mxPIC Quick-Reference Style Guide

Welcome to the collaboration of the repository of **mxPIC** program. This repository is aimed for a self-designed PDK that based on ***nazca***, ***gdstk*** and other libraries for the functional build up of photonic integrated circuits (PIC).

This is the fast-track guide to contributing to **mxPIC**. Adherence to these rules is mandatory to pass the automated CI/CD pipeline.

0. Installation dependencies

- python 3.10.1
- nazca 0.6.1
- numpy 1.12.0
- pandas 0.12.0
-

1. File & architecture

The repository have an archtecture like this.

```
mxpic_forge/
├── primitives/
│   ├── edge_couplers/
│   │   └── EC_dual_layer_px3.py
│   ├── multimode_interferomters/
│   ├── directional_couplers/
│   ├── grating_couplers/
│   └── microring_modulators/
├── composite/
│   └── mach_zender_modulators/
├── electronics/
│   ├── resistors/
│   ├── inductors/
│   ├── vias/
│   └── pads/
├── others/
├── structures/
├── routing/
├── foundries/
└── docs/
    └── source/
        └── conf.py
```

2. Annotation & Documentation

We strictly use **Type Hints** (PEP 484) and **NumPy-Style Docstrings**.

- **Rule:** Every function MUST have its parameter types and return type explicitly declared.
- **Rule:** Use `"""` for docstrings, broken into `Parameters` and `Returns` sections.

```
- class Waveguide(length, width):
-     """Builds a Mach-Zehnder."""
-     pass

+ class Waveguide(length: float, width: float = 0.5) -> nazca.Cell:
+     """
+     Generates an MZM layout.
+
+     Parameters
+     ----------
+     length : float
+         Arm length in microns.
+     width : float, optional
+         Waveguide width in microns.
+
+     Returns
+     -------
+     list[tuple]
+         Polygon vertices.
+     """
+     pass
```

---

3. Class mangement of devices

The devices are divided into four major types, which is **primitives, composite, electronics, and others**. All photonic and electronic devices are built within the for types.
In the definitial of classes, compulsary keys are required, including **name** , **show_pins** , all defaults are None.
All classes are cored at the **cell** class of **nazca**, which is generated through an internal method named "generate_gds"

```python
class GratingCoupler():
    def __init__(self,name:"str"=None,....,show_pins:bool=None)
        ...
        ...

        self.cell = self.generate_gds() ## the generation of cell

    def generate_gds(self) -> nazca.Cell:
        """ Creating the cell of the device """
        with nd.Cell(name=self.name,inst) as C:

        return C
```

---

4. Port information formatting

The basic routing algorthium is based the information between nodes, which is the **pin** attribute inside each file. The formatting of **pin** name will be important.

Note: The pin name should only be related to different functionalities instead of material, layer or shaped. Say, the pin name of a *silicon directional coupler* is the same as a *silicon nitride directional coupler*.

Catagory	**Prefix**	**name**	**index**	**expample**	**device instance**
Optical	opt_	a	1~x	opt_a1	direction_coupler
Optical (Sinlge IO to waveguide)	opt_	a	1~x	opt_a1	grating_coupler
Electrical	ele_	a	1~x	ele_a1	heater/resistor
PN diodes	ele_	ka/an	1~x	ele_an1	modulator/p-i-n waveuigde
Transistors (BJT)	ele_	ba/em/cl	1~x	ele_cl1	BJT
Transistors (MOS)	ele_	gt/su/dr	1~x	ele_gt1	MOSFET