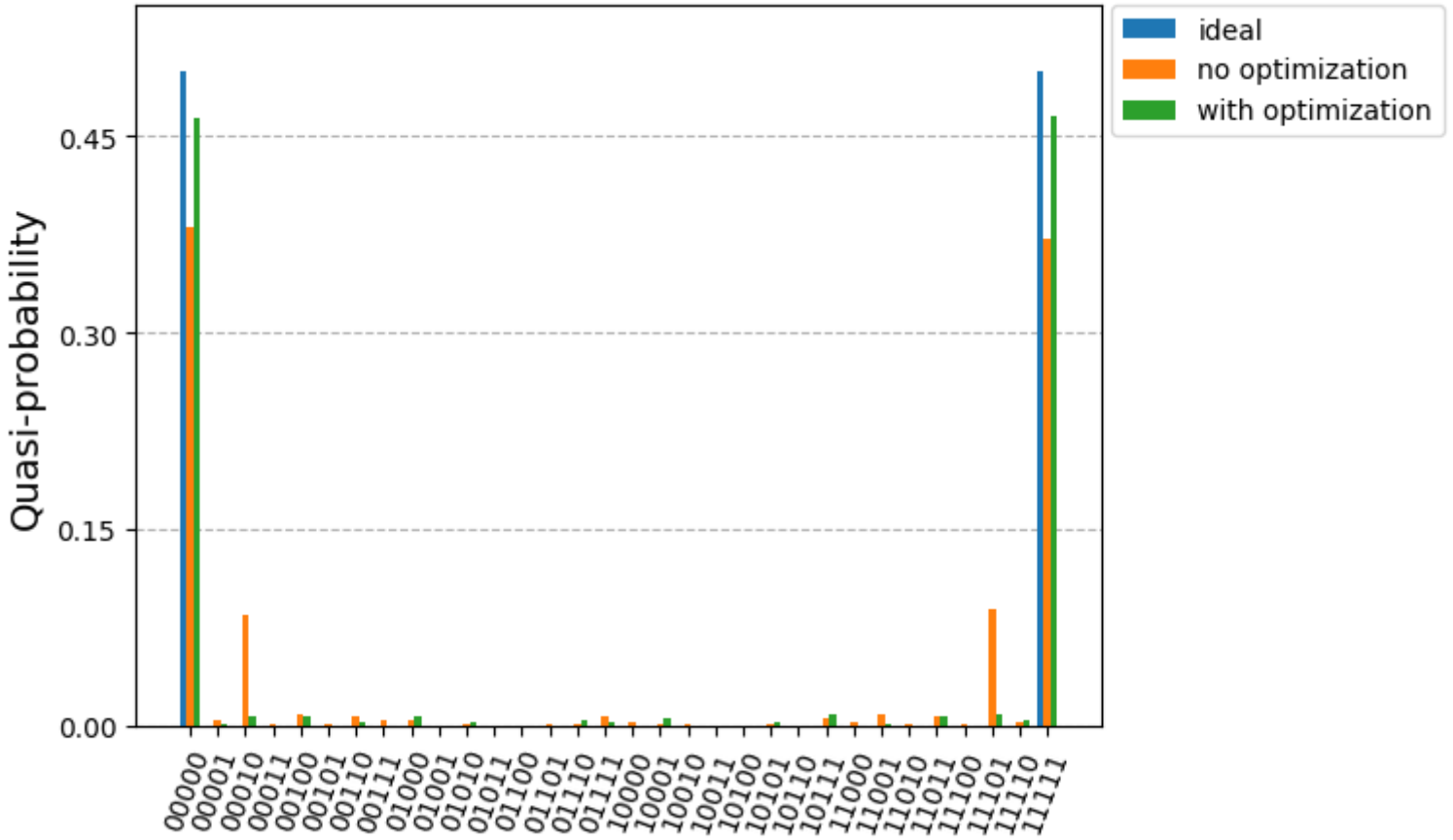
```
Job ID: 882f563b-30c4-44ae-a375-d4850c2ed355
```

In [10]:
```
# get results
result = job.result()
unoptimized_result = result[0].data.meas.get_counts()
optimized_result = result[1].data.meas.get_counts()
```

In [11]:

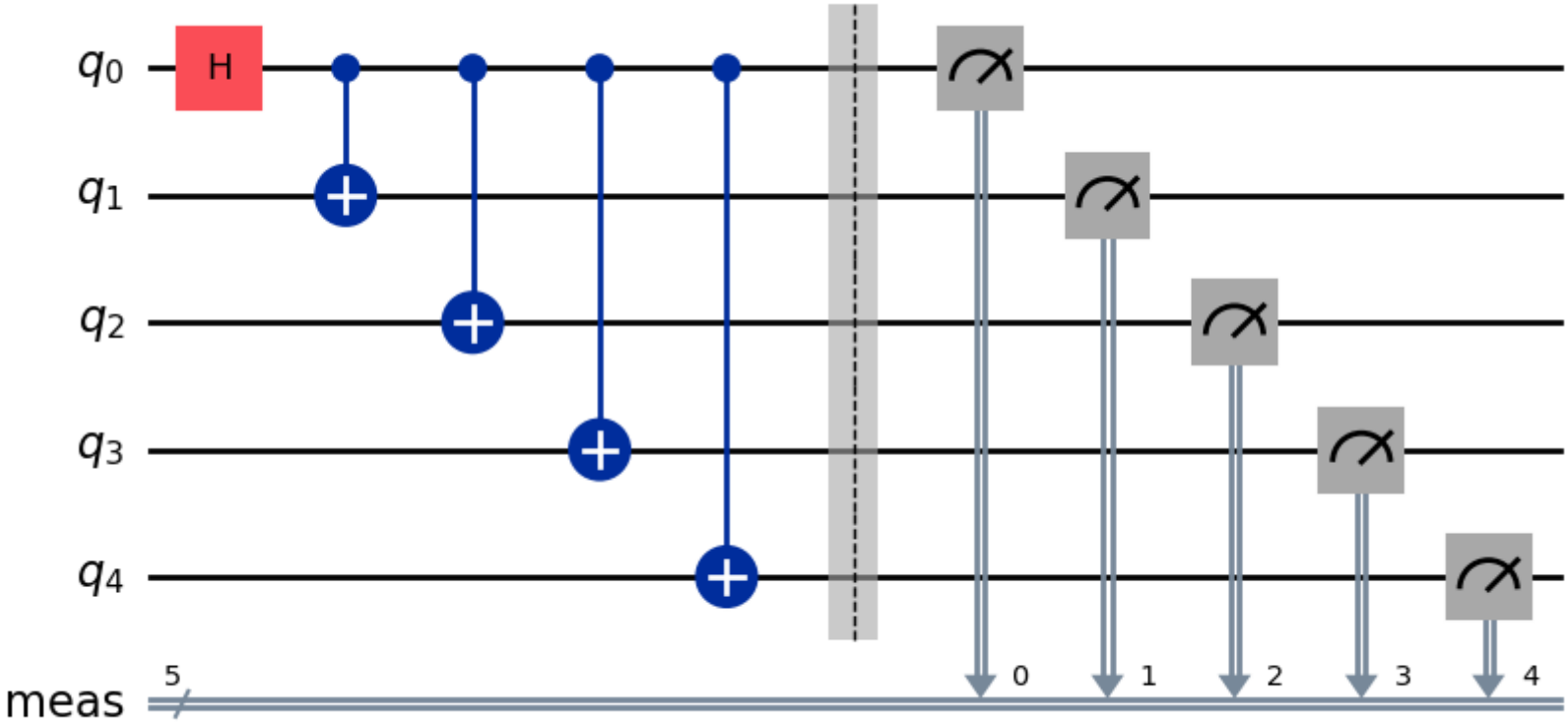
```
from qiskit.visualization import plot_histogram
# plot
sim_result = {'0'*5:0.5, '1'*5:0.5}
plot_histogram(
    [result for result in [sim_result, unoptimized_result, optimized_result]],
    bar_labels=False,
    legend=[
        "ideal",
        "no optimization",
        "with optimization",
    ],
)
```

Out[11]:



In [12]:
```
1 # Original GHZ circuit (naive synthesis)
2 ghz_circ.draw('mpl')
```

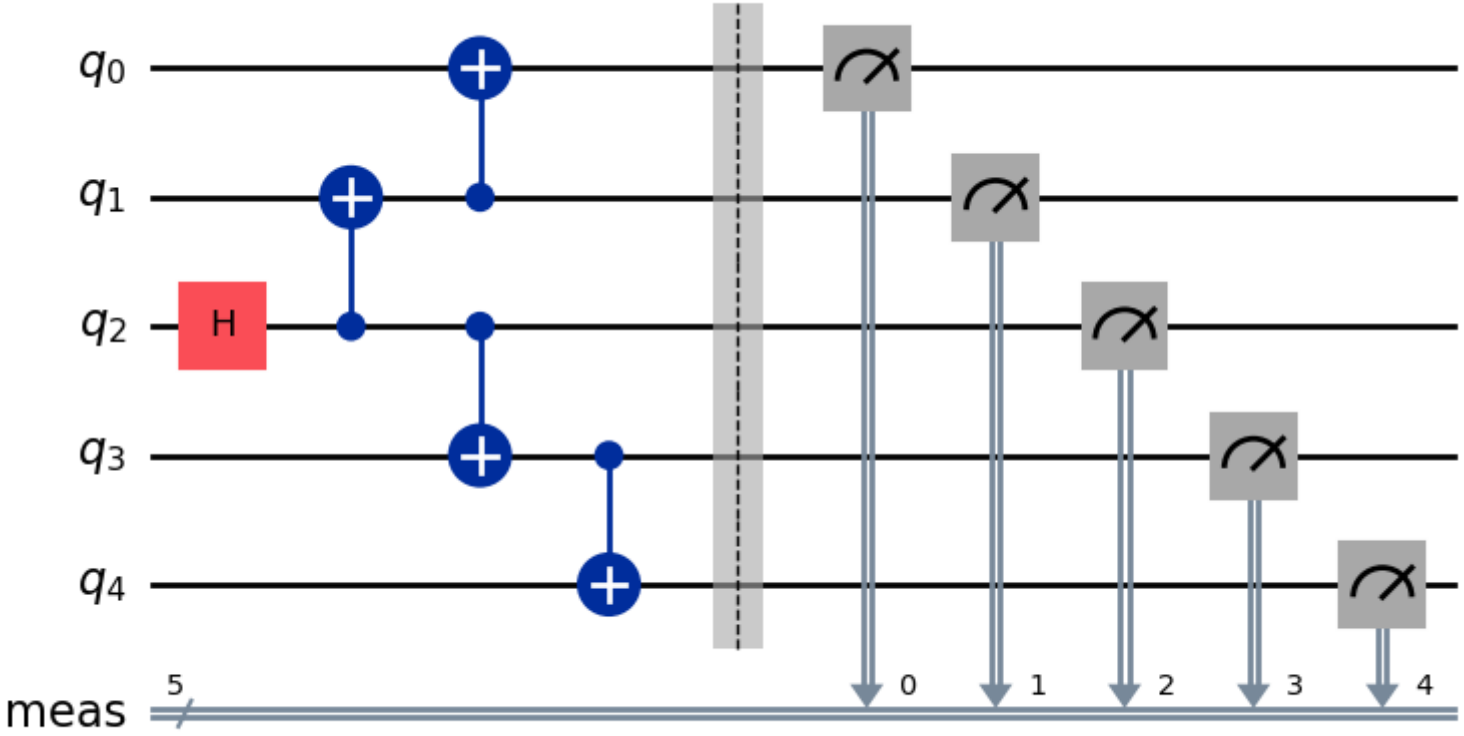
Out[12]:



In [13]:

```
# A cleverly-synthesized GHZ circuit
ghz_circ2 = QuantumCircuit(5)
ghz_circ2.h(2)
ghz_circ2.cx(2, 1)
ghz_circ2.cx(2, 3)
ghz_circ2.cx(1, 0)
ghz_circ2.cx(3, 4)
ghz_circ2.measure_all()
ghz_circ2.draw('mpl')
```

