|  |
| --- |
|  |
| 程序设计报告 |
| 项目名称：硅光统计模型的开发与验证  技术团队： Luceda NV.  项目经理： Pieter Dumon |
| 2025年 06月 17日 |

**目录**

[1 摘要 1](#_Toc205650423)

[2 程序框架 2](#_Toc205650424)

[2.1 DoE线路设计程序框架 2](#_Toc205650425)

[2.2 DoE器件设计程序框架 7](#_Toc205650426)

[2.3 DoE线路仿真程序框架 12](#_Toc205650427)

[3 DoE程序设计 13](#_Toc205650428)

[3.1 Grating coupler DoE程序设计 13](#_Toc205650429)

[3.2 Waveguide 有效折射率和群折射率 DoE程序设计 17](#_Toc205650430)

[3.3 Waveguide 传输损耗 DoE程序设计 20](#_Toc205650431)

[3.4 Waveguide 弯曲损耗 DoE程序设计 24](#_Toc205650432)

[3.5 Waveguide Crossing DoE程序设计 30](#_Toc205650433)

[3.6 Transition DoE程序设计 33](#_Toc205650434)

[3.7 1x2 MMI DoE程序设计 39](#_Toc205650435)

[3.8 2x2 MMI DoE程序设计 47](#_Toc205650436)

[3.9 Ring Resonator DoE程序设计 52](#_Toc205650437)

[3.10 Directional Coupler DoE程序设计 63](#_Toc205650438)

[3.11 Thermal Phase Shifter DoE程序设计 73](#_Toc205650439)

[3.12 Phase Shifter DoE程序设计 84](#_Toc205650440)

[3.13 MZM DoE程序设计 94](#_Toc205650441)

[3.14 RM DoE程序设计 99](#_Toc205650442)

[3.15 PhotoDetector DoE程序设计 105](#_Toc205650443)

[4 紧凑模型程序设计 109](#_Toc205650444)

[4.1 Grating Coupler 紧凑模型程序设计 109](#_Toc205650445)

[4.2波导紧凑模型程序设计 113](#_Toc205650446)

[4.3 Crossing紧凑模型程序设计 118](#_Toc205650447)

[4.4 Transition紧凑模型程序设计 121](#_Toc205650448)

[4.5 MMI1x2紧凑模型程序设计 123](#_Toc205650449)

[4.6 MMI2x2紧凑模型程序设计 127](#_Toc205650450)

[4.7 Ring紧凑模型程序设计 128](#_Toc205650451)

[4.8 DirectionalCoupler 紧凑模型程序设计 132](#_Toc205650452)

[4.9 Thermal phase shifter 紧凑模型程序设计 137](#_Toc205650453)

[4.10 PN Phase shifter 紧凑模型程序设计 143](#_Toc205650454)

[4.11 MZM 紧凑模型程序设计 148](#_Toc205650455)

[4.12 RingModulator紧凑模型程序设计 149](#_Toc205650456)

[4.13 PhotoDetector 紧凑模型程序设计 154](#_Toc205650457)

[5 验证程序设计 157](#_Toc205650458)

[5.1验证线路版图程序设计 158](#_Toc205650459)

[5.1.1光开关的设计 158](#_Toc205650460)

[5.1.2可调延时线的设计 162](#_Toc205650461)

[5.2验证线路仿真程序设计 164](#_Toc205650462)

[5.2.1 光开关仿真程序设计 164](#_Toc205650463)

[5.2.2 可调延时线的仿真程序设计 166](#_Toc205650464)

[6 测试数据处理程序设计 169](#_Toc205650465)

[6.1 测试数据分类程序设计 170](#_Toc205650466)

[6.2 测试数据可视化程序设计 182](#_Toc205650467)

[7模型参数提取程序设计 190](#_Toc205650468)

[7.1 损耗与带宽提取程序设计 190](#_Toc205650469)

[7.2 有效折射率和群折射率提取程序设计 194](#_Toc205650470)

[7.3 有源器件带宽与调制效率提取程序设计 201](#_Toc205650471)

[8 统计学建模程序设计 207](#_Toc205650472)

[8.1 基于样品数量的统计学建模程序设计 207](#_Toc205650473)

[8.2 基于位置分布的统计学建模程序设计 216](#_Toc205650474)

[8.3 基于工艺波动的统计学建模程序设计 222](#_Toc205650475)

# 1 摘要

本程序设计报告系统性地阐述了DoE（Design of Experiment）设计、模型参数提取、模型精度分析以及器件建模等关键环节的程序设计和实现过程。报告首先详尽地介绍了DoE程序设计，讨论了实验设计方法的选择、参数优化的策略，以及在确保实验结果具有统计显著性的前提下，如何进行设计流程的迭代和完善。DoE设计是整个程序开发的基础，旨在通过优化实验设计实现高效的数据收集与可靠的实验结果。

在模型参数提取程序部分，报告重点讲述了如何通过实验数据的采集与分析，提取出能够准确描述器件物理特性的重要参数。提取过程不仅包括数据处理技术，还涉及多种参数优化算法，确保所获得的模型参数在不同工作条件下具有广泛的适用性和准确性。参数的精确提取是后续建模和仿真的核心，因此本部分在细节和方法论上进行了深入探讨。

模型精度分析程序旨在评估所提取的模型参数的精度与可靠性。报告通过将仿真结果与实验数据进行对比，采用多种定量和定性分析方法，验证了模型的准确性。该部分内容对模型在实际应用中的可行性做出了详细评估，并提出了可能的优化方案，以进一步提升模型的预测能力。

器件建模程序则结合了前述参数提取与精度分析的结果，构建了一个全面而精确的器件模型。该模型能够在仿真环境中有效地模拟器件的工作状态和行为，为进一步的设计优化与工艺开发提供了可靠的依据。报告在此部分详细描述了模型构建的理论基础、实现步骤，以及如何通过模型的迭代优化，实现对器件特性的精确描述。

总体而言，本程序设计报告通过对各个环节的详细阐述，展现了从DoE设计到最终器件模型构建的完整流程。每个环节相互联系、相互作用，形成了一个闭环系统，为器件的高效设计和性能优化提供了科学的理论依据和实践指导。

# 2 程序框架

此次测试线路DoE（Design of Experiment）的程序主要采用IPKISS的线路设计函数i3.Circuit的框架进行设计，对于一些辅助的测试器件会采用IPKISS的器件测设计函数i3.Pcell的框架进行设计，除此之外还有专门的用于线路仿真的代码框架来对测试线路进行仿真。

## 2.1 DoE线路设计程序框架

测试线路DoE采用的IPKISS中的i3.Circuit的代码模板有如下结构：

class DoE(i3.Circuit):  
# 在这里定义线路的属性  
 def \_default\_insts(self):  
 insts = {  
 # 在这里添加器件  
 }  
 return insts  
  
 def \_default\_specs(self):  
 specs = [  
 # 在这里放置器件和连线  
 ]  
  
 return specs  
  
 def \_default\_exposed\_ports(self):  
 exposed\_ports = {  
 # 在这里给端口命名  
 }  
 return exposed\_ports

该代码框架中主要包含四个部分的内容：  
1、定义线路的属性  
2、线路中的器件  
3、器件的放置与连接  
4、端口命名

第一部分是定义线路的属性，线路的属性是用于定义整个线路结构的参数，其中包括器件参数、波导参数、线路结构参数等。器件参数是定义用于线路设计的所有子器件种类，如线路中的光栅耦合器：

gc = i3.ChildCellProperty()  
def \_default\_gc(self):  
 return pdk.GC\_TE\_1550()

这些子器件可以是PDK中的器件也可以是自定义的器件，甚至可以是定义的子线路，器件属性的参数化方便测试线路器件的调整，比如将C波段的光栅耦合器替换到O波段等，除此之外，器件还可以赋值自定义的线路，可用于层级化的线路设计，从而大大简化大型复杂线路的设计。

波导参数主要是定义用于器件之间连接的波导，在设计测试线路时，可以调用CSIP180A pdk中已经定义好的波导样板来连接器件。

trace\_template = i3.TraceTemplateProperty()  
def \_default\_trace\_template(self):  
 return pdk.SWG450\_CTE()

结构参数是用来定义线路的结构，如光栅耦合器的间隔，弯曲半径，MZI的臂长差、螺旋线的长度等。将设计线路的属性参数化可以方便参数的扫描和调整。

bend\_radius = i3.PositiveNumberProperty(default=10.0, doc="the bend radius in the circuit")  
space\_x = i3.PositiveNumberProperty(default=450.0)  
space\_y = i3.PositiveNumberProperty(default=50.0)  
y\_offset = i3.PositiveNumberProperty(default=300)  
waveguide\_space = i3.PositiveNumberProperty(default=20)  
number = i3.IntProperty(default=4)

第二部分是给线路添加器件，第一部分已经定义了线路中的器件种类，接下来就是定义线路中具体的器件，比如在线路中的多个光栅耦合器：

def \_default\_insts(self):  
 insts = {}  
 for m in range(self.number):  
 insts["gc\_in\_{}".format(m)] = self.gc  
 insts["gc\_out\_{}".format(m)] = self.gc  
 return insts

第三部分是将器件放置在所需位置以及连接对应的端口。可以用IPKISS中的i3.Place函数将器件放置在设定的坐标点上，如：

import si\_fab.all as pdk

import ipkiss3.all as i3

x\_coord = 20

y\_coord = 40

# the output port of the grating coupler will be placed at (x\_coord, y\_coord)

place\_spec = i3.Place("gc:out", (x\_coord, y\_coord))

gc = pdk.FC\_TE\_1550()

circuit = i3.Circuit(insts={"gc": gc}, specs=[place\_spec])

circuit.Layout().visualize(annotate=True)

该代码运行结果如下：

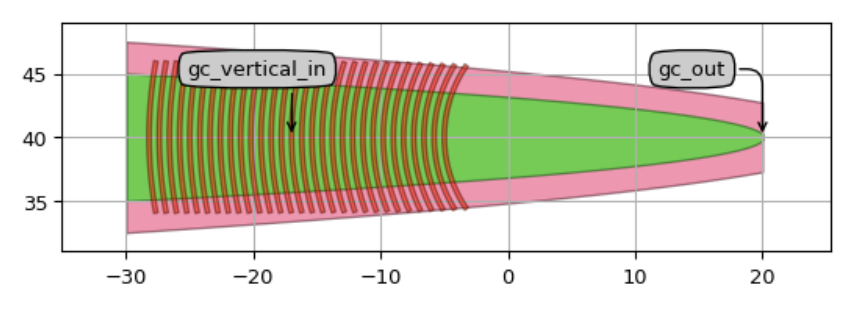


图 1 器件放置函数代码运行结果

通过i3.Place中也可以实现器件的相对放置：

import si\_fab.all as pdk

import ipkiss3.all as i3

import matplotlib.pyplot as plt

# the second grating coupler will be shifted by (20, 40) with respect to the first one

placerelative\_spec = i3.Place("gc2", (20, 40), relative\_to="gc1")

gc = pdk.FC\_TE\_1550()

circuit = i3.Circuit(

insts={"gc1": gc, "gc2": gc},

specs=[i3.Place("gc1:out", (0, 0)), placerelative\_spec],

)

circuit.Layout().visualize(annotate=True)

i3.Place中的relative\_to接口可以实现器件的相对放置，该代码运行结果：

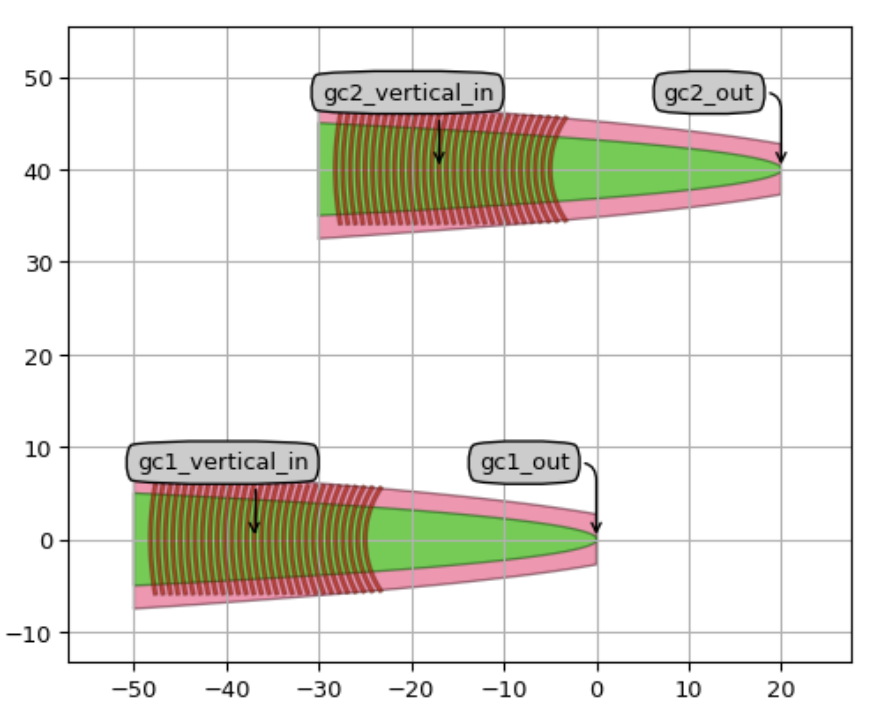


图 2 器件相对放置代码运行结果

器件的连接可以用IPKISS中的i3.ConnectManhattan函数，该函数可以通过一段或者多段横平竖直的波导连接两个器件的端口，该函数可以通过bend\_radius设置连接波导的弯曲半径，通过rounding\_algorithm设置弯曲算法，可选的波导弯曲算法为，圆弧弯、欧拉弯以及贝塞尔弯曲，在DoE的设计中我们采用的是欧拉弯曲来降低弯曲波导的反射：

ra = i3.EulerRoundingAlgorithm(p=0.70)

i3.ConnectManhattan(  
 "gc\_in\_{}:wg".format(m),  
 "gc\_out\_{}:wg".format(m),  
 control\_points=[i3.V(i3.END - self.space\_x / 2.0 -20+ self.waveguide\_space \* m)],  
 bend\_radius=self.bend\_radius,  
 rounding\_algorithm=self.ra,  
),

此外，还可以通过control\_points**来控制波导的路由，这也是在**DoE**设计程序中经常用到的, 其用法如下：**

import si\_fab.all as pdk # noqa: F401

from picazzo3.fibcoup.curved import FiberCouplerCurvedGrating

from ipkiss3 import all as i3

from pylab import plt

# Defining the pcells

gr = FiberCouplerCurvedGrating()

obstacle = pdk.BONDPAD\_5050()

control\_points = [

i3.V(-20, relative\_to="obstacle@W"),

i3.H(20, relative\_to="obstacle@N"),

i3.V(20, relative\_to="obstacle@E"),

]

circuit = i3.Circuit(

insts={"obstacle": obstacle, "grb1": gr, "grb2": gr},

specs=[

i3.Place("grb1", position=(0, 0)),

i3.Place("obstacle", position=(100, 0)),

i3.Place("grb2", position=(200, 0), angle=180),

i3.ConnectManhattan(

"grb1:out",

"grb2:out",

control\_points=control\_points,

),

],

)

circuit\_layout = circuit.Layout()

fig = plt.figure()

circuit\_layout.visualize(figure=fig, show=**False**)

# plot annotations

ax = fig.gca()

ax.plot([25, 175], [45, 45], color="black", linestyle="--", linewidth=1.0)

ax.plot([25, 175], [25, 25], color="black", linestyle="--", linewidth=1.0)

ax.annotate(

"", xy=(80, 25), xycoords="data", xytext=(80, 45), textcoords="data", arrowprops={"arrowstyle": "<->"}

)

ax.annotate("20", xy=(91, 30), xycoords="data", xytext=(-4, 5), textcoords="offset points")

ax.annotate(

'i3.H(20, relative\_to="obstacle@N")',

xy=(70, 48),

xycoords="data",

xytext=(-4, 5),

textcoords="offset points",

)

ax.set\_ylim([-40, 70])

plt.show()

代码运行结果如下：

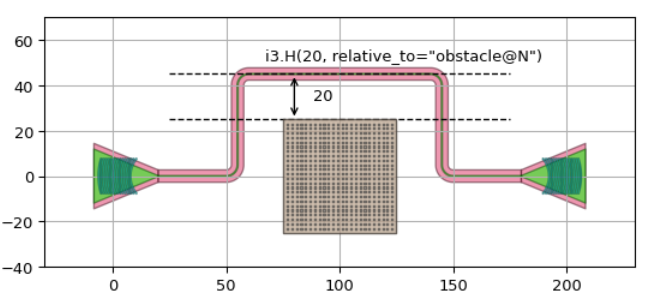


图 3 线路路由程序运行结果

第四部分是给定义的线路端口命名，这部分代码的目的是给线路所有端口按一定规律命名，方便后续线路仿真以及作为子线路在其他线路中参与布线，如给Waceguide crossingde DoE中所有光栅耦合器端口的命名：

def \_default\_exposed\_ports(self):  
 exposed\_ports = {}  
 for m in range(4):  
 exposed\_ports["fgc\_in\_{}:vertical\_io".format(m)] = "in{}".format(m)  
 exposed\_ports["fgc\_out\_{}:vertical\_io".format(m)] = "out{}".format(m)  
 return exposed\_ports

命名后的线路端口名称如下：

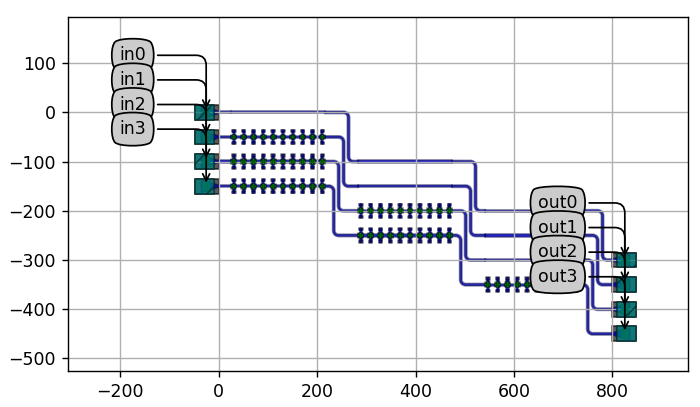


图 4 测试线路端口命名

基于i3.Circuit的代码框架，我们能够高效地完成测线电路的版图设计。这一框架不仅提供了高度灵活的设计功能，还能够轻松集成各种设计元素，从而加速了整个电路设计流程。通过使用i3.Circuit，设计者可以快速定义和优化测线电路的布局，将复杂的电路结构清晰地展现在版图中。这种高效的设计方式显著减少了设计周期，并确保了版图的准确性和可制造性，为后续的制造和测试环节奠定了坚实的基础。

