Qubit and chip design with KQCircuits®

github.com/iqm-finland/KQCircuits

CI passing DOI 10.5281/zenodo.4944796 License GPLv3

Alessandro Landra

Technical Lead for QPU design

Schedule

Monday	Tuesday	Wednesday	Thursday	Friday
Caspar & Pavel	Alessandro	Alessandro	Niko & Eelis	Caspar
Introduction to QPU design Installing KQCircuits First look around	Introduction to designing Create a custom qubit element	Design a custom chip	Finite element simulations	Mask export Composite waveguides GUI

Workshop format

Introductions + hands-on exercises

- Follow along
- Ask for help if you are stuck

IQM

Questions?!

Ask questions any time!

✋ Raise hand (or just interrupt)

💬 Zoom chat

Discord

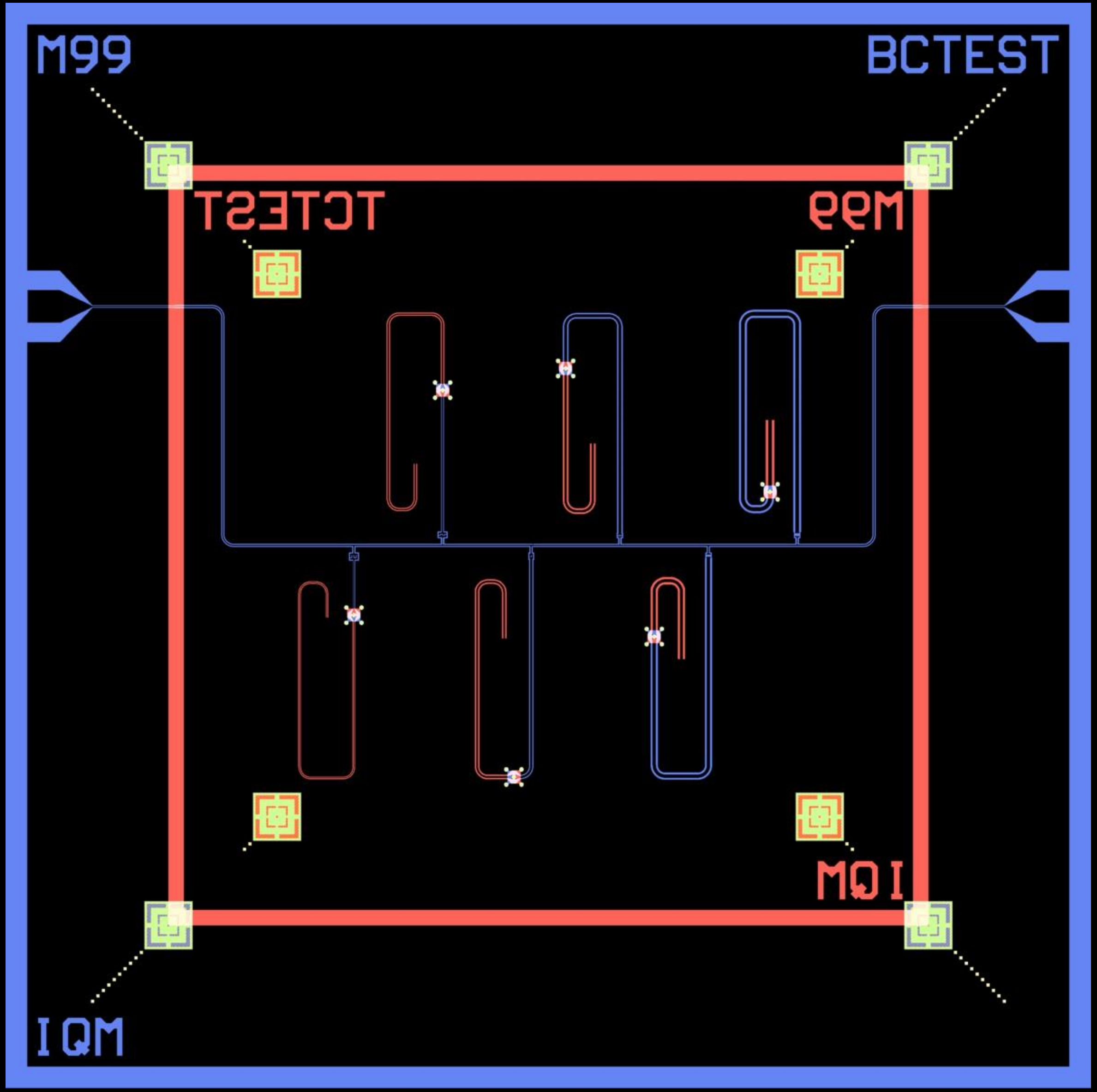
🔴 Presentations are recorded

Design steps

- FEM simulations
- Microwave circuit modelling
- Hamiltonian calculations
- Geometry building

Problem

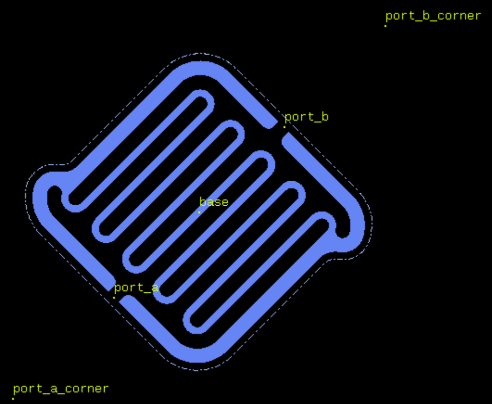
- Design structures in a parametrized way
- Repeatability
- Shared platform
- Human errors
- Inefficiency