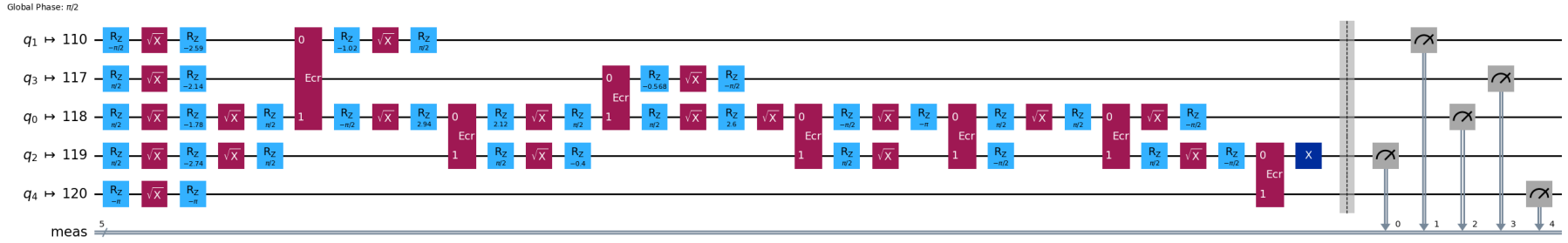
Out[13]:



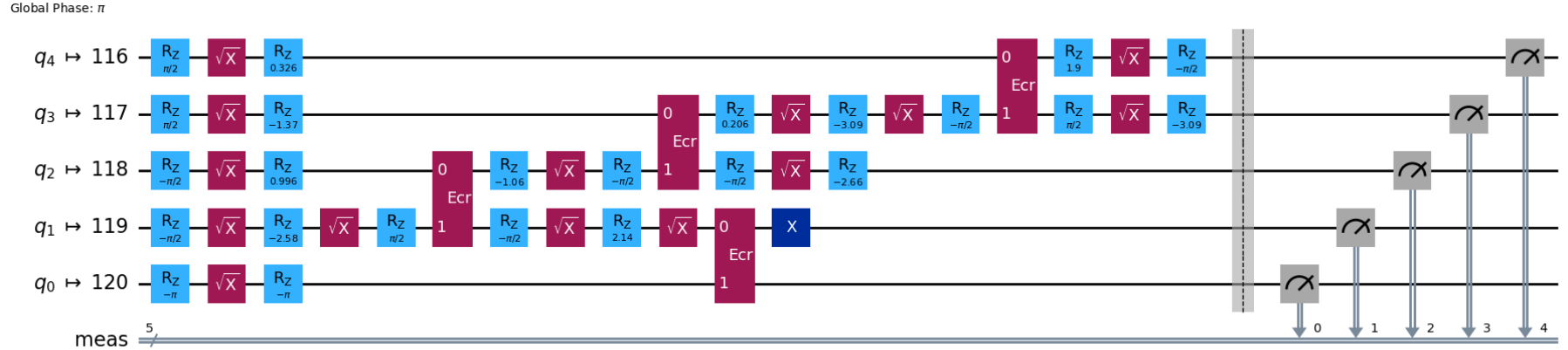
In [14]:

```
# transpile both with the same optimization level 2
circ_org = pm2.run(ghz_circ)
circ_new = pm2.run(ghz_circ2)
print("original synthesis:"); display(circ_org.draw('mpl', idle_wires=False, fold=-1))
print("new synthesis:"); display(circ_new.draw('mpl', idle_wires=False, fold=-1))
```

```
original synthesis:
```




```
new synthesis:
```



In [15]:
```
# run the circuits
sampler = Sampler(backend=backend)
job = sampler.run([circ_org, circ_new], shots=10000)
print(f"Job ID: {job.job_id()}")
```

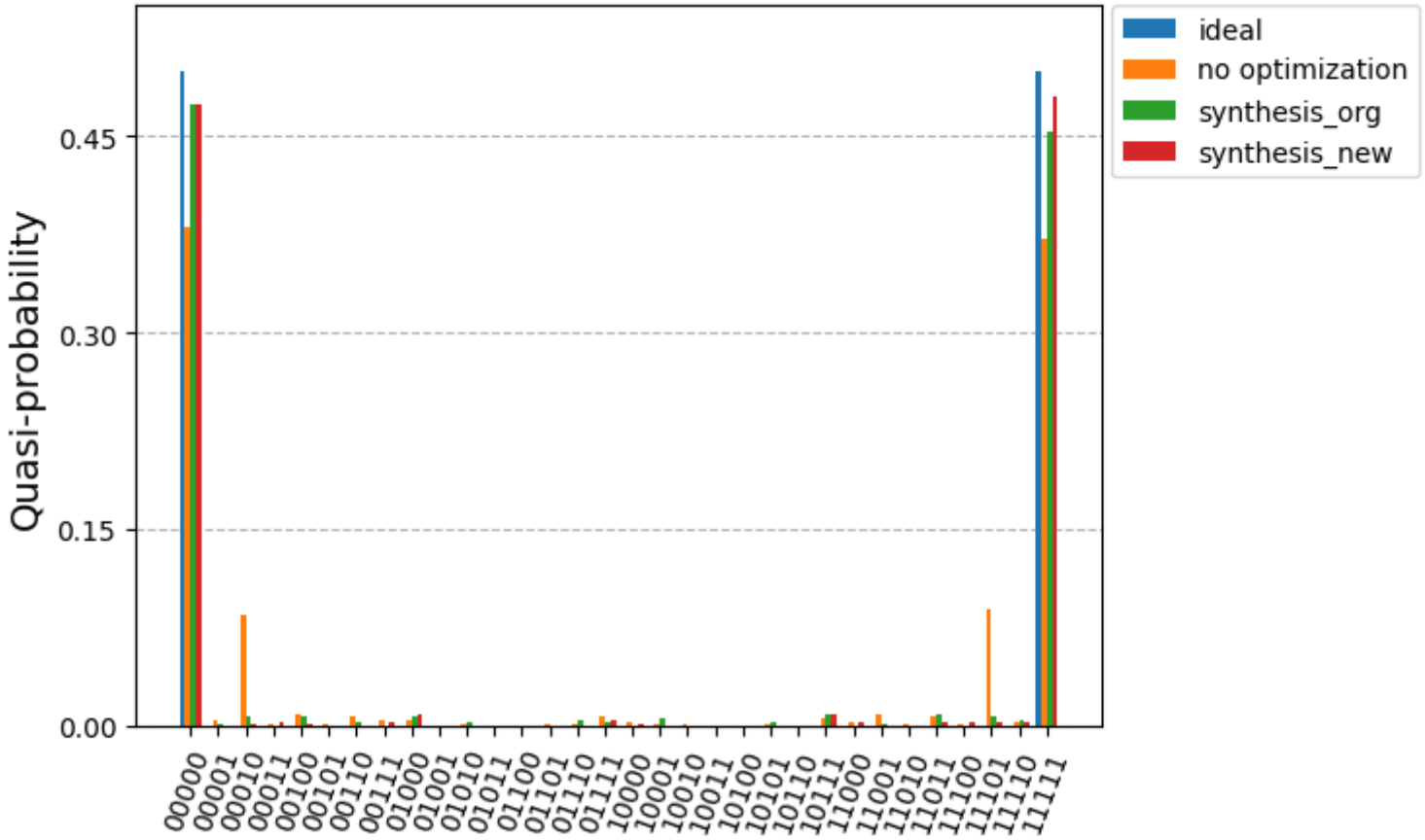
```
Job ID: cd3dc6fa-15a3-4025-9fd9-b57cd65df6f4
```

In [16]:
```
# get results
result = job.result()
synthesis_org_result = result[0].data.meas.get_counts()
synthesis_new_result = result[1].data.meas.get_counts()
```

In [17]:

```
# plot
sim_result = {'0'*5:0.5, '1'*5:0.5}
plot_histogram(
    [result for result in [sim_result, unoptimized_result, synthesis_org_result, synthesis_new_result]]
    bar_labels=False,
    legend=[
        "ideal",
        "no optimization",
        "synthesis_org",
        "synthesis_new",
    ],
)
```

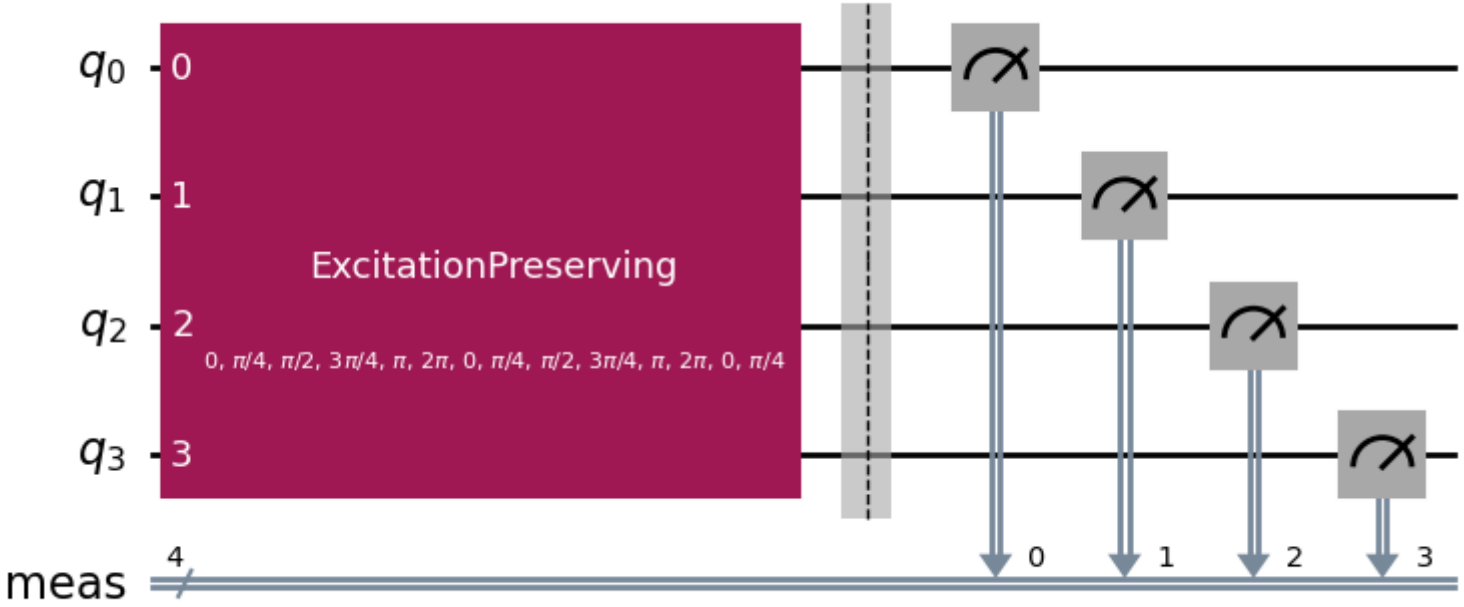
Out[17]:



In [18]:

```
# Build a toy example circuit
from math import pi
import itertools
from qiskit.circuit import QuantumCircuit
from qiskit.circuit.library import ExcitationPreserving

circuit = QuantumCircuit(4, name="Example circuit")
circuit.append(ExcitationPreserving(4, reps=1, flatten=True), range(4))
circuit.measure_all()

value_cycle = itertools.cycle([0, pi / 4, pi / 2, 3*pi / 4, pi, 2* pi])
circuit.assign_parameters([x[1] for x in zip(range(len(circuit.parameters)), value_cycle)], inplace=Tru
circuit.draw('mpl')
```