## 2.2 DoE器件设计程序框架

在DoE的设计中，同样需要设计一些器件来测试，如设计微环谐振腔等。IPKISS中器件设计通常采用i3.Pcell的框架，其结构如下：

class DoE\_device(i3.PCell):  
 \_name\_prefix = "demo"  
 class Layout(i3.LayoutView):  
 def \_generate\_elements(self, elems):  
 return elems  
 def \_generate\_ports(self, ports):  
 ports += i3.OpticalPort()  
 return ports  
 class Netlist(i3.NetlistFromLayout):  
 pass  
 class CircuitModel(i3.CircuitModelView):  
 def \_generate\_model(self):  
 return CompactModel()

通过i3.Pcell的框架中主要包含三部分内容：

1. 布局视图（Layout)）
2. 网表视图（Netlist）
3. 行为模型视图（CircuitModel）

在i3.Pcell的框架下，布局给出器件的基本形状，网表视图给出器件与外界的连接关系，行为模型视图表征器件的光学性能，以微环谐振腔为例，其设计程序如下：

from csip130c import all as pdk  
import ipkiss3.all as i3  
import numpy as np  
class RingResonator(i3.PCell):  
 trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")  
 def \_default\_trace\_template(self):  
 tt = pdk.StripWaveguideTemplate()  
 tt.Layout(core\_width=0.45)  
 return tt  
 class Layout(i3.LayoutView):  
 ring\_radius = i3.PositiveNumberProperty(default=20, doc="The bend radius of the

ring")  
 spacing = i3.PositiveNumberProperty(default=0.2, doc="The spacing between the ring

and the bus waveguides")  
 wrapping\_angle = i3.NonNegativeNumberProperty(  
 default=0,  
 doc="The angle around which the bus waveguides are wrapped"  
 )  
 wrapping\_bend\_radius = i3.PositiveNumberProperty(  
 default=5,  
 doc="Maximum bend radius for the wrap around"  
 )  
 wrapping\_straight\_length = i3.PositiveNumberProperty(  
 default=2,  
 doc="Straight length for the wrap around"  
 )  
  
 def \_generate\_instances(self, insts):  
  
 trace\_template = self.trace\_template  
  
 ring\_radius = self.ring\_radius  
 spacing = self.spacing  
  
 wrapping\_angle = self.wrapping\_angle  
 wrapping\_bend\_radius = self.wrapping\_bend\_radius  
  
 waveguide\_ring = i3.Waveguide(trace\_template=trace\_template)  
 waveguide\_ring.Layout(shape=i3.ShapeCircle(radius=ring\_radius))  
 bus\_y\_distance\_from\_origin = ring\_radius + trace\_template.core\_width + spacing  
 if wrapping\_angle != 0:  
 shape = i3.ShapeArc(  
 radius=bus\_y\_distance\_from\_origin,  
 start\_angle=-90-wrapping\_angle,  
 end\_angle=-90+wrapping\_angle  
 )  
 else:  
 shape = i3.Shape(  
 points=[  
 (-ring\_radius, -bus\_y\_distance\_from\_origin),  
 (+ring\_radius, -bus\_y\_distance\_from\_origin)  
 ]  
 )  
 bus\_add\_filter\_coupling\_segment = i3.Waveguide(trace\_template=trace\_template)  
 bus\_add\_filter\_coupling\_segment\_lv =

bus\_add\_filter\_coupling\_segment.Layout(shape=shape)  
 bus\_add\_filter\_waveguides = [bus\_add\_filter\_coupling\_segment]  
 if wrapping\_angle != 0:  
 for coupling\_segment\_port in bus\_add\_filter\_coupling\_segment\_lv.ports:  
 route\_bus\_add\_filter = i3.RouteToHorizontal(  
 start\_port=coupling\_segment\_port,  
 bend\_radius=wrapping\_bend\_radius,  
 rounding\_algorithm=i3.SplineRoundingAlgorithm(adiabatic\_angles=(15.0, 15.0))  
 )  
 routing\_waveguide =

i3.RoundedWaveguide(trace\_template=self.trace\_template)  
 routing\_waveguide.Layout(  
 shape=route\_bus\_add\_filter,  
 bend\_radius=wrapping\_bend\_radius,  
 rounding\_algorithm=i3.SplineRoundingAlgorithm(adiabatic\_angles=(15.0, 15.0))  
 )  
 bus\_add\_filter\_waveguides.append(routing\_waveguide)  
 insts += i3.SRef(reference=waveguide\_ring, name="waveguide\_ring")  
 for i, wav in enumerate(bus\_add\_filter\_waveguides):  
 insts += i3.SRef(  
 reference=wav,  
 name="bus\_add\_filter\_{}".format(i)  
 )  
 return insts  
 def \_generate\_ports(self, ports):  
 ports += i3.expose\_ports(  
 instances=self.instances,  
 port\_name\_map={  
 "bus\_add\_filter\_0:in": "in",  
 "bus\_add\_filter\_0:out": "out",  
 }  
 )  
 return ports

class Netlist(i3.NetlistFromLayout):  
 pass

该程序定义的Ringresonator继承自i3.Pcell：

class RingResonator(i3.PCell):

代码开始首先对线路中微环用到的波导进行了定义：

trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")  
 def \_default\_trace\_template(self):  
 tt = pdk.StripWaveguideTemplate()  
 tt.Layout(core\_width=0.45)  
 return tt

接下来是定义器件的布局视图（Layout），对于微环的布局视图，可以通过来构建， generate\_instances是给器件添加子器件（波导在IPKISS中也被当作器件）， 如果器件是通过基本图形（如正方形，圆形等）来搭建，则应该用generate\_elements，微环可以全由波导构建因此代码中只有generate\_instances。

def \_generate\_instances(self, insts):

return insts

添加完基本图形，还需要给器件添加端口来与外界连接，其代码实现如下：

def \_generate\_ports(self, ports):  
 ports += i3.expose\_ports(  
 instances=self.instances,  
 port\_name\_map={  
 "bus\_add\_filter\_0:in": "in",  
 "bus\_add\_filter\_0:out": "out",  
 }  
 )  
 return ports”

对于网表视图，由于在Layout中已经定义了端口信息，因此网表可以直接从Layout中继承：

class Netlist(i3.NetlistFromLayout):  
 pass

行为模型视图只用于器件的仿真就没有添加，有了布局视图和网表视图，就可以来绘制器件的版图和设计对应的DoE线路，通过定义的微环就可以绘制出如下微环的版图：

图示

AI 生成的内容可能不正确。

图 5 基于i3.Pcell代码框架实现的微环的设计

## 2.3 DoE线路仿真程序框架

通过IPKISS的线路仿真引擎Caphe以及CSIP130C PDK中器件的模型数据，可以对测试线路进行仿真来预测测试结果，同时也可以验证测试线路设计的合理性，由于本次流片只涉及频域的测试，这里就只介绍IPKISS的频域仿真，IPKISS频域仿真代码框架如下：

wavelengths = np.linspace(1.5, 1.6, 2001)  
cm = DoE\_circuit().CircuitModel()  
S = cm.get\_smatrix(wavelengths=wavelengths)  
plt.plot(wavelengths, i3.signal\_power\_dB(S["out", "in"]), label="in -> out")  
plt.legend()  
plt.xlabel("Wavelengths [um]")  
plt.ylabel("Transmission")  
plt.show()

该代码中只需要给出仿真的波段，获取线路的行为模型和散射矩阵，就可以绘制端口的透射谱。

# 3 DoE程序设计

## 3.1 Grating coupler DoE程序设计

为提取光栅耦合器（Grating coupler）的带宽、中心波长、插损以及反射，其DoE设计程序采用IPKISS中i3.Circuit框架进行设计，具体见附录1。线路中波导的弯曲损耗bend\_radius、线路中的待测试光栅耦合器，以及为自动化测试设置的space\_x、space\_y以及y\_offset，这三个参数分别对应光栅耦合器横向和纵向的间隔，以及输入与输出光栅对在纵向上的偏移。此外包括光栅连接波导的间隔waveguide\_space，光栅对的数目number，光栅对的初始位置int\_xy，以及为减少弯曲处的损耗设置的欧拉弯曲算法。

参数定义完成就可以在线路中添加器件，通过定义的光栅对的数目以及循环控制可以在线路中添加对应数目的光栅对：

def \_default\_insts(self):  
 insts = {}  
 for m in range(self.number):  
 insts["gc\_in\_{}".format(m)] = self.gc  
 insts["gc\_out\_{}".format(m)] = self.gc  
 return insts

接下来是光栅耦合器的放置与连接，这里用到的线路放置和器件连接函数分别是IPKISS中的i3.Place以及i3.ConnectManhattan。

器件放置：使用i3.Place()将输入、输出光栅耦合器放置指定坐标位置，其中angle可以调整器件的放置角度；

使用 i3.ConnectManhattan 方法连接输入和输出光栅耦合器。连接时使用了控制点control\_points来确定路径。此外，bend\_radius 和rounding\_algorithm可以指定连接路径的弯曲半径和弯曲算法，来保证连接路径的平滑弯曲。整体实现代码如下：

def \_default\_specs(self):  
 specs = []  
 for m in range(self.number):  
 specs += [  
 i3.Place("gc\_in\_{}:wg".format(m), (self.int\_xy[0], self.int\_xy[1] + self.space\_y \* m)),  
 i3.Place(  
 "gc\_out\_{}:wg".format(m),  
 (self.int\_xy[0] + self.space\_x, self.int\_xy[1] + self.space\_y \* m - self.y\_offset),  
 angle=180,  
 ),  
 ]  
 specs += [  
 i3.ConnectManhattan(  
 "gc\_in\_{}:wg".format(m),  
 "gc\_out\_{}:wg".format(m),  
 control\_points=[i3.V(i3.END - self.space\_x / 2.0 -20+ self.waveguide\_space \* m)],  
 bend\_radius=self.bend\_radius,  
 rounding\_algorithm=self.ra,  
 ),  
 ]  
 return specs

最后，为了方便在测试和分析过程中快速识别器件的类型和功能，可以在版图上添加“GC\_C”的TEXT，具体实现代码如下。其中TEXT的图层、内容、位置、对其方式、字体以及高度可以在代码中设置。

class Layout(i3.Circuit.Layout):  
 def \_generate\_elements(self, elems):  
 elems += super().\_generate\_elements(elems)  
 x0 = self.instances["gc\_in\_0"].ports["wg"].position.x  
 elems += i3.PolygonText(  
 layer=i3.TECH.PPLAYER.NONE.LOGO ,  
 text="GC\_C",  
 coordinate=(-30, -60.0),  
 alignment=(i3.TEXT.ALIGN.LEFT, i3.TEXT.ALIGN.BOTTOM),  
 font=i3.TEXT.FONT.STANDARD,  
 height=40.0,  
 )  
 return elems

至此，线路设计工作已圆满完成。通过上述设计流程，我们对光栅耦合器及相关元件进行了详细的布局和建模，完成了所有必需的设计步骤。接下来，将基于这些设计架构进行实例化，并通过write\_gdsii函数一键输出DoE的GDS版图文件，用于后续的制造和测试，最终输出的GDS文件如图 6。同时，我们使用IPKISS中size\_info()函数导出了设计的尺寸信息，以便后续进行合版。

doe\_gc\_c = DOE\_gc\_C(y\_offset=300.0, space\_x=400.0).Layout()  
doe\_gc\_c.visualize(annotate=True)  
doe\_gc\_c.write\_gdsii("gratingcoupler\_Cband.gds")  
si = doe\_gc\_c.size\_info()  
np.savetxt("gratingcoupler\_Cband\_size.txt", si.bounding\_box.points)  
doe\_gc\_o = DOE\_gc\_O(y\_offset=300.0, space\_x=400.0).Layout()  
doe\_gc\_o.visualize(legacy=True)  
doe\_gc\_o.write\_gdsii("gratingcoupler\_Oband.gds")  
si = doe\_gc\_o.size\_info()  
np.savetxt("gratingcoupler\_Oband\_size.txt", si.bounding\_box.points)

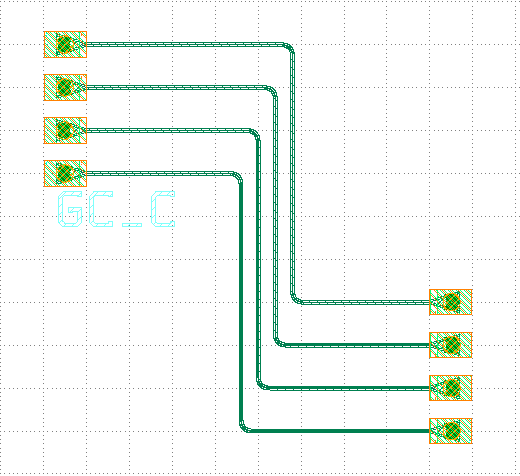


图 6 输出的光栅耦合器的GDS版图

最后，还可以通过如下代码来对设计的线路进行线路仿真，具体过程分为以下四个部分：

1. 定义波长范围: wavelength\_range = np.linspace(1.5, 1.60, 1001) 创建了一个包含1001个点的波长范围，从1.5微米到1.6微米；
2. 提取S矩阵:用CircuitModel().get\_smatrix(wavelengths=wavelength\_range)提取指定波长范围内的散射矩阵（S矩阵）；
3. 绘制传输特性图: 使用 plt.plot 绘制光栅耦合器的传输特性。

整体实现的代码如下，基于该代码生成的光栅耦合器仿真结果如图 7所示。

wavelength\_range = np.linspace(1.5, 1.60, 1001)  
from matplotlib import pyplot as plt  
s\_matrix = DOE\_gc\_C(y\_offset=300.0, space\_x=400.0).CircuitModel().get\_smatrix(wavelengths=wavelength\_range)  
for m in range(4):  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["gc\_in\_{}\_vertical\_io".format(m), "gc\_out\_{}\_vertical\_io".format(m)]), linewidth=2, )  
plt.xlabel("Wavelength [\u03BCm]")  
plt.xlim(1.5, 1.6)  
plt.ylabel("Transmission [dB]")  
plt.show()

代码运行结果图 7：

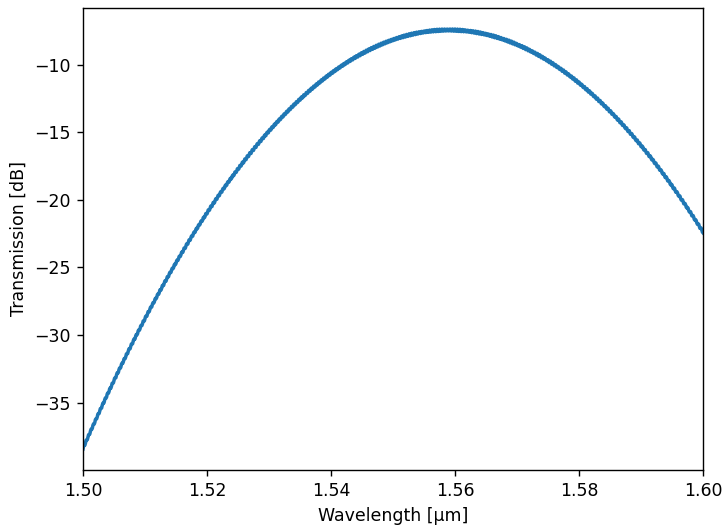


图 7 光栅耦合器测试线路仿真结果

## 3.2 Waveguide 有效折射率和群折射率 DoE程序设计

为提取波导的有效折射率和群折射率，首先需要搭建一个MZI线路，MZI程序设计见附录2，该线路的设计同样采用的是i3.Circuit的设计框架， 线路中定义了两个重要的结构参数：

delta\_l = i3.PositiveNumberProperty(default=20, doc="delta L in one of the arms")  
core\_width = i3.PositiveNumberProperty(default=0.8, doc="width of waveguide under test")

分别为MZI的长度差delta\_l以及波导芯层的宽度core\_width，方便不同类型以及不同宽度波导有效折射率和群折射率的提取。该代码运行结果：

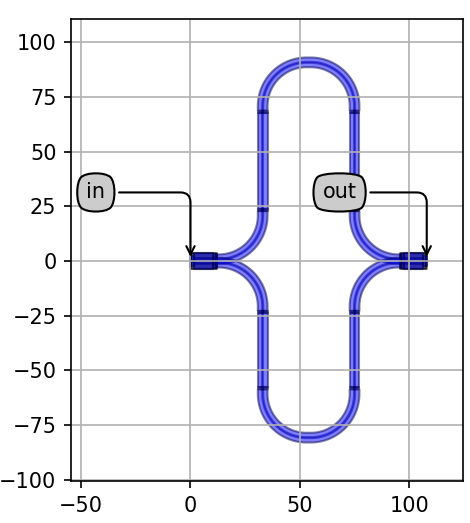


图 8 单个MZI 代码运行结果

设计好单个MZI线路，就可以在此基础上得到MZI阵列以及连接光栅耦合器，来测试不同干涉级次的光谱，MZI阵列设计代码见附录3，该代码运行结果如下：

图示

AI 生成的内容可能不正确。

图 9 代码GenericMZI\_array得到的MZI测试阵列结构

通过扫描波导的宽度，可以得到不同宽度波导对应的MZI测试阵列结构：

allowed\_values = [0.45, 0.55, 0.7, 0.8, 1.0, 1.2]  
for core\_width in allowed\_values:  
 doe = WgDoe\_C(space\_x=400, core\_width=core\_width)  
 lay = doe.Layout()  
 elems\_add, elems\_subt = i3.get\_stub\_elements(lay, layers=[i3.TECH.PPLAYER.FETCH.CLD], stub\_width=0.5,

angle\_threshold=40, grow\_amount=0.6, limit\_stub\_length=False)i3.LayoutCell().Layout(elements=elems\_add + lay.layout).write\_gdsii("wg\_doe\_c\_{}.gds".format(core\_width))  
 si = lay.size\_info()  
 np.savetxt("wg\_doe\_c\_{}\_size.txt".format(core\_width), si.bounding\_box.points)  
  
allowed\_values = [0.35, 0.45, 0.55, 0.65, 0.8, 1.0]  
for core\_width in allowed\_values:  
 doe = WgDoe\_O(space\_x=400, core\_width=core\_width)  
 lay = doe.Layout()  
 elems\_add, elems\_subt = i3.get\_stub\_elements(lay, layers=[i3.TECH.PPLAYER.FETCH.CLD], stub\_width=0.5, angle\_threshold=40, grow\_amount=0.6, limit\_stub\_length=False)i3.LayoutCell().Layout(elements=elems\_add + lay.layout).write\_gdsii("wg\_doe\_o\_{}.gds".format(core\_width))  
 si = lay.size\_info()  
 np.savetxt("wg\_doe\_o\_{}\_size.txt".format(core\_width), si.bounding\_box.points)