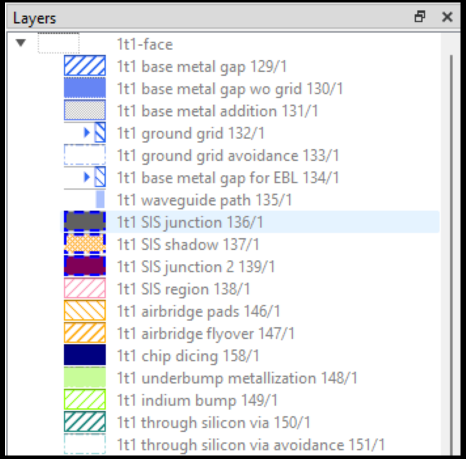


Concepts

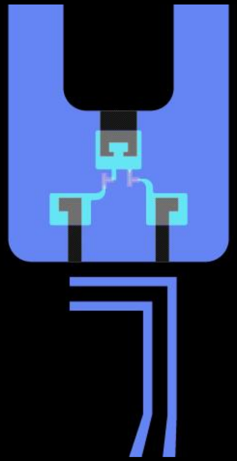
- PCell Library
- Reference points
- Standard for layers
- Face

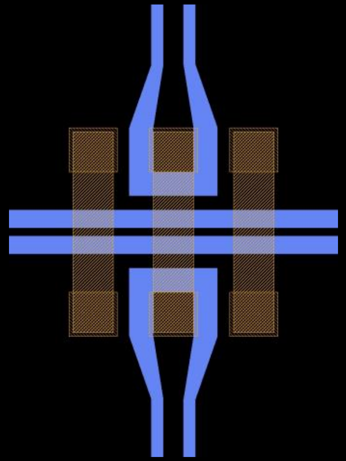
```
class Element(pya.PCellDeclarationHelper):
    """Element PCell declaration.

    PCell parameters for an element are defined as class attributes of Param type.
    Elements have ports.
    """

    LIBRARY_NAME = "Element Library"
    LIBRARY_DESCRIPTION = "Superconducting quantum circuit library for elements."
    LIBRARY_PATH = "elements"

    a = Param(pdt.TypeDouble, "Width of center conductor", 10, unit="μm")
    b = Param(pdt.TypeDouble, "Width of gap", 6, unit="μm")
    n = Param(pdt.TypeInt, "Number of points on turns", 64)
    r = Param(pdt.TypeDouble, "Turn radius", 100, unit="μm")
    margin = Param(pdt.TypeDouble, "Margin of the protection layer", 5, unit="μm")
    face_ids = Param(pdt.TypeList, "Chip face IDs list", ["1t1", "2b1", "1b1", "2t1"])
    display_name = Param(pdt.TypeString, "Name displayed in GUI (empty for default)", "")
    protect_opposite_face = Param(pdt.TypeBoolean, "Add opposite face protection too", False)
```

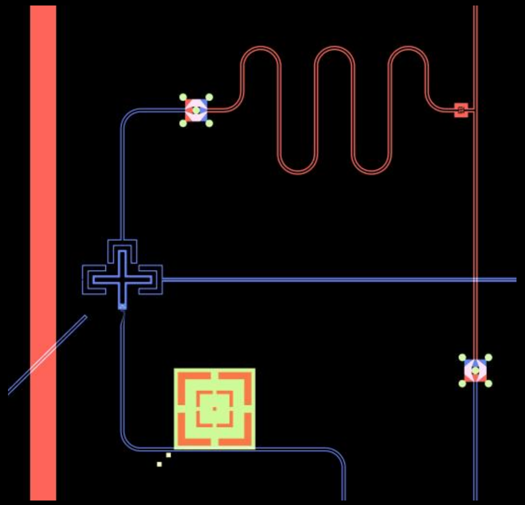



Layers

- 1t1-face
  - ▨ 1t1 base metal gap 129/1
  - ■ 1t1 base metal gap wo grid 130/1
  - ▨ 1t1 base metal addition 131/1
  - ▸ 1t1 ground grid 132/1
  - ▢ 1t1 ground grid avoidance 133/1
  - ▸ 1t1 base metal gap for EBL 134/1
  - 1t1 waveguide path 135/1
  - ■ 1t1 SIS junction 136/1
  - ▨ 1t1 SIS shadow 137/1
  - ■ 1t1 SIS junction 2 139/1
  - ▨ 1t1 SIS region 138/1
  - ▨ 1t1 airbridge pads 146/1
  - ▨ 1t1 airbridge flyover 147/1
  - ■ 1t1 chip dicing 158/1
  - ■ 1t1 underbump metallization 148/1
  - ▨ 1t1 indium bump 149/1
  - ▨ 1t1 through silicon via 150/1
  - ▢ 1t1 through silicon via avoidance 151/1