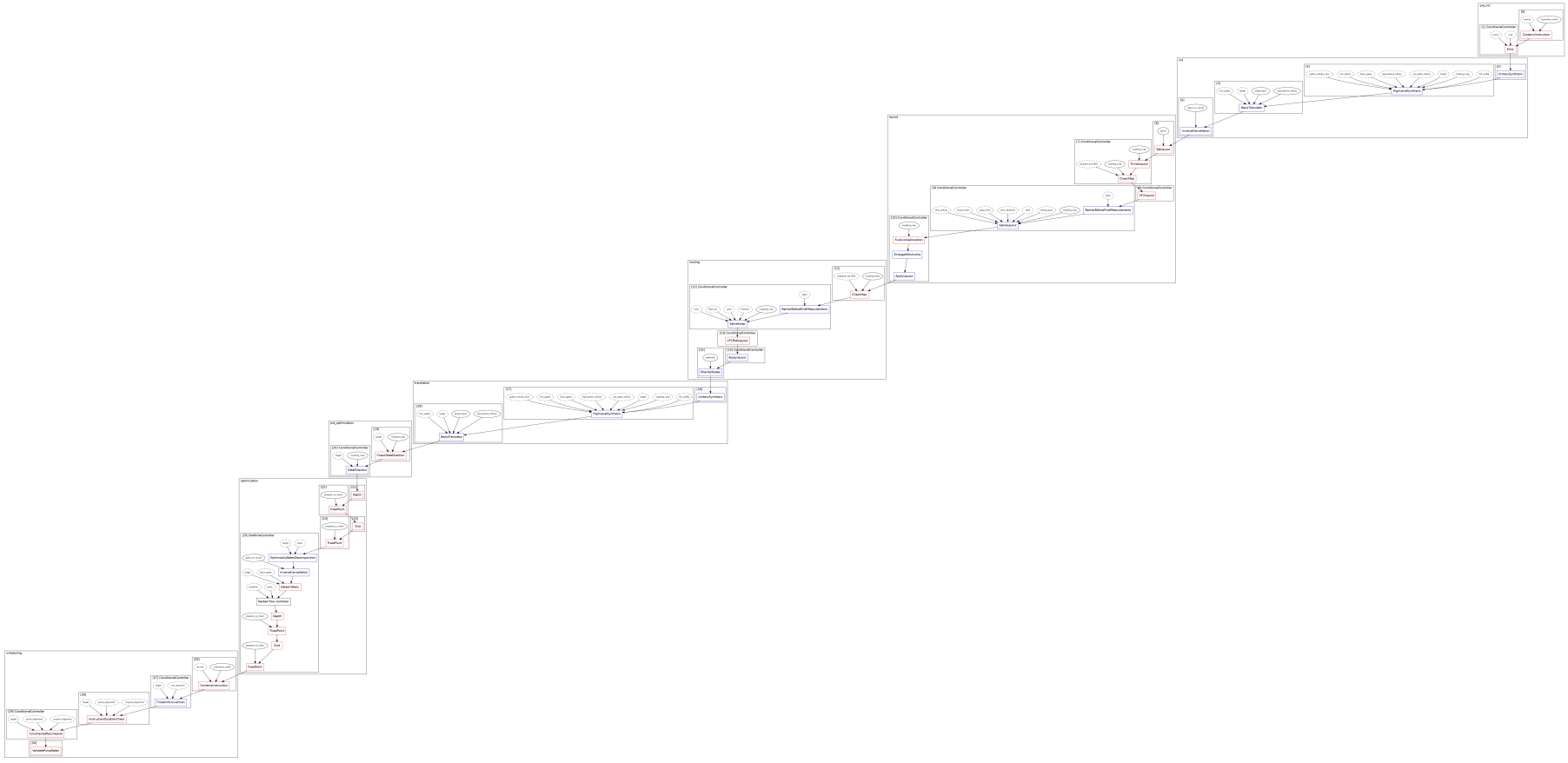
Out[18]:



In [19]:

```
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager

# There is no need to read this entire image, but this outputs all the steps in the transpile() call
# for optimization level 1
pm = generate_preset_pass_manager(1, backend, seed_transpiler=42)
pm.draw()
```

Out[19]:



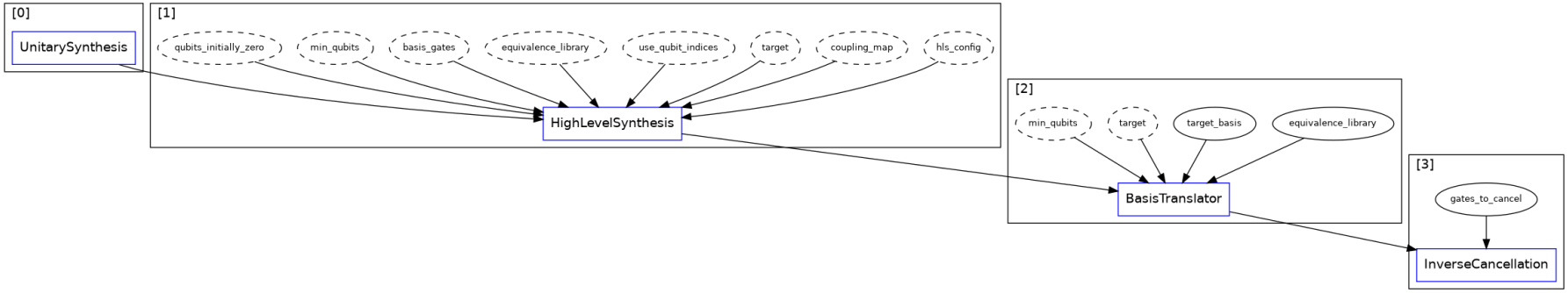
In [21]:

```
print(pm.stages)
```

```
('init', 'layout', 'routing', 'translation', 'optimization', 'scheduling')
```

In [22]:
```
1 pm.init.draw()
```

Out[22]:

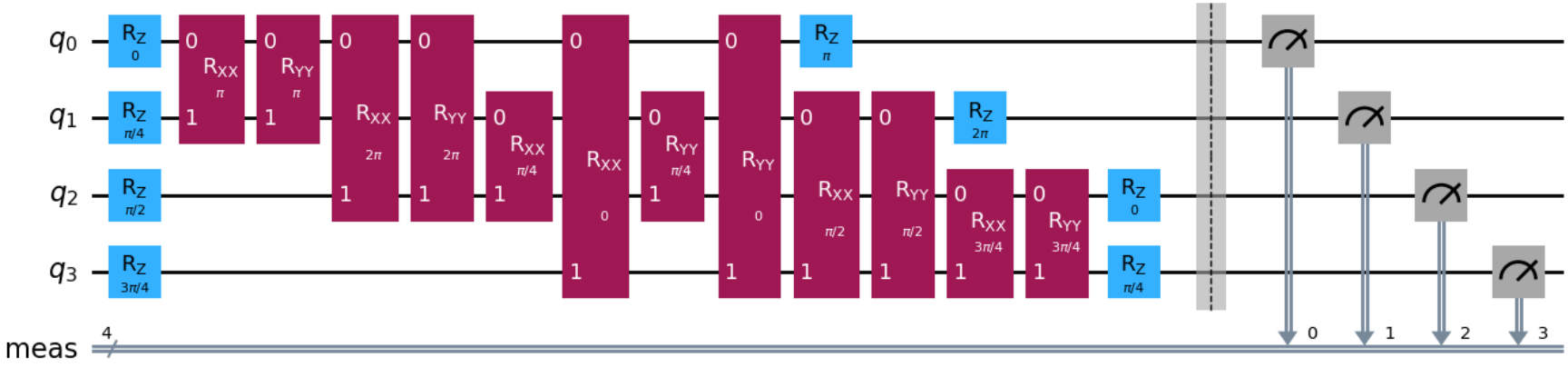


In [23]:

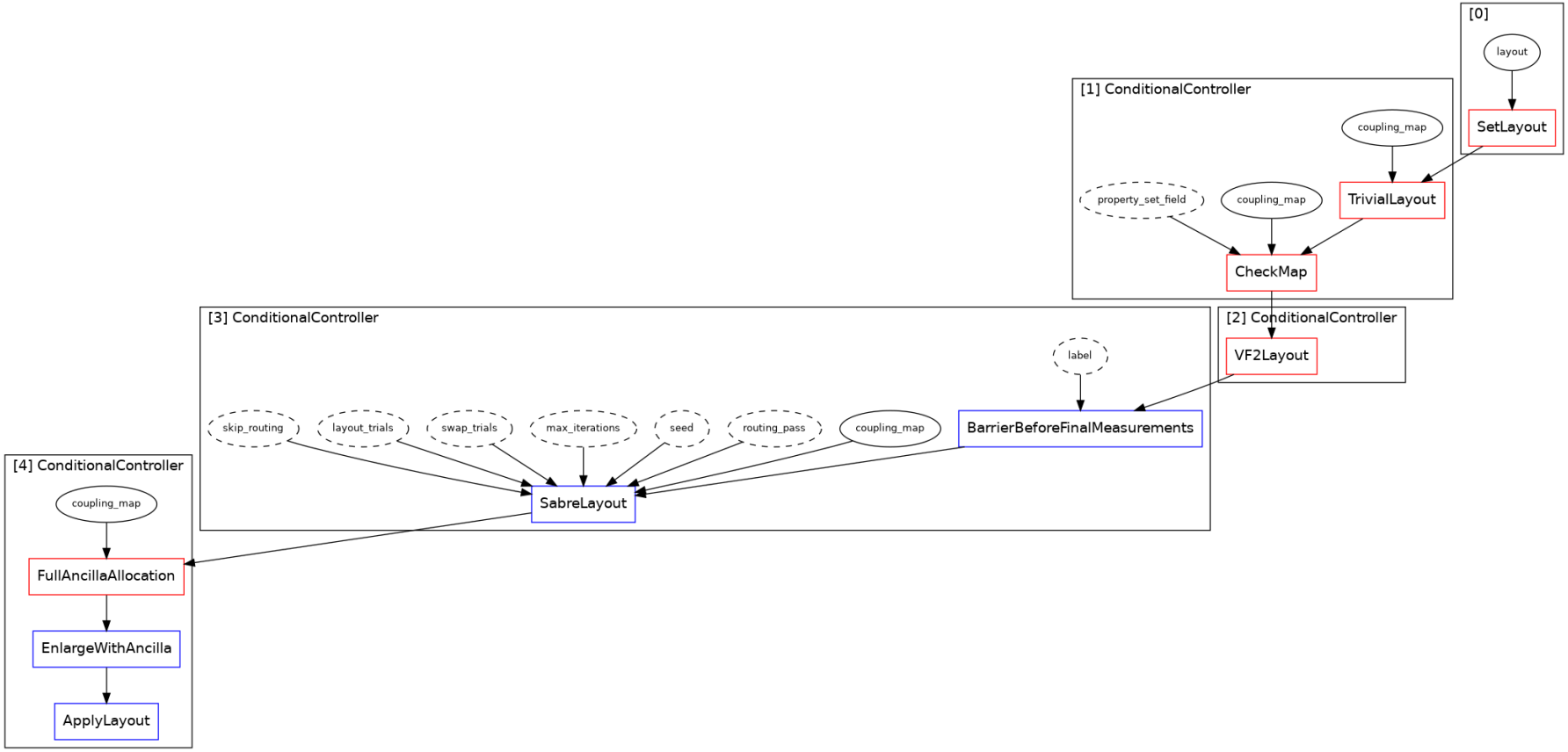
```
1 import logging
2
3 logger = logging.getLogger()
4 logger.setLevel("INFO")
5
6 init_out = pm.init.run(circuit)
7 init_out.draw('mpl', fold=-1)
```

```
INFO:qiskit.passmanager.base_tasks:Pass: UnitarySynthesis - 0.02551 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: HighLevelSynthesis - 4.45867 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: BasisTranslator - 0.07963 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: InverseCancellation - 0.20981 (ms)
```

Out[23]:

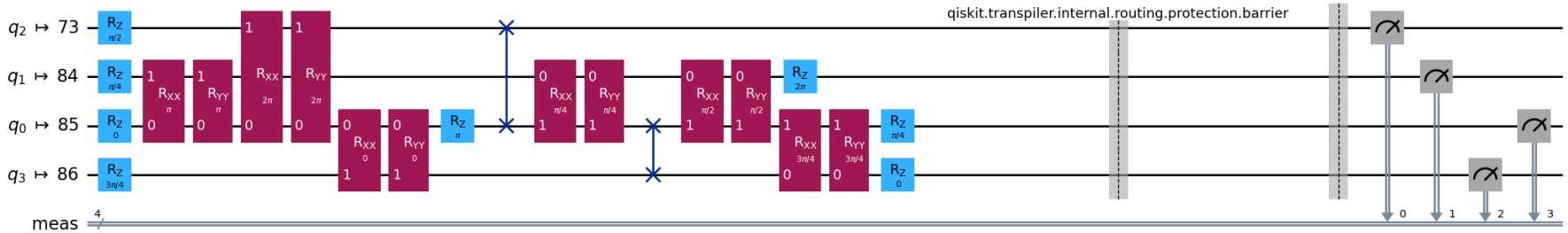


In [24]:
```
1 display(pm.layout.draw())
2 layout_out = pm.layout.run(init_out)
3 layout_out.draw('mpl', idle_wires=False, fold=-1)
```