如代码所示，条波导在O波段的扫描宽度为：0.35 μm、0.45 μm、0.55 μm、0.65 μm、0.8 μm以及1.0 μm。C波段的条波导宽度为：0.45 μm、0.55 μm、0.7 μm、0.8 μm、1.0 μm以及1.2 μm。

此外，还可以IPKISS的线路仿真引擎Caphe来对设计的测试结构进行仿真，如C波段条波导的仿真代码如下：

allowed\_values = [0.45, 0.55, 0.7, 0.8, 1.0, 1.2]a = 0  
for core\_width in allowed\_values:  
 a += 1  
 doe = WgDoe\_C(space\_x=400, core\_width=core\_width)  
 doe.Layout().visualize(annotate=True)  
 wavelength\_range = np.linspace(1.5, 1.6, 1001)  
 s\_matrix = doe.CircuitModel().get\_smatrix(wavelengths=wavelength\_range)  
 plt.figure(a)for m in range(4):  
 if m == 1:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]),  
 linewidth=2, label="m=15")  
 if m == 2:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]),  
 linewidth=2, label="m=16")  
 if m == 3:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]),  
 linewidth=2, label="m=110")  
  
 plt.xlabel("Wavelength [\u03BCm]")  
 plt.xlim(1.5, 1.6)  
 plt.ylabel("Transmission [dB]")  
 plt.legend(fontsize=14, loc=0)  
 plt.show()

仿真结果如下图：

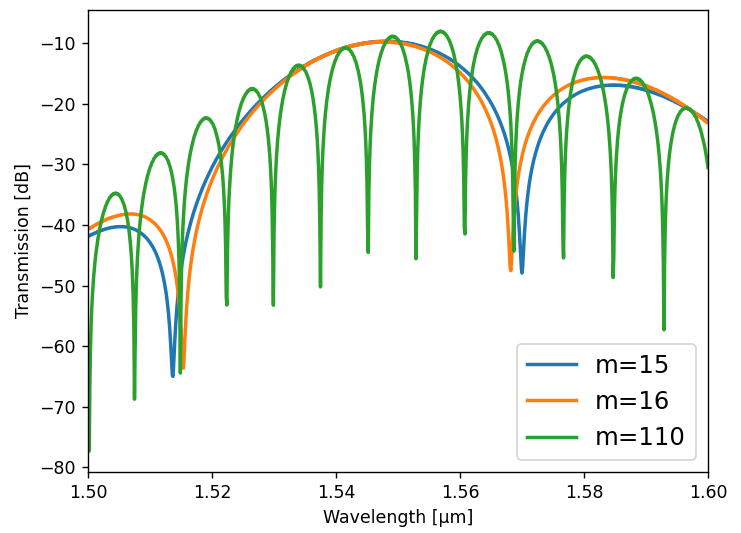


图 10 MZI测试结构仿真代码运行结果

## 3.3 Waveguide 传输损耗 DoE程序设计

在波导传输损耗的测量过程中，为了提高损耗的可测量性，通常需要较长的光路径。螺旋形波导（spiral）结构是实现这一目标的有效工具。与直线型波导相比，螺旋形波导通过其螺旋形路径有效地延长了光的传播距离，并且在有限的空间内提供了较长的光路径，兼顾了紧凑的尺寸设计。为了方便设计和实验，我们可以直接使用 IPKISS 库中现有的spiral组件：FixedLengthSpiralRounded，具体如下：

from CSiP180Al import all as pdk  
from picazzo3.wg.spirals import FixedLengthSpiralRounded  
from ipkiss3 import all as i3  
  
cell = FixedLengthSpiralRounded(total\_length=4000, n\_o\_loops=6, trace\_template=i3.TECH.PCELLS.WG.DEFAULT)  
layout = cell.Layout(  
 incoupling\_length=10.0,  
 bend\_radius=10.0,  
 spacing=4,  
 stub\_direction="H", *# either H or V* growth\_direction="V", *# either H or V*)  
layout.visualize(annotate=True)

该代码主要分为以下四个部分：

1. 导入库：导入PDK（CSiP180Al）、IPKISS以及器件库picazzo3中的FixedLengthSpiralRounded类；
2. 创建spiral实例：指定spiral总长度、螺旋圈数以及波导模板，创建一个具体的spiral实例；
3. 生成布局：通过定义弯曲半径、波导耦合区域的长度以及螺旋圈之间的间距等布局参数，为实例生成相应的布局；
4. 可视化布局：使用visualize(annotate=True) 方法显示生成的布局，并在布局中添加端口注释，方便理解和检查设计。

最终，该代码运行后得到的单个spiral结构布局如图 11所示。

图示

AI 生成的内容可能不正确。

图 11 IPKISS自带spiral 运行结果

基于IPKISS提供的内置spiral组件，可以设计并创建测试线路以提取波导的弯曲损耗。具体的程序实现和详细代码请参见附录4，该程序可允许自定义线路中spiral的数目、长度、长度步长以及波导类型、光栅耦合器的类型等结构参数，具体如下：

core\_width = i3.PositiveNumberProperty(default=0.45, doc=" waveguide core width ")  
bend\_radius = i3.PositiveNumberProperty(default=10.0, doc="the bend radius in the circuit")  
Spiral = i3.ChildCellProperty(doc="Spiral in circuit")  
gc\_distance\_x = i3.PositiveNumberProperty(default=500.0, doc="distance between the grating couplers ")  
gc\_distance\_y = i3.PositiveNumberProperty(default=50.0, doc="distance between the grating couplers ")  
space\_y = i3.PositiveNumberProperty(default=50, doc="gc space in y direction")  
space\_x = i3.PositiveNumberProperty(default=100, doc="space between gc and spiral ")  
space\_y\_spiral = i3.PositiveNumberProperty(default=150, doc="spiral space in y direction")  
space\_x\_spiral = i3.PositiveNumberProperty(default=1100, doc="spiral space in x direction")  
spiral\_length = i3.PositiveNumberProperty(default=10612, doc="length of sigle spiral")  
num\_Spiral\_y = i3.IntProperty(default=3, doc="number of spiral in y direction")  
num\_Spiral\_x = i3.IntProperty(default=1, doc="number of spiral in x direction")  
y\_offset = i3.PositiveNumberProperty(default=200)  
waveguide\_space = i3.PositiveNumberProperty(default=8)  
spiral\_length\_step = i3.PositiveNumberProperty(default=10000, doc="step length of waveguide under test")  
ra = i3.EulerRoundingAlgorithm(p=0.7)fgc = i3.ChildCellProperty()  
trace\_template = i3.TraceTemplateProperty()

通过自定义以上参数，并基于附录4中的测试线路代码，分别设计了长度为 0、1、2、3 cm 的spiral，进行waveguide DoE。运行代码后，得到的线路布局如图 12所示。

图示

AI 生成的内容可能不正确。

图 12 波导传输损耗测试程序运行结果

最后，还可以通过如下代码来对图 12所示线路进行仿真，具体过程分为以下四个部分：

1. 定义波长范围: wavelength\_range = np.linspace(1.5, 1.60, 1001) 创建了一个包含1001个点的波长范围，从1.5微米到1.6微米；
2. 提取S矩阵:用CircuitModel().get\_smatrix(wavelengths=wavelength\_range)提取指定波长范围内的散射矩阵（S矩阵）；
3. 绘制传输特性图: 使用i3.SpectrumAnalyzer提取传输特性信息，并通过 plt.plot 绘制。

整体实现的代码如下，基于该代码生成的不同长度的spiral线路仿真结果如图 13所示。

Spiral\_c = GenericSpiral(fgc=pdk.GC\_TE\_1550(), trace\_template=pdk.SWG450\_CTE(), gc\_distance\_x=4800 / 6, gc\_distance\_y=300)  
wavelength\_range = np.linspace(1.5, 1.60, 1001)  
dl=[3,2,1,0]  
s\_matrix = Spiral\_c.CircuitModel().get\_smatrix(wavelengths=wavelength\_range)  
fig, ax = plt.subplots(2)  
for m in range(4):  
 analyzer = i3.SpectrumAnalyzer(  
 *# using the spectrum analyser on our s-matrix between the input and output ports* smatrix=s\_matrix, input\_port\_mode="in{}".format(m), output\_port\_modes=["out{}".format(m)], dB=True,  
 bandpass=True  
 )  
 insertion\_loss = analyzer.min\_insertion\_losses()["out{}".format(m)]  
 ax[0].plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]),  
 linewidth=2,  
 label="spiral length = {} cm ".format(dl[m]),  
 )  
 ax[1].plot(dl[m], insertion\_loss, marker="o", color="b")  
ax[0].set\_xlabel("Wavelength [\u03BCm]")  
ax[0].set\_xlim(1.52, 1.6)  
ax[0].set\_ylabel("Transmission [dB]")  
ax[0].legend(fontsize=14)  
ax[1].set\_xlabel("Spiral length [\u03BCm]")  
ax[1].set\_xlim(0, 3)  
ax[1].set\_ylabel("Insertion losses at 1550 nm [dB]")  
plt.show()

图表, 图示

AI 生成的内容可能不正确。

图 13 波导传输损耗仿真程序运行结果

## 3.4 Waveguide 弯曲损耗 DoE程序设计

波导的弯曲损耗可以通过设计固定数量和弯曲半径的波导来进行测试。为了实现这一目标，首先需要创建具有特定弯曲半径的波导。该设计程序如下，其结构与2.1节所述相一致，继承自 i3.Circuit，并包含以下四个主要部分：

1. 属性声明：设置线路的属性参数，如实例名称、定义波导模板、弯曲半径等；
2. 器件实例化：定义线路中的器件实例。例如该线路是由四个弯曲波导bend1、bend2、bend3、bend4组成的，每段弯曲波导都是arc的实例；
3. 布局连接：定义器件在线路中的位置和连接。该线路中，使用i3.Place("bend1:in", (0, 0), 180)将 bend1 的输入端放置在 (0, 0) 位置，并旋转 180 度；使用i3.Join("bend2:out", "bend1:out")将 bend2 的输出端与 bend1 的输出端连接；同理，通过i3.Join依次连接剩下的弯曲波导，形成一个连续的测试线路；
4. 定义端口：将bend1:in 端口作为整体线路的输入端口 "in"；将bend4:out端口作为整体线路的输出端口 "out"。

基于上述代码，设计了四个弯曲半径为10 μm的弯曲波导级联，其运行结果如图 14所示。

class s\_circle(i3.Circuit):  
 \_name\_prefix = "s\_circle"  
 bend\_radius = i3.PositiveNumberProperty(default=10.0, doc="the bend radius in the circuit")  
 trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")  
 arc = i3.ChildCellProperty(doc="arc in circuit")  
  
 def \_default\_trace\_template(self):  
 tt = pdk.StripWaveguideTemplate()  
 tt.Layout(core\_width=0.45)  
 return tt  
  
 def \_default\_arc(self):  
 arc = pdk.RoundedWaveguide(trace\_template=self.trace\_template)  
 arc.Layout(shape=i3.ShapeArc(radius=self.bend\_radius,  
 start\_angle=0,  
 end\_angle=90,  
 start\_face\_angle=90,  
 end\_face\_angle=180), )  
 return arc  
  
  
 def \_default\_insts(self):  
 insts = {  
 "bend1": self.arc,  
 "bend2": self.arc,  
 "bend3": self.arc,  
 "bend4": self.arc,  
 }  
 return insts  
  
 def \_default\_specs(self):  
 specs = []  
 specs += [  
 i3.Place("bend1:in", (0, 0),180),  
 i3.Join("bend2:out", "bend1:out"),  
 i3.Join("bend3:out", "bend2:in"),  
 i3.Join("bend4:in", "bend3:in"),  
 ]  
 return specs  
  
 def \_default\_exposed\_ports(self): *# port definition* exposed\_ports = {  
 "bend1:in": "in",  
 "bend4:out": "out",  
 }  
 return exposed\_ports

图表

AI 生成的内容可能不正确。

图 14 弯曲波导生成代码运行结果

基于单个弯曲波导结构设计，可以将多个波导按照横向（x方向）和纵向（y方向）排列，形成阵列。通过这种弯曲波导阵列，可以有效累积弯曲损耗，从而更精确地提取波导的弯曲损耗数据。

弯曲波导阵列的测试线路代码结构与单个弯曲波导相似，依然继承自i3.Circuit。通过循环操作，将多个弯曲波导放置在指定位置并进行级联。具体实现代码如下：

class s\_circle\_X\_Y(i3.Circuit):  
 \_name\_prefix = "s\_circle\_X\_Y"  
 radius = i3.PositiveNumberProperty(default=10.0, doc="the bend radius in the circuit")  
 trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")  
 wg1 = i3.ChildCellProperty(doc="waveguide in circuit")  
 wg2 = i3.ChildCellProperty(doc="waveguide in circuit")  
 wg3 = i3.ChildCellProperty(doc="waveguide in circuit")  
 num\_bend\_x = i3.IntProperty(default=20, doc="number of s\_cirle in x direction")  
 num\_bend\_y = i3.IntProperty(default=5, doc="number of s\_cirle in y direction")  
  
 def \_default\_trace\_template(self):  
 tt = pdk.StripWaveguideTemplate()  
 tt.Layout(core\_width=0.45)  
 return tt  
  
 def \_default\_wg1(self):  
 return s\_circle\_X(num\_bend\_x=self.num\_bend\_x, bend\_radius=self.radius, trace\_template=self.trace\_template)  
  
 def \_default\_wg2(self):  
 bw = i3.RoundedWaveguide(trace\_template=self.trace\_template)  
 bw.Layout(shape=i3.ShapeArc(radius=self.radius, start\_angle=270, end\_angle=90, start\_face\_angle=0, end\_face\_angle=180))  
 return bw  
 def \_default\_wg3(self):  
 bw = i3.RoundedWaveguide(trace\_template=self.trace\_template)  
 bw.Layout(shape=i3.ShapeArc(radius=self.radius, start\_angle=90, end\_angle=270, start\_face\_angle=180, end\_face\_angle=0))  
 return bw  
  
 def \_default\_insts(self):  
 insts = {}  
 for m in range(self.num\_bend\_y):  
 insts["bend\_{}".format(m)] = self.wg1  
 for m in range(self.num\_bend\_y // 2):  
 insts["bend\_in\_{}".format(m)] = self.wg3  
 insts["bend\_out\_{}".format(m)] = self.wg2  
 return insts  
  
 def \_default\_specs(self):  
 specs = []  
 for m in range(self.num\_bend\_y):  
 specs += [i3.Place("bend\_{}:in".format(m), (0, m \* self.radius \* 2))]  
 for m in range(self.num\_bend\_y // 2):  
 specs += [i3.Join("bend\_in\_{}:out".format(m), "bend\_{}:in".format(2 \* m + 1))]  
 specs += [i3.Join("bend\_out\_{}:in".format(m), "bend\_{}:out".format(2 \* m))]  
 return specs  
 def \_default\_exposed\_ports(self): *# port definition* exposed\_ports = {  
 "bend\_0:in": "in",  
 "bend\_{}:out".format(self.num\_bend\_y - 1): "out",  
 }  
 return exposed\_ports

在该代码中，弯曲波导阵列的横向（x方向）和纵向（y方向）波导数量都是可调的。通过修改参数 num\_bend\_x 和 num\_bend\_y，可以控制阵列的布局。例如，将 num\_bend\_x 设置为 20、num\_bend\_y 设置为 5，可以生成一个 20 列、5 行的弯曲波导阵列。此阵列布局如图 15所示

图标

AI 生成的内容可能不正确。

图 15 弯曲波导阵列生成程序运行结果

基于上述代码，分别设计了弯曲波导总个数为0、1560、2040和2520的弯曲波导阵列，并给每个弯曲波导阵列添加光栅耦合器，可以得到最终的测试线路，具体代码见附录5。最终，得到波导弯曲损耗线路布局如图 16所示。

背景图案

AI 生成的内容可能不正确。

图 16 波导弯曲损耗线路设计程序运行结果

最后，还可以通过如下代码来对图 16所示线路进行仿真，具体过程分为以下四个部分：

1. 定义波长范围: wavelength\_range = np.linspace(1.5, 1.60, 1001) 创建了一个包含1001个点的波长范围，从1.5微米到1.6微米；
2. 提取S矩阵:用CircuitModel().get\_smatrix(wavelengths=wavelength\_range)提取指定波长范围内的散射矩阵（S矩阵）；
3. 绘制传输特性图: 使用plt.plot 绘制弯曲波导阵列的传输特性。

整体实现的代码如下，基于该代码生成的不同数目的弯曲波导阵列线路仿真结果如图 17所示。

doe = s\_circle\_X\_Y\_routed(gc\_space\_x=4800 / 6,num\_bend\_y=13, num\_bend\_x=30, radius=5, bend\_radius=10, gc=pdk.GC\_TE\_1550(), num\_gc=3,  
 trace\_template=pdk.StripWaveguideTemplate().Layout(core\_width=0.45).cell)  
doe.Layout().visualize(annotate=True)  
wavelength\_range = np.linspace(1.50, 1.60, 1001)  
s\_matrix = doe.CircuitModel().get\_smatrix(wavelengths=wavelength\_range)  
plt.figure(1)  
for m in range(4):  
 if m==0:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]), linewidth=2,label="bend number: 0" )  
 if m==1:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]), linewidth=2,label="bend number: 1560" )  
 if m==2:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]), linewidth=2,label="bend number: 2040" )  
 if m==3:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]), linewidth=2,label="bend number: 2520" )  
plt.xlabel("Wavelength [\u03BCm]")  
plt.xlim(1.5, 1.6)  
plt.ylabel("Transmission [dB]")  
plt.legend()  
plt.show()

