EDA-Q

This tutorial introduces the usage of EDA-Q.

Table of Contents

Basic Interface

EDA-Q abstracts each designed object into a class, and operations are performed based on these classes. These design objects are located in the `./api` directory, including `Design`, `Topology`, `EquivalentCircuit`, and `Gds`. In most cases, you should use the `Design` class as the foundation for the entire chip design. The usage is as follows:

```
from api.design import Design
design = Design()
```

Topology Design

EDA-Q integrates topology design functionality, with the `Topology` class including a series of modules for automated topology design. Typically, the `Topology` class relies on the `Design` class for designing. The usage is as follows:

```
# Generate a topology by specifying the number of qubits. The qubits layout will
aim to form a square.
design.generate_topology(qubits_num=144)

# Generate a topology by specifying the number of qubits and the number of
columns.
design.generate_topology(qubits_num=50, topo_col=10)

# Generate a topology by specifying the number of rows and columns.
design.generate_topology(topo_col=13, topo_row=12)
design.generate_topology1(num_cols=13, num_rows=12)
```

```
# Generate a customized topology structure based on a quantum circuit algorithm.
design.generate_topology(qasm_path="./qasm_files/dj_indep_qiskit_8.qasm")

# Generate a customized topology structure with specified rows and columns.
design.generate_topology(qasm_path="./qasm_files/dj_indep_qiskit_8.qasm", row=3,
col=3)

# Generate a regular hexagonal topology structure.
design.generate_topology(num=7, shape="hex")

# Add a topology edge.
design.topology.add_edge(q0_name="q0", q1_name="q2")

# Remove a topology edge.
design.topology.remove_edge(edge=["q0", "q2"])

# Add multiple topology edges in batch.
design.topology.add_edges(edges=[["q0", "q1"], ["q0", "q9"], ["q1", "q2"]])

# Randomly generate topology edges.
design.topology.generate_random_edges(edges_num=100)

# Generate all topology edges
design.topology.generate_full_edges()

# Generate topology edges for a specific row.
design.topology.batch_add_edges(y=4)

# Generate topology edges for a specific column.
design.topology.batch_add_edges(x=4)

# Generate topology edges for specific rows/columns.
design.topology.batch_add_edges_list(y=[0, 1, 2, 3])
design.topology.batch_add_edges_list(x=[1, 2])
design.topology.batch_add_edges_list(y=[0, 1, 2, 3], x=[1, 2])

# Generate all edges for a hexagonal structure.
design.topology.generate_hex_full_edges()

# Find the name of a qubit based on its coordinates.
design.topology.find_qname(pos=[1, 2])

# Check if a specific edge exists in the current topology structure.
design.topology.if_edge(edge=["q0", "q3"])

# Display the current topology structure image.
design.topology.show_image()

# Save the current topology structure image.
design.topology.save_image(path="./image/topo.png")

# Display topology structure parameters.
design.topology.show_options(topology=True)
```

Equivalent Circuit Design

```
# Create an equivalent circuit based on the existing topology.
design.generate_equivalent_circuit()

# Display the equivalent circuit structure.
design.equivalent_circuit.show()

# Clear the equivalent circuit components.
design.equivalent_circuit.clear()

# Modify qubit parameters.
design.equivalent_circuit.change_qubit_options(qubit_name="q0", value=65)

# Modify coupling component parameters.
design.equivalent_circuit.change_coupling_options(coupling_line_name="c21",
op_name="L", op_value=30)
design.equivalent_circuit.change_coupling_options(coupling_line_name="c21",
op_name="C", op_value=50)

# Output qubit parameters.
design.equivalent_circuit.find_qubit_options(qubit_name="q0")

# Output coupling component parameters.
design.equivalent_circuit.find_coupling_options(coupling_line_name="c21",
op_name="L")
design.equivalent_circuit.find_coupling_options(coupling_line_name="c21",
op_name="C")

# Calculate qubit parameters.
design.equivalent_circuit.caculate_qubits_parms(f_q=65, Ec=30)

# Save the equivalent circuit diagram as an image.
design.equivalent_circuit.save_image(path="./picture/equivalent_circuit.png")  #
This feature currently has errors.
```