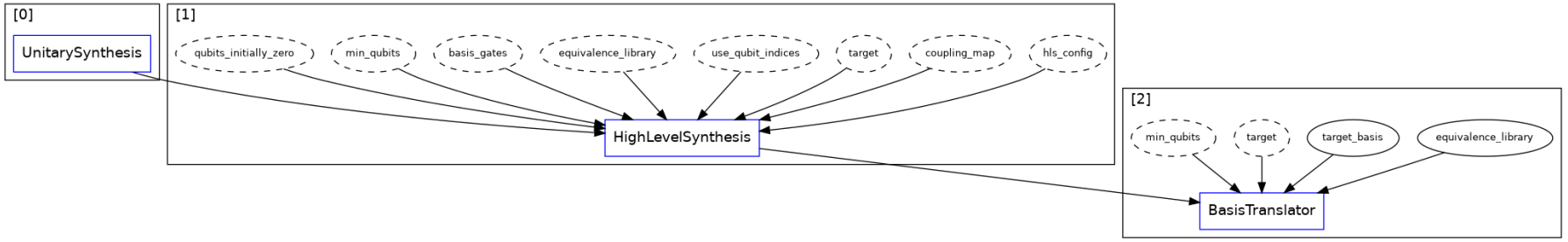

```
INFO:qiskit.passmanager.base_tasks:Pass: SetLayout - 0.01121 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: TrivialLayout - 0.08893 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: CheckMap - 0.11373 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: VF2Layout - 4.09365 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: BarrierBeforeFinalMeasurements - 0.09632 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: SabreLayout - 44.33680 (ms)
```

Out[24]:



In [25]:

```
1 display(pm.translation.draw())
2 basis_out = pm.translation.run(layout_out)
3 basis_out.draw('mpl', idle_wires=False, fold=-1)
```



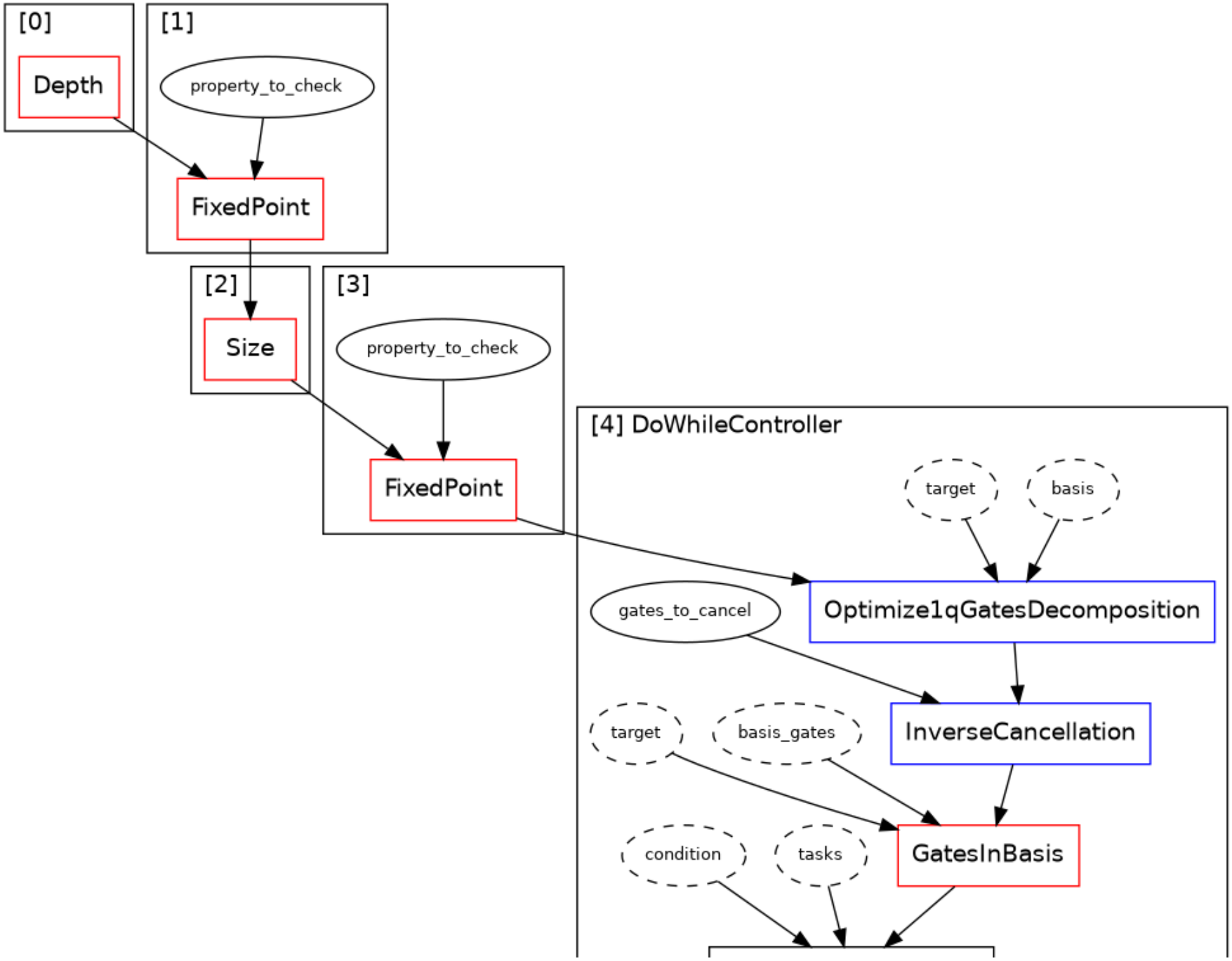

```
INFO:qiskit.passmanager.base_tasks:Pass: UnitarySynthesis - 0.01955 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: HighLevelSynthesis - 0.94318 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: BasisTranslator - 9.23014 (ms)
```

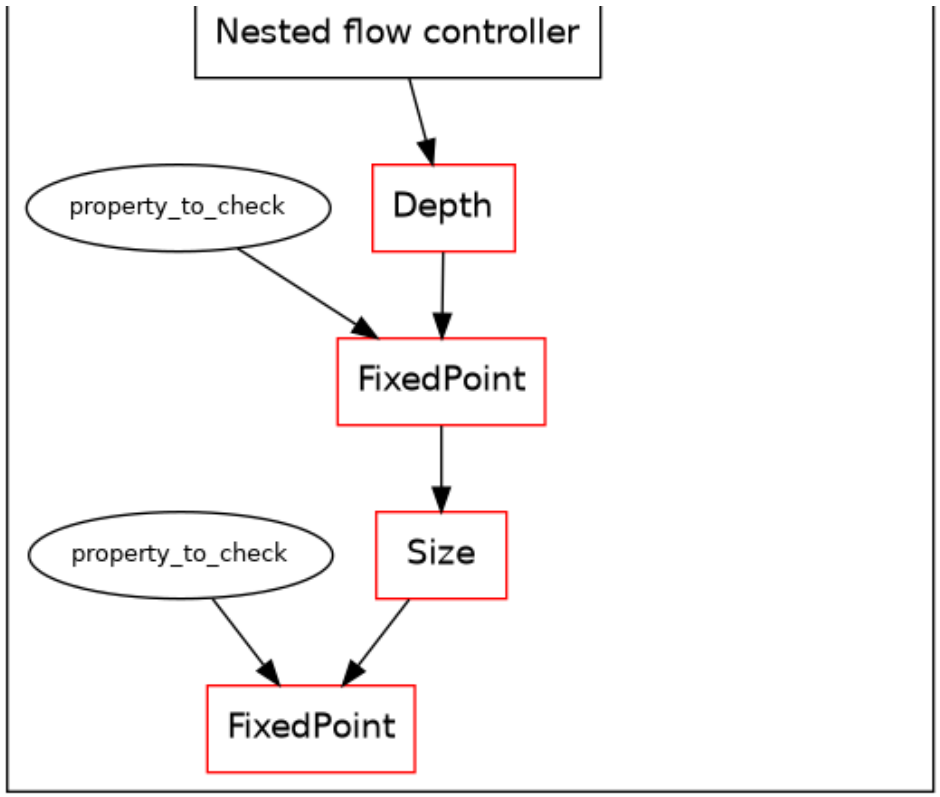
Out[25]:

In [26]:

```
# pm.pre_optimization.draw()
pm.optimization.draw()
```

Out[26]:


[0]
Depth
[1]
property_to_check
FixedPoint
[2]
Size
[3]
property_to_check
FixedPoint
[4] DoWhileController
target
basis
gates_to_cancel
Optimize1qGatesDecomposition
target
basis_gates
InverseCancellation
condition
tasks
GatesInBasis


Nested flow controller
property_to_check
Depth
FixedPoint
property_to_check
Size
FixedPoint

In [27]:
```
logger = logging.getLogger()
logger.setLevel("INFO")
pre_opt_out = pm.pre_optimization.run(basis_out)
opt_out = pm.optimization.run(pre_opt_out)
```

```
INFO:qiskit.passmanager.base_tasks:Pass: CheckGateDirection - 0.02956 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: GateDirection - 4.47989 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: Depth - 4.33159 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: FixedPoint - 0.01764 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: Size - 0.01955 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: FixedPoint - 0.01907 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: Optimize1qGatesDecomposition - 0.38862 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: InverseCancellation - 0.38767 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: GatesInBasis - 0.07463 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: Depth - 0.10514 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: FixedPoint - 0.02384 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: Size - 1.69039 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: FixedPoint - 0.01955 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: Optimize1qGatesDecomposition - 0.23031 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: InverseCancellation - 0.27680 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: GatesInBasis - 0.06771 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: Depth - 0.09441 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: FixedPoint - 0.01454 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: Size - 0.01550 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: FixedPoint - 0.01955 (ms)
```

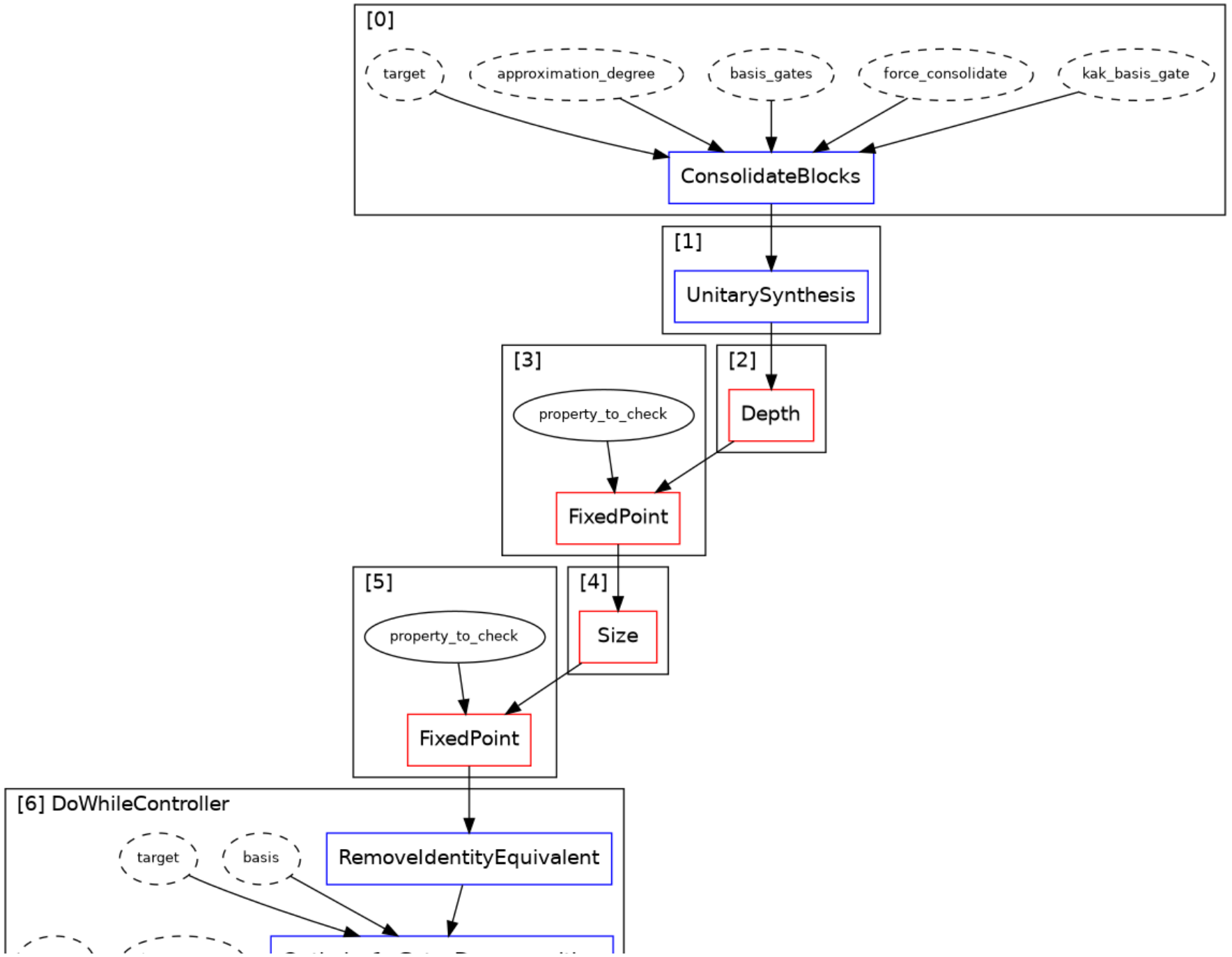
In [28]:
```
opt_out.draw('mpl', idle_wires=False, fold=-1)
```

