三角形网络可编程光子链路的设计

Aolong Sun

除了矩形可编程光子链路，基于三角形网络的可编程光子链路同样得到了广泛的应用。这一部分主要详细讲解如何从单元元件 MZI 出发，基于 PhotoCAD 搭建三角形网络可编程光子链路。

目录

第一部分 构建 MZI unit

在构建三角形网络之前，需要先补充 gpdk 里的单元器件。在这里我们选择构建一个单臂可调的片上集成马赫曾德尔干涉仪（MZI）。这一器件的可调性是通过 PN 结（电可调）实现的，当然也可以使用热可调的 MZI 来构建该三角形网络。

首先，导入库文件，在这里我们需要用到定向耦合器、直波导、移相器：

```
from dataclasses import dataclass
from typing import Tuple

from fnpcell import all as fp
from gpdk.components.directional_coupler.directional_coupler_sbend import DirectionalCouplerSBend
from gpdk.components.straight.straight import Straight
from gpdk.components.pn_phase_shifter.pn_phase_shifter import PnPhaseShifter
from gpdk.technology import WG, get_technology
```

定义网络的基础单元器件 MZI 类：

```
@dataclass(eq=False)
class MZI(fp.PCell, band="C"):
    p_width: float = fp.PositiveFloatParam(default=1)
    n_width: float = fp.PositiveFloatParam(default=1)
    np_offset: float = fp.FloatParam(default=0)
    wg_length: float = fp.PositiveFloatParam(default=100)
    arm_spacing: float = fp.PositiveFloatParam(default=60)
    dc_length: float = fp.FloatParam(default=100)
    waveguide_type: WG.FWG.C = fp.WaveguideTypeParam(type=WG.FWG.C)
    pn_phase_shifter: fp.IDevice = fp.DeviceParam(type=PnPhaseShifter, port_count=2, pin_count=2, required=False)
    straight_waveguide: fp.IDevice = fp.DeviceParam(type=Straight, port_count=2, required=False)
    directional_coupler_left: fp.IDevice = fp.DeviceParam(type=DirectionalCouplerSBend, port_count=4, required=False)
    directional_coupler_right: fp.IDevice = fp.DeviceParam(type=DirectionalCouplerSBend, port_count=4, required=False)
    port_names: fp.IPortOptions = fp.PortOptionsParam(count=4,default=["op_0", "op_1", "op_2", "op_3"])

    def _default_waveguide_type(self):
        return get_technology().WG.FWG.C.WIRE

    def _default_pn_phase_shifter(self):
        return PnPhaseShifter(
            name="ps", p_width=self.p_width, n_width=self.n_width, np_offset=self.np_offset,
            wg_length=self.wg_length-10, waveguide_type=self.waveguide_type,
            transform=fp.translate(5, 0)
        )
    def _default_straight_waveguide(self):
        return Straight(
            name="straight", waveguide_type=self.waveguide_type, length=self.wg_length, transform=fp.translate(0, -self.arm_spacing)