Generate Qubits

```
# Manually generate a readout line
design.gds.readout_lines.add(name="readout_line0", type="ReadoutCavityPlus")

# Generate a specified number of qubits with a layout as close to a square as
possible
design.generate_qubits(num=10)

# Generate a specified number of qubits and specify the number of columns in the
layout
design.generate_qubits(num=8, col=4)

# Generate a specified number of qubits, specify the number of columns, and the
spacing between adjacent qubits
design.generate_qubits(num=8, col=4, dist=2000)
```

```
# Generate a specified number of qubits, specify the number of columns, spacing,
and the geometric parameters of the qubits
from addict import Dict
options = Dict(
    type="Transmon",
    chip="chip0",
    cpw_width=[10]*6,
    cpw_extend=[100]*6,
    width=455,
    height=200,
    gap=30,
    pad_options=[1]*6,
    pad_gap=[15]*6,
    pad_width=[125]*6,
    pad_height=[30]*6,
    cpw_pos=[[0,0]]*6,
    control_distance=[10]*4,
    subtract_gap=20,
    subtract_width=600,
    subtract_height=600
)
design.generate_qubits(num=8, col=4, dist=2000, geometric_ops=options)

# Generate a specified number of qubits and specify the type of qubits
design.generate_qubits(num=8, qubits_type="Xmon")

# Generate qubits based on the already created topology
design.generate_qubits(topo_positions=True)

# Generate qubits based on the already created topology and specify the type of
qubits
design.generate_qubits(topo_positions=True, qubits_type="Transmon")

# Generate qubits based on the already created topology and specify the type of
qubits and chip name
design.generate_qubits(topo_positions=True, chip_name="chip0",
qubits_type="Transmon")

# Generate qubits based on the already created topology and specify the spacing
and type of qubits
design.generate_qubits(topo_positions=True, dist=2000, qubits_type="Transmon")

# Generate qubits based on the already created topology and specify the chip
name, spacing, and type of qubits
design.generate_qubits(topo_positions=True, chip_name="chip0", dist=2000,
qubits_type="Transmon")

# Generate qubits based on the existing topology
design.generate_qubits_from_topo1(qubits_type="Transmon", chip_name="chip0",
dist=2000)

# Generate qubits based on the already created topology, and specify the type of
qubits and geometric parameters
options = Dict(
    chip="chip0",
    cpw_width=[10]*6,
```

```
    cpw_extend=[100]*6,
    width=455,
    height=200,
    gap=30,
    pad_options=[1]*6,
    pad_gap=[15]*6,
    pad_width=[125]*6,
    pad_height=[30]*6,
    cpw_pos=[[0,0]]*6,
    control_distance=[10]*4,
    subtract_gap=20,
    subtract_width=600,
    subtract_height=600
)
design.generate_qubits(topo_positions=True, qubits_type="Transmon",
geometric_ops=options)
```

Generate Chip

```
# Manually generate a chip
design.gds.chips.add(name="chip0", type="RecChip")

# Generate a chip based on the quantum bit scale
design.generate_chip(qubits=True)

# Generate a chip based on the quantum bit scale and specify the chip name
design.generate_chip(qubits=True, chip_name="chip0")

# Generate a chip based on the quantum bit scale and specify the distance from
the center of the outermost quantum bits to the chip edge
design.generate_chip(qubits=True, dist=4000)

# Generate a chip based on the quantum bit scale, specify the chip name, and the
distance from the center of the outermost quantum bits to the chip edge
design.generate_chip(qubits=True, chip_name="chip0", dist=4000)

# Use the center point of the quantum bit layout as the chip center point,
specify the chip name, and define the chip height and width
design.generate_chip(qubits=True, chip_name="chip0", height=20000, width=20000)
```

Generate Coupling Lines

```
design.generate_coupling_lines(topology=True, qubits=True)
design.generate_coupling_lines(topology=True, qubits=True,
cpls_type="CouplingCavity")
design.generate_coupling_lines(topology=True, qubits=True,
cpls_type="CouplingCavity", chip_name="chip1")
# Generate coupling components based on existing topologies and qubits
design.generate_coupling_lines_from_topo_and_qubits1(cpls_type="CouplingLineStrai
ght",
                                                     chip_name="chip0")
```

Generate Readout Lines

```
# Generate readout cavity based on the existing qubits
design.generate_readout_lines(qubits=True)

# Generate readout cavity and specify the readout cavity type
design.generate_readout_lines(qubits=True, rdls_type="ReadoutCavityPlus")

# Generate readout cavity by specifying `pin_num` (requires the corresponding
`readout_pins` to be generated beforehand)
design.generate_readout_lines(qubits=True, rdls_type="ReadoutCavityPlus",
pin_num=1)

# Specify the chip where the readout cavity is generated
design.generate_readout_lines(qubits=True, rdls_type="ReadoutCavityPlus",
chip_name="chip0")

# Specify the type and geometric parameters of the readout cavity
from addict import Dict
options = Dict(
    chip = "chip0",  # Chip name
    coupling_length = 300,  # Coupling length
    coupling_dist = 26.5,  # Coupling distance
    width = 10,  # Readout cavity width
    gap = 6,  # Gap
    outline = [],  # Outline
    start_dir = "up",  # Starting direction
    height = 700,  # Height
    length = 3000,  # Total length
    start_length = 300,  # Starting length
    space_dist = 200,  # Space distance
    radius = 90,  # Corner radius
    orientation = 90  # Orientation
)
design.generate_readout_lines(qubits=True, rdls_type="ReadoutCavityPlus",
geometric_options=options)

# Specify the type, chip, and geometric parameters of the readout cavity
from addict import Dict
options = Dict(
    chip = "chip0",  # Chip name
    coupling_length = 300,  # Coupling length
    coupling_dist = 26.5,  # Coupling distance
    width = 10,  # Readout cavity width
    gap = 6,  # Gap
    outline = [],  # Outline
    start_dir = "up",  # Starting direction
    height = 700,  # Height
    length = 3000,  # Total length
    start_length = 300,  # Starting length
    space_dist = 200,  # Space distance
    radius = 90,  # Corner radius
    orientation = 90  # Orientation
)
```

```
design.generate_readout_lines(qubits=True, rdls_type="ReadoutCavityPlus",
chip_name="chip0", geometric_options=options)
```