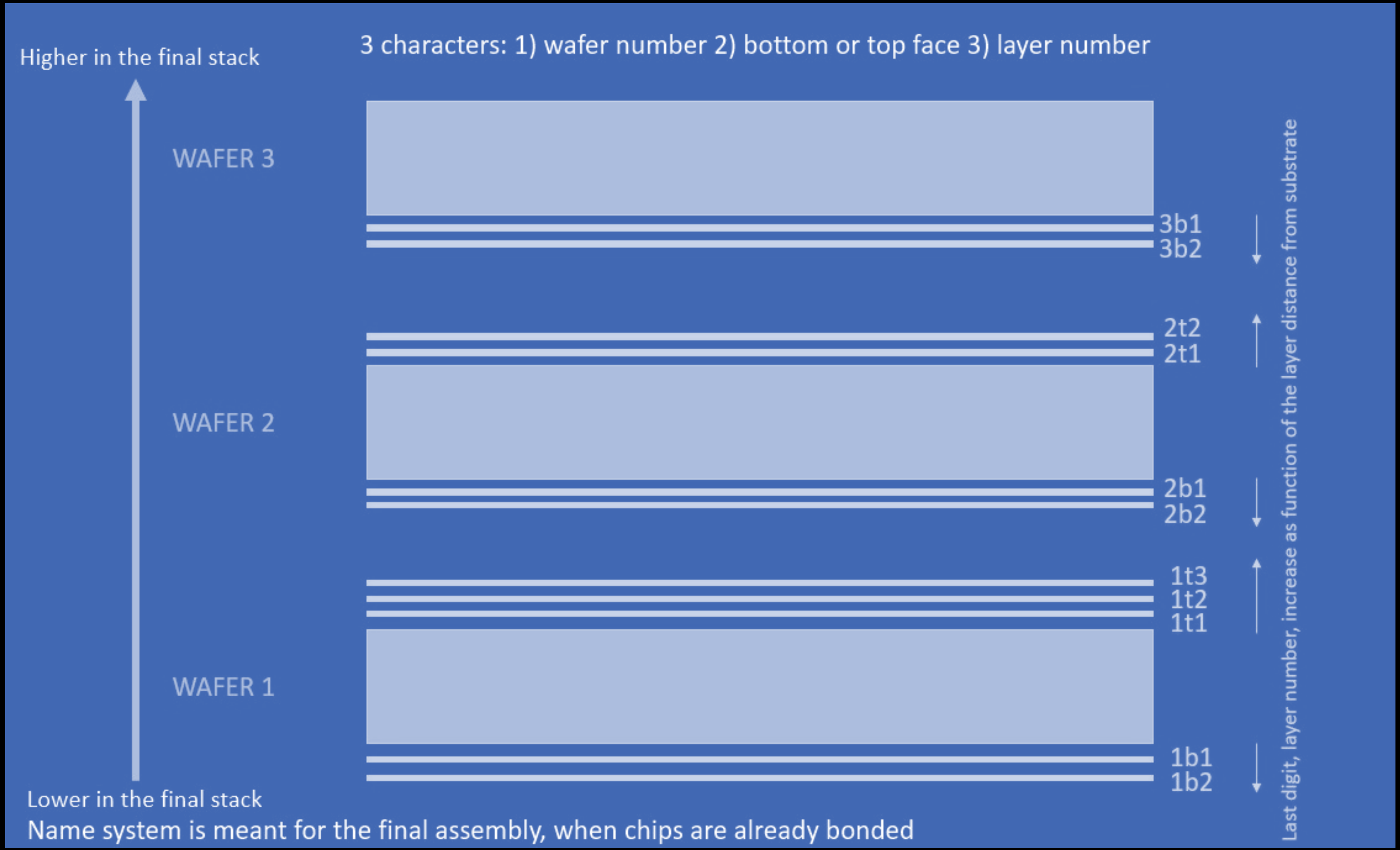






Concepts

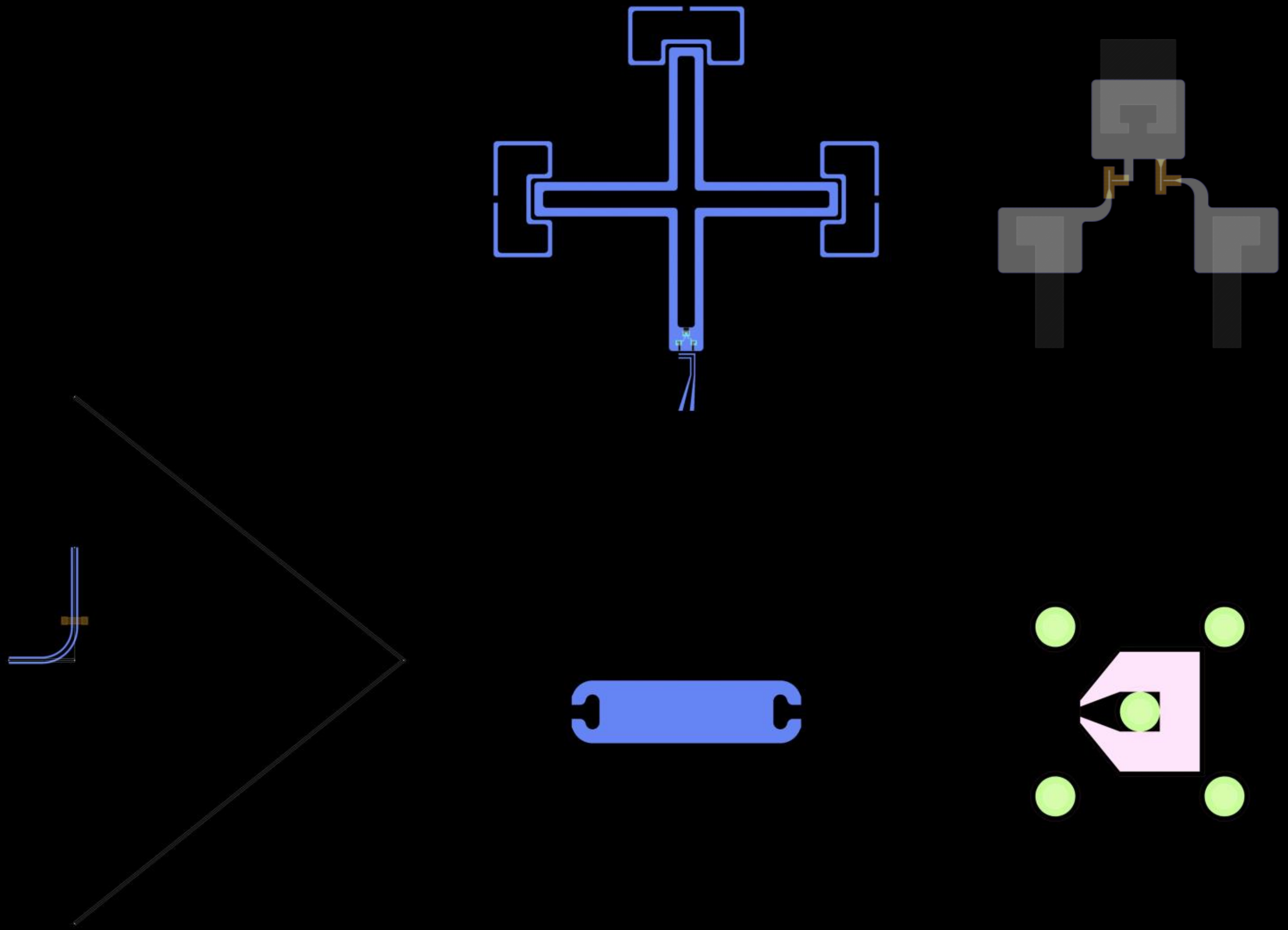
- Face in details


Higher in the final stack
3 characters: 1) wafer number 2) bottom or top face 3) layer number
WAFER 3
3b1
3b2
2t2
2t1
WAFER 2
2b1
2b2
1t3
1t2
1t1
WAFER 1
1b1
1b2
Last digit, layer number, increase as function of the layer distance from substrate
Lower in the final stack
Name system is meant for the final assembly, when chips are already bonded