图表, 图示

AI 生成的内容可能不正确。

图 17 波导弯曲损耗测试线路仿真结果

## 3.5 Waveguide Crossing DoE程序设计

交叉波导（Waveguide Crossing）的主要模型参数是插损。由于单个交叉波导的插损通常较小（约0.1 dB），为了实现有效测量，通常需要将多个交叉波导级联。以下是级联结构的设计程序，继承自 i3.Circuit，并包含以下四个主要部分：

1. 属性声明：设置线路的属性参数，如实例名称、定义波导模板、弯曲半径、 Crossing的数目以及每个crossing之间的间隔等参数；
2. 器件实例化：定义线路需要的crossing和ABSORBER；
3. 布局连接：给每个交叉点放置一个crossing，并在crossing上下两个悬空端口放置 ABSORBER来减少端口的反射。将每个ABSORBER连接到相应的crossing端口，并将相邻的crossing连接在一起；
4. 定义端口：将第一个crossing的左端口作为整体线路的输入端口 "in"；将最后一个crossing的右端口作为整体线路的输出端口 "out"。

该代码运行结果如图 18所示。

class Crossing(i3.Circuit):  
 \_name\_prefix = "crossing"  
 bend\_radius = i3.PositiveNumberProperty(default=20.0, doc="the bend radius in the circuit")  
 num\_crossing = i3.IntProperty(default=10, doc="munber of crossing")  
 crossing\_space = i3.PositiveNumberProperty(default=15.0)  
 spacing\_y = i3.PositiveNumberProperty(default=127.0)  
 abs\_offset = i3.PositiveNumberProperty(default=5.0)  
 absorber = i3.ChildCellProperty()  
 crossing = i3.ChildCellProperty()  
  
 def \_default\_absorber(self):  
 return pdk.ABSORBER()  
 def \_default\_crossing(self):  
 return pdk.XP\_TE\_1550()  
 def \_default\_insts(self):  
 insts = {}  
 for m in range(self.num\_crossing):  
 insts["X\_{}".format(m)] = self.crossing  
 insts["abs\_north\_{}".format(m)] = self.absorber  
 insts["abs\_south\_{}".format(m)] = self.absorber  
 return insts  
 def \_default\_specs(self):  
 specs = []  
 for m in range(self.num\_crossing):  
 specs += [  
 i3.Place("X\_{}:east".format(m), (self.crossing\_space \* m+m\*m/2, 0)),  
 i3.PlaceRelative("abs\_north\_{}:in".format(m), "X\_{}:north".format(m), (0, self.abs\_offset), angle=-90),  
 i3.PlaceRelative("abs\_south\_{}:in".format(m), "X\_{}:south".format(m), (0, -self.abs\_offset), angle=90),  
 i3.ConnectManhattan("abs\_south\_{}:in".format(m), "X\_{}:south".format(m)),  
 i3.ConnectManhattan("abs\_north\_{}:in".format(m), "X\_{}:north".format(m)),  
 ]  
 for m in range(self.num\_crossing - 1):  
 specs += [  
 i3.ConnectManhattan("X\_{}:east".format(m), "X\_{}:west".format(m + 1)),  
 ]  
 return specs  
  
 def \_default\_exposed\_ports(self):  
 exposed\_ports = {}  
 exposed\_ports["X\_{}:west".format(0)] = "in"  
 exposed\_ports["X\_{}:east".format(self.num\_crossing - 1)] = "out"  
 return exposed\_ports

图表, 箱线图

AI 生成的内容可能不正确。

图 18 Crossing 阵列生成程序运行结果

基于上述代码，分别设计了crossing个数为0、10、20和30的Waveguide Crossing测试线路，并给每个测试线路两端添加光栅耦合器，具体代码见附录6。最终，得到Waveguide Crossing测试线路布局如图 19所示。

图片包含 图表

AI 生成的内容可能不正确。

图 19 Waveguide Crossing测试线路设计程序运行结果

最后，通过如下代码来对图 19所示线路进行仿真，具体过程分为以下四个部分：

1. 定义波长范围: wavelength\_range = np.linspace(1.5, 1.60, 1001) 创建了一个包含1001个点的波长范围，从1.5微米到1.6微米；
2. 提取S矩阵:用CircuitModel().get\_smatrix(wavelengths=wavelength\_range)提取指定波长范围内的散射矩阵（S矩阵）；
3. 绘制传输特性图: 使用plt.plot 绘制Waveguide Crossing测试线路的传输特性。

整体实现的代码如下，基于该代码生成的不同数目的Waveguide Crossing测试线路仿真结果如图 20所示。

x\_offset = 4800 / 6  
y\_offset = 300  
doe = Crossing\_C\_fgc(x\_offset=x\_offset, y\_offset=y\_offset)  
doe.Layout().visualize(annotate=True)  
wavelength\_range = np.linspace(1.50, 1.60, 1001)  
s\_matrix = doe.CircuitModel().get\_smatrix(wavelengths=wavelength\_range)  
plt.figure(1)  
for m in range(4):  
 if m==0:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]), linewidth=2,label="Crossing number:0" )  
 if m==1:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]), linewidth=2,label="Crossing number:10" )  
 if m==2:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]), linewidth=2,label="Crossing number:20" )  
 if m==3:  
 plt.plot(wavelength\_range, i3.signal\_power\_dB(s\_matrix["in{}".format(m), "out{}".format(m)]), linewidth=2,label="Crossing number:30" )  
  
plt.xlabel("Wavelength [\u03BCm]")  
plt.xlim(1.5, 1.6)  
plt.ylabel("Transmission [dB]")  
plt.legend()  
plt.show()

图表

AI 生成的内容可能不正确。

图 20 Waveguide Crossing测试线路设计程序仿真代码运行结果

## 3.6 Transition DoE程序设计

与Waveguide Crossing类似，单个Transition插损通常也很小（0.1 dB左右），因此Transition的插损测试线路中也需要级联多个来实现有效的测量，多个Transition级联的设计程序如下，其结构与2.1节所述相一致，继承自 i3.Circuit，并包含以下四个主要部分：

1. 属性声明：设置线路的属性参数，如实例名称、定义波导模板、弯曲半径等；
2. 器件实例化：定义线路中的器件实例。例如该线路是由四个弯曲波导bend1、bend2、bend3、bend4组成的，每段弯曲波导都是arc的实例；
3. 布局连接：定义器件在线路中的位置和连接。该线路中，使用i3.Place("bend1:in", (0, 0), 180)将 bend1 的输入端放置在 (0, 0) 位置，并旋转 180 度；使用i3.Join("bend2:out", "bend1:out")将 bend2 的输出端与 bend1 的输出端连接；同理，通过i3.Join依次连接剩下的弯曲波导，形成一个连续的测试线路；
4. 定义端口：将bend1:in 端口作为整体线路的输入端口 "in"；将bend4:out端口作为整体线路的输出端口 "out"。

基于上述代码，设计了四个弯曲半径为10 μm的弯曲波导级联，其运行结果如图 21所示。

class TransitionTest(i3.Circuit):  
 \_name\_prefix = "transition"  
 dut = i3.ChildCellProperty(doc="the end trace template tapered to")  
 wg = i3.ChildCellProperty(doc="the end trace template tapered to")  
 bend\_radius = i3.PositiveNumberProperty(default=10.0, doc="the bend radius in the circuit")  
 rib\_length = i3.PositiveNumberProperty(default=10.0, doc="length of rib waveguide between transitions")  
 rib\_length\_increment = i3.NonNegativeNumberProperty(  
 default=1.0,  
 doc="gradual length increase of the rib waveguide between transitions, to reduce coherent backreflection effects",  
 )  
 strip\_length = i3.PositiveNumberProperty(default=10.0, doc="length of strip waveguide between transitions")  
 strip\_length\_increment = i3.NonNegativeNumberProperty(  
 default=2.0,  
 doc="gradual length increase of the strip waveguide between transitions, to reduce coherent backreflection effects",  
 )  
 spacing\_y = i3.PositiveNumberProperty(default=150.0)  
 num\_transition = i3.IntProperty(default=35, doc="number of transitions")  
 num\_transition\_x = i3.IntProperty(default=4, doc="number of transitions")  
 transition = i3.ChildCellProperty(doc="the end trace template tapered to")  
 ra = i3.EulerRoundingAlgorithm(p=0.70)  
 trace\_template = i3.TraceTemplateProperty()  
 def \_default\_trace\_template(self):  
 return pdk.SWG450\_CTE()  
 def \_default\_transition(self):  
 return pdk.TR\_S\_R\_TE\_1550()  
 def \_default\_dut(self):  
 return TransitionUnit(length=self.strip\_length, transition=self.transition)  
 def \_default\_wg(self):  
 wg = i3.RoundedWaveguide(trace\_template=self.trace\_template).Layout(  
 shape=[(0, 0), (100, 0)]  
 )  
 return wg  
 def \_default\_insts(self):  
 insts = {}  
 for m in range(35):  
 if self.num\_transition == 20:  
 if m < self.num\_transition:  
 insts["TR\_{}".format(m)] = self.dut.modified\_copy(  
 length=self.rib\_length + self.rib\_length\_increment \* m)  
 else:  
 insts["TR\_{}".format(m)] = self.wg  
 else:  
 insts["TR\_{}".format(m)] = self.dut.modified\_copy(  
 length=self.rib\_length + self.rib\_length\_increment \* m)  
 return insts  
 def \_default\_specs(self): *# we place the ports in the same way as the tapers* specs = []  
 for m in range(35):  
 size = self.dut.modified\_copy(  
 length=self.rib\_length + self.rib\_length\_increment \* m).get\_default\_view(i3.LayoutView).size\_info()  
 length = size.east - size.west  
 if m % self.num\_transition\_x == 0:  
 specs += [i3.Place("TR\_{}:in".format(m), (0, -m // self.num\_transition\_x \* (34)))]  
 else:  
 specs += [i3.PlaceRelative("TR\_{}:in".format(m), "TR\_{}:in".format(m - 1),  
 (length + self.strip\_length + self.strip\_length\_increment \* m, 0)),  
 i3.ConnectManhattan("TR\_{}:in".format(m), "TR\_{}:out".format(m - 1)),]  
 for m in range(35 // (self.num\_transition\_x \* 2)):  
 specs += [i3.ConnectManhattan("TR\_{}:out".format(self.num\_transition\_x \* (2 \* m + 1) - 1),  
 "TR\_{}:out".format(self.num\_transition\_x \* (2 \* m + 2) - 1),  
 bend\_radius=self.bend\_radius,  
 start\_straight=0,  
 end\_straight=0,  
 control\_points=[i3.V(590)],  
 rounding\_algorithm=self.ra,  
 ),  
 i3.ConnectManhattan("TR\_{}:in".format(self.num\_transition\_x \* (2 \* m + 1)),  
 "TR\_{}:in".format(self.num\_transition\_x \* (2 \* m + 2)),  
 bend\_radius=self.bend\_radius,  
 rounding\_algorithm=self.ra, ),  
 ]  
 return specs  
 def \_default\_exposed\_ports(self): *# we place the ports in the same way as the tapers* exposed\_ports = {  
 "TR\_0:in": "in",  
 "TR\_{}:out".format(35 - 1): "out",  
 }  
 return exposed\_ports

该代码运行结果如下：

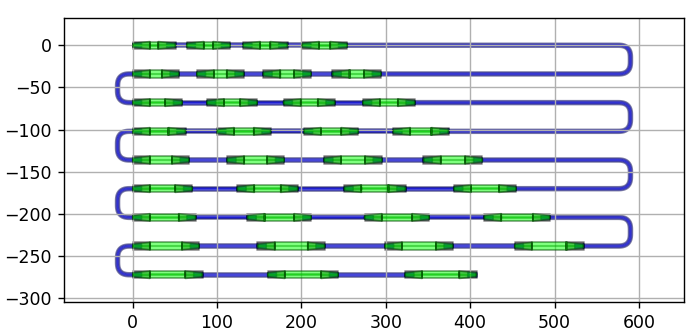


图 21 Transition测试线路设计程序仿真代码运行结果

为了满足自动化测试条件，该线路采用“折叠”的设计，在代码中中是通过遍历每个transition，并通过判断其数目是否为num\_transition\_x（代码中设置的值为4）整数倍来实现。

for m in range(35):  
 size = self.dut.modified\_copy(  
 length=self.rib\_length + self.rib\_length\_increment \* m).get\_default\_view(i3.LayoutView).size\_info()  
 length = size.east - size.west  
 if m % self.num\_transition\_x == 0:  
 specs += [i3.Place("TR\_{}:in".format(m), (0, -m // self.num\_transition\_x \* (34)))]  
 else:  
 specs += [i3.PlaceRelative("TR\_{}:in".format(m), "TR\_{}:in".format(m - 1),  
 (length + self.strip\_length + self.strip\_length\_increment \* m, 0)),  
 i3.ConnectManhattan("TR\_{}:in".format(m), "TR\_{}:out".format(m - 1)),  
 ]

同时为了方便调整Transition的数目、以及减小transtion的反射，设置了transition之间连接的条波导于脊波导的长度，以及长度增加步长：

rib\_length = i3.PositiveNumberProperty(default=10.0, doc="length of rib waveguide between transitions")  
rib\_length\_increment = i3.NonNegativeNumberProperty(  
 default=1.0,  
 doc="gradual length increase of the rib waveguide between transitions, to reduce coherent backreflection effects",  
)  
strip\_length = i3.PositiveNumberProperty(default=10.0, doc="length of strip waveguide between transitions")  
strip\_length\_increment = i3.NonNegativeNumberProperty(  
 default=2.0,  
 doc="gradual length increase of the strip waveguide between transitions, to reduce coherent backreflection effects",  
)  
spacing\_y = i3.PositiveNumberProperty(default=150.0)  
num\_transition = i3.IntProperty(default=35, doc="number of transitions")

通过对modified\_copy方法的参数进行不同的赋值，你可以生成多个具有不同过渡（transition）结构的DUT实例。这些实例在rib\_length、rib\_length\_increment、strip\_length、strip\_length\_increment和num\_transition等参数上有所不同，从而可以模拟和测试各种过渡结构的变化对器件性能的影响。

def \_default\_insts(self):  
 insts = {  
 "dut\_0": self.dut.modified\_copy(rib\_length=40,  
 rib\_length\_increment=0,  
 strip\_length=40,  
 strip\_length\_increment=0,  
 num\_transition=35),  
 "dut\_1": self.dut.modified\_copy(rib\_length=40,  
 rib\_length\_increment=0,  
 strip\_length=40,  
 strip\_length\_increment=0,  
 num\_transition=20),  
 "dut\_2": self.dut.modified\_copy(rib\_length=10,  
 rib\_length\_increment=2,  
 strip\_length=10,  
 strip\_length\_increment=1,  
 num\_transition=35),  
 "dut\_3": self.dut.modified\_copy(rib\_length=10,  
 rib\_length\_increment=2,  
 strip\_length=10,  
 strip\_length\_increment=1,  
 num\_transition=20),  
 }  
 return insts

代码中，\_default\_insts 方法返回一个字典，其中包含四个不同的DUT实例：dut\_0 和 dut\_1 保持固定的rib和strip长度，但使用不同的transition数量（35和20）。dut\_2 和 dut\_3 则在rib和strip长度上有一个递增值，同时也使用不同的transition数量（35和20）。

