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 1](#_Toc123672774)

[第二部分 构建可编程三角形MZI网络 5](#_Toc123672775)

[带有8个外置光学端口的MZI Mesh（MZI\_triangle\_mesh） 5](#_Toc123672776)

[带有8个光栅耦合器的MZI Mesh（MZI\_triangle\_mesh\_with\_GC） 8](#_Toc123672777)

[附录：基于comp scan进行批量化光栅耦合器连接 13](#_Toc123672778)

# 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"],

],

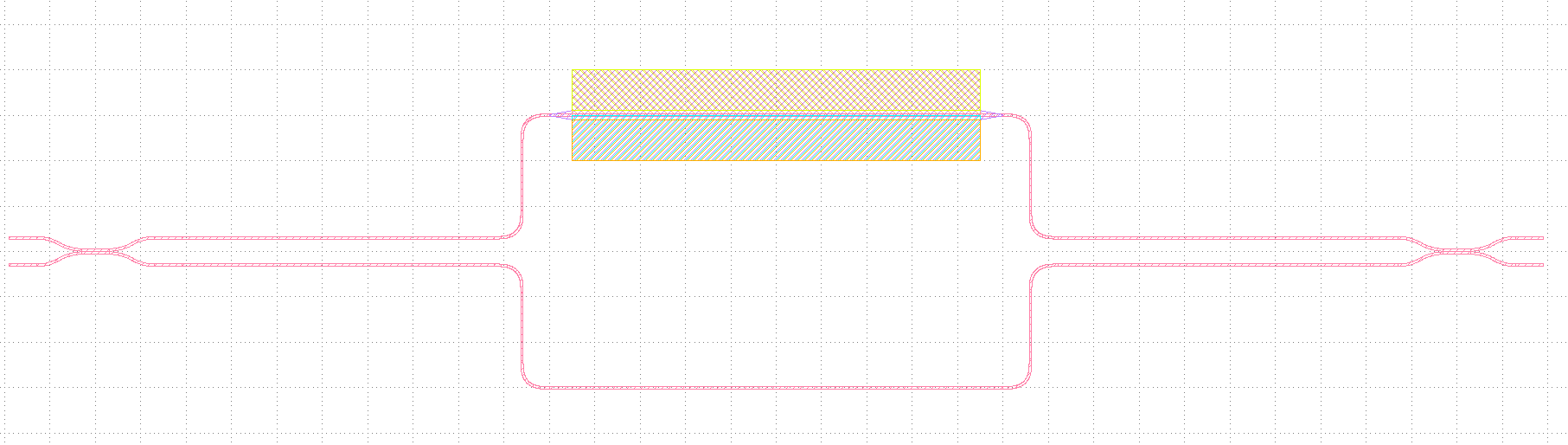
ports=[],

)