IQM

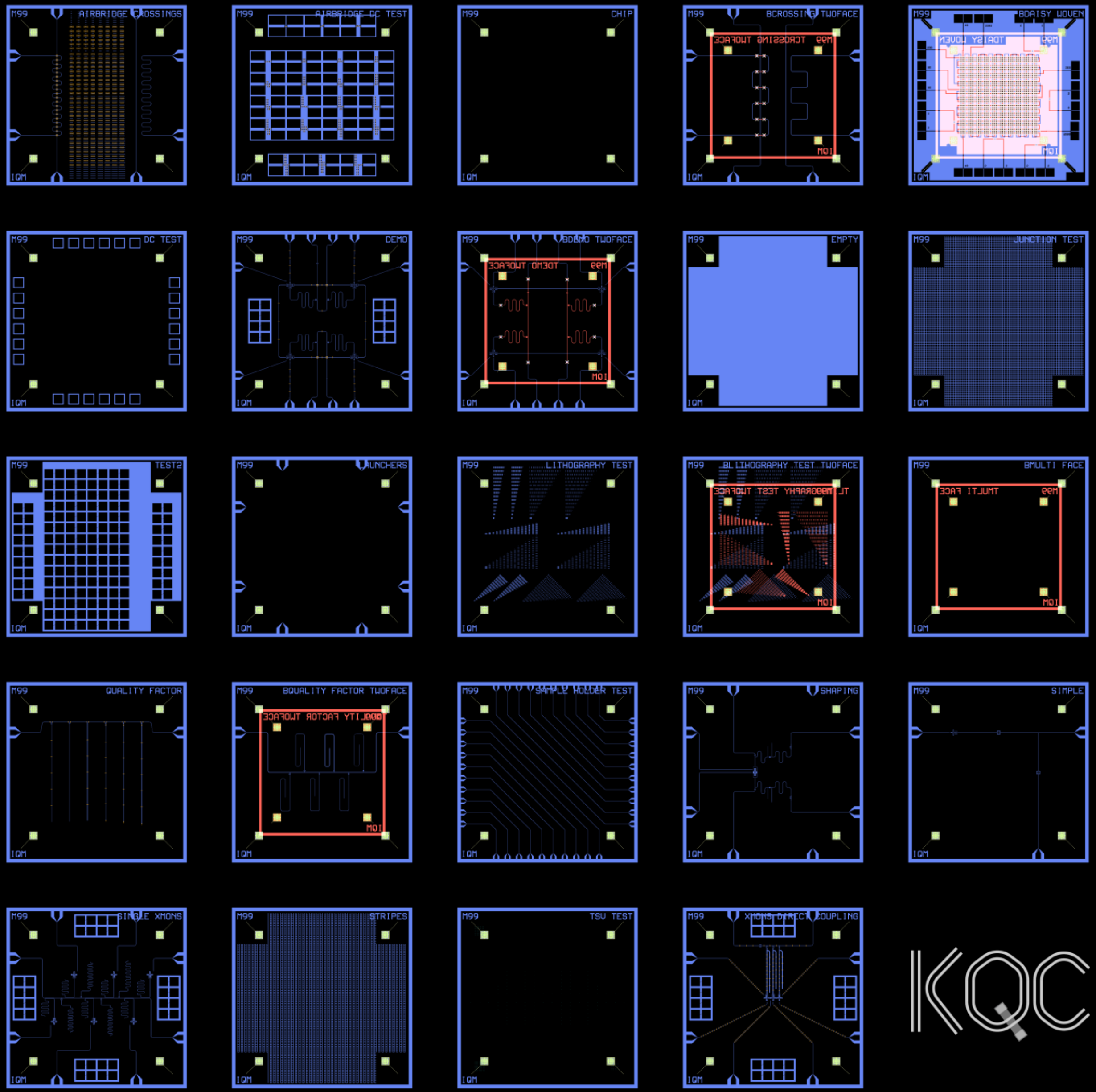
Element Library

- Qubits
- SQUIDs
- Complex waveguides
- Coplanar capacitors
- Face-to-face connectors
- Alignment markers
- …