        )

    def _default_directional_coupler_left(self):
        return DirectionalCouplerSBend(
            name="dc_l", bend_radius=10, waveguide_type=self.waveguide_type, transform=fp.translate(-self.dc_length, -self.arm_spacing / 2)
        )

    def _default_directional_coupler_right(self):
        return DirectionalCouplerSBend(
            name="dc_r", bend_radius=10, waveguide_type=self.waveguide_type, transform=fp.translate(self.wg_length + self.dc_length, -self.arm_spacing / 2)
        )
```

```
    def build(self) -> Tuple[fp.InstanceSet, fp.ElementSet, fp.PortSet]:
        insts, elems, ports = super().build()
        # fmt: off

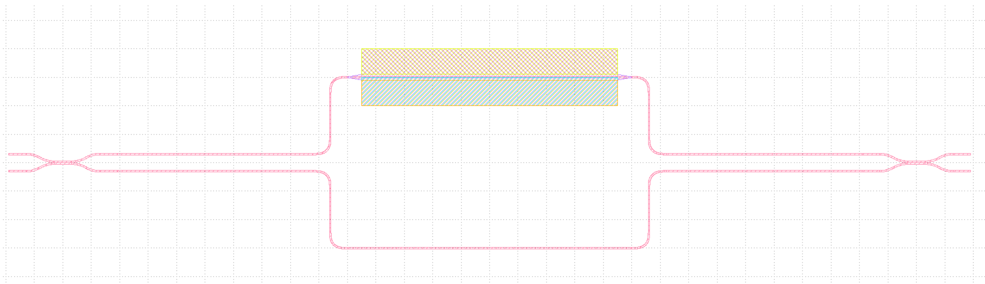
        waveguide_type = self.waveguide_type
        pn_phase_shifter = self.pn_phase_shifter.translated(-self.wg_length / 2, self.arm_spacing / 2)
        straight_waveguide = self.straight_waveguide.translated(-self.wg_length / 2, self.arm_spacing / 2)
        directional_coupler_left = self.directional_coupler_left.translated(-self.wg_length / 2,
self.arm_spacing / 2)
        directional_coupler_right = self.directional_coupler_right.translated(-self.wg_length / 2,
self.arm_spacing / 2)

        port_names = self.port_names
        ports += directional_coupler_left["op_0"].with_name(port_names[0])
        ports += directional_coupler_left["op_1"].with_name(port_names[1])
        ports += directional_coupler_right["op_2"].with_name(port_names[2])
        ports += directional_coupler_right["op_3"].with_name(port_names[3])

        insts += fp.Linked(
            link_type=waveguide_type,
            bend_factory=self.waveguide_type.BEND_EULER,
            links=[
                directional_coupler_left["op_3"] >> pn_phase_shifter["op_0"],
                directional_coupler_left["op_2"] >> straight_waveguide["op_0"],
                directional_coupler_right["op_0"] >> pn_phase_shifter["op_1"],
                directional_coupler_right["op_1"] >> straight_waveguide["op_1"],
            ],
            ports=[],
        )

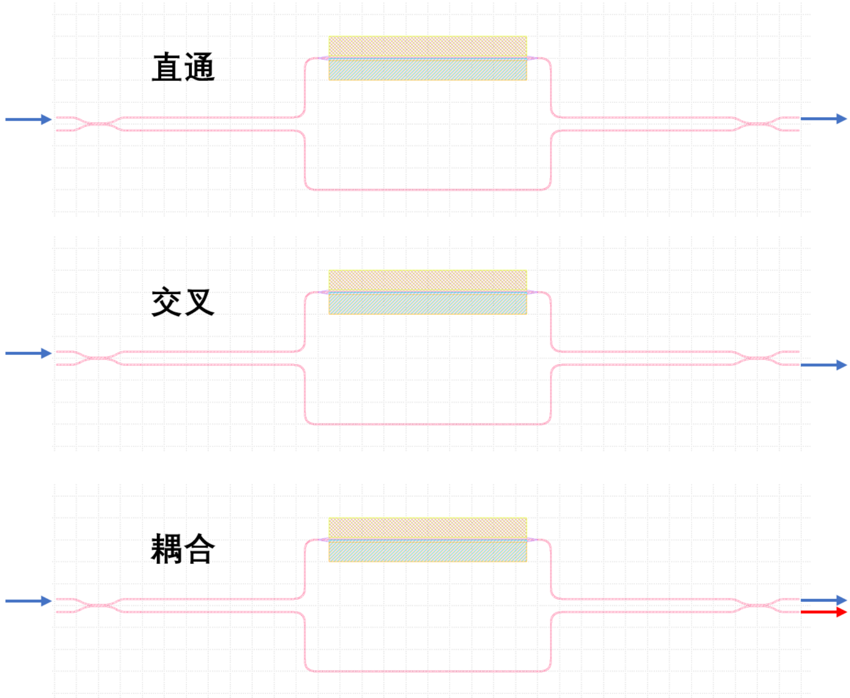
        # fmt: on
        return insts, elems, ports
```