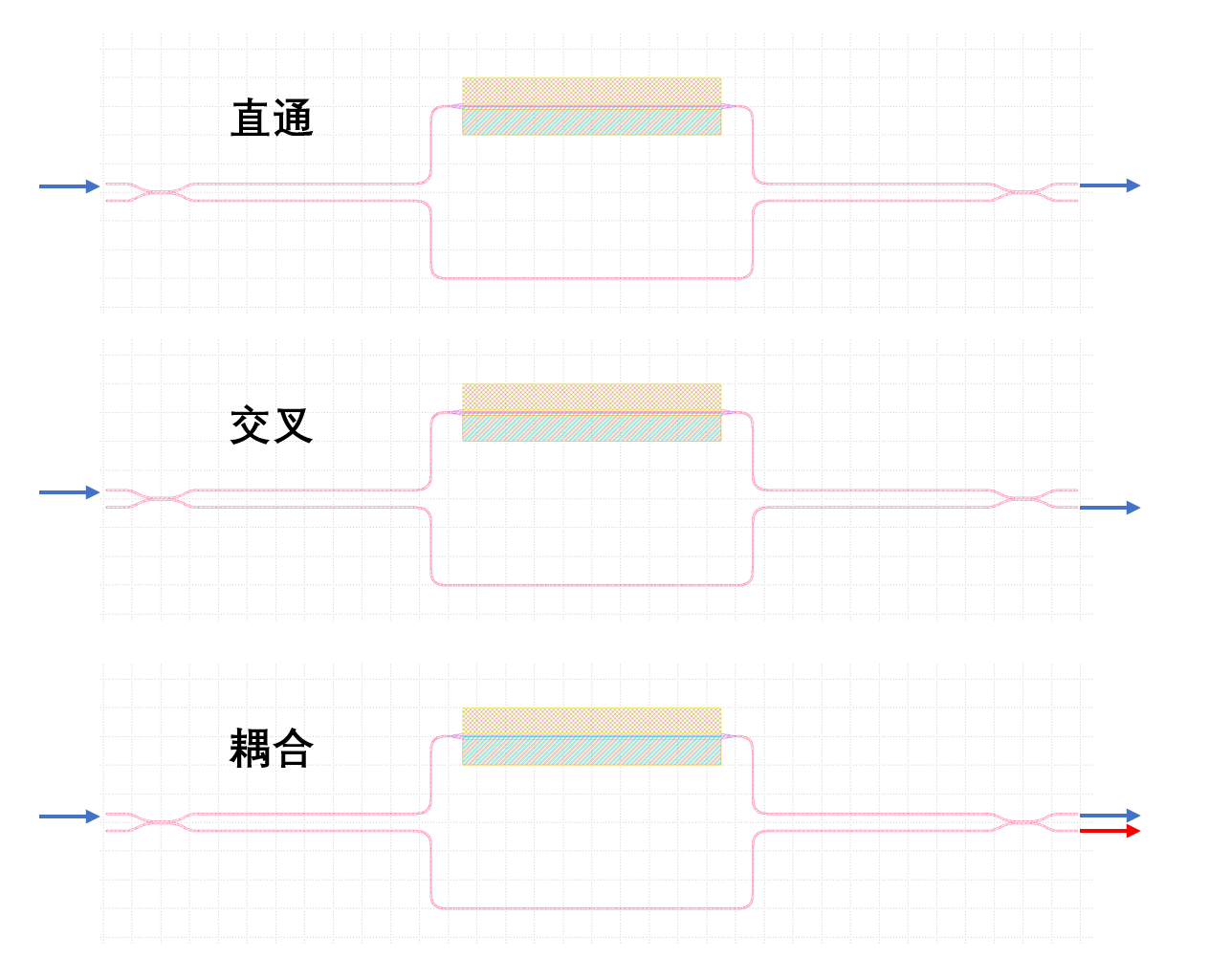
# fmt: on

return insts, elems, ports

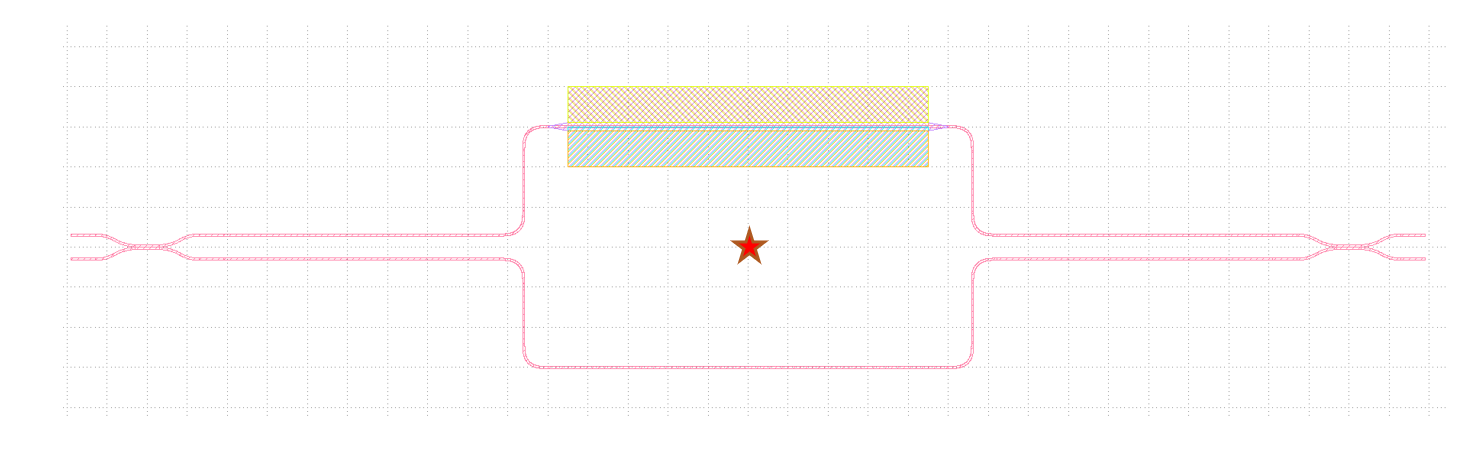
在本案例中，MZI由两个定向耦合器、一个移相器以及一个直波导构成，基本结构如下图所示：



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()