这种方式可以很好地用于测试不同过渡结构的级联组合，以评估它们在不同数量和配置下的行为表现。就可以得到用于测试不同数量transition级联结构。

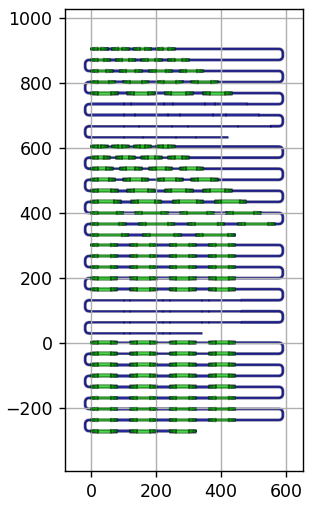


图 22 不同数量transition级联结构

## 3.7 1x2 MMI DoE程序设计

1x2 MMI DOE设计包括1x2 MMI 功率不平衡DOE设计，1x2 MMI 损耗DOE设计以及1x2 MMI 反射率DOE设计。

首先是1x2 MMI 功率不平衡DOE设计，1x2 MMI 功率不平衡DOE采用树状分光器的设计，其设计代码如下：

class MMI1x2Fgc(i3.Circuit):  
 mmi\_c = i3.ChildCellProperty()  
 fgc = i3.ChildCellProperty()  
 # doe\_text = Text(text="MMI\_DOE\_1x2\_C")  
  
 levels = i3.IntProperty(default=6, doc="Number of fgcs")  
 spacing\_x = i3.PositiveNumberProperty(default=100.0)  
 spacing\_y = i3.PositiveNumberProperty(default=127.0)  
 bend\_radius = i3.PositiveNumberProperty(default=25.0, doc="the bend radius in the circuit")  
 text = i3.StringProperty(default="mmi")  
  
 trace\_template = i3.TraceTemplateProperty()  
 mmi = i3.ChildCellProperty()  
  
 def \_default\_mmi(self):  
 return pdk.Fixed\_M1X2\_TE\_1550()  
  
 def \_default\_trace\_template(self):  
 return pdk.SWG450()  
  
 def \_default\_mmi\_c(self):  
 return MMI1x2(mmi=self.mmi, trace\_template=self.trace\_template)  
  
 def \_default\_fgc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_insts(self):  
 insts = dict()  
 insts["mmi\_c"] = self.mmi\_c  
 n\_fgcs = self.levels  
 for m in range(n\_fgcs + 2):  
 insts["fgc\_{}".format(m)] = self.fgc  
  
 insts["fgc\_up"] = self.fgc  
 insts["fgc\_down"] = self.fgc  
 insts["text"] = self.text,  
 return insts  
  
 def \_default\_specs(self):  
 x = self.spacing\_x  
 y\_up = self.spacing\_y  
 y\_down = self.spacing\_y  
  
 specs = [i3.Place("mmi\_c:in", (0, 0)),  
 i3.PlaceRelative("fgc\_0:wg", "mmi\_c:in", (-100, 0), 0),  
 i3.PlaceRelative("fgc\_1:wg", "mmi\_c:out1", (300, y\_up / 2), 180),  
 i3.PlaceRelative("fgc\_up:wg", "fgc\_1:wg", (0, y\_up), 180),  
 # i3.PlaceRelative("doe\_text:loc", "mmi\_c:in", (30, 60), 0),  
 i3.ConnectManhattan("fgc\_1:wg", "mmi\_c:out1", bend\_radius=self.bend\_radius, ),  
 i3.ConnectManhattan("mmi\_c:in", "fgc\_0:wg", bend\_radius=self.bend\_radius, ),  
 ]  
  
 n\_fgcs = self.levels + 2  
 for m in range(2, n\_fgcs):  
 specs.append(i3.PlaceRelative("fgc\_{}:wg".format(m), "fgc\_{}:wg".format(m - 1), (0, -self.spacing\_y), 180))  
 specs.append(  
 i3.ConnectManhattan(  
 "fgc\_{}:wg".format(m), "mmi\_c:out{}".format(m),  
 bend\_radius=self.bend\_radius,  
 )  
 )  
 specs.append(i3.PlaceRelative("fgc\_down:wg", "fgc\_{}:wg".format(n\_fgcs - 1), (0, -self.spacing\_y), 180))  
 specs.append(  
 i3.ConnectManhattan(  
 "fgc\_up:wg", "fgc\_down:wg",  
 bend\_radius=self.bend\_radius,  
 control\_points=[i3.V(i3.START - 3 \* self.bend\_radius),  
 i3.H(i3.START + 3 \* self.bend\_radius),  
 i3.V(i3.START + 3 \* self.bend\_radius),  
 i3.H(i3.END - 3 \* self.bend\_radius),  
 i3.V(i3.END - 3 \* self.bend\_radius),  
 ]  
 )  
 )  
 return specs

该电路主要由以下核心组件构成：

* 1x2 多模干涉器 (MMI)：实现光信号的 1 分 2 功能
* 多个光纤光栅耦合器 (FGC)：实现光波导与光纤之间的高效耦合
* 光波导连接：采用曼哈顿布线方式连接各组件

电路的主要功能是将输入光信号通过 MMI 分路后，由多个 FGC 耦合到对应的输出光纤中，同时支持额外的上下行光路连接。

1. 参数设计与配置

线路通过一系列参数化设计实现灵活配置：

levels = i3.IntProperty(default=6, doc="Number of fgcs")

spacing\_x = i3.PositiveNumberProperty(default=100.0)

spacing\_y = i3.PositiveNumberProperty(default=127.0)

bend\_radius = i3.PositiveNumberProperty(default=25.0, doc="the bend radius in the circuit")

text = i3.StringProperty(default="mmi")

* levels：控制输出 FGC 的数量，默认值为 6
* spacing\_x：水平方向的间距参数，影响组件布局
* spacing\_y：垂直方向的间距参数，影响组件布局
* bend\_radius：光波导弯曲半径，影响光信号传输质量
* text：电路标识文本

**2. 组件实例化**

线路通过默认方法实现各组件的实例化：

def \_default\_mmi(self):

return pdk.Fixed\_M1X2\_TE\_1550()

def \_default\_trace\_template(self):

return pdk.SWG450()

def \_default\_mmi\_c(self):

return MMI1x2(mmi=self.mmi, trace\_template=self.trace\_template)

def \_default\_fgc(self):

return pdk.Fixed\_GC\_TE\_1550()

* MMI 采用 pdk 库中的 Fixed\_M1X2\_TE\_1550 组件，适用于 1550nm 波长的 TE 模式
* 光波导模板采用 pdk 库中的 SWG450
* FGC 采用 pdk 库中的 Fixed\_GC\_TE\_1550 组件，适用于 1550nm 波长的 TE 模式

**3. 电路布局与连接**

线路采用层次化布局策略：

def \_default\_insts(self):

insts = dict()

insts["mmi\_c"] = self.mmi\_c

n\_fgcs = self.levels

for m in range(n\_fgcs + 2):

insts["fgc\_{}".format(m)] = self.fgc

insts["fgc\_up"] = self.fgc

insts["fgc\_down"] = self.fgc

insts["text"] = self.text,

return insts

* 中心位置放置 MMI 组件
* 左侧放置输入 FGC (fgc\_0)
* 右侧按垂直方向排列多个输出 FGC (fgc\_1 至 fgc\_n)
* 顶部和底部分别放置上行和下行 FGC (fgc\_up 和 fgc\_down)

线路采用曼哈顿布线策略实现光波导连接：

def \_default\_specs(self):

x = self.spacing\_x

y\_up = self.spacing\_y

y\_down = self.spacing\_y

specs = [i3.Place("mmi\_c:in", (0, 0)),

i3.PlaceRelative("fgc\_0:wg", "mmi\_c:in", (-100, 0), 0),

i3.PlaceRelative("fgc\_1:wg", "mmi\_c:out1", (300, y\_up / 2), 180),

i3.PlaceRelative("fgc\_up:wg", "fgc\_1:wg", (0, y\_up), 180),

i3.ConnectManhattan("fgc\_1:wg", "mmi\_c:out1", bend\_radius=self.bend\_radius, ),

i3.ConnectManhattan("mmi\_c:in", "fgc\_0:wg", bend\_radius=self.bend\_radius, ),

]

n\_fgcs = self.levels + 2

for m in range(2, n\_fgcs):

specs.append(i3.PlaceRelative("fgc\_{}:wg".format(m), "fgc\_{}:wg".format(m - 1), (0, -self.spacing\_y), 180))

specs.append(

i3.ConnectManhattan(

"fgc\_{}:wg".format(m), "mmi\_c:out{}".format(m),

bend\_radius=self.bend\_radius,

)

)

specs.append(i3.PlaceRelative("fgc\_down:wg", "fgc\_{}:wg".format(n\_fgcs - 1), (0, -self.spacing\_y), 180))

specs.append(

i3.ConnectManhattan(

"fgc\_up:wg", "fgc\_down:wg",

bend\_radius=self.bend\_radius,

control\_points=[i3.V(i3.START - 3 \* self.bend\_radius),

i3.H(i3.START + 3 \* self.bend\_radius),

i3.V(i3.START + 3 \* self.bend\_radius),

i3.H(i3.END - 3 \* self.bend\_radius),

i3.V(i3.END - 3 \* self.bend\_radius),

]

)

)

return specs

该代码运行结果如图3-7-1所示：

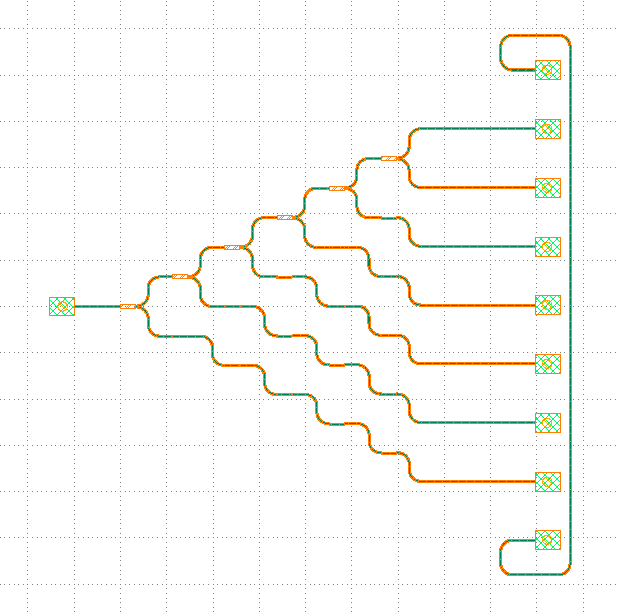


图 3-7-1 1x2 MMI 功率不平衡DOE设计

接下来是，1x2 MMI 损耗DOE设计，1x2 MMI 损耗DOE设计与transition的设计类似采用多段级联的设计,代码设计与transition也一样，就替换了基本单元。

class TransitionUnit(i3.Circuit):  
   
 \_name\_prefix = "transition2"  
 transition = i3.ChildCellProperty(doc="the end trace template tapered to")  
 length = i3.PositiveNumberProperty(default=40.0, doc="length of strip waveguide between mmi1x2loss")  
  
 def \_default\_transition(self):  
 return pdk.Fixed\_M1X2\_TE\_1550()  
  
 def \_default\_insts(self):  
 insts = {"mmi1": self.transition,  
 "mmi2": self.transition,  
 }  
 return insts  
  
 def \_default\_specs(self):  
 specs = []  
  
 specs += [  
 i3.Place("mmi1:in\_1", (0, 0)),  
 i3.Place("mmi2:out\_1", (self.length, 0), relative\_to="mmi1:out\_1"),  
 i3.FlipH("mmi2"),  
 i3.ConnectManhattan("mmi1:out\_1", "mmi2:out\_1"),  
 i3.ConnectManhattan("mmi1:out\_2", "mmi2:out\_2"),  
 ]  
 return specs  
  
 def \_default\_exposed\_ports(self):  
 exposed\_ports = {  
 "mmi1:in\_1": "in",  
 "mmi2:in\_1": "out",  
 }  
 return exposed\_ports

该线路主要由以下核心组件构成：

* 两个背对背放置的 1x2 多模干涉器 (MMI)：实现光波导模式转换
* 直波导连接：连接两个 MMI，提供指定长度的传输路径

线路通过一系列参数化设计实现灵活配置：

\_name\_prefix = "transition2"

transition = i3.ChildCellProperty(doc="the end trace template tapered to")

length = i3.PositiveNumberProperty(default=40.0, doc="length of strip waveguide between mmi1x2loss")

* transition：指定过渡组件，默认为 pdk.Fixed\_M1X2\_TE\_1550 ()
* length：两个 MMI 之间直波导的长度，影响光信号传输特性

电路通过默认方法实现各组件的实例化：

def \_default\_transition(self):

return pdk.Fixed\_M1X2\_TE\_1550()

def \_default\_insts(self):

insts = {"mmi1": self.transition,

"mmi2": self.transition,

}

return insts

* 采用两个相同的 MMI 组件 (mmi1 和 mmi2)，背对背放置
* 默认使用 pdk 库中的 Fixed\_M1X2\_TE\_1550 组件，适用于 1550nm 波长的 TE 模式

**电路布局与连接**

def \_default\_specs(self):

specs = []

specs += [

i3.Place("mmi1:in\_1", (0, 0)),

i3.Place("mmi2:out\_1", (self.length, 0), relative\_to="mmi1:out\_1"),

i3.FlipH("mmi2"),

i3.ConnectManhattan("mmi1:out\_1", "mmi2:out\_1"),

i3.ConnectManhattan("mmi1:out\_2", "mmi2:out\_2"),

]

return specs

* mmi1 放置在原点 (0,0) 位置
* mmi2 相对于 mmi1 的 out\_1 端口，沿 x 轴正方向偏移 length 距离放置
* mmi2 进行水平翻转 (FlipH)，实现背对背放置
* 两个 MMI 的 out\_1 端口和 out\_2 端口分别通过曼哈顿布线方式连接

线路定义了输入输出端口：

def \_default\_exposed\_ports(self):

exposed\_ports = {

"mmi1:in\_1": "in",

"mmi2:in\_1": "out",

}

return exposed\_ports

* 将 mmi1 的 in\_1 端口暴露为电路的输入端口 (in)
* 将 mmi2 的 in\_1 端口暴露为电路的输出端口 (out)

代码运行结果如图3-7-2



图3-7-2 1x2 MMI 损耗DOE设计基本单元

与transition折叠的代码类似，替换1x2 MMI 损耗DOE设计基本单元后，就可以得到1x2 MMI 损耗DOE：

图示

AI 生成的内容可能不正确。

图3-7-3 1x2 MMI 损耗DOE设计

## 3.8 2x2 MMI DoE程序设计

2x2 MMI DoE程序包括2x2 MMI 损耗DoE程序以及2x2 MMI 功率不平衡DoE程序。

2x2 MMI 损耗的测试和1x2 MMI 损耗的测试类似，采用多组级联的方式来测其损耗，其DOE的设计同样包括基本单元的设计和弯曲级联结构DOE设计。

2x2 MMI的损耗测试基本单元如图3-8-2：

图示

AI 生成的内容可能不正确。

图3-8-2 2x2 MMI 损耗DOE设计基本单元

其实现代码如下：

class TransitionUnit(i3.Circuit):  
  
 \_name\_prefix = "transition2"  
 transition = i3.ChildCellProperty(doc="the end trace template tapered to")  
 gc = i3.ChildCellProperty(doc="the end trace template tapered to")  
 length = i3.PositiveNumberProperty(default=120.0, doc="length of strip waveguide between mmi1x2loss")  
  
 def \_default\_transition(self):  
 return pdk.Fixed\_M2X2\_TE\_1550()  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_insts(self):  
 insts = {"mmi1": self.transition,  
 "mmi2": self.transition,  
 "gc1": self.gc,  
 "gc2": self.gc,  
 }  
 return insts  
  
 def \_default\_specs(self):  
 specs = []  
  
 specs += [  
 i3.Place("mmi1:in\_1", (0, 0)),  
 i3.Place("mmi2:out\_1", (self.length, 0), relative\_to="mmi1:out\_1"),  
 i3.Place("gc1:wg", (0, -30), relative\_to="mmi1:in\_2", angle=180),  
 i3.Place("gc2:wg", (0, 30), relative\_to="mmi2:in\_1", angle=0),  
 i3.FlipH("mmi2"),  
 i3.ConnectManhattan("gc1:wg", "mmi1:in\_2", bend\_radius=10),  
 i3.ConnectManhattan("gc2:wg", "mmi2:in\_1", bend\_radius=10),  
 i3.ConnectManhattan("mmi1:out\_1", "mmi2:out\_1"),  
 i3.ConnectManhattan("mmi1:out\_2", "mmi2:out\_2"),  
 ]  
 return specs  
  
 def \_default\_exposed\_ports(self):  
 exposed\_ports = {  
 "mmi1:in\_1": "in",  
 "mmi2:in\_2": "out",  
 }  
 return exposed\_ports

1. 属性定义

transition = i3.ChildCellProperty(doc="the end trace template tapered to")

gc = i3.ChildCellProperty(doc="the end trace template tapered to")

length = i3.PositiveNumberProperty(default=120.0, doc="length of strip waveguide between mmi1x2loss")

* **transition**：MMI2x2 器件实例（多模干涉仪），实现光的分束 / 合束。
* **gc**：光栅耦合器（Grating Coupler），用于芯片与光纤的光接口。
* **length**：两个 MMI 之间的直波导长度（单位：μm），影响光程差和干涉效应。

2. 默认组件实现

def \_default\_transition(self):

return pdk.Fixed\_M2X2\_TE\_1550()

def \_default\_gc(self):

return pdk.Fixed\_GC\_TE\_1550()

**PDK 依赖**：使用工艺设计套件（PDK）中的标准组件：

Fixed\_M2X2\_TE\_1550：针对 1550nm 波长优化的 TE 模式 MMI2x2。

Fixed\_GC\_TE\_1550：针对 1550nm 波长优化的 TE 模式光栅耦合器。

3. 实例化与布局设计

def \_default\_insts(self):

insts = {"mmi1": self.transition, "mmi2": self.transition,

"gc1": self.gc, "gc2": self.gc}

return insts

组件布局：

* mmi1和mmi2：两个背对背的 MMI2x2 器件。
* gc1和gc2：输入 / 输出端的光栅耦合器。

4. 位置与连接规范

def \_default\_specs(self):

specs = [

i3.Place("mmi1:in\_1", (0, 0)),

i3.Place("mmi2:out\_1", (self.length, 0), relative\_to="mmi1:out\_1"),

i3.FlipH("mmi2"), # 水平翻转MMI2，形成背对背结构

i3.Place("gc1:wg", (0, -30), relative\_to="mmi1:in\_2", angle=180),

i3.Place("gc2:wg", (0, 30), relative\_to="mmi2:in\_1", angle=0),

# 连接路径（曼哈顿走线，避免锐角弯曲）

i3.ConnectManhattan("gc1:wg", "mmi1:in\_2", bend\_radius=10),

i3.ConnectManhattan("gc2:wg", "mmi2:in\_1", bend\_radius=10),

i3.ConnectManhattan("mmi1:out\_1", "mmi2:out\_1"),

i3.ConnectManhattan("mmi1:out\_2", "mmi2:out\_2"),

]

return specs

光学路径设计：

**MMI2x2 背对背连接**：mmi1的输出连接到mmi2的输入，形成 "strip→rib→strip→rib" 的模式转换路径。

**光栅耦合器位置**：gc1和gc2分别位于上下两侧，通过 180° 旋转实现光的垂直输入 / 输出。

**弯曲半径控制**：所有波导弯曲半径设为 10μm，避免因过小弯曲导致的辐射损耗。

5. 端口定义

def \_default\_exposed\_ports(self):

exposed\_ports = {

"mmi1:in\_1": "in",

"mmi2:in\_2": "out",

}

return exposed\_ports

输入 / 输出端口：

* in：光从mmi1的输入端口进入。
* out：光从mmi2的输入端口（实际为背对背后的输出）输出。

至于，2x2MMI弯曲级联结构用于损耗测试的DOE，其程序设计与1x2MMI类似，代码中只需要将1x2 MMI替换为2x2 MMI即可：

def \_default\_transition(self):  
 return pdk.Fixed\_M2X2\_TE\_1550()  
  
def \_default\_dut(self):  
 return TransitionUnit(length=self.strip\_length, transition=self.transition)