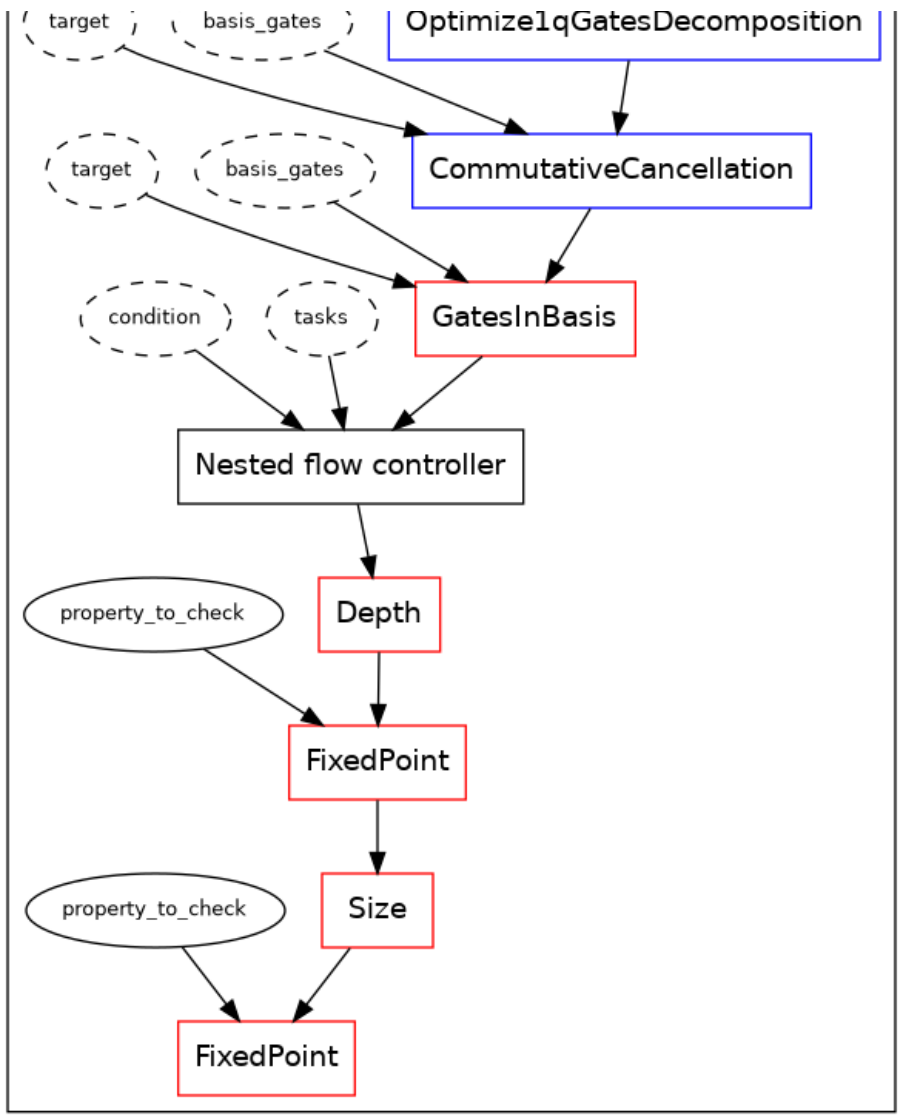
Out[28]:

In [29]:

```
pm2 = generate_preset_pass_manager(2, backend, seed_transpiler=42)
pm2.optimization.draw()
```

Out[29]:


[0]
target
approximation_degree
basis_gates
force_consolidate
kak_basis_gate
ConsolidateBlocks
[1]
UnitarySynthesis
[3]
property_to_check
FixedPoint
[2]
Depth
[5]
property_to_check
FixedPoint
[4]
Size
[6] DoWhileController
target
basis
RemoveIdentityEquivalent

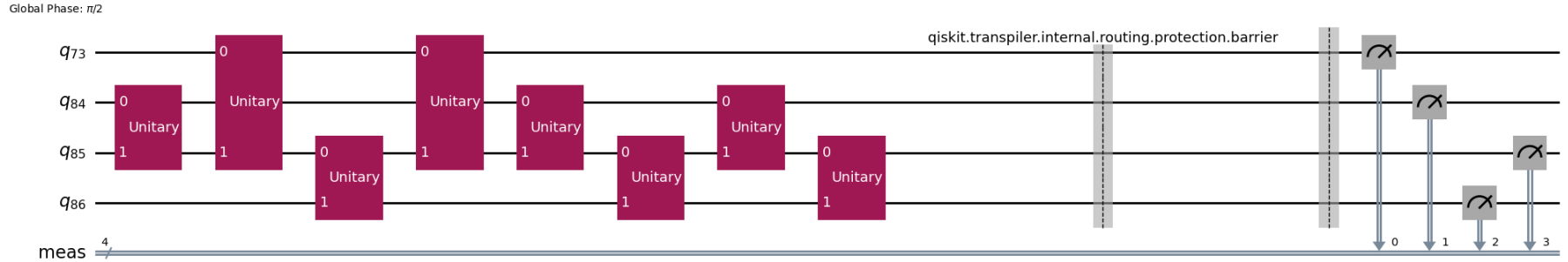

target
basis_gates
Optimize1qGatesDecomposition
target
basis_gates
CommutativeCancellation
condition
tasks
GatesInBasis
Nested flow controller
property_to_check
Depth
FixedPoint
property_to_check
Size
FixedPoint

In [30]:
```
from qiskit.transpiler import PassManager
from qiskit.transpiler.passes import Collect2qBlocks, ConsolidateBlocks, UnitarySynthesis

# Collect 2q blocks and consolidate to unitary when we expect that we can reduce the 2q gate count for t
consolidate_pm = PassManager([
    Collect2qBlocks(),
    ConsolidateBlocks(target=backend.target),
])
```

In [31]:
```
display(pre_opt_out.draw('mpl', idle_wires=False, fold=-1))
consolidated = consolidate_pm.run(pre_opt_out)
consolidated.draw('mpl', idle_wires=False, fold=-1)
```

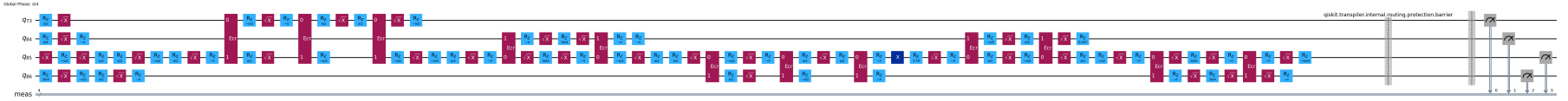
```
INFO:qiskit.passmanager.base_tasks:Pass: Collect2qBlocks - 0.43726 (ms)
INFO:qiskit.passmanager.base_tasks:Pass: ConsolidateBlocks - 1.13511 (ms)
```

Out[31]:



In [32]:
```
# Synthesize unitaries
UnitarySynthesis(target=backend.target)(consolidated).draw('mpl', idle_wires=False, fold=-1)
```

Out[32]:



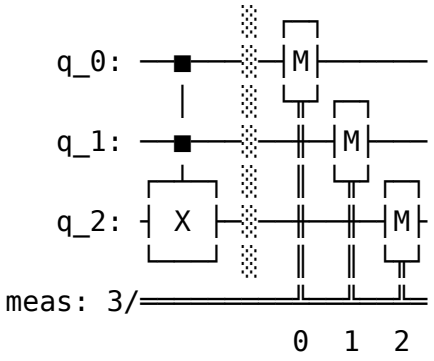
In [33]:
```
logger.setLevel("WARNING")
```

In [34]:
```
from qiskit.circuit import QuantumCircuit
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler
```

In [35]:
```
service = QiskitRuntimeService()
backend = service.backend('ibm_torino')
sampler = Sampler(backend)
```

In [36]:
```
circ = QuantumCircuit(3)
circ.ccx(0, 1, 2)
circ.measure_all()
circ.draw()
```

Out[36]:



In [37]:
```
1 sampler.run([circ])  # IBMInputValueError will be raised
```

```
---------------------------------------------------------------------------
IBMInputValueError                        Traceback (most recent call last)
Cell In[37], line 1
----> 1 sampler.run([circ])  # IBMInputValueError will be raised

File /path/to/virtual/environment/lib/python3.12/site-packages/qiskit_ibm_runtime/sampler.py:110, in SamplerV2.run(self, pubs, shots)
    106 coerced_pubs = [SamplerPub.coerce(pub, shots) for pub in pubs]
    108 validate_classical_registers(coerced_pubs)
--> 110 return self._run(coerced_pubs)

File /path/to/virtual/environment/lib/python3.12/site-packages/qiskit_ibm_runtime/base_primitive.py:157, in BasePrimitiveV2._run(self, pubs)
    155 for pub in pubs:
    156     if getattr(self._backend, "target", None) and not is_simulator(self._backend):
--> 157         validate_isa_circuits([pub.circuit], self._backend.target)
    159     if isinstance(self._backend, IBMBackend):
    160         self._backend.check_faulty(pub.circuit)

File /path/to/virtual/environment/lib/python3.12/site-packages/qiskit_ibm_runtime/utils/validations.py:90, in validate_isa_circuits(circuits, target)
     88 message = is_isa_circuit(circuit, target)
     89 if message:
---> 90     raise IBMInputValueError(
     91         message
     92         + " Circuits that do not match the target hardware definition are no longer "
     93         "supported after March 4, 2024. See the transpilation documentation "
     94         "(https://docs.quantum.ibm.com/guides/transpile) for instructions "
     95         "to transform circuits and the primitive examples "
     96         "(https://docs.quantum.ibm.com/guides/primitives-examples) to see "
     97         "this coupled with operator transformations."
     98     )

IBMInputValueError: 'The instruction ccx on qubits (0, 1, 2) is not supported by the target system. Circuits that do not match the target hardware definition are no longer supported after March 4, 2024. See the transpilation documentation (https://docs.quantum.ibm.com/guides/transpile) for instructions to transform circuits and the primitive examples (https://docs.quantum.ibm.com/guides/primitives-examples) to see this coupled with operator transformations.'
```

In [38]:
```
from qiskit.circuit import QuantumCircuit
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler
```