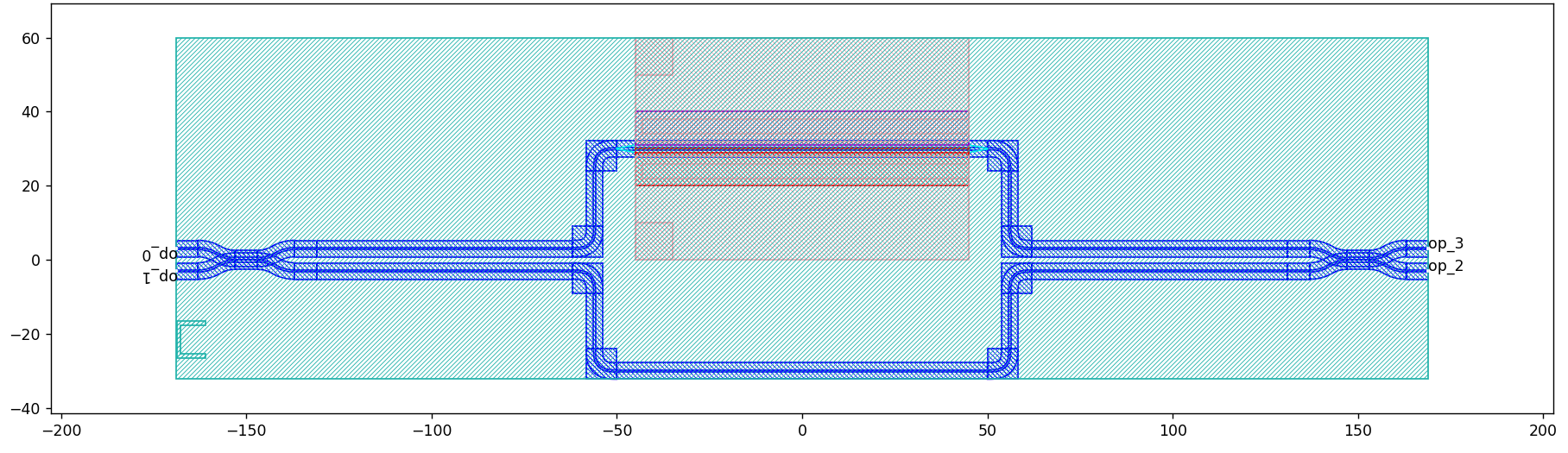
# fmt: on

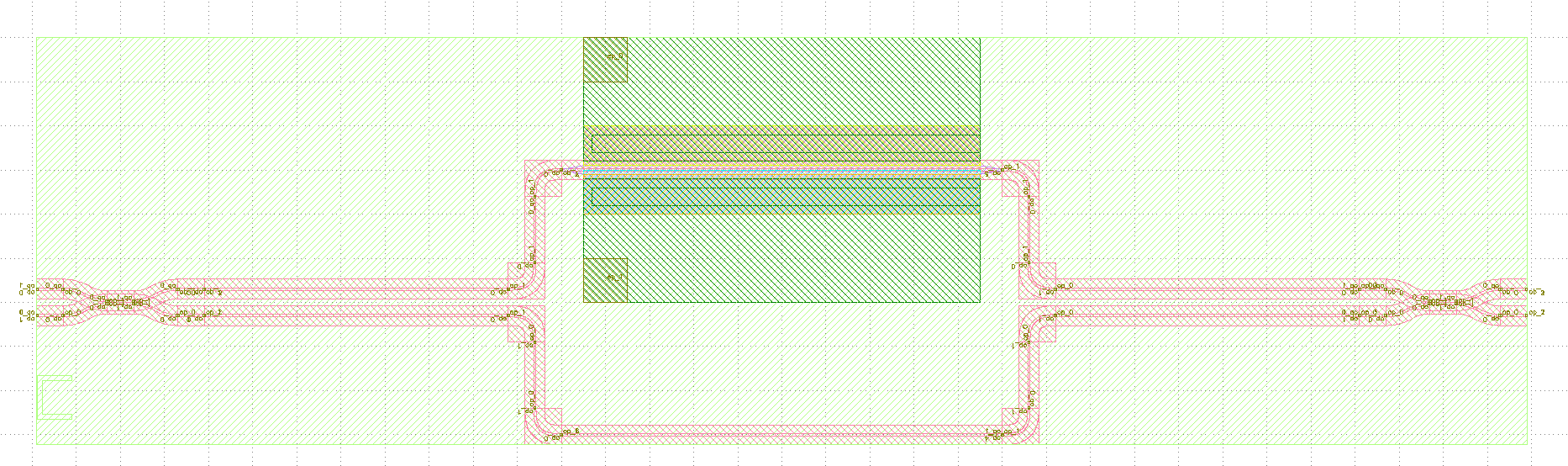
# =============================================================