在本案例中，MZI 由两个定向耦合器、一个移相器以及一个直波导构成，基本结构如下图所示：

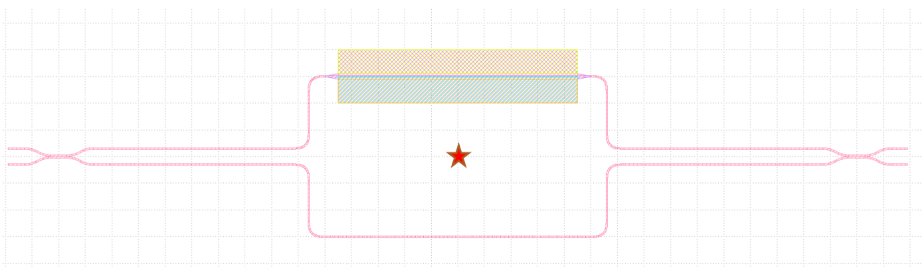


其中移相器基于 PN 结构建，通过改变施加到 PN 结两端的电压，即可改变载流

子浓度，从而改变波导的折射率，最终改变相对相位。因此，通过改变电压，可以使 MZI 工作在不同的工作模式，如直通、交叉或耦合模式，如下图所示：



在本例中，依次构建四个单元器件并设置好位置参数后，可以直接使用 PhotoCAD 自带的 Linked 方法实现端口的自动连接。其次，需要给 MZI 分配四个端口并指定名称。注意，我们在 class 里对各个元件进行了平移，使整个 MZI 单元的原点位于 MZI 的中心处，如不这么做，在对 MZI 进行旋转操作时会出错，因为 PhotoCAD 里的旋转操作默认是以单元器件的原点为中心的。本例中 MZI 的原点位置如下图所示：



在主函数里调用 MZI 并生成 gds 文件：

```
if __name__ == "__main__":
    from gpdk.util.path import local_output_file
```

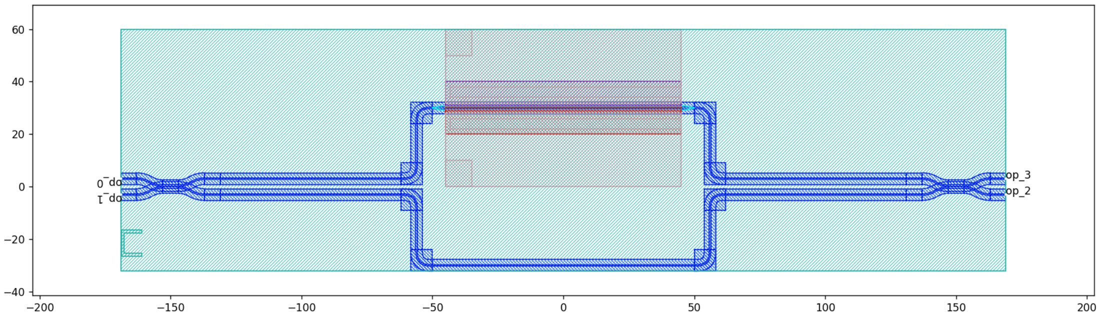
```
gds_file = local_output_file(__file__).with_suffix(".gds")
  library = fp.Library()

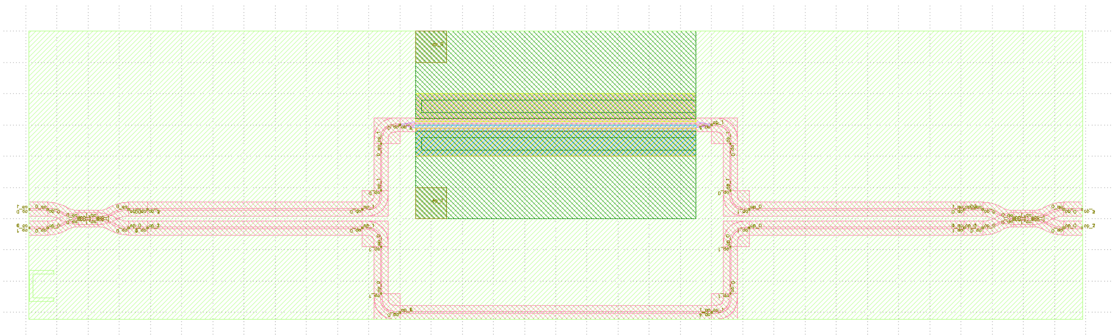
TECH = get_technology()
# =============================================================
# fmt: off

library += MZI()

# fmt: on
# =============================================================
fp.export_gds(library, file=gds_file)
fp.plot(library)
```

自动生成的版图如下：





第二部分 构建可编程三角形 MZI 网络

在这一步里将调用上一步构建的 MZI 类，实现三角形 MZI 网络的构建。我们提供两种实现方案，一种是不加光栅耦合器的 MZI 网络，提供了 8 个外置光学端口（op0~op7），可以方便用户自定义拓展该网络；第二种实现方案是给 MZI 网络加上了 8 个光栅耦合器，构成了一个完整闭环的光子链路版图。