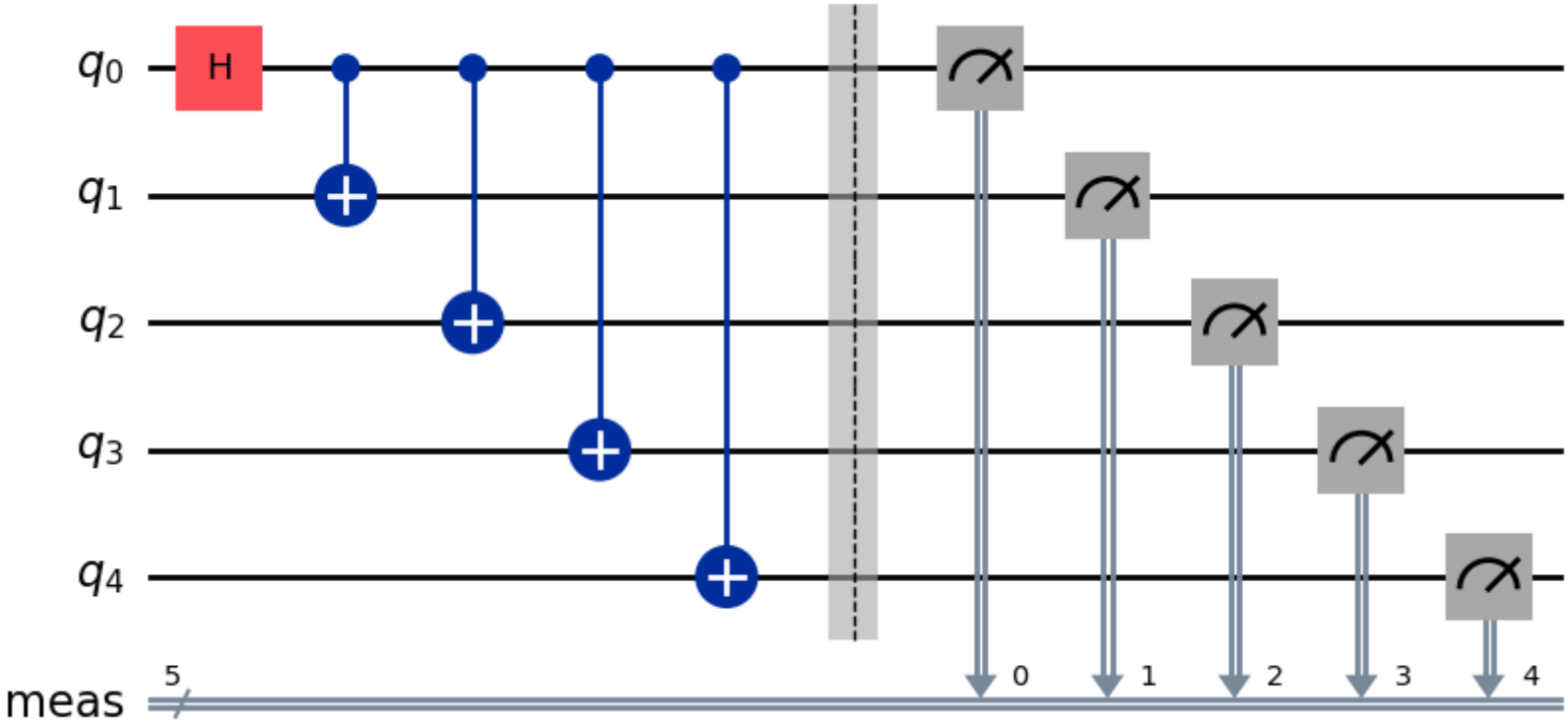
In [39]:
```
service = QiskitRuntimeService()
```

In [46]:
```
backend = service.backend('ibm_brisbane')
```

In [47]:

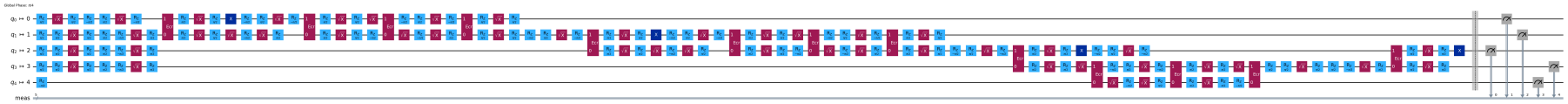
```
num_qubits = 5

ghz_circ = QuantumCircuit(num_qubits)
ghz_circ.h(0)
[ghz_circ.cx(0,i) for i in range(1,num_qubits)]
ghz_circ.measure_all()
ghz_circ.draw('mpl')
```

Out[47]:

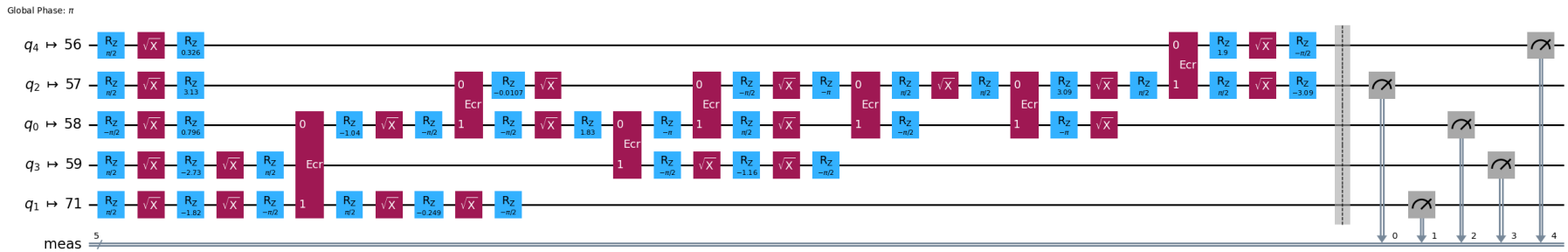


In [48]:
```
pm0 = generate_preset_pass_manager(optimization_level=0, backend=backend, seed_transpiler=777)
pm2 = generate_preset_pass_manager(optimization_level=2, backend=backend, seed_transpiler=777)
circ0 = pm0.run(ghz_circ)
circ2 = pm2.run(ghz_circ)
print("optimization_level=0:"); display(circ0.draw('mpl', idle_wires=False, fold=-1))
print("optimization_level=2:"); display(circ2.draw('mpl', idle_wires=False, fold=-1))
```

```
optimization_level=0:
```



```
optimization_level=2:
```



In [49]:
```
# run the circuits
sampler = Sampler(backend)
job = sampler.run([circ0, circ2], shots=10000)
print(f"Job ID: {job.job_id()}")
```

```
Job ID: d0vhenumu1ms73d8ijog
```

In [50]:
```
# REPLACE WITH YOUR OWN JOB IDS
job = service.job("d0vhenumu1ms73d8ijog")
```

In [51]:

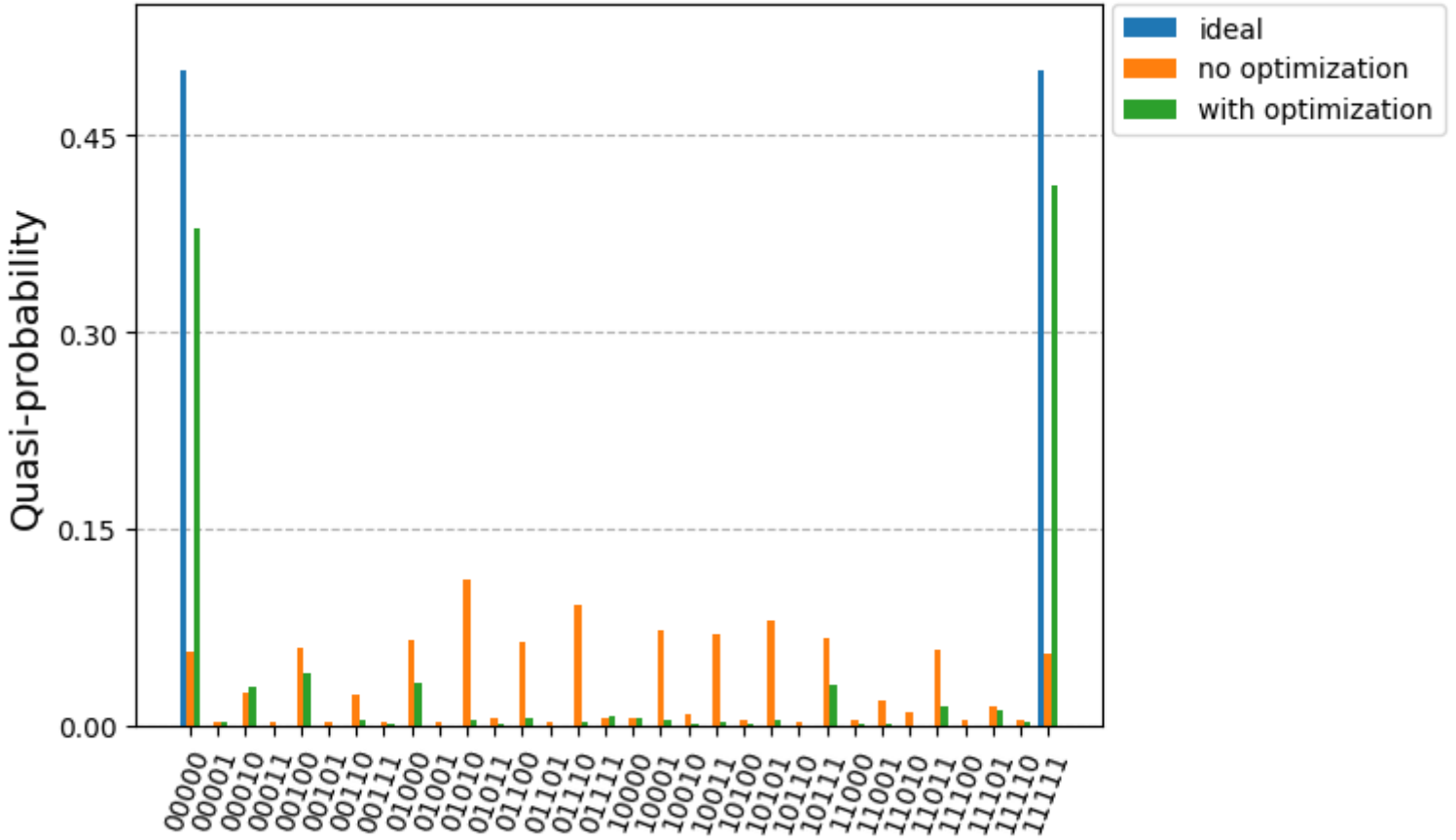
```
# get results
result = job.result()
unoptimized_result = result[0].data.meas.get_counts()
optimized_result = result[1].data.meas.get_counts()
print (result)
```

```
PrimitiveResult([SamplerPubResult(data=DataBin(meas=BitArray(<shape=(), num_shots=10000, num_bits=5>)), me
tadata={'circuit_metadata': {}}), SamplerPubResult(data=DataBin(meas=BitArray(<shape=(), num_shots=10000,
num_bits=5>)), metadata={'circuit_metadata': {}})], metadata={'execution': {'execution_spans': ExecutionSp
ans([DoubleSliceSpan(<start='2025-06-03 15:40:17', stop='2025-06-03 15:40:23', size=20000>)])}, 'version':
2})
```

In [52]:

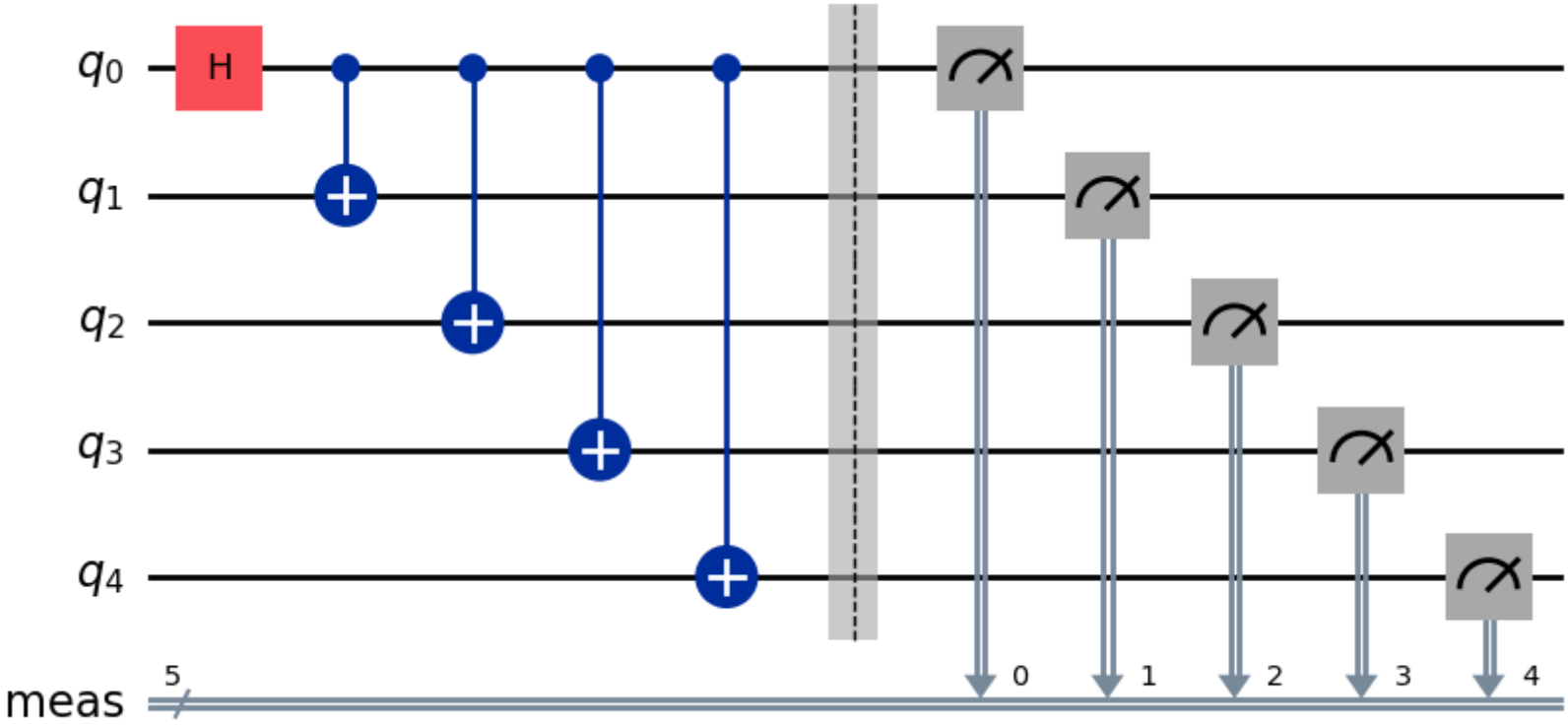
```
from qiskit.visualization import plot_histogram
# plot
sim_result = {'0'*5:0.5, '1'*5:0.5}
plot_histogram(
    [result for result in [sim_result, unoptimized_result, optimized_result]],
    bar_labels=False,
    legend=[
        "ideal",
        "no optimization",
        "with optimization",
    ],
)
```