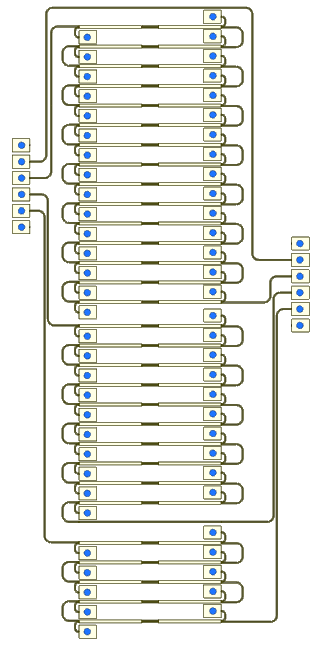


图3-8-3 2x2 MMI 损耗DOE设计

## 3.9 Ring Resonator DoE程序设计

Ring Resonator DOE设计主要用于提取Ring Resonator损耗、FSR等参数。其DOE程序的第一步就是设计微环的基本结构：

class RingResonators(i3.Circuit):  
 ring = i3.ChildCellProperty()  
 ab = i3.ChildCellProperty(default=pdk.Fixed\_GC\_TE\_1550())  
 ring\_radius = i3.PositiveNumberProperty(default=20, doc="The bend radius of the ring")  
 spacing = i3.PositiveNumberProperty(default=0.2, doc="The spacing between the ring and the bus waveguides")  
 trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")  
  
 def \_default\_trace\_template(self):  
 tt = pdk.StripWaveguideTemplate()  
 tt.Layout(core\_width=0.45)  
 return tt  
  
 def \_default\_ring(self):  
 return RingResonator(trace\_template=self.trace\_template).Layout(ring\_radius=self.ring\_radius,  
 spacing=self.spacing).cell  
  
 def \_default\_insts(self):  
 insts = {  
 "ring": self.ring,  
 "ab1": self.ab,  
 "ab2": self.ab,  
 }  
 return insts  
  
 def \_default\_specs(self):  
 specs = [  
 i3.Place("ring", (0, 0)),  
 i3.Place("ab1:wg", (-15, 15), relative\_to="ring:out1", angle=-90),  
 i3.Place("ab2:wg", (15, 15), relative\_to="ring:in1", angle=-90),  
 # i3.FlipH("ab2"),  
 i3.ConnectManhattan("ab1:wg", "ring:out1"),  
 i3.ConnectManhattan("ab2:wg", "ring:in1"),  
 ]  
  
 return specs  
  
 def \_default\_exposed\_ports(self):  
 exposed\_ports = {  
 "ring:in": "in",  
 "ring:out": "out",  
 }  
 return exposed\_ports

**该设计代码主要包括：**

**类定义与属性声明**

class RingResonators(i3.Circuit):

ring = i3.ChildCellProperty()

ab = i3.ChildCellProperty(default=pdk.Fixed\_GC\_TE\_1550())

ring\_radius = i3.PositiveNumberProperty(default=20, doc="The bend radius of the ring")

spacing = i3.PositiveNumberProperty(default=0.2, doc="The spacing between the ring and the bus waveguides")

trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")

这部分定义了一个名为RingResonators的电路类，继承自i3.Circuit。类中定义了几个重要属性：

* ring：表示微环谐振器的子单元
* ab：默认使用 PDK 提供的 1550nm TE 模式固定光栅耦合器 (Fixed\_GC\_TE\_1550)，用于将光从光纤耦合到芯片上
* ring\_radius：微环的弯曲半径，默认 20 微米
* spacing：微环与总线波导之间的间距，默认 0.2 微米，这个参数对光的耦合效率有重要影响
* trace\_template：光波导的模板，定义了光波导的参数

**光波导模板的默认设置**

def \_default\_trace\_template(self):

tt = pdk.StripWaveguideTemplate()

tt.Layout(core\_width=0.45)

return tt

这个方法设置了默认的光波导模板，使用 PDK 提供的条形光波导模板 (StripWaveguideTemplate)，并将波导芯层宽度设置为 0.45 微米。

**微环谐振器子单元的默认设置**

def \_default\_ring(self):

return RingResonator(trace\_template=self.trace\_template).Layout(ring\_radius=self.ring\_radius,

spacing=self.spacing).cell

这个方法创建并返回一个微环谐振器实例。使用前面定义的光波导模板，设置微环的弯曲半径和间距参数。RingResonator应该是一个预定义的微环谐振器类，它的具体实现决定了微环的结构和性能。

**电路元件的默认设置**

def \_default\_insts(self):

insts = {

"ring": self.ring,

"ab1": self.ab,

"ab2": self.ab,

}

return insts

这个方法定义了电路中包含的元件，包括一个微环谐振器 (ring) 和两个光栅耦合器 (ab1和ab2)。光栅耦合器用于将光从光纤输入到芯片，以及将光从芯片输出到光纤。

**元件布局和连接的默认设置**

def \_default\_specs(self):

specs = [

i3.Place("ring", (0, 0)),

i3.Place("ab1:wg", (-15, 15), relative\_to="ring:out1", angle=-90),

i3.Place("ab2:wg", (15, 15), relative\_to="ring:in1", angle=-90),

# i3.FlipH("ab2"),

i3.ConnectManhattan("ab1:wg", "ring:out1"),

i3.ConnectManhattan("ab2:wg", "ring:in1"),

]

return specs

**这部分定义了元件的布局和连接方式：**

i3.Place：用于放置元件。将微环放置在坐标原点 (0,0)，将两个光栅耦合器分别放置在微环的左右两侧

i3.ConnectManhattan：使用曼哈顿布线方式连接元件。将左侧的光栅耦合器与微环的输出端口连接，将右侧的光栅耦合器与微环的输入端口连接

注释掉的i3.FlipH("ab2")：可能是用于水平翻转右侧的光栅耦合器，以确保光的传播方向正确

**电路端口的默认设置**

def \_default\_exposed\_ports(self):

exposed\_ports = {

"ring:in": "in",

"ring:out": "out",

}

return exposed\_ports

这个方法定义了电路对外暴露的端口，将微环的输入和输出端口分别重命名为 "in" 和 "out"，作为整个电路的输入输出端口。

基于已上代码，设计出的微环谐振器的结构如下：

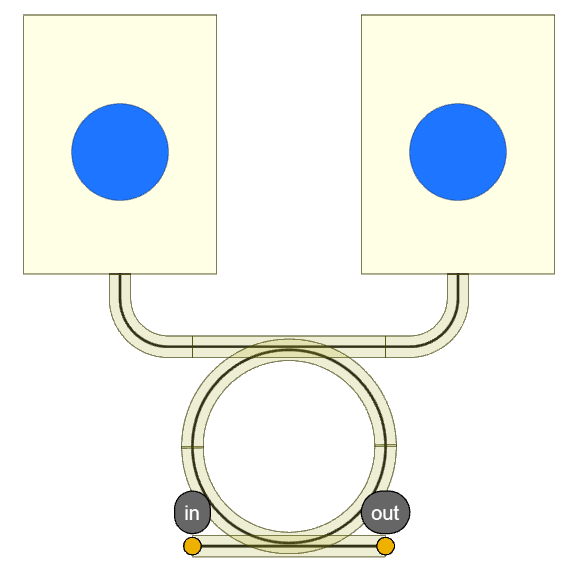


图 2-9-1 微环谐振器的基本结构

基于以上的微环谐振器的基本机构，就可以来设计用于提取微环损耗等参数的DOE，其代码实现过程如下：

class RingResonator\_routed(i3.Circuit):  
 \_name\_prefix = "RingResonator\_routed"  
 bend\_radius = i3.PositiveNumberProperty(default=10.0, doc="the bend radius in the circuit")  
 trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")  
 num\_gc = i3.IntProperty(default=3, doc="number of grating coupler")  
 gc\_space\_y = i3.PositiveNumberProperty(default=50, doc="the gc space in y direction")  
 gc\_space\_x = i3.PositiveNumberProperty(default=450, doc="the gc space in x direction")  
 gc\_y\_offset = i3.PositiveNumberProperty(default=300)  
 gc = i3.ChildCellProperty(doc="grating coupler in circuit")  
 dut = i3.ChildCellProperty(doc="dut in circuit")  
 radius = i3.ListProperty(default=[10, 20, 50], doc="filter orders of mzi", locked=True)  
 ra = i3.EulerRoundingAlgorithm(p=0.70)  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_trace\_template(self):  
 tt = pdk.StripWaveguideTemplate()  
 tt.Layout(core\_width=0.45)  
 return tt  
  
 def \_default\_dut(self):  
 return RingResonators(trace\_template=self.trace\_template)  
  
 def \_default\_insts(self):  
 insts = {}  
 for m in range(self.num\_gc):  
 insts["gc\_in\_{}".format(m)] = self.gc  
 insts["gc\_out\_{}".format(m)] = self.gc  
 insts["dut\_{}".format(m)] = self.dut.modified\_copy(ring\_radius=self.radius[m])  
 for m in range(2):  
 insts["fgc\_dummy\_in\_{}".format(m)] = self.gc  
 insts["fgc\_dummy\_out\_{}".format(m)] = self.gc  
  
 insts["R\_10"]=Text(text="R10", height=30.0)  
 insts["R\_20"]=Text(text="R20", height=30.0)  
 insts["R\_50"]=Text(text="R50", height=30.0)  
 insts["gc\_re\_in"] = self.gc  
 insts["gc\_re\_out"] = self.gc  
 return insts  
  
 def \_default\_specs(self):  
 specs = []  
 for m in range(self.num\_gc):  
 specs += [  
 i3.Place("gc\_in\_{}:vertical\_io".format(m), (0, m \* self.gc\_space\_y)),  
 i3.PlaceRelative("gc\_out\_{}:vertical\_io".format(m), "gc\_in\_{}:vertical\_io".format(m),  
 (self.gc\_space\_x, - self.gc\_y\_offset), angle=180, ),  
 ]  
 specs += [i3.Place("dut\_0:in", (200, -250)),  
 i3.PlaceRelative("gc\_re\_in:wg", "gc\_in\_0:wg", (0, -self.gc\_space\_y)),  
 i3.PlaceRelative("gc\_re\_out:wg", "gc\_out\_0:wg", (0, -self.gc\_space\_y), 180),  
 i3.ConnectManhattan("gc\_re\_in:wg", "gc\_re\_out:wg", bend\_radius=self.bend\_radius,  
 control\_points=[i3.V(i3.START + self.bend\_radius \* 2 + 60),  
 i3.H(i3.START - self.bend\_radius \* 2 - 200),  
 i3.V(i3.START + self.bend\_radius \* 2 + 180),],  
 rounding\_algorithm=self.ra,),  
 ]  
  
 for m in range(1, self.num\_gc):  
 specs += [  
 i3.PlaceRelative("dut\_{}:in".format(m), "dut\_{}:in".format(m - 1),  
 (0, 120), ),  
 ]  
 for m in range(self.num\_gc):  
 specs += [  
 i3.ConnectManhattan("dut\_{}:in".format(m), "gc\_in\_{}:wg".format(m),  
 control\_points=[i3.V(i3.START - self.bend\_radius \* 2 - 10 \* (self.num\_gc-m+1))],  
 bend\_radius=self.bend\_radius,  
 rounding\_algorithm=self.ra,),  
 i3.ConnectManhattan("dut\_{}:out".format(m), "gc\_out\_{}:wg".format(m),  
 control\_points=[i3.V(i3.START + self.bend\_radius \* 2 )],  
 bend\_radius=self.bend\_radius,  
 rounding\_algorithm=self.ra,  
 ),  
 ]  
 specs += [  
 i3.PlaceRelative("R\_10:loc", "gc\_in\_0:wg",  
 (45, - 25), angle=0, ),  
 i3.PlaceRelative("R\_20:loc", "gc\_in\_1:wg",  
 (45, - 25), angle=0, ),  
 i3.PlaceRelative("R\_50:loc", "gc\_in\_2:wg",  
 (45, - 25), angle=0, ),  
 ]  
 specs += [i3.PlaceRelative("fgc\_dummy\_in\_0:vertical\_io", "gc\_re\_in:vertical\_io",  
 (0, -self.gc\_space\_y)),  
 i3.PlaceRelative("fgc\_dummy\_in\_1:vertical\_io", "gc\_in\_2:vertical\_io",  
 (0, self.gc\_space\_y)),  
 i3.PlaceRelative("fgc\_dummy\_out\_0:vertical\_io", "gc\_re\_out:vertical\_io",  
 (0, -self.gc\_space\_y), 180),  
 i3.PlaceRelative("fgc\_dummy\_out\_1:vertical\_io", "gc\_out\_2:vertical\_io",  
 (0, self.gc\_space\_y), 180),  
 ]  
  
 return specs

这段代码定义了一个名为RingResonator\_routed的光子集成电路，它将多个不同半径的微环谐振器集成在一起，并通过光波导和光栅耦合器实现光信号的输入输出。这种设计常用于光通信和光传感系统中，用于波长选择或多参数检测。下面我将分段介绍这段代码的内容。

**类定义与参数设置**

class RingResonator\_routed(i3.Circuit):

\_name\_prefix = "RingResonator\_routed"

bend\_radius = i3.PositiveNumberProperty(default=10.0, doc="the bend radius in the circuit")

trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")

num\_gc = i3.IntProperty(default=3, doc="number of grating coupler")

gc\_space\_y = i3.PositiveNumberProperty(default=50, doc="the gc space in y direction")

gc\_space\_x = i3.PositiveNumberProperty(default=450, doc="the gc space in x direction")

gc\_y\_offset = i3.PositiveNumberProperty(default=300)

gc = i3.ChildCellProperty(doc="grating coupler in circuit")

dut = i3.ChildCellProperty(doc="dut in circuit")

radius = i3.ListProperty(default=[10, 20, 50], doc="filter orders of mzi", locked=True)

ra = i3.EulerRoundingAlgorithm(p=0.70)



这部分定义了电路的基本参数：

* bend\_radius：光波导弯曲半径，默认为 10 微米
* num\_gc：光栅耦合器数量，默认为 3 个
* gc\_space\_y和gc\_space\_x：光栅耦合器在 y 和 x 方向上的间距
* radius：定义了 3 个不同半径的微环 (10、20、50 微米)，不同半径的微环会有不同的谐振波长
* ra：使用 Euler 算法进行光波导转弯，p=0.70 控制转弯的平滑度

**元件默认设置**

def \_default\_gc(self):

return pdk.Fixed\_GC\_TE\_1550()

def \_default\_trace\_template(self):

tt = pdk.StripWaveguideTemplate()

tt.Layout(core\_width=0.45)

return tt

def \_default\_dut(self):

return RingResonators(trace\_template=self.trace\_template)

**这部分设置了电路中使用的基本元件：**

* \_default\_gc：使用 PDK 提供的 1550nm TE 模式固定光栅耦合器
* \_default\_trace\_template：设置光波导模板，芯层宽度为 0.45 微米
* \_default\_dut：使用之前定义的RingResonators类作为被测器件 (DUT)

**电路元件实例化**

def \_default\_insts(self):

insts = {}

for m in range(self.num\_gc):

insts["gc\_in\_{}".format(m)] = self.gc

insts["gc\_out\_{}".format(m)] = self.gc

insts["dut\_{}".format(m)] = self.dut.modified\_copy(ring\_radius=self.radius[m])

for m in range(2):

insts["fgc\_dummy\_in\_{}".format(m)] = self.gc

insts["fgc\_dummy\_out\_{}".format(m)] = self.gc

insts["R\_10"]=Text(text="R10", height=30.0)

insts["R\_20"]=Text(text="R20", height=30.0)

insts["R\_50"]=Text(text="R50", height=30.0)

insts["gc\_re\_in"] = self.gc

insts["gc\_re\_out"] = self.gc

return insts

这部分创建了电路中的所有元件实例：

* 为每个微环创建输入输出光栅耦合器
* 创建 3 个不同半径的微环实例
* 创建一些文本标签用于标识不同半径的微环
* 创建参考光栅耦合器 (gc\_re\_in和gc\_re\_out) 和虚拟光栅耦合器

**元件布局设置**

def \_default\_specs(self):

specs = []

for m in range(self.num\_gc):

specs += [

i3.Place("gc\_in\_{}:vertical\_io".format(m), (0, m \* self.gc\_space\_y)),

i3.PlaceRelative("gc\_out\_{}:vertical\_io".format(m), "gc\_in\_{}:vertical\_io".format(m),

(self.gc\_space\_x, - self.gc\_y\_offset), angle=180, ),

]

specs += [i3.Place("dut\_0:in", (200, -250)),

i3.PlaceRelative("gc\_re\_in:wg", "gc\_in\_0:wg", (0, -self.gc\_space\_y)),

i3.PlaceRelative("gc\_re\_out:wg", "gc\_out\_0:wg", (0, -self.gc\_space\_y), 180),

i3.ConnectManhattan("gc\_re\_in:wg", "gc\_re\_out:wg", bend\_radius=self.bend\_radius,

control\_points=[i3.V(i3.START + self.bend\_radius \* 2 + 60),

i3.H(i3.START - self.bend\_radius \* 2 - 200),

i3.V(i3.START + self.bend\_radius \* 2 + 180),],

rounding\_algorithm=self.ra,),

]

for m in range(1, self.num\_gc):

specs += [

i3.PlaceRelative("dut\_{}:in".format(m), "dut\_{}:in".format(m - 1),

(0, 120), ),

]

这部分设置了元件的位置：

* 将输入光栅耦合器沿 y 轴排列
* 将输出光栅耦合器放置在输入光栅耦合器的右下方
* 设置 3 个微环的位置，使它们垂直排列
* 创建一个参考连接，用于校准或参考测量

**光波导连接设置**

for m in range(self.num\_gc):

specs += [

i3.ConnectManhattan("dut\_{}:in".format(m), "gc\_in\_{}:wg".format(m),

control\_points=[i3.V(i3.START - self.bend\_radius \* 2 - 10 \* (self.num\_gc-m+1))],

bend\_radius=self.bend\_radius,

rounding\_algorithm=self.ra,),

i3.ConnectManhattan("dut\_{}:out".format(m), "gc\_out\_{}:wg".format(m),

control\_points=[i3.V(i3.START + self.bend\_radius \* 2 )],

bend\_radius=self.bend\_radius,

rounding\_algorithm=self.ra,

),

]