带有 8 个外置光学端口的 MZI Mesh（MZI_triangle_mesh）

首先导入必要的库文件：

```
from dataclasses import dataclass
from typing import Tuple

from fnpcell import all as fp
from gpdk.components.mzm.mzi import MZI
from gpdk.technology import WG, get_technology
from gpdk.routing.extended.extended import Extended
from gpdk.technology.waveguide_factory import EulerBendFactory
from gpdk.components.grating_coupler.grating_coupler import GratingCoupler
from gpdk.routing.comp_scan.comp_scan import CompScan,Block
```

随后构建 MZI_triangle_mesh 类：

```
@dataclass(eq=False)
class MZI_triangle_mesh(fp.PCell, band="C"):
    """
    Attributes:
        p_width: defaults to 1
        n_width: defaults to 1
        np_offset: defaults to 0
        wg_length: defaults to 25
        arm_spacing: defaults to 100
        dc_length: defaults to 100
        waveguide_type: type of waveguide
        pn_phase_shifter: instance of `PnPhaseShifter`, port_count=2, pin_count=2, required=False
        straight_waveguide: instance of `Straight`, port_count=2, required=False
        directional_coupler_left: instance of `DirectionalCouplerSBend`, port_count=2, required=False
        directional_coupler_right: instance of `DirectionalCouplerSBend`, port_count=2, required=False
        port_names: defaults to ["op_0", "op_1", "op_2", "op_3"]

    Examples:
    ```python
    TECH = get_technology()
    mzi = MZI(wg_length=600, waveguide_type=TECH.WG.FWG.C.WIRE)
    fp.plot(mzi)
    ```
    ![MZI](images/mzi.png)
    """

    side_length: float = fp.PositiveFloatParam(default=400)
    dc_length: float = fp.FloatParam(default=100)
    arm_spacing: float = fp.FloatParam(default=60)
    wg_length: float = fp.FloatParam(default=100)
    waveguide_type: WG.FWG.C = fp.WaveguideTypeParam(type=WG.FWG.C)
    MZI_unit: fp.IDevice = fp.DeviceParam(type=MZI, port_count=4, required=False)
    port_names: fp.IPortOptions = fp.PortOptionsParam(count=8,
```

```
            default=["op_0", "op_1", "op_2", "op_3", "op_4", "op_5", "op_6", "op_7"])