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"],

],

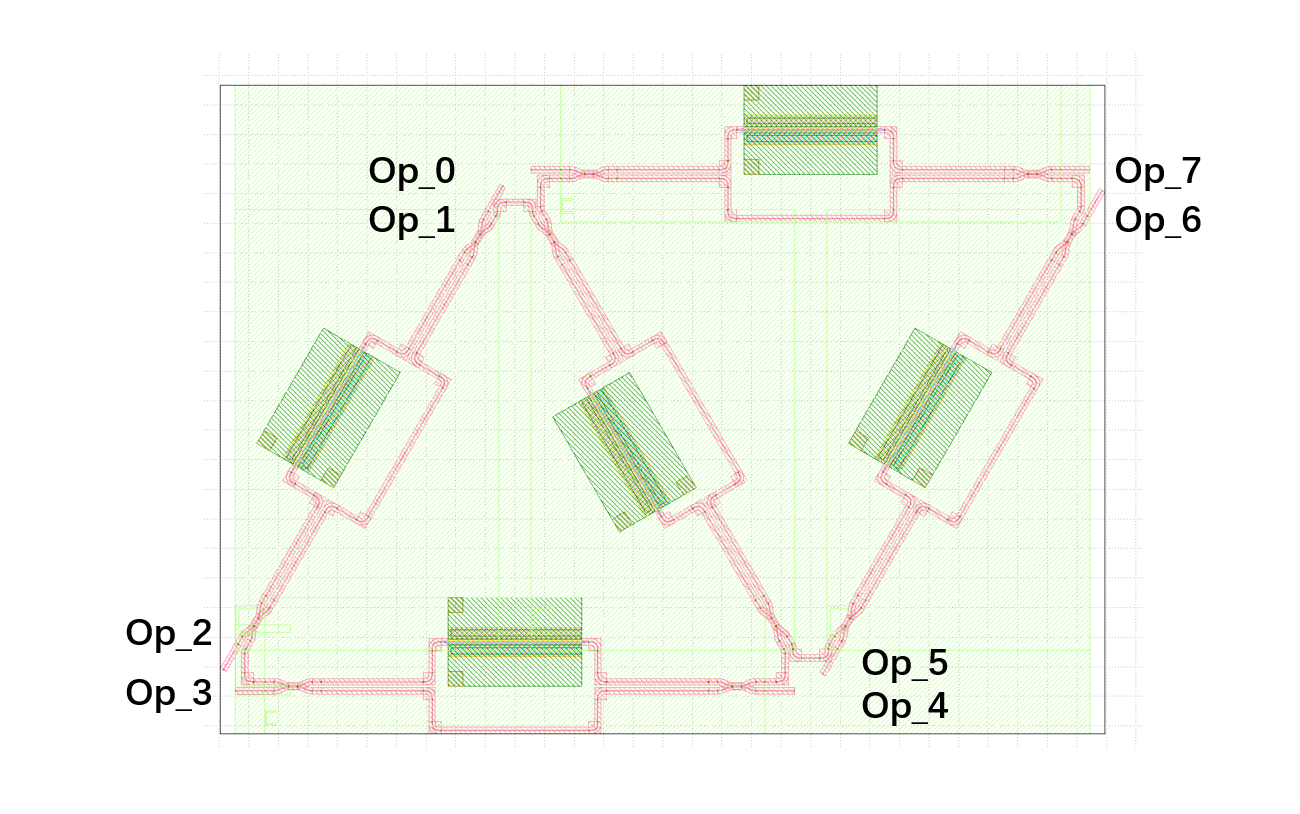
ports=[],

)

# 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"):

"""