这部分设置了光波导的连接：

* 使用曼哈顿布线方式连接每个微环的输入端口到对应的输入光栅耦合器
* 同样连接每个微环的输出端口到对应的输出光栅耦合器
* 通过control\_points参数精确控制光波导的路径形状
* 使用指定的弯曲半径和 Euler 算法确保光波导转弯平滑

**元件布局**

specs += [

i3.PlaceRelative("R\_10:loc", "gc\_in\_0:wg",

(45, - 25), angle=0, ),

i3.PlaceRelative("R\_20:loc", "gc\_in\_1:wg",

(45, - 25), angle=0, ),

i3.PlaceRelative("R\_50:loc", "gc\_in\_2:wg",

(45, - 25), angle=0, ),

]

specs += [i3.PlaceRelative("fgc\_dummy\_in\_0:vertical\_io", "gc\_re\_in:vertical\_io",

(0, -self.gc\_space\_y)),

i3.PlaceRelative("fgc\_dummy\_in\_1:vertical\_io", "gc\_in\_2:vertical\_io",

(0, self.gc\_space\_y)),

i3.PlaceRelative("fgc\_dummy\_out\_0:vertical\_io", "gc\_re\_out:vertical\_io",

(0, -self.gc\_space\_y), 180),

i3.PlaceRelative("fgc\_dummy\_out\_1:vertical\_io", "gc\_out\_2:vertical\_io",

(0, self.gc\_space\_y), 180),

]

return specs

这部分设置了标签元件的位置：

* 在每个微环附近放置文本标签，标明其半径大小
* 放置虚拟光栅耦合器，用于改善芯片边缘的光学性能或对称性

基于以上代码设计微环谐振器损耗等参数提取DOE结构如下图所示：

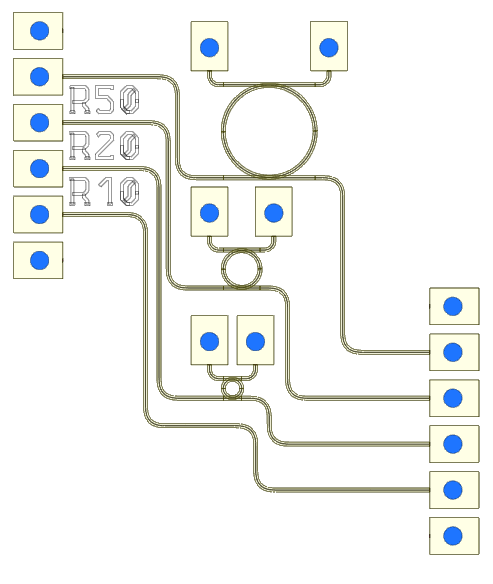


图 2-9-2 微环微环谐振器损耗等参数提取DOE结构

## 3.10 Directional Coupler DoE程序设计

Directional Coupler DoE设计主要用于提取Directional Coupler耦合系数等参数。其DOE程序的第一步就是设计Directional Coupler的基本结构，其设计代码如下：

class DirectionalCoupler(i3.PCell):  
 """Directional coupler or evanescent coupler where the gap, bending radius and coupling length are user defined.  
 This PCell does not have a circuit model.  
 """  
  
 \_name\_prefix = "DC"  
 gap = i3.PositiveNumberProperty(doc="Gap between the waveguides", default=0.3)  
 coupling\_length = i3.NonNegativeNumberProperty(doc="Length of the coupling straight section", default=10.0)  
 radius = i3.PositiveNumberProperty(doc="Gap between the waveguides", default=15.0)  
 trace\_template = i3.TraceTemplateProperty(default=pdk.SWG450())  
  
 class Layout(i3.LayoutView):  
 def \_generate\_instances(self, insts):  
 wg\_bot = i3.RoundedWaveguide(trace\_template=self.trace\_template)  
 wg\_bot.Layout(  
 bend\_radius=self.radius,  
 shape=[  
 (0.0, 0.0),  
 (0.0, self.radius),  
 (self.radius, self.radius),  
 (self.radius + self.coupling\_length, self.radius),  
 (2.0 \* self.radius + self.coupling\_length, self.radius),  
 (2.0 \* self.radius + self.coupling\_length, 0.0),  
 ],  
 )  
 insts += i3.SRef(wg\_bot, name="wg\_bot")  
 wg\_top = i3.RoundedWaveguide(trace\_template=self.trace\_template)  
 width = self.trace\_template.core\_width  
 wg\_top.Layout(  
 bend\_radius=self.radius,  
 shape=[  
 (0.0, 2.0 \* self.radius + width + self.gap),  
 (0.0, self.radius + width + self.gap),  
 (self.radius, self.radius + width + self.gap),  
 (self.radius + self.coupling\_length, self.radius + width + self.gap),  
 (  
 2.0 \* self.radius + self.coupling\_length,  
 self.radius + width + self.gap,  
 ),  
 (  
 2.0 \* self.radius + self.coupling\_length,  
 2.0 \* self.radius + width + self.gap,  
 ),  
 ],  
 )  
 insts += i3.SRef(wg\_top, name="wg\_top")  
 return insts  
  
 def \_generate\_ports(self, ports):  
 ports += i3.expose\_ports(  
 self.instances,  
 {  
 "wg\_bot:in": "in\_1",  
 "wg\_bot:out": "out\_1",  
 "wg\_top:in": "in\_2",  
 "wg\_top:out": "out\_2",  
 },  
 )  
 return ports  
  
 class Netlist(i3.NetlistFromLayout):  
 pass

该代码有由第二节介绍的ipkiss中的i3.PCell器件设计结构设计，其运行结果如下：

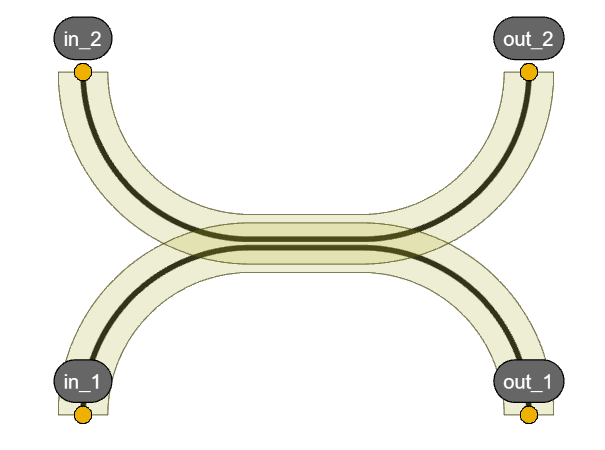


图3-10-1 Directional Coupler 基本结构

基于设计好的Directional Coupler 基本结构可以得到多个Directional Coupler的级联结构：

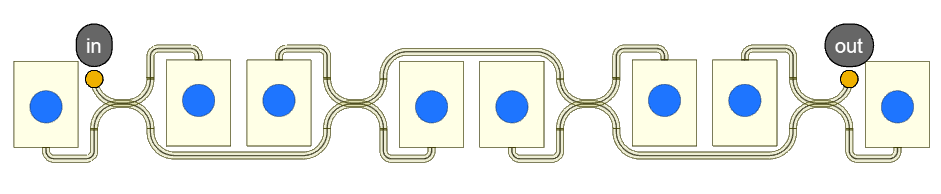


图3-10-2 多个Directional Coupler bar型级联结构

其实现代码如下：

class DirectionalCoupler\_cascaded\_bar(i3.Circuit):  
 """Directional coupler or evanescent coupler where the gap, bending radius and coupling length are user defined.  
 This PCell does not have a circuit model.  
 """  
  
 \_name\_prefix = "DC\_cascaded\_bar"  
 gap = i3.PositiveNumberProperty(doc="Gap between the waveguides", default=0.3)  
 coupling\_length = i3.NonNegativeNumberProperty(doc="Length of the coupling straight section", default=10.0)  
 radius = i3.PositiveNumberProperty(doc="Gap between the waveguides", default=15.0)  
 trace\_template = i3.TraceTemplateProperty(default=pdk.SWG450())  
 dc = i3.ChildCellProperty(doc="directional coupler in circuit")  
 dc\_space = i3.PositiveNumberProperty(doc="the space between each directional coupler", default=110.0)  
 dc\_num = i3.IntProperty(doc="the number of directional coupler", default=4)  
 gc = i3.ChildCellProperty(doc="grating coupler in circuit")  
  
 def \_default\_dc(self):  
 return DirectionalCoupler(  
 gap=self.gap,  
 coupling\_length=self.coupling\_length,  
 radius=self.radius,  
 trace\_template=self.trace\_template,  
 )  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_insts(self):  
 insts = {}  
 for m in range(self.dc\_num):  
 insts[f"dc\_{m}"] = self.dc  
 for m in range(self.dc\_num \* 2):  
 insts[f"gc\_{m}"] = self.gc  
 return insts  
  
 def \_default\_specs(self):  
 specs = []  
 for m in range(self.dc\_num):  
 if m == 0:  
 specs += [i3.Place(f"dc\_{m}:in\_1", (0, 0)),  
 ]  
 else:  
 specs += [i3.Place(f"dc\_{m}:in\_1", (self.dc\_space, 0), relative\_to=f"dc\_{m - 1}:out\_1"),  
 i3.ConnectManhattan(f"dc\_{m}:in\_2", f"dc\_{m - 1}:out\_2", bend\_radius=self.radius),  
 ]  
 specs += [i3.Place(f"gc\_{2 \* m}:wg", (-30, -15), relative\_to=f"dc\_{m}:in\_1", angle=-90),  
 i3.Place(f"gc\_{2 \* m + 1}:wg", (30, -15), relative\_to=f"dc\_{m}:out\_1", angle=-90),  
 i3.ConnectManhattan(f"gc\_{2 \* m}:wg", f"dc\_{m}:in\_1", bend\_radius=5),  
 i3.ConnectManhattan(f"gc\_{2 \* m + 1}:wg", f"dc\_{m}:out\_1", bend\_radius=5),  
 ]  
 return specs  
  
 def \_default\_exposed\_ports(self):  
 exposed\_ports = {}  
 exposed\_ports["dc\_0:in\_2".format(0)] = "in"  
 exposed\_ports[f"dc\_{self.dc\_num - 1}:out\_2"] = "out"  
  
 return exposed\_ports

该代码，是基于ipkiss i3.Circuit设计，同样的方式可以实现cross型的级联：

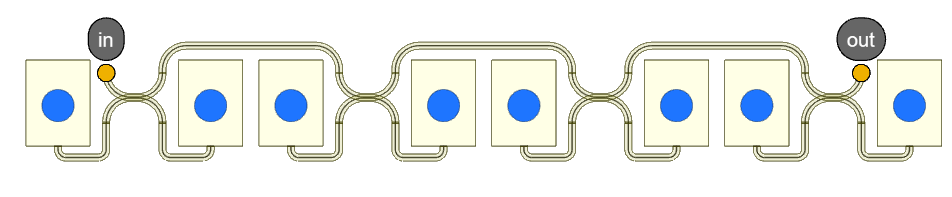


图3-10-3 多个Directional Coupler cross型级联结构

其代码实现如下：

class DirectionalCoupler\_cascaded\_cross(i3.Circuit):  
 """Directional coupler or evanescent coupler where the gap, bending radius and coupling length are user defined.  
 This PCell does not have a circuit model.  
 """  
  
 \_name\_prefix = "DC\_cascaded\_cross"  
 gap = i3.PositiveNumberProperty(doc="Gap between the waveguides", default=0.3)  
 coupling\_length = i3.NonNegativeNumberProperty(doc="Length of the coupling straight section", default=10.0)  
 radius = i3.PositiveNumberProperty(doc="Gap between the waveguides", default=15.0)  
 trace\_template = i3.TraceTemplateProperty(default=pdk.SWG450())  
 dc = i3.ChildCellProperty(doc="directional coupler in circuit")  
 dc\_space = i3.PositiveNumberProperty(doc="the space between each directional coupler", default=110.0)  
 dc\_num = i3.IntProperty(doc="the number of directional coupler", default=4)  
 gc = i3.ChildCellProperty(doc="grating coupler in circuit")  
  
 def \_default\_dc(self):  
 return DirectionalCoupler(  
 gap=self.gap,  
 coupling\_length=self.coupling\_length,  
 radius=self.radius,  
 trace\_template=self.trace\_template,  
 )  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_insts(self):  
 insts = {}  
 for m in range(self.dc\_num):  
 insts[f"dc\_{m}"] = self.dc  
 for m in range(self.dc\_num \* 2):  
 insts[f"gc\_{m}"] = self.gc  
 return insts  
  
 def \_default\_specs(self):  
 specs = []  
 for m in range(self.dc\_num):  
 if m == 0:  
 specs += [i3.Place(f"dc\_{m}:in\_1", (0, 0)),  
 ]  
 else:  
 specs += [i3.Place(f"dc\_{m}:in\_1", (self.dc\_space, 0), relative\_to=f"dc\_{m - 1}:out\_1"),  
 # i3.ConnectManhattan(f"dc\_{m}:in\_2", f"dc\_{m - 1}:out\_2", bend\_radius=self.radius),  
 ]  
  
 if m % 2 == 0:  
 specs += [i3.Place(f"gc\_{2 \* m}:wg", (-30, -12), relative\_to=f"dc\_{m}:in\_1", angle=-90),  
 i3.Place(f"gc\_{2 \* m + 1}:wg", (30, 12), relative\_to=f"dc\_{m}:out\_2", angle=90),  
 i3.ConnectManhattan(f"gc\_{2 \* m}:wg", f"dc\_{m}:in\_1", bend\_radius=5),  
 i3.ConnectManhattan(f"gc\_{2 \* m + 1}:wg", f"dc\_{m}:out\_2", bend\_radius=5),  
 ]  
 else:  
 specs += [i3.Place(f"gc\_{2 \* m}:wg", (-30, 12), relative\_to=f"dc\_{m}:in\_2", angle=90),  
 i3.Place(f"gc\_{2 \* m + 1}:wg", (30, -12), relative\_to=f"dc\_{m}:out\_1", angle=-90),  
 i3.ConnectManhattan(f"gc\_{2 \* m}:wg", f"dc\_{m}:in\_2", bend\_radius=5),  
 i3.ConnectManhattan(f"gc\_{2 \* m + 1}:wg", f"dc\_{m}:out\_1", bend\_radius=5),  
 ]  
 for m in range(self.dc\_num - 1):  
 if m % 2 == 0:  
 specs += [  
 i3.ConnectManhattan(f"dc\_{m + 1}:in\_1", f"dc\_{m}:out\_1", bend\_radius=self.radius),  
 ]  
 else:  
 specs += [  
 i3.ConnectManhattan(f"dc\_{m + 1}:in\_2", f"dc\_{m}:out\_2", bend\_radius=self.radius),  
 ]  
  
 return specs  
  
 def \_default\_exposed\_ports(self):  
 exposed\_ports = {}  
 exposed\_ports["dc\_0:in\_2".format(0)] = "in"  
 if self.dc\_num % 2 == 0:  
 exposed\_ports[f"dc\_{self.dc\_num - 1}:out\_2"] = "out"  
 else:  
 exposed\_ports[f"dc\_{self.dc\_num - 1}:out\_1"] = "out"  
  
 return exposed\_ports

基于设计好的基本单元，连接光栅耦合器，可以得到Directional Coupler测试线路：

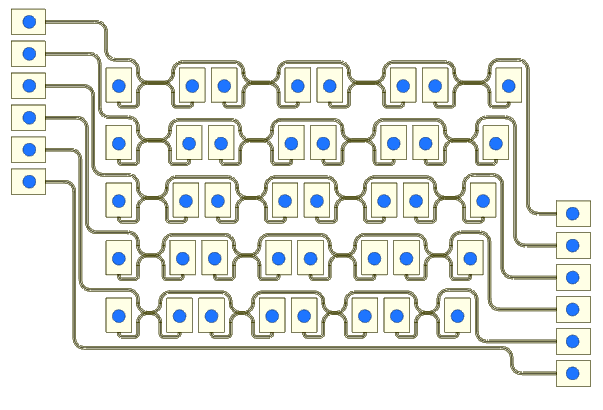


图3-10-4 bar型Directional Coupler测试结构

其代码实现如下：

import numpy as np  
from csip130c import all as pdk  
import ipkiss3.all as i3  
from cells import DirectionalCoupler, DirectionalCoupler\_cascaded\_bar  
  
  
class DC\_routed(i3.Circuit):  
 \_name\_prefix = "DC\_routed"  
 gap = i3.PositiveNumberProperty(doc="Gap between the waveguides", default=0.2)  
 bend\_radius = i3.PositiveNumberProperty(default=15.0, doc="the bend radius in the circuit")  
 trace\_template = i3.TraceTemplateProperty(doc="trace\_template in circuit")  
 num\_gc = i3.IntProperty(default=5, doc="number of grating coupler")  
 gc\_space\_y = i3.PositiveNumberProperty(default=50, doc="the gc space in y direction")  
 gc\_space\_x = i3.PositiveNumberProperty(default=850, doc="the gc space in x direction")  
 gc\_y\_offset = i3.PositiveNumberProperty(default=300)  
 gc = i3.ChildCellProperty(doc="grating coupler in circuit")  
 dut = i3.ChildCellProperty(doc="dut in circuit")  
 coupling\_length = i3.ListProperty(default=[5, 10, 15, 20, 25], doc="filter orders of mzi", locked=True)  
 ra = i3.EulerRoundingAlgorithm(p=0.30)  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_trace\_template(self):  
 tt = pdk.StripWaveguideTemplate()  
 tt.Layout(core\_width=0.45)  
 return tt  
  
 def \_default\_dut(self):  
 return DirectionalCoupler\_cascaded\_bar(gap=self.gap, trace\_template=self.trace\_template, gc=self.gc)  
  
 def \_default\_insts(self):  
 insts = {}  
 for m in range(self.num\_gc):  
 insts["gc\_in\_{}".format(m)] = self.gc  
 insts["gc\_out\_{}".format(m)] = self.gc  
 # insts["gc\_ab\_in\_{}".format(m)] = self.gc  
 # insts["gc\_ab\_out\_{}".format(m)] = self.gc  
 insts["dut\_{}".format(m)] = self.dut.modified\_copy(  
 coupling\_length=self.coupling\_length[m])  
  