    def _default_waveguide_type(self):
        return get_technology().WG.FWG.C.WIRE

    def _default_MZI_unit(self):
        return MZI(arm_spacing=self.arm_spacing, dc_length=self.dc_length, wg_length=self.wg_length)

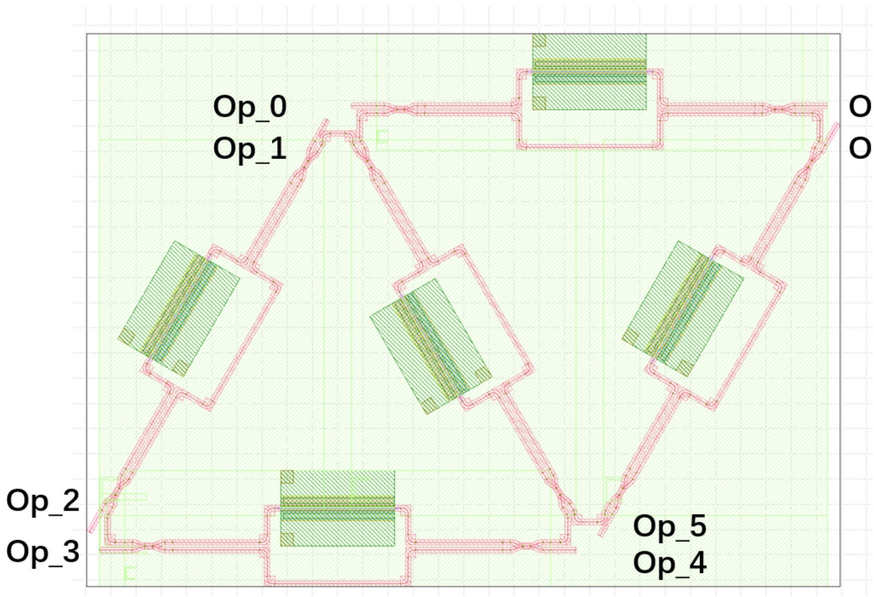
    def build(self) -> Tuple[fp.InstanceSet, fp.ElementSet, fp.PortSet]:
        insts, elems, ports = super().build()
        # fmt: off

        waveguide_type = self.waveguide_type
        port_names = self.port_names
        MZI_0 = self.MZI_unit.translated(0,0)
        MZI_1 = self.MZI_unit.rotated(degrees=120).translated(self.side_length / 4, self.side_length / 4 *
(3) ** (0.5))
        MZI_2 = self.MZI_unit.rotated(degrees=60).translated(-self.side_length / 4, self.side_length / 4 *
(3) ** (0.5))
        MZI_3 = self.MZI_unit.translated(self.side_length / 2, self.side_length / 2 * (3) ** (0.5))
        MZI_4 = self.MZI_unit.rotated(degrees=60).translated(self.side_length * 3 / 4, self.side_length / 4
* (3) ** (0.5))
        ports += MZI_3["op_0"].with_name(port_names[0])
        ports += MZI_2["op_3"].with_name(port_names[1])
        ports += MZI_2["op_0"].with_name(port_names[2])
        ports += MZI_0["op_1"].with_name(port_names[3])
        ports += MZI_0["op_2"].with_name(port_names[4])
        ports += MZI_4["op_1"].with_name(port_names[5])
        ports += MZI_4["op_2"].with_name(port_names[6])
        ports += MZI_3["op_3"].with_name(port_names[7])

        insts += fp.Linked(
            link_type=waveguide_type,
            bend_factory=self.waveguide_type.BEND_EULER,
            links=[
                MZI_0["op_0"] >> MZI_2["op_1"],
                MZI_0["op_3"] >> MZI_1["op_0"],
                MZI_1["op_1"] >> MZI_4["op_0"],
                MZI_4["op_3"] >> MZI_3["op_2"],
                MZI_1["op_2"] >> MZI_3["op_1"],
                MZI_1["op_3"] >> MZI_2["op_2"],
            ],
            ports=[],
        )
```

```
# fmt: on
return insts, elems, ports
```