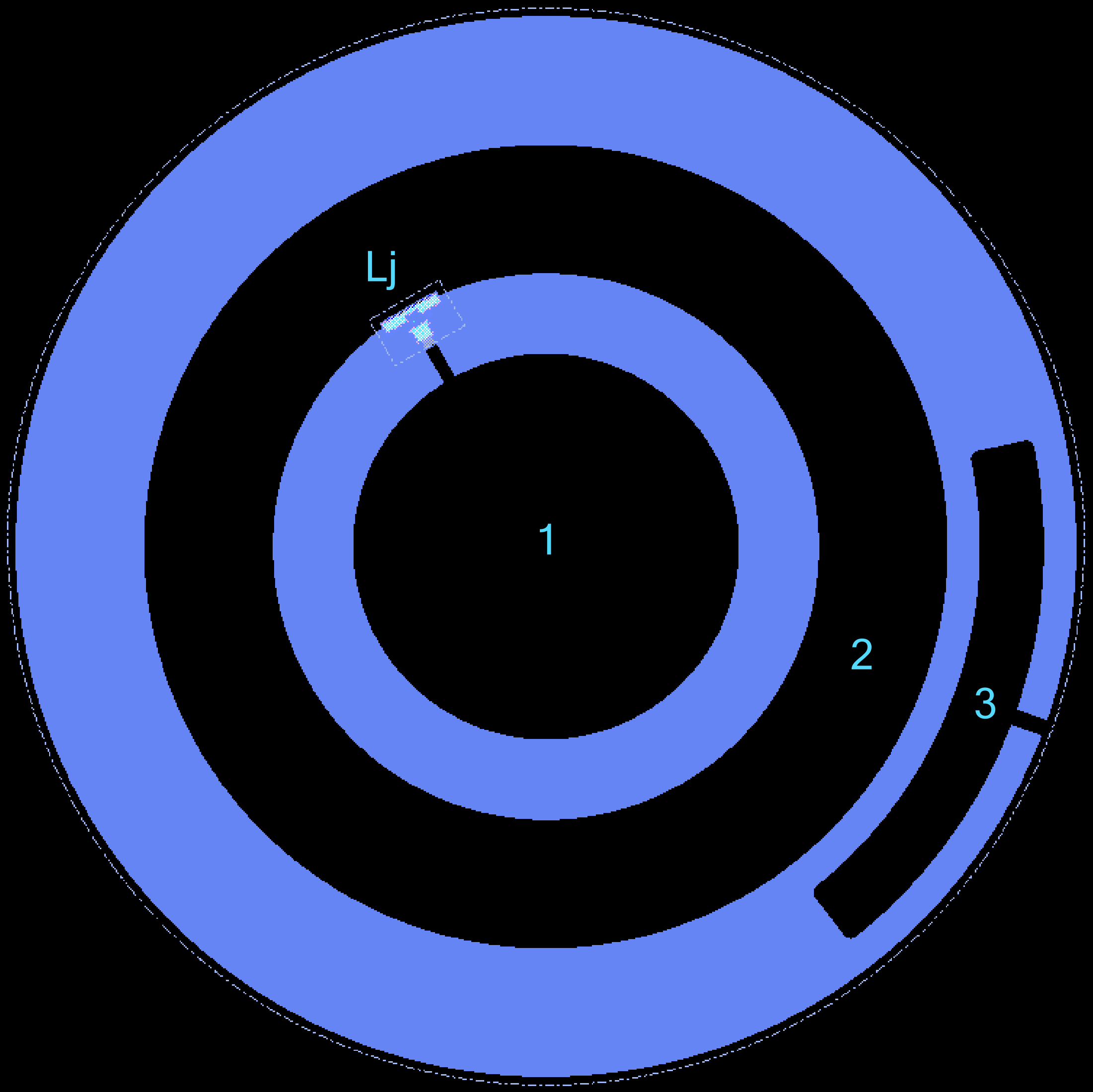


Chip Library

- Code use examples
- Fabrication test standards

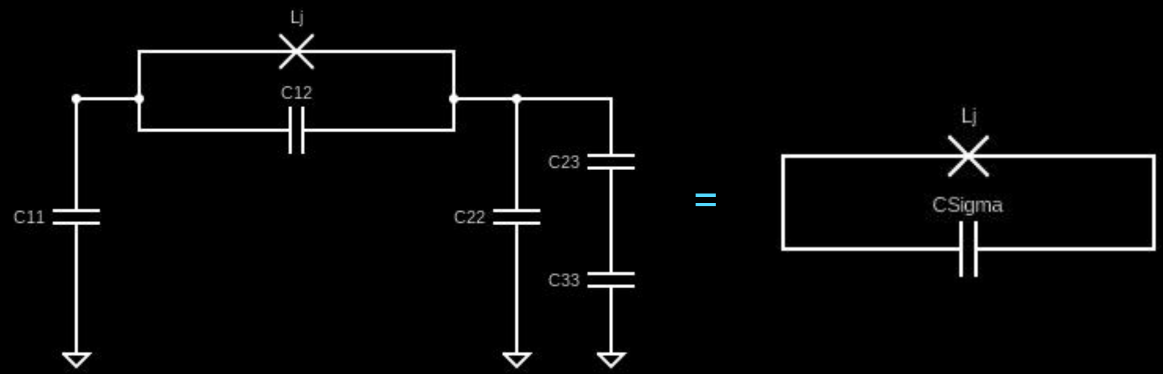


Floating transmon and chip coding demo



Floating transmon qubit

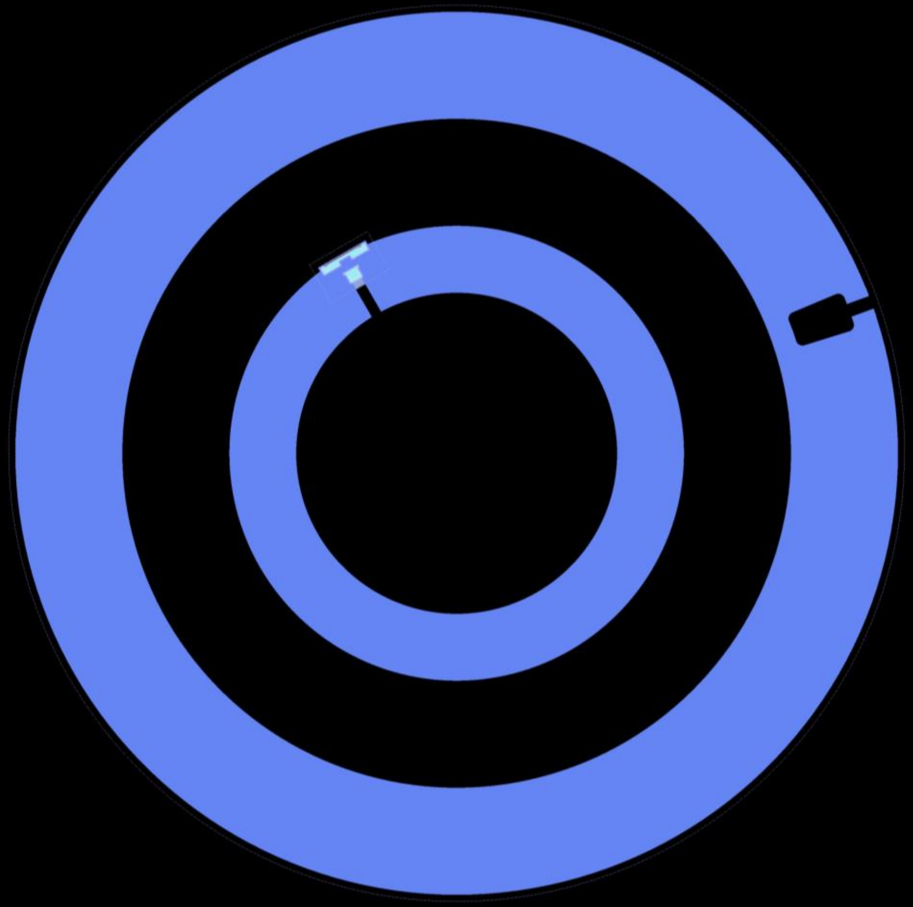
- Two SC islands (1 is the inner, 2 is the outer) which capacitively shunt the Josephson Junction or SQUID (Lj).
- Islands size and distance between them determines the charging energy, which depends on CSigma.
- We can add one (island 3) coupler or more to couple the qubit to the readout resonator or other qubits.
- Example circuit ($C13\approx0$)



Parametrized design

Using PCell parameters in the concentric transmon class we want to be able to control the following dimensions.

```
class ConcentricTransmon(Qubit):
    """The PCell declaration for a concentric transmon.

    A concentric transmon consists of two islands, one inner and one outer, connected by a Josephson Junction/s. Multiple
    couplers can be defined. They can have custom waveguide impedance, size and shape.
    Each coupler has reference points, numbered starting from 1. Driveline can be connected to the drive port.

    """
    # Qubit geometry
    r_inner = Param(pdt.TypeDouble, "Internal island radius", 120, unit="µm",
                    docstring="Radius of the outer edge of the inner island")
    r_outer = Param(pdt.TypeDouble, "External island radius, measured at the outer edge", 250, unit="µm",
                    docstring="Radius of the external qubit island")
    outer_island_width = Param(pdt.TypeDouble, "Outer island radial width", 80, unit="µm",
                               docstring="Width of the external island")
    ground_gap = Param(pdt.TypeDouble, "Ground plane padding", 80, unit="µm")
    squid_angle = Param(pdt.TypeDouble, "Angular position of the Josephson Junction/s, where the positive x-axis is zero",
                        120, unit="degrees")

    # Couplers parameters (the list size define the number of couplers)
    couplers_r = Param(pdt.TypeDouble, "Radius of the couplers positioning", 290, unit="µm")
    couplers_a = Param(pdt.TypeList, "Width of the coupler waveguide's center conductors", [10, 3, 4.5], unit="[µm]")
    couplers_b = Param(pdt.TypeList, "Width of the coupler waveguide's gaps", [6, 32, 20], unit="[µm]")
    couplers_angle = Param(pdt.TypeList,