Out[52]:



In [53]:
```
# Original GHZ circuit (naive synthesis)
ghz_circ.draw('mpl')
```

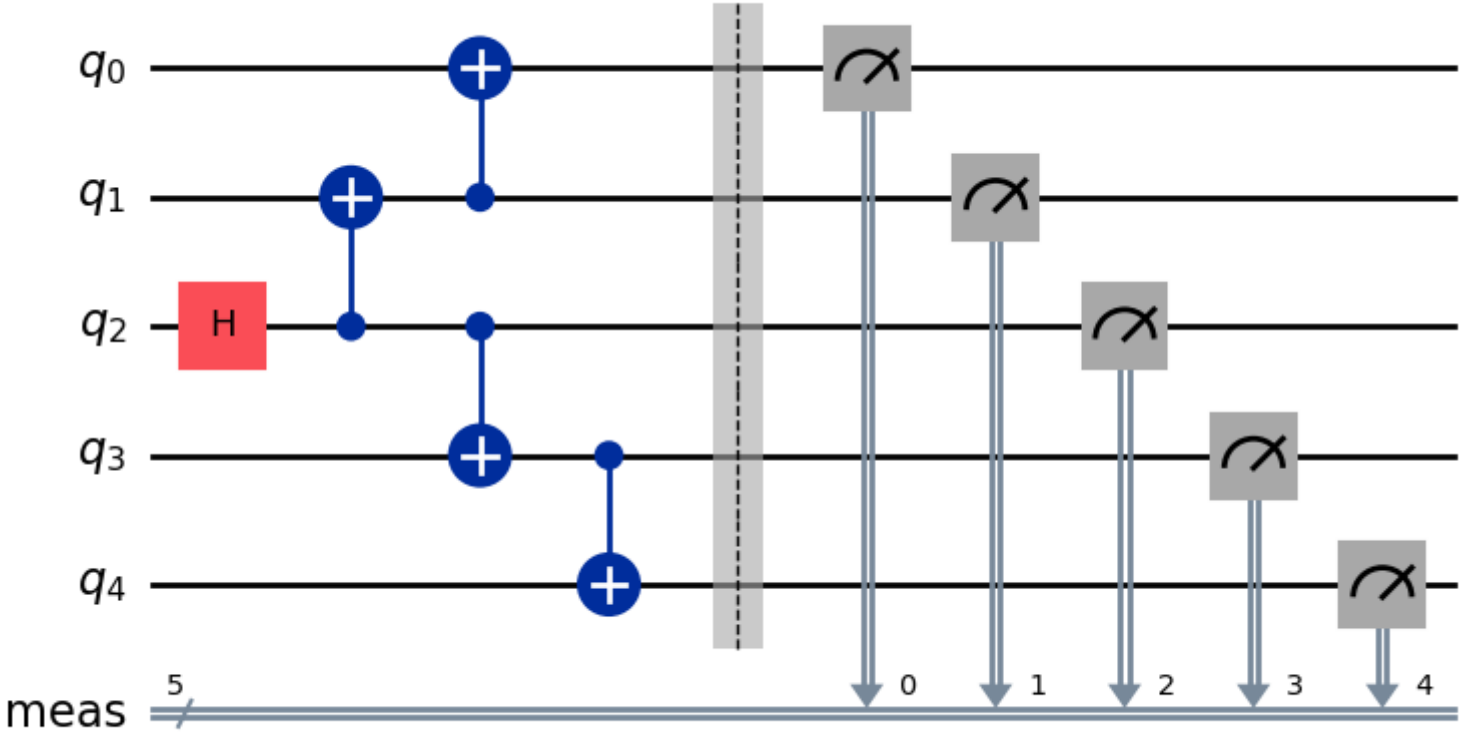
Out[53]:



In [55]:

```
# A better GHZ circuit (smarter synthesis), you learned in a previous lecture
ghz_circ2 = QuantumCircuit(5)
ghz_circ2.h(2)
ghz_circ2.cx(2, 1)
ghz_circ2.cx(2, 3)
ghz_circ2.cx(1, 0)
ghz_circ2.cx(3, 4)
ghz_circ2.measure_all()
ghz_circ2.draw('mpl')
```

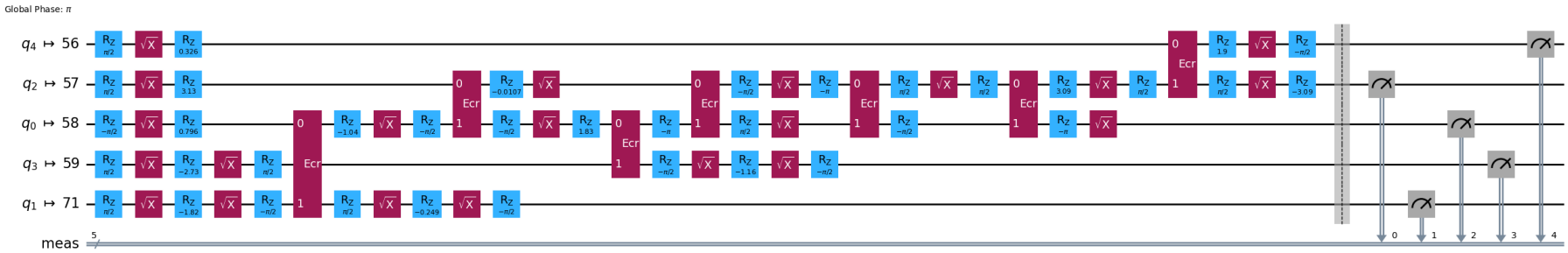
Out[55]:



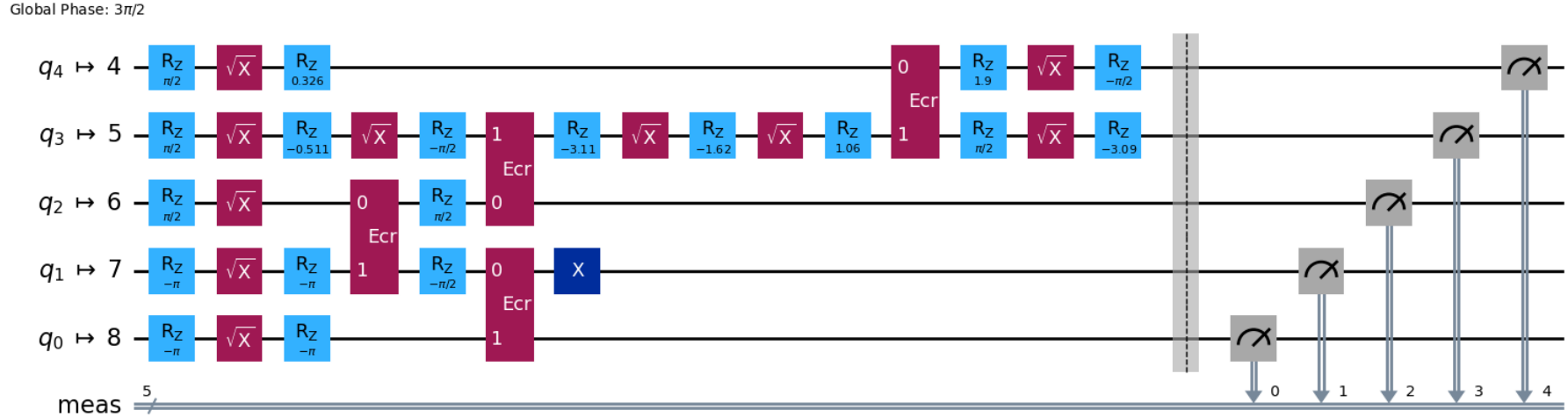
In [57]:

```
1 circ_org = pm2.run(ghz_circ)
2 circ_new = pm2.run(ghz_circ2)
3 print("original synthesis:"); display(circ_org.draw('mpl', idle_wires=False, fold=-1))
4 print("new synthesis:"); display(circ_new.draw('mpl', idle_wires=False, fold=-1))
```

original synthesis:



new synthesis:



In [58]:
```
# run the circuits
sampler = Sampler(backend)
job = sampler.run([circ_org, circ_new], shots=10000)
print(f"Job ID: {job.job_id()}")
```

```
Job ID: d0vhgta4p3dc73f5d5t0
```

In [59]:
```
# REPLACE WITH YOUR OWN JOB IDS
job = service.job("d0vhgta4p3dc73f5d5t0")
```

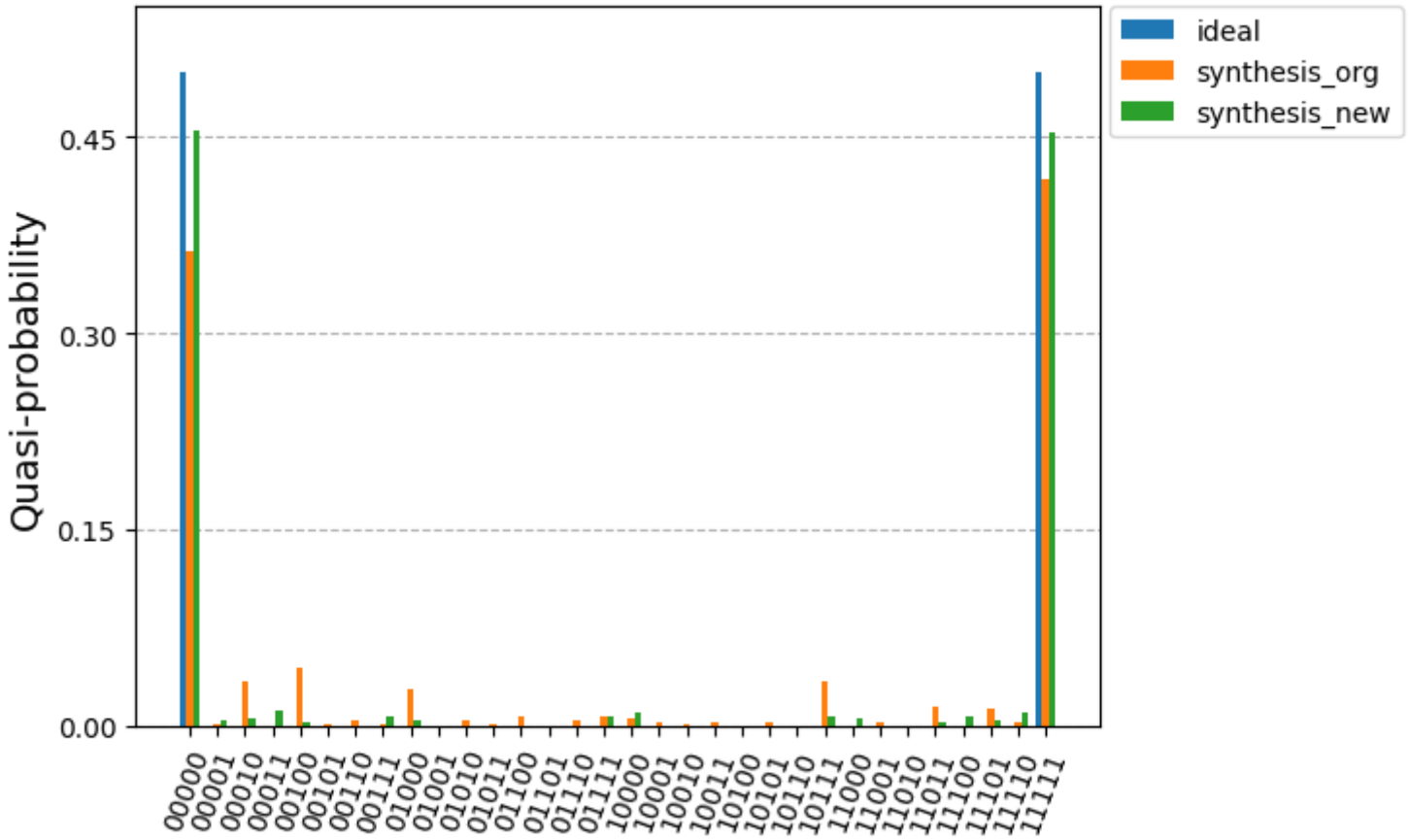
In [60]:
```
# get results
result = job.result()
synthesis_org_result = result[0].data.meas.get_counts()
synthesis_new_result = result[1].data.meas.get_counts()
print(result)
```

```
PrimitiveResult([SamplerPubResult(data=DataBin(meas=BitArray(<shape=(), num_shots=10000, num_bits=5>)), me
tadata={'circuit_metadata': {}}), SamplerPubResult(data=DataBin(meas=BitArray(<shape=(), num_shots=10000,
num_bits=5>)), metadata={'circuit_metadata': {}})], metadata={'execution': {'execution_spans': ExecutionSp
ans([DoubleSliceSpan(<start='2025-06-03 15:44:54', stop='2025-06-03 15:45:00', size=20000>)])}, 'version':
2})
```