在这一部分，我们构建了一个基于五个 MZI 的三角形网络光子链路，八个端口及其标号都已在图中标注出：



在这个 Class 的定义里我们并未引入光栅耦合器，需要在主函数里对光栅耦合器（或者其他类型的耦合器）进行定义；或者也可以与其他自定义的单元器件进行连接。

带有 8 个光栅耦合器的 MZI Mesh（MZI_triangle_mesh_with_GC）

同样也是先导入必要的库文件：

```
from dataclasses import dataclass
from typing import Tuple

from fnpcell import all as fp
from gpdk.components.mzm.mzi import MZI
from gpdk.technology import WG, get_technology
from gpdk.routing.extended.extended import Extended
from gpdk.technology.waveguide_factory import EulerBendFactory
from gpdk.components.grating_coupler.grating_coupler import GratingCoupler
from gpdk.routing.comp_scan.comp_scan import CompScan,Block
```

随后构建 MZI_triangle_mesh_with_GC 类：

```
@dataclass(eq=False)
class MZI_triangle_mesh_with_GC(fp.PCell, band="C"):
```

````python
"""
Attributes:
    p_width: defaults to 1
    n_width: defaults to 1
    np_offset: defaults to 0
    wg_length: defaults to 25
    arm_spacing: defaults to 100
    dc_length: defaults to 100
    waveguide_type: type of waveguide
    pn_phase_shifter: instance of `PnPhaseShifter`, port_count=2, pin_count=2, required=False
    straight_waveguide: instance of `Straight`, port_count=2, required=False
    directional_coupler_left: instance of `DirectionalCouplerSBend`, port_count=2, required=False
    directional_coupler_right: instance of `DirectionalCouplerSBend`, port_count=2, required=False
    port_names: defaults to ["op_0", "op_1", "op_2", "op_3"]

Examples:
```python
TECH = get_technology()
mzi = MZI(wg_length=600, waveguide_type=TECH.WG.FWG.C.WIRE)
fp.plot(mzi)
```
![MZI](images/mzi.png)
"""

side_length: float = fp.PositiveFloatParam(default=400)
dc_length: float = fp.FloatParam(default=100)
arm_spacing: float = fp.FloatParam(default=60)
wg_length: float = fp.FloatParam(default=100)
gc_spacing: float = fp.FloatParam(default=50)
waveguide_type: WG.FWG.C = fp.WaveguideTypeParam(type=WG.FWG.C)
MZI_unit: fp.IDevice = fp.DeviceParam(type=MZI, port_count=4, required=False)
grating_coupler: fp.IDevice = fp.DeviceParam(type=GratingCoupler, port_count=1, required=False)

def _default_waveguide_type(self):
    return get_technology().WG.FWG.C.WIRE

def _default_MZI_unit(self):
    return MZI(waveguide_type=self.waveguide_type, arm_spacing=self.arm_spacing,
             dc_length=self.dc_length, wg_length=self.wg_length)

def _default_grating_coupler(self):
    return GratingCoupler(waveguide_type=self.waveguide_type)

def build(self) -> Tuple[fp.InstanceSet, fp.ElementSet, fp.PortSet]:
    insts, elems, ports = super().build()
````

```
        # fmt: off

        waveguide_type = self.waveguide_type
        port_names = self.port_names
        MZI_0 = self.MZI_unit.translated(0,0)
        MZI_1 = self.MZI_unit.rotated(degrees=120).translated(self.side_length / 4, self.side_length / 4 *
(3) ** (0.5))
        MZI_2 = self.MZI_unit.rotated(degrees=60).translated(-self.side_length / 4, self.side_length / 4 *
(3) ** (0.5))
        MZI_3 = self.MZI_unit.translated(self.side_length / 2, self.side_length / 2 * (3) ** (0.5))
        MZI_4 = self.MZI_unit.rotated(degrees=60).translated(self.side_length * 3 / 4, self.side_length / 4
* (3) ** (0.5))
        gc_0 = self.grating_coupler.rotated(degrees=180).translated(-self.side_length / 4 * 3, -
self.gc_spacing)
        gc_1 = self.grating_coupler.rotated(degrees=180).translated(-self.side_length / 4 * 3, 10)
        gc_2 = self.grating_coupler.rotated(degrees=180).translated(-self.side_length / 4 * 3,
self.side_length / 2 * (3) ** (0.5) - 10 )
        gc_3 = self.grating_coupler.rotated(degrees=180).translated(-self.side_length / 4 * 3,
self.side_length / 2 * (3) ** (0.5) + self.gc_spacing)
        gc_4 = self.grating_coupler.translated(-self.side_length / 4 * 3 + self.side_length * 2, -
self.gc_spacing)
        gc_5 = self.grating_coupler.translated(-self.side_length / 4 * 3 + self.side_length * 2, 10)
        gc_6 = self.grating_coupler.translated(-self.side_length / 4 * 3 + self.side_length * 2,
                                                              self.side_length / 2 * (3) ** (0.5) - 10)
        gc_7 = self.grating_coupler.translated(-self.side_length / 4 * 3 + self.side_length * 2,
                                                              self.side_length / 2 * (3) ** (0.5) +
self.gc_spacing)
        insts += fp.Linked(
            link_type=waveguide_type,
            bend_factory=self.waveguide_type.BEND_EULER,
            links=[
                MZI_0["op_1"] >> gc_0["op_0"],
                MZI_2["op_0"] >> gc_1["op_0"],
                MZI_2["op_3"] >> gc_2["op_0"],
                MZI_3["op_0"] >> gc_3["op_0"],
                MZI_0["op_2"] >> gc_4["op_0"],
                MZI_4["op_1"] >> gc_5["op_0"],
                MZI_4["op_2"] >> gc_6["op_0"],
                MZI_3["op_3"] >> gc_7["op_0"],
                MZI_0["op_0"] >> MZI_2["op_1"],
                MZI_0["op_3"] >> MZI_1["op_0"],
                MZI_1["op_1"] >> MZI_4["op_0"],
                MZI_4["op_3"] >> MZI_3["op_2"],
                MZI_1["op_2"] >> MZI_3["op_1"],
                MZI_1["op_3"] >> MZI_2["op_2"],
```