                           "Positioning angles of the couplers, where 0deg corresponds to positive x-axis",
                           [340, 60, 210], unit="[degrees]")
    couplers_width = Param(pdt.TypeList, "Radial widths of the arc couplers", [10, 20, 30], unit="[µm]")
    couplers_arc_amplitude = Param(pdt.TypeList, "Couplers angular extension", [35, 45, 15], unit="[degrees]")

    # Drive port parameters
    drive_angle = Param(pdt.TypeDouble, "Angle of the drive port, where 0deg corresponds to positive x-axis", 300,
                        unit="degrees")
    drive_distance = Param(pdt.TypeDouble, "Distance of the driveline, measured from qubit centre", 400, unit="µm")
```



IQM

Build function

```
def build(self):
    self.x_end = self.r_outer + self.ground_gap  # Define the outermost qubit coordinate

    # Generate the qubit islands (they are the negative shape of the final geometry)
    qubit_negative = self._make_qubit_islands()

    # Generate the coupler islands
    coupler_islands_region = self._make_coupler_island()

    # Add the waveguides connecting the couplers to external waveguides
    waveguide, waveguide_gap = self._make_waveguides()

    # Add the Josephson Junction/s
    self._add_junction(qubit_negative)

    # Define the capacitor in the ground (final polarity)
    ground_region = self._make_ground_region()
    qubit = ground_region - qubit_negative + waveguide_gap - coupler_islands_region - \
        waveguide  # Operations order is important!
    self.cell.shapes(self.get_layer("base_metal_gap_wo_grid")).insert(qubit)

    # Protection region from the ground grid
    region_protection = self._get_protection_region(ground_region)
    self.cell.shapes(self.get_layer("ground_grid_avoidance")).insert(region_protection)

    # Couplers and driveline ports
    self._add_ports()
```

- In the previous pictures, the blue colored area represent the etched gap and not the metal.
- We want to draw the metal Regions instead, then subtract them from a round ground plane region (blue).
- In order we will make the qubit island (as positive regions), the n-couplers islands, the connecting n-waveguides and then subtracting all of them from a circular (blue) ground region, basically defining our qubit picture.
- Finally, we will add the JJ or SQUID, the protection region from the automatic ground holes grid generation and the ports (refpoints with a direction) to connect ports and waveguides seamlessly.