In [61]:

```
# plot
sim_result = {'0'*5:0.5, '1'*5:0.5}
plot_histogram(
    [result for result in [sim_result, synthesis_org_result, synthesis_new_result]],
    bar_labels=False,
    legend=[
        "ideal",
        "synthesis_org",
        "synthesis_new",
    ],
)
```

Out[61]:



In [12]:
```
from qiskit import QuantumCircuit, transpile
from qiskit.circuit import Parameter
from qiskit.circuit.library.standard_gates import UGate
phi, theta, lam = Parameter("φ"), Parameter("θ"), Parameter("λ")
```

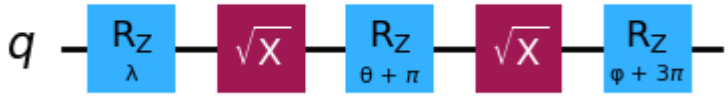
In [63]:
```
qc = QuantumCircuit(1)
qc.append(UGate(theta, phi, lam), [0])
qc.draw(output="mpl")
```

Out[63]:

In [64]:
```
transpile(qc, basis_gates=["rz", "sx"]).draw(output="mpl")
```

Out[64]:

Global Phase: λ/2 + φ/2 - π/2



In [3]:

```
from qiskit import QuantumCircuit
qc = QuantumCircuit(1)
qc.x(0)
qc.y(0)
qc.z(0)
qc.rx(1.23, 0)
qc.ry(1.23, 0)
qc.rz(1.23, 0)
qc.h(0)
qc.s(0)
qc.t(0)
qc.sx(0)
qc.sdg(0)
qc.tdg(0)
qc.draw(output="mpl")
```

Out[3]:

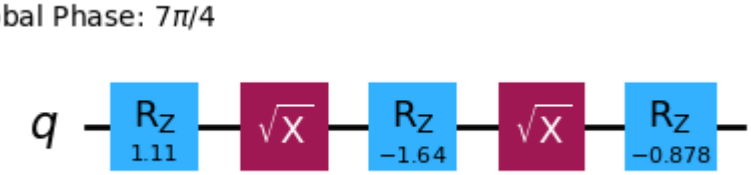
In [4]:

```
from qiskit.quantum_info import Operator
Operator(qc)
```

```
Operator([[ 0.45292511-0.57266982j, -0.66852684-0.14135058j],
          [ 0.14135058+0.66852684j, -0.57266982+0.45292511j]],
         input_dims=(2,), output_dims=(2,))
```

In [6]:
```
from qiskit import transpile
qc_opt = transpile(qc, basis_gates=["rz", "sx"])
qc_opt.draw(output="mpl")
```

Out[6]:



In [7]:
```
Operator(qc_opt)
```

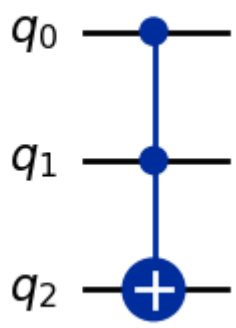
```
Operator([[-0.45292511+0.57266982j,  0.66852684+0.14135058j],
          [-0.14135058-0.66852684j,  0.57266982-0.45292511j]],
         input_dims=(2,), output_dims=(2,))
```

In [8]:
```
Operator(qc).equiv(Operator(qc_opt))
```

Out[8]:
```
True
```

In [9]:
```
1 qc = QuantumCircuit(3)
2 qc.ccx(0, 1, 2)
3 qc.draw(output="mpl")
```

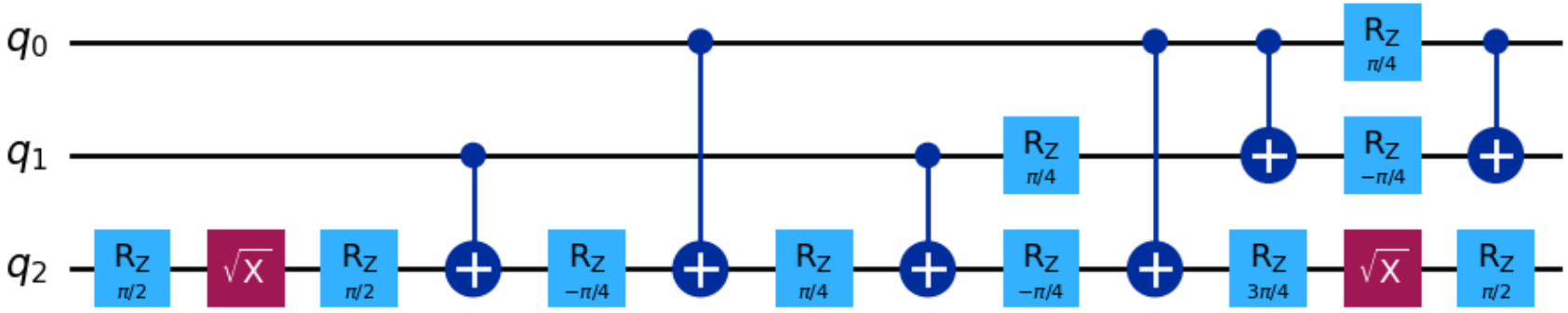
Out[9]:



In [10]:
```
1 from qiskit import QuantumCircuit, transpile
2 qc = QuantumCircuit(3)
3 qc.ccx(0, 1, 2)
4 qc = transpile(qc, basis_gates=["rz", "sx", "cx"])
5 qc.draw(output="mpl")
```

Out[10]:

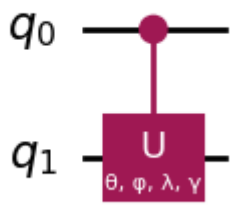
Global Phase: 5π/8



In [13]:

```
from qiskit.circuit.library.standard_gates import CUGate
phi, theta, lam, gamma = Parameter("φ"), Parameter("θ"), Parameter("λ"), Parameter("γ")
qc = QuantumCircuit(2)
# qc.cu(theta, phi, lam, gamma, 0, 1)
qc.append(CUGate(theta, phi, lam, gamma), [0, 1])
qc.draw(output="mpl")
```

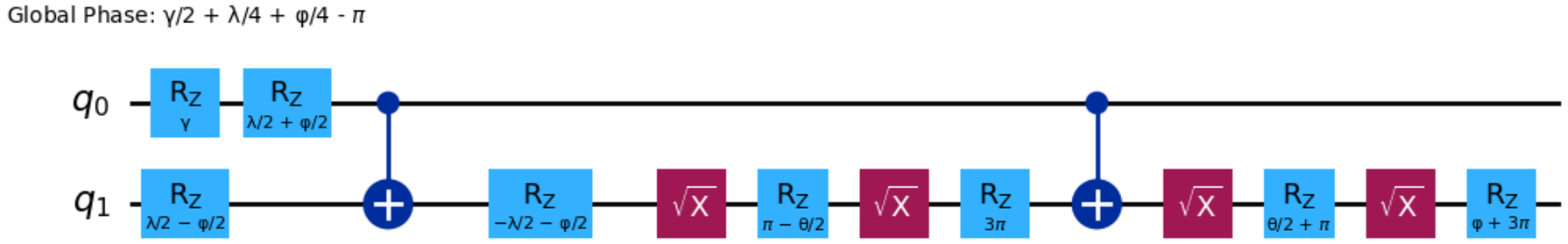
Out[13]:



In [14]:

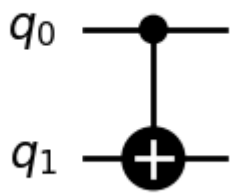
```
from qiskit.circuit.library.standard_gates import CUGate
phi, theta, lam, gamma = Parameter("φ"), Parameter("θ"), Parameter("λ"), Parameter("γ")
qc = QuantumCircuit(2)
qc.append(CUGate(theta, phi, lam, gamma), [0, 1])
qc = transpile(qc, basis_gates=["rz", "sx", "cx"])
qc.draw(output="mpl")
```

Out[14]:



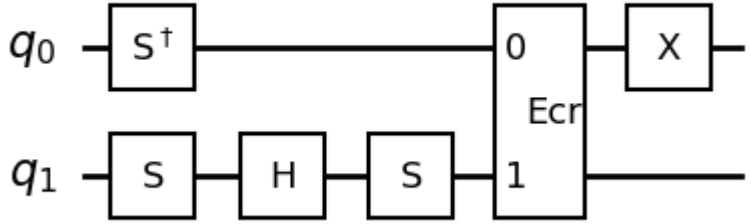
In [15]:
```
qc = QuantumCircuit(2)
qc.cx(0, 1)
qc.draw(output="mpl", style="bw")
```

Out[15]:



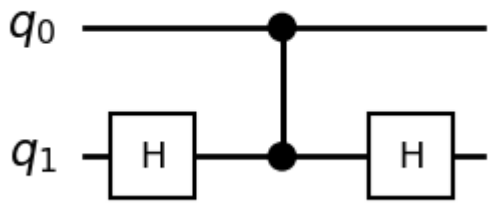
In [16]:
```
qc = QuantumCircuit(2)
qc.cx(0, 1)
transpile(qc, basis_gates=[ "x", "s", "h", "sdg", "ecr"]).draw(output="mpl", style="bw")
```

Out[16]:



In [17]:
```
1 qc = QuantumCircuit(2)
2 qc.cx(0, 1)
3 transpile(qc, basis_gates=["h", "cz"]).draw(output="mpl", style="bw")
```

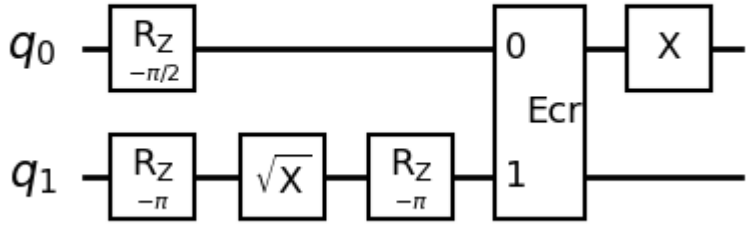
Out[17]:



In [18]:
```
1 qc = QuantumCircuit(2)
2 qc.cx(0, 1)
3 transpile(qc, basis_gates=[ "rz", "sx", "x", "ecr"]).draw(output="mpl", style="bw")
```

Out[18]:

Global Phase: $\pi/2$



In [19]:

```
1 qc = QuantumCircuit(2)
2 qc.cx(0, 1)
3 transpile(qc, basis_gates=[ "rz", "sx", "x", "cz"]).draw(output="mpl", style="bw")
```

Out[19]:

Global Phase: $\pi/2$