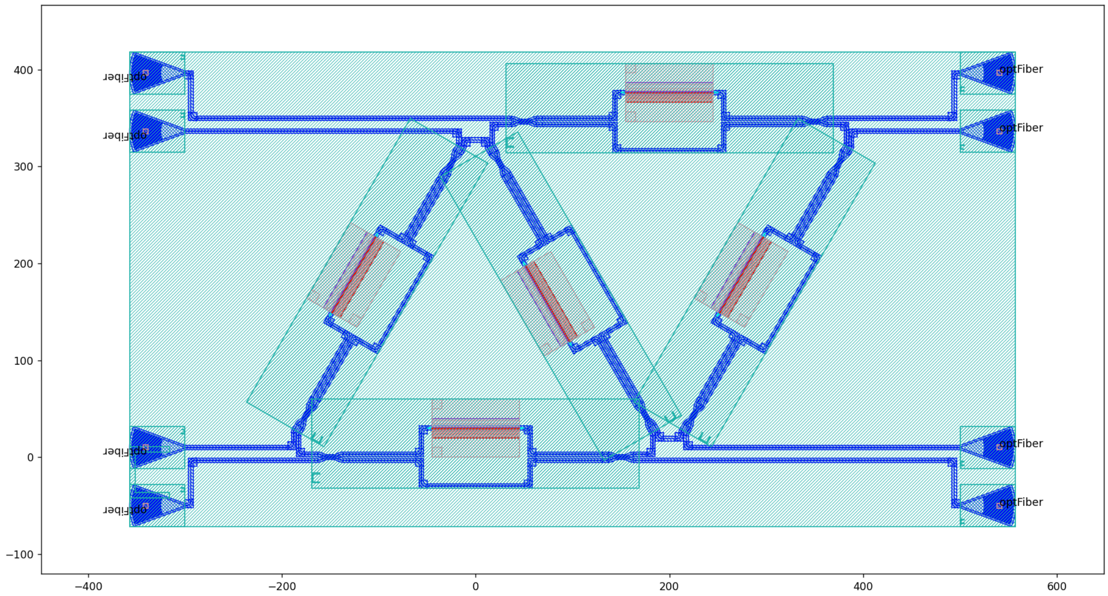
```
        ],
        ports=[],
    )
    # fmt: on
    return insts, elems, ports
```

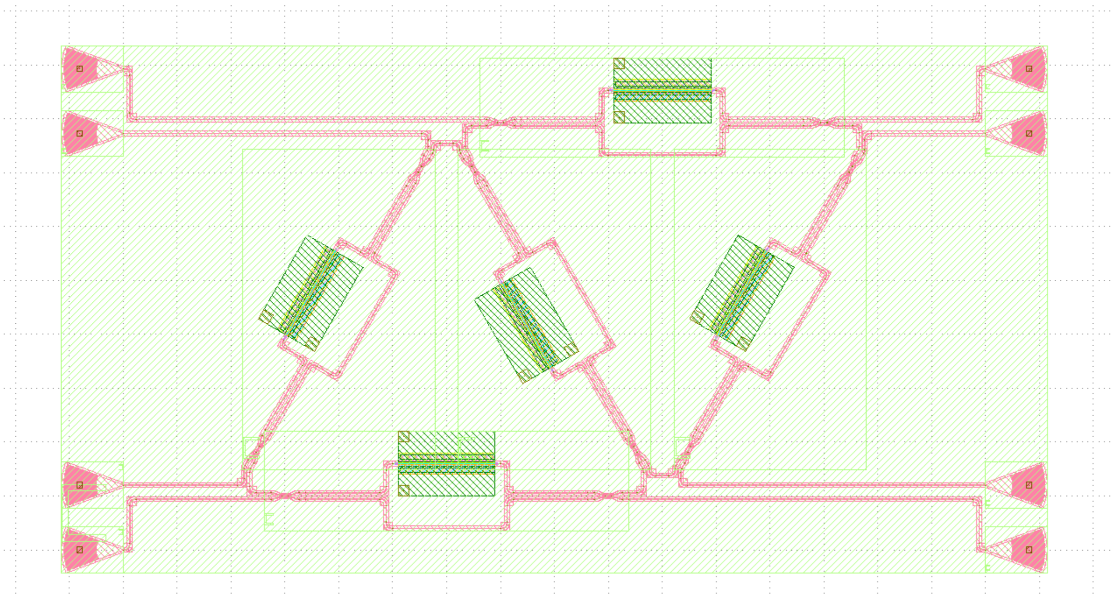
在这段代码包含以下几个部分：首先是进行默认参数的设置，比如波导类型、光栅耦合器类型等；随后实例化了各个单元器件，包括五个 MZI 和八个光栅耦合器，并且在实例化的过程中就定义好了坐标和旋转角度。在这一部分，光栅耦合器的坐标设置需微调，以避免飞线的产生。另外，在本例中，三角形网络的边长（side length）以及光栅耦合器的间隔（gc spacing）都是可调的。随后进行器件的连接（Linked）。