Let’s now code the functions together

We will now use the concentric_transmon.py file which is partially written.

```
def _make_arc_island(self, island_outer_radius, island_width, swept_angle):
    # Generate a polygon arc of any size and angle
    angle_rad = math.radians(swept_angle)
    points_outside = arc_points(island_outer_radius, ..., ..., ...)
    points_inside = arc_points(island_outer_radius - island_width, ..., ..., ...)
    points = ...
    arc_island = ...

    return arc_island
```

```
def _make_qubit_islands(self):
    # Generate a region of the qubit shunting capacitor
```

```
def _make_waveguides(self):
    # Make the waveguides for each coupler with custom impedance and return the region
```

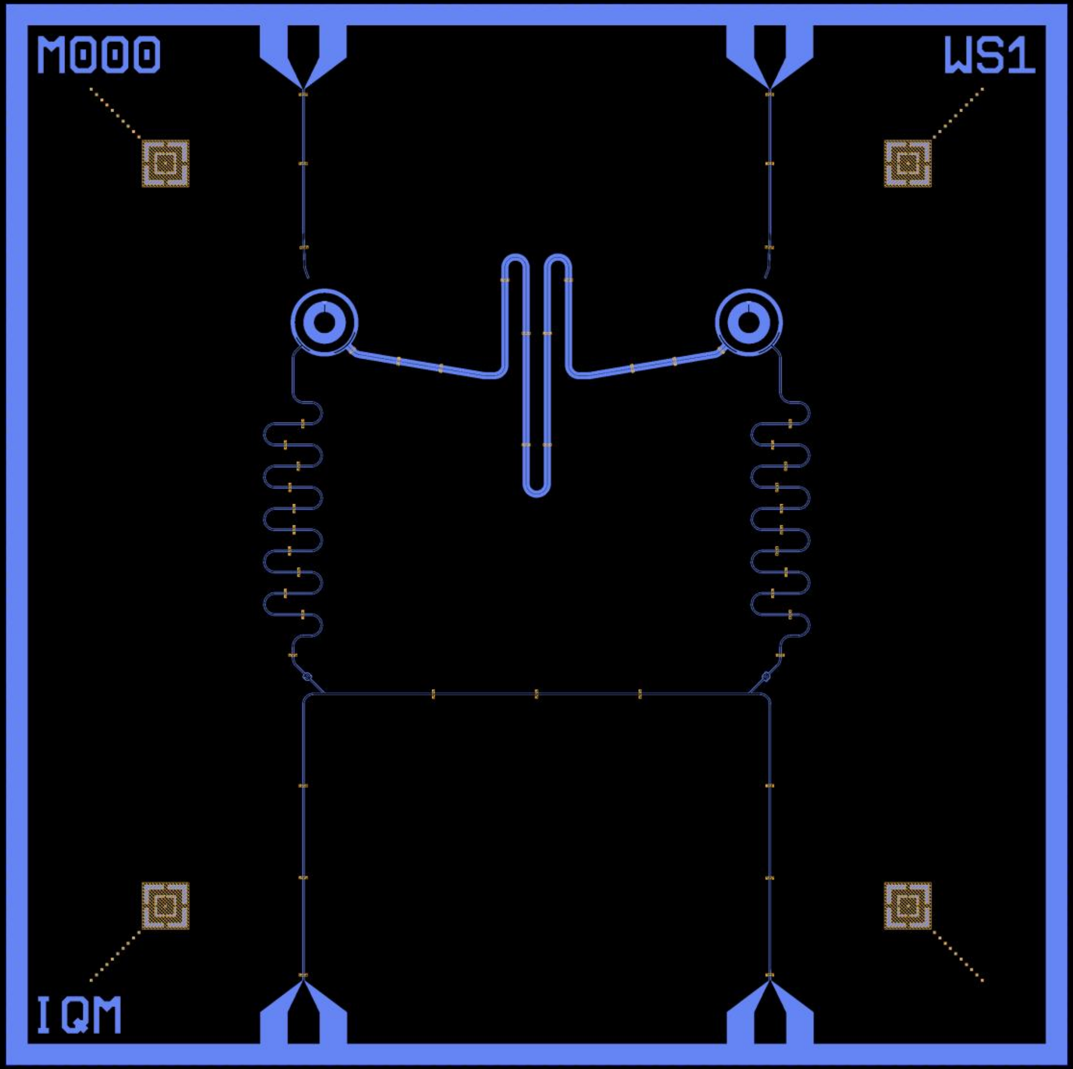
```
def _add_junction(self, region):
    # Add the junction to the qubit islands
```

```
def _add_ports(self):
    # Add couplers ports
```

```
def _make_coupler_island(self):
    # Generate the regions of the coupler islands.
```

```
def _get_protection_region(self, region):
    # Region which we don't want to cover with the automatically generated ground grid
```

```
def _make_ground_region(self):
    # Generate the ground region as a filled circle of the maximum size
```