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

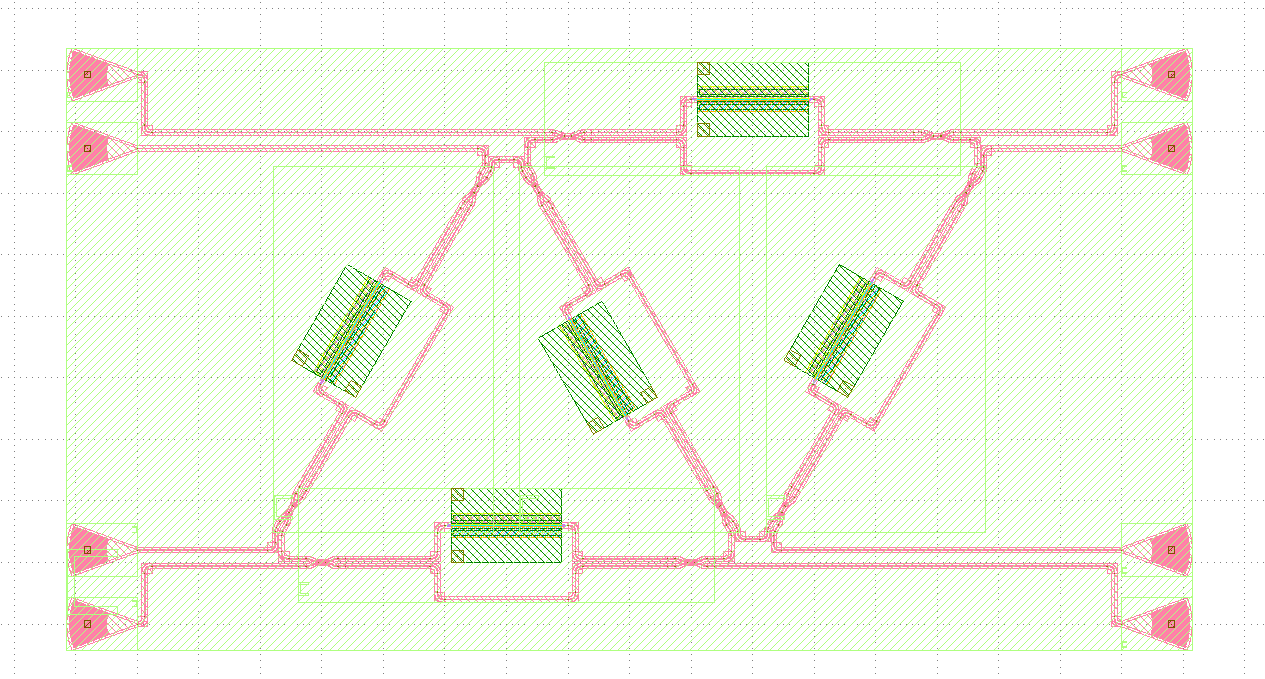
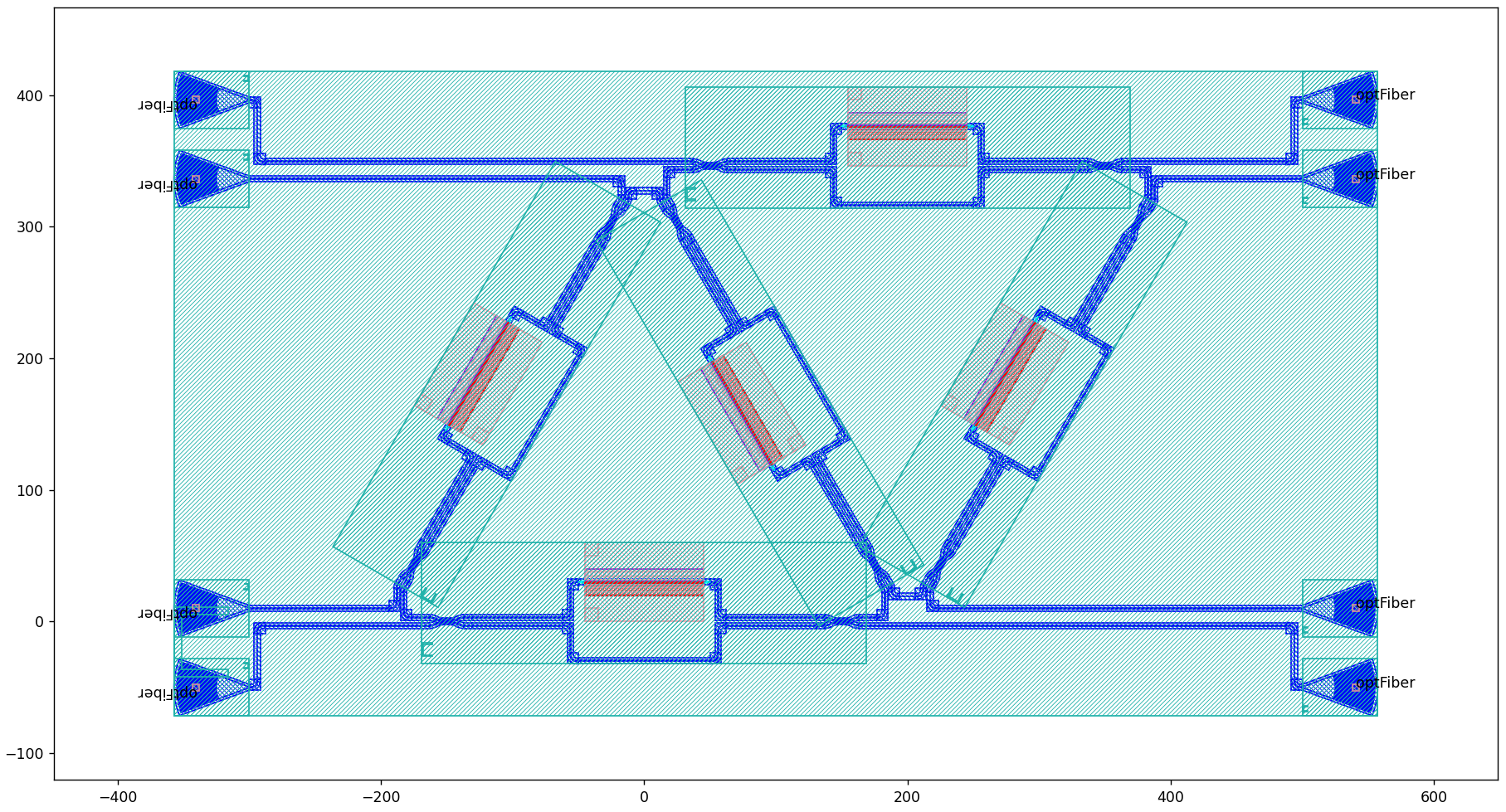
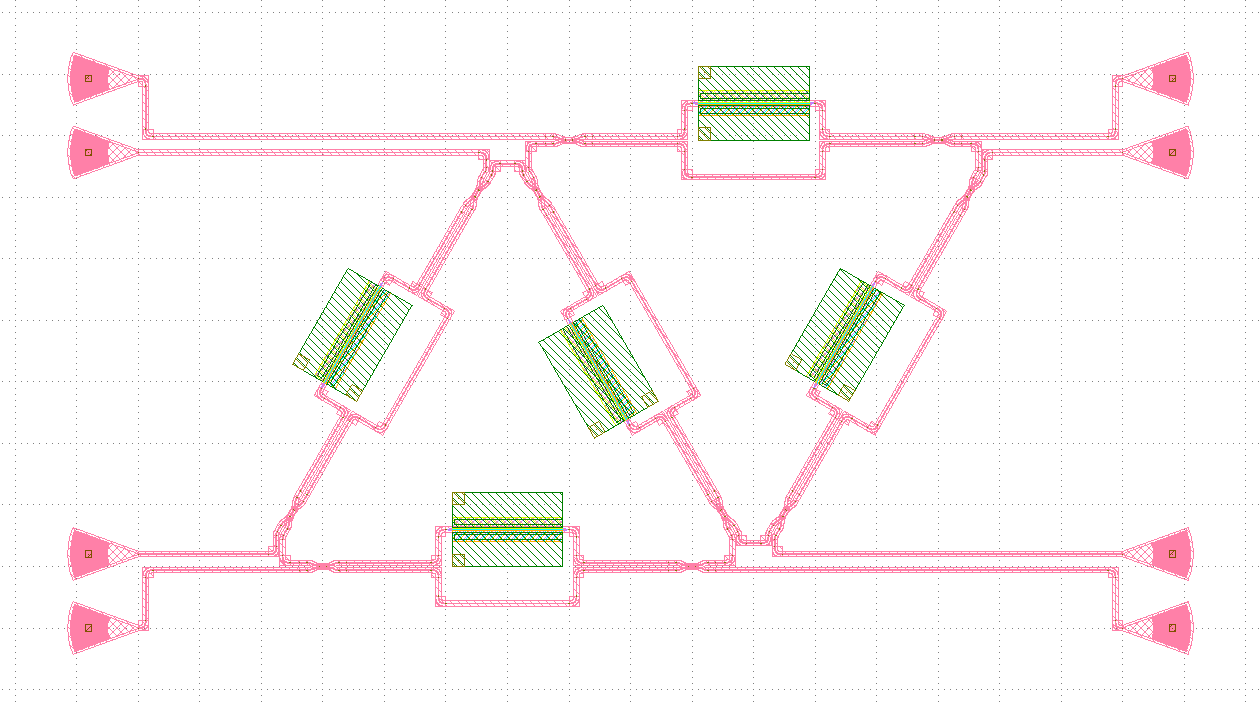
# fmt: on

# =============================================================

fp.export\_gds(library, file=gds\_file)

fp.plot(library)

生成的版图如下所示：

# 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。