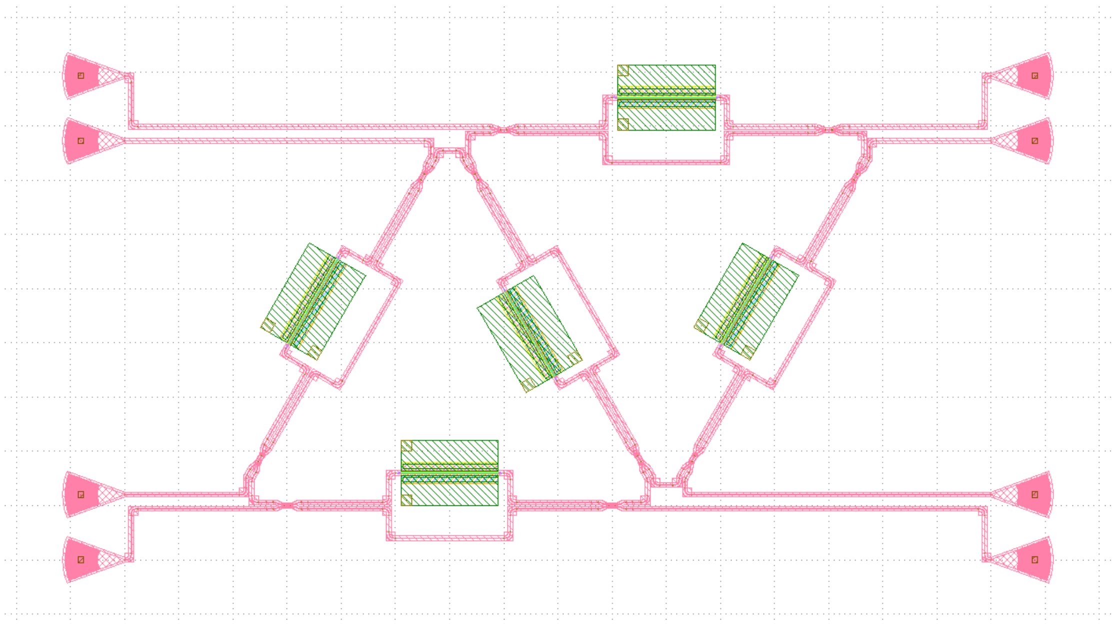
最后在主函数里生成 gds 文件：

```
if __name__ == "__main__":
    from gpdk.util.path import local_output_file

    gds_file = local_output_file(__file__).with_suffix(".gds")
    library = fp.Library()

    TECH = get_technology()
    # =============================================================
    # fmt: off

    mesh = MZI_triangle_mesh_with_GC()
    library += mesh

    # fmt: on
    # =============================================================
    fp.export_gds(library, file=gds_file)
    fp.plot(library)
```

生成的版图如下所示：


400
300
200
100
0
−100
−400
−200
0
200
400
600
optFiber
optFiber
optFiber
optFiber





附录：基于 comp scan 进行批量化光栅耦合器连接

构建光栅耦合器的方法除了上面所讲的方法（即手动生成八个光栅耦合器，指定它们的坐标和旋转角度，最后使用 Linked 方法进行连接），还可以基于 comp scan 方法来完成。

在主函数里基于批量化函数 Comp_scan 可以直接自动生成 8 个光栅耦合器，并进行自动连接和布线：

```
if __name__ == "__main__":
    from gpdk.util.path import local_output_file

    gds_file = local_output_file(__file__).with_suffix(".gds")
    library = fp.Library()

    TECH = get_technology()
    # =============================================================
    # fmt: off
    def bend_factories(waveguide_type: fp.IWaveguideType):
        return TECH.WG.FWG.C.WIRE.BEND_EULER

    def gc_factory(at: fp.IRay, device: fp.IDevice):
        return GratingCoupler(), "op_0"

    mesh = MZI_triangle_mesh()

    blocks = [Block(mesh)]
    library += CompScan(
        name="comp_scan",
        spacing=200,
        width=1000,
        blocks=blocks,
        bend_factories=bend_factories,
        waveguide_type=TECH.WG.FWG.C.WIRE,
        connection_type=TECH.WG.FWG.C.WIRE,
        fiber_coupler_factory=gc_factory,
    )



    # fmt: on
    # =============================================================
    fp.export_gds(library, file=gds_file)
    fp.plot(library)
```

但这一功能仍在开发中，直接使用会出现布线交错的问题。