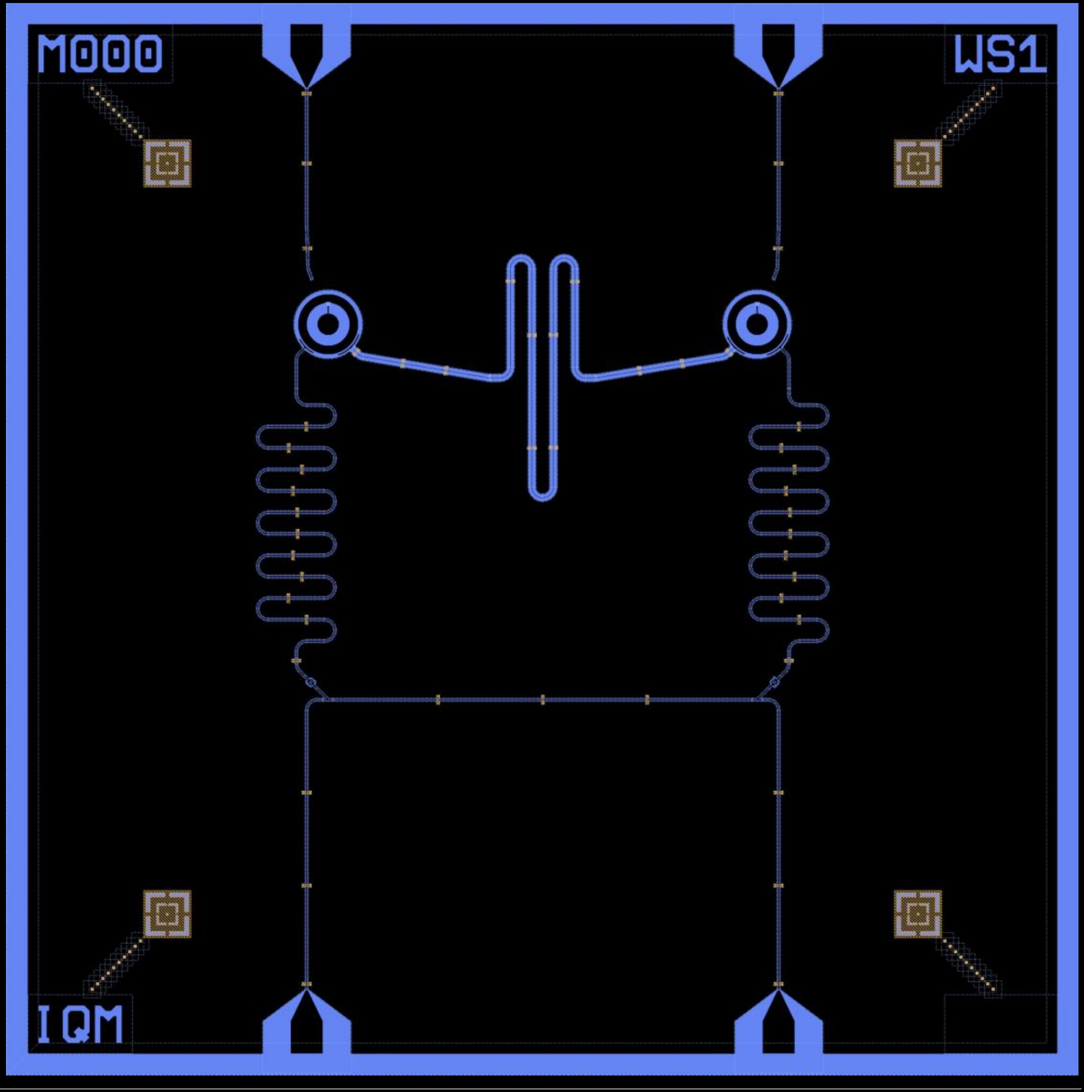
Two coupled transmons

- Two concentric transmon qubits connected by a fixed frequency resonator bus.
- Each concentric transmon should have two couplers.
- Each concentric transmon is read out by its own readout resonator.
- The readout resonators are multiplexed to the same feedline.

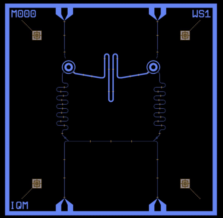
Parametrized design

Using PCell parameters in the demo chip class we want to be able to control the following dimensions.

```
class ConcentricQubitsDemo(Chip):
    """Demonstration chip with two concentric qubits, two readout resonators, one probe line,
    two drivelines and one resonant coupler."""

    name_chip = Param(pdt.TypeString, "Name of the chip", "WS1")

    readout_res_lengths = Param(pdt.TypeList, "Readout resonator lengths", [8000, 10000], unit="[μm]")
    kappa_finger_control = Param(pdt.TypeList, "Finger control for the input capacitor",
                                 default=[1.99, 2.035], unit="[μm]")

    coupler_length = Param(pdt.TypeDouble, "Resonant coupler length", 10000)

    couplers_a = Param(pdt.TypeList, "Width of the coupler waveguide's center conductors", [[10, 3], [10, 3]],
                  unit="[μm]")
    couplers_b = Param(pdt.TypeList, "Width of the coupler waveguide's gaps", [[6, 32], [6, 32]], unit="[μm]")
    couplers_angle = Param(pdt.TypeList,
                      "Positioning angles of the couplers, where 0deg corresponds to positive x-axis",
                      [[225, 315], [315, 225]], unit="[degrees]")
    couplers_width = Param(pdt.TypeList, "Radial widths of the arc couplers", [[10, 10], [10, 10]], unit="[μm]")
    couplers_arc_amplitude = Param(pdt.TypeList, "Couplers angular extension", [[35, 55], [55, 45]],
unit="[degrees]")

    drive_line_offsets = Param(pdt.TypeList, "Distance between the end of a drive line and the qubit pair",
[450.0] * 2)
```



Let’s now code the functions together



We will now use the concentric_transmon_demo_chip.py file which is partially written.

```
def build(self):
    # Define launchpads positioning and function
    launcher_assignments = {
        1: "DL-QB1",
        2: "DL-QB2",
        5: "PL-OUT",
        6: "PL-IN",
    }
    # Use an 8 port default launcher
    self.produce_launchers("SMA8", launcher_assignments)
    self.produce_qubits()
    self.produce_coupler()
    self.produce_drivelines()
    self.produce_probeline()
    self.produce_readout_resonators()
```

```
def produce_qubits(self):
    # Position the qubits
```

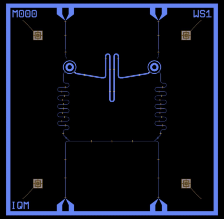
```
def produce_coupler(self):
    # Insert a fixed coupler of a variable meander size in between qubits
```

```
def produce_probeline(self):
```

```
def produce_drivelines(self):
    # Connect the drivelines to the qubit ports
```

```
def produce_readout_resonators(self):
    # Break down the resonator in few parts for simplicity
```

What if we want a three qubits chip?



You can try to code a three qubits processor:

- Use the existing chip design
- Add a third readout resonator in the probeline
- Connect the qubit to the readout resonator
- Either connect the three qubits with two meanders, or delete the existing meander to make three directly coupled qubits
- Don't forget to change the launcher assignment and add a new driveline to play with the added qubit!

Thank you for your participation!