Generate Control Lines

```
# Manually generate a control line
design.control_lines.add(name="ctl0", type="ChargeLine")
```

Generate Transmission Lines

```
# Manually generate a transmission line
design.transmission_lines.add(name="tml0", type="TransmissionPath")
```

Auto Routing

```
# Automatically route to generate transmission lines (only applicable under
certain conditions)
design.routing(method="Control_off_chip_routing")

# Automatically route to generate transmission lines and specify the chip where
they are located (only applicable under certain conditions)
design.routing(method="Control_off_chip_routing", chip_name="chip0")

# Automatically arrange control lines and pins using IBM flip-chip method
design.routing(method="Flipchip_routing_IBM", chip_name="chip1",
ctls_type="ControlLineCircle1")

# Automatically arrange transmission lines, control lines, and pins using flip-
chip method
design.routing(method="Flipchip_routing",
               chip_name="chip1",
               ctls_type="ChargeLine",
               pins_type="LaunchPad",
               tmls_type="TransmissionPath")
```

Simulation

```
# Perform capacitance simulation for Xmon-type qubits in a flip-chip structure
design.simulation(sim_module="Flipchip_Xmon", ctl_name="charge_line_0",
q_name="q0")

# Perform capacitance simulation for Xmon-type qubits in a flip-chip structure
and specify the path to save the capacitance matrix
design.simulation(sim_module="Flipchip_Xmon", ctl_name="charge_line_0",
q_name="q0", path="./results.txt")

# Perform capacitance simulation for Xmon-type qubits in a planar structure
design.simulation(sim_module="PlaneXmonSim", qubit_name="q0")

# Perform capacitance simulation for Transmon-type qubits
design.simulation(sim_module="TransmonSim", frequency=5.6, qubit_name="q0")
```

```
# Perform capacitance simulation for Transmon-type qubits and specify the path to save the capacitance matrix
design.simulation(sim_module="TransmonSim", frequency=5.6, qubit_name="q0", path="./results.txt")
```

Modify The Gds Layout

```
# Only the usage of manipulating qubits is listed here, other types of components are similar to use

# Calculate the general parameters of the component framework
design.gds.qubits.calc_general_ops()
# Move every components
design.gds.qubits.move(pos_name="gds_pos", dx=-200, dy=500)
# Modifies the value of a single option
design.gds.qubits.change_option(op_name="width", op_value="800")
# Modifies the values of multiple options
options = Dict(
    cpw_width=[15]*6,
    cpw_extend=[80]*6,
    width=600,
    height=300,
    gap=10,
)
design.gds.qubits.change_options(new_options=options)
# Add a new component
options = Dict(
    name = "q0",
    type = "Transmon",
    gds_pos = (0, 0),
    topo_pos = (0, 0),
    chip = "chip0",
)
design.gds.qubits.add(options=options)
# Copy a specified component
design.gds.qubits.copy_component(old_name="q0", new_name="q1")
# Generate a row of components
design.gds.qubits.generate_row(start_pos=(0, 0),
                               dist=2000,
                               key="gds_pos",
                               num="10",
                               pre_name="qubits_row_",
                               type="Transmon",
                               geometric_options=None)
# Batch generate components
design.gds.qubits.batch_generate(pos_list=[(0, 0), (2000, 0)],
                                 key="gds_pos",
                                 pre_name="qubits_batch_",
                                 type="Transmon",
                                 geometric_options=None)
# Batch modify component options
design.gds.qubits.batch_change(name_list=["q0", "q1"],
                               op_name="type",
                               op_value="Xmon")
# Fast add components
```

```
op0 = Dict(
    name = "q0",
    type = "Transmon",
    gds_pos = (0, 0),
    topo_pos = (0, 0),
    chip = "chip0",
)
op1 = Dict(
    name = "q1",
    type = "Transmon",
    gds_pos = (2000, 0),
    topo_pos = (1, 0),
    chip = "chip0",
)
options_list = [op0, op1]
design.gds.qubits.batch_add(options_list=options_list)
```