 insts["gc\_re\_in"] = self.gc  
 insts["gc\_re\_out"] = self.gc  
 # insts["gc\_dm\_up\_in"] = self.gc  
 # insts["gc\_dm\_up\_out"] = self.gc  
 # insts["gc\_dm\_down\_in"] = self.gc  
 # insts["gc\_dm\_down\_out"] = self.gc  
 return insts  
  
 def \_default\_specs(self):  
 specs = []  
 for m in range(self.num\_gc):  
 specs += [  
 i3.Place("gc\_in\_{}:vertical\_io".format(m), (0, m \* self.gc\_space\_y)),  
 i3.PlaceRelative("gc\_out\_{}:vertical\_io".format(m), "gc\_in\_{}:vertical\_io".format(m),  
 (self.gc\_space\_x, - self.gc\_y\_offset)),  
  
 i3.FlipH("gc\_out\_{}".format(m)),  
 ]  
 specs += [i3.Place("dut\_0:in", (170, -240)),  
 i3.PlaceRelative("gc\_re\_in:wg", "gc\_in\_0:wg", (0, -self.gc\_space\_y)),  
 i3.PlaceRelative("gc\_re\_out:wg", "gc\_out\_0:wg", (0, -self.gc\_space\_y), 180),  
 i3.ConnectManhattan("gc\_re\_in:wg", "gc\_re\_out:wg", bend\_radius=self.bend\_radius,  
 control\_points=[i3.V(i3.START + self.bend\_radius \* 2 + 15),  
 i3.H(i3.START - self.bend\_radius \* 2 - 230),  
 i3.V(i3.END - 70), ],  
 rounding\_algorithm=self.ra, ),  
 ]  
  
 for m in range(1, self.num\_gc):  
 specs += [  
 i3.PlaceRelative("dut\_{}:in".format(m), "dut\_{}:in".format(m - 1), (0, 90), ),  
 # i3.PlaceRelative(f"gc\_ab\_in\_{m}:wg", f"dut\_{m}:in\_2",  
 # (-self.bend\_radius \* 3 - 5, 10), angle=90),  
 # i3.PlaceRelative(f"gc\_ab\_out\_{m}:wg", f"dut\_{m}:out\_2",  
 # (self.bend\_radius \* 3 + 5, 10), angle=90),  
 # i3.FlipH(f"gc\_ab\_out\_{m}"),  
 ]  
 for m in range(self.num\_gc):  
 specs += [  
 i3.ConnectManhattan("dut\_{}:in".format(m), "gc\_in\_{}:wg".format(m),  
 control\_points=[i3.V(i3.START - 30 - 10 \* (self.num\_gc - m + 1))],  
 bend\_radius=self.bend\_radius,  
 rounding\_algorithm=self.ra,  
 ),  
 i3.ConnectManhattan("dut\_{}:out".format(m), "gc\_out\_{}:wg".format(m),  
 control\_points=[i3.V(i3.START + 60 - 0 \* (self.num\_gc - m + 1))],  
 bend\_radius=self.bend\_radius,  
 rounding\_algorithm=self.ra,  
 ),  
 ]  
 # specs += [  
 # i3.PlaceRelative("gc\_dm\_down\_in:vertical\_io", "gc\_re\_in:vertical\_io",  
 # (0, - self.gc\_space\_y)),  
 # i3.PlaceRelative("gc\_dm\_up\_in:vertical\_io", "gc\_in\_4:vertical\_io",  
 # (0, self.gc\_space\_y)),  
 # i3.PlaceRelative("gc\_dm\_down\_out:vertical\_io", "gc\_re\_out:vertical\_io",  
 # (0, -self.gc\_space\_y)),  
 # i3.FlipH("gc\_dm\_down\_out"),  
 # i3.PlaceRelative("gc\_dm\_up\_out:vertical\_io", "gc\_out\_4:vertical\_io",  
 # (0, self.gc\_space\_y)),  
 # i3.FlipH("gc\_dm\_up\_out"),  
 # ]  
 return specs

将代码中的default\_dut，替换为DirectionalCoupler\_cascaded\_bar，即：

def \_default\_dut(self):  
 return DirectionalCoupler\_cascaded\_bar(gap=self.gap, trace\_template=self.trace\_template, gc=self.gc)

可以得到：

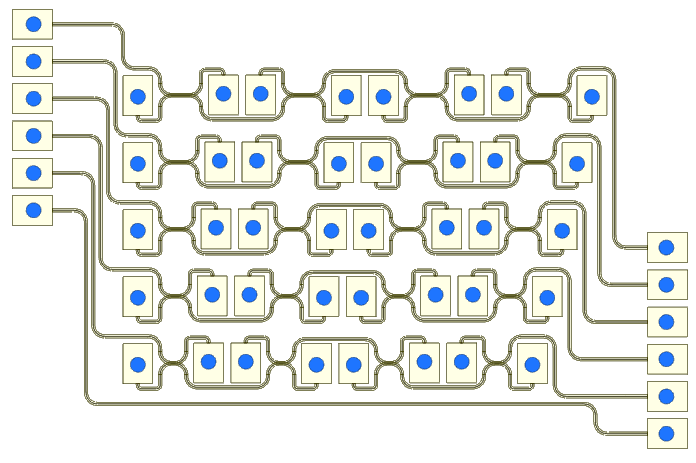


图3-10-5 cross型Directional Coupler测试结构

## 3.11 Thermal Phase Shifter DoE程序设计

Thermal Phase Shifter DoE设计主要用于提取Thermal Phase Shifter 调制效率、损耗等参数。

其中调制效率提取设计代码如下：

class Optical\_MZI\_Frame\_C(i3.Circuit):  
 \_name\_prefix = "Optical MZI frame for C band"  
  
 trace\_template = i3.ChildCellProperty("Trace\_template for the whole structure")  
  
 gc = i3.ChildCellProperty("grating coupler used")  
 splitter = i3.ChildCellProperty("power splitter")  
 length = i3.NonNegativeNumberProperty(default=2000, doc="grating couplers distance")  
 path\_diff = i3.NonNegativeNumberProperty(doc="Length difference between the two arms")  
  
 def \_default\_trace\_template(self):  
 return pdk.SWG450()  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_splitter(self):  
 return pdk.Fixed\_M1X2\_TE\_1550()  
  
 def \_default\_path\_diff(self):  
 return 50  
  
 def \_default\_insts(self):  
 insts = {  
 "gc1": self.gc,  
 "gc2": self.gc,  
 "sp": self.splitter,  
 "cb": self.splitter  
 }  
 return insts  
  
 def \_default\_specs(self):  
 io = self.gc.Layout().ports["vertical\_io"]  
 placement\_spec = [i3.Place("gc1", position=(-io.x, 0)),  
 i3.Place("gc2", position=(self.length+io.x, -30), angle=180),  
 i3.Place("sp", position=(10-io.x, 0)),  
 i3.Place("cb", position=(self.length +io.x- 10, -30), angle=180),  
 ]  
  
 optical\_spec = [i3.ConnectManhattan("gc1:wg", "sp:in\_1", trace\_template=self.trace\_template),  
 i3.ConnectManhattan("cb:in\_1", "gc2:wg", trace\_template=self.trace\_template),  
 i3.ConnectManhattan("sp:out\_1", "cb:out\_2", trace\_template=self.trace\_template,  
 control\_points=[i3.H(25)], bend\_radius=8),  
 i3.ConnectManhattan("sp:out\_2", "cb:out\_1", trace\_template=self.trace\_template,  
 control\_points=[i3.H(-25 - self.path\_diff)], bend\_radius=8)  
 ]  
 return placement\_spec + optical\_spec  
  
 class Layout(i3.Circuit.Layout):  
 def \_generate\_elements(self, elems):  
 super(Optical\_MZI\_Frame\_C.Layout, self).\_generate\_elements(elems)  
  
 elems += i3.Rectangle(layer = i3.TECH.PPLAYER.FETCH.CLD,  
 center = (50+25,0),  
 box\_size=(5,5))  
  
 elems += i3.Rectangle(layer=i3.TECH.PPLAYER.FETCH.CLD,  
 center=(self.length-50-25, -30),  
 box\_size=(5, 5))  
 elems += i3.Rectangle(layer=i3.TECH.PPLAYER.DEVREC,  
 center=(50+25, 0),  
 box\_size=(5, 5))  
  
 elems += i3.Rectangle(layer=i3.TECH.PPLAYER.DEVREC,  
 center=(self.length - 50-25, -30),  
 box\_size=(5, 5))  
 class Metal\_heater\_width\_sweep\_MZI\_tests\_C(Optical\_MZI\_Frame\_C):  
 heater1 = i3.DefinitionProperty(doc="Used heater in this sweep")  
 heater2 = i3.DefinitionProperty(doc="Used heater in this sweep")  
 bp = i3.ChildCellProperty(doc="Bondpad Frame")  
 heater\_width = i3.DefinitionProperty(doc="Used metal heater width")  
 heater\_offset = i3.DefinitionProperty(doc="Used metal heater offset")  
 heater\_length = i3.NonNegativeNumberProperty(doc="Used metal heater length")  
  
 def \_default\_name(self):  
 return f"Metal\_heater\_length\_{self.heater\_length}\_width\_sweep\_MZI\_tests\_C"  
  
 def \_default\_heater1(self):  
 return heaters.Metal\_heater\_straight  
  
 def \_default\_heater2(self):  
 return heaters.Metal\_heater\_straight\_with\_thermal\_trench  
 def \_default\_heater\_length(self):  
 return 150  
  
 def \_default\_heater\_width(self):  
 return Width  
  
 def \_default\_heater\_offset(self):  
 return 0  
  
 def \_default\_bp(self):  
 bp\_needed = len(self.heater\_width)\*2 + 2  
 return Bondpad\_Frame.PadFrame(no\_pads\_per\_row=bp\_needed)  
  
 def \_default\_insts(self):  
 insts = super(Metal\_heater\_width\_sweep\_MZI\_tests\_C,self).\_default\_insts()  
  
 for i in range(len(self.heater\_width)):  
 insts[f"heater1\_{i}"] = self.heater1(name=self.name + f"\_heater1\_{i}").Layout(  
 length=self.heater\_length, width=self.heater\_width[i])  
 insts[f"heater2\_{i}"] = self.heater2(name=self.name + f"\_heater2\_{i}").Layout(  
 length=self.heater\_length, width=self.heater\_width[i])  
 insts["BP"] = self.bp  
 return insts  
  
 def \_default\_specs(self):  
 specs = super(Metal\_heater\_width\_sweep\_MZI\_tests\_C,self).\_default\_specs()  
 placement\_specs = []  
 for i in range(len(self.heater\_width)):  
 placement\_specs.extend([  
 i3.Place(f"heater1\_{i}", position=(110+(self.heater\_length+30)\*i, 25 + self.heater\_offset-self.heater\_width[i]/2)),  
 i3.Place(f"heater2\_{i}", position=(110 + (self.heater\_length + 30) \* (i+len(self.heater\_width)), 25 + self.heater\_offset - self.heater\_width[i] / 2))  
 ])  
  
 placement\_specs.extend([  
 i3.Place("BP", position=(30,-400))  
 ])  
  
 electrical\_routing = []  
 wire\_template1 = i3.ElectricalWireTemplate()  
 wire\_template2 = i3.ElectricalWireTemplate()  
 wire\_template1.Layout(width=5.0, layer=i3.TECH.PPLAYER.M1.DRW)  
 wire\_template2.Layout(width=5.0, layer=i3.TECH.PPLAYER.M2.DRW)  
  
 for i in range(len(self.heater\_width)):  
 electrical\_routing.extend([  
 i3.ConnectElectrical(f"heater1\_{i}:elec\_1","BP:elec\_BP\_1\_1",  
 control\_points= [  
 i3.VIA((30,60),trace\_template=wire\_template2,layout = VIA())  
 ],  
 start\_angle=90, end\_angle=90,  
 trace\_template=wire\_template1),  
 i3.ConnectElectrical(f"heater2\_{i}:elec\_1", "BP:elec\_BP\_1\_1",  
 control\_points=[  
 i3.VIA((30, 60), trace\_template=wire\_template2, layout=VIA())  
 ],  
 start\_angle=90, end\_angle=90,  
 trace\_template=wire\_template1),  
 i3.ConnectElectrical(f"heater1\_{i}:elec\_2", f"BP:elec\_BP\_1\_{i+2}",  
 control\_points=[  
 i3.VIA((10+190\*(i+1), -350+i\*15), trace\_template=wire\_template2, layout=VIA())  
 ],  
 start\_angle=-90, end\_angle=90,  
 trace\_template=wire\_template1),  
 i3.ConnectElectrical(f"heater2\_{i}:elec\_2", f"BP:elec\_BP\_1\_{i+2+len(self.heater\_width)}",  
 control\_points=[  
 i3.VIA((10+190\*(i+1+len(self.heater\_width)), -350+(i+len(self.heater\_width))\*15), trace\_template=wire\_template2, layout=VIA())  
 ],  
 start\_angle=-90, end\_angle=90,  
 trace\_template=wire\_template1),  
 i3.ConnectElectrical("BP:elec\_BP\_1\_1", f"BP:elec\_BP\_1\_{len(self.heater\_width)\*2 + 2}",start\_angle=-90, end\_angle=-90,trace\_template=wire\_template1,  
 control\_points = [i3.H(-450)])  
  
 ])  
  
 return specs + placement\_specs+electrical\_routing  
  
 class Layout(Optical\_MZI\_Frame\_C.Layout):  
  
 def \_generate\_elements(self, elems):  
 super(Metal\_heater\_width\_sweep\_MZI\_tests\_C.Layout,self).\_generate\_elements(elems)  
 elems += i3.PolygonText(layer=i3.TECH.PPLAYER.FETCH.CLD,  
  
 coordinate=(1000, -300),  
 text=self.name,  
 alignment=(i3.TEXT.ALIGN.CENTER, i3.TEXT.ALIGN.BOTTOM),  
 font=i3.TEXT.FONT.DEFAULT,  
 height=20,)  
 elems += i3.PolygonText(layer=i3.TECH.PPLAYER.FETCH.CLD,  
  
 coordinate=(1000, -350),  
 text=f"heater\_length\_{self.heater\_length}\_heater\_width" + str(Width),  
 alignment=(i3.TEXT.ALIGN.CENTER, i3.TEXT.ALIGN.BOTTOM),  
 font=i3.TEXT.FONT.DEFAULT,  
 height=15, )  
 return elems

return elems

这段代码定义了一个用于C波段(O波段的类似)的光学马赫-曾德尔干涉仪(MZI)框架，并集成了金属加热器进行宽度扫描测试。以下是代码的核心内容：

**基础MZI结构 (Optical\_MZI\_Frame\_C类)**：

* 包含两个光栅耦合器(gc1/gc2)和两个分束器(sp/cb)
* 使用SWG450波导模板连接这些组件
* 布局中包含两个臂，路径差(path\_diff)默认为50μm
* 添加了标记矩形(FETCH.CLD和DEVREC层)

**金属加热器宽度扫描测试 (Metal\_heater\_width\_sweep\_MZI\_tests\_C类)**：

* 继承自基础MZI结构并添加加热器功能
* 支持两种加热器类型：普通金属加热器和带热沟槽的金属加热器
* 可以扫描不同宽度的加热器(Width参数)
* 包含键合焊盘框架(Bondpad Frame)用于电连接

**关键特性**：

* 电气布线使用M1和M2金属层，通过VIA连接
* 自动生成测试结构的名称和标注信息
* 加热器按指定宽度数组进行排列和连接
* 支持自定义加热器长度和偏移量

**技术实现**：

* 使用i3.Circuit作为基类
* 通过ChildCellProperty管理子组件
* 使用Place和Connect指令进行组件放置和连接
* 支持曼哈顿布线(ConnectManhattan)

这个设计主要用于测试不同宽度金属加热器对MZI性能的影响，是光子集成电路中常见的热光调谐测试结构。

代码运行结果如下：

图示

AI 生成的内容可能不正确。

图3-11-1 Thermal Phase Shifter 调制效率提取线路设计

Thermal Phase Shifter损耗提取代码如下：

class Metal\_heater\_loss\_sweep(i3.Circuit):  
 \_name\_prefix = "Metal\_heater\_loss\_sweep\_C"  
  
 ref = i3.ChildCellProperty(doc="standard reference waveguide")  
 heater\_lengths = i3.DefinitionProperty(doc="sweep lengths list")  
 bp = i3.ChildCellProperty(doc="Bondpads")  
 heater = i3.DefinitionProperty(doc="define the heater class")  
 heater\_offset = i3.DefinitionProperty(doc = "gap between heater center to waveguide center")  
 heater\_width = i3.DefinitionProperty(doc= "heater width")  
  
 def \_default\_name(self):  
 return "Metal\_heater\_loss\_sweep"  
 def \_default\_ref(self):  
 return Reference\_straight\_waveguide(name = self.name + "\_waveguide")  
 def \_default\_heater\_lengths(self):  
 return lengths  
 def \_default\_heater\_offset(self):  
 return 0  
 def \_default\_heater\_width(self):  
 return 2.5  
 def \_default\_heater(self):  
 return heaters.Metal\_heater\_straight  
 def \_default\_bp(self):  
 bp\_needed = (len(self.heater\_lengths)-1)\*2  
 return Bondpad\_Frame.PadFrame(no\_pads\_per\_row = bp\_needed)  
 #  
  
 def \_default\_insts(self):  
 insts = {}  
 insts["ref"] = self.ref  
 for i in range(len(lengths)-1):  
 insts[f"heater\_{i+1}"] = self.heater(name=self.name+f"\_heater\_{i}").Layout(length = self.heater\_lengths[i+1], width = self.heater\_width)  
 insts[f"optical\_wg\_{i+1}"] = self.ref  
 insts["BP"] = self.bp  
 return insts  
  
 def \_default\_specs(self):  
 placement\_specs = []  
  
 offset\_x = 55  
 offset\_y = 80  
 for i in range(len(self.heater\_lengths)-1):  
 placement\_specs.extend(  
 [i3.Place(f"heater\_{i+1}",(i%2\*offset\_x,offset\_y\*i-1.25)),  
 i3.Place(f"optical\_wg\_{i+1}",(i%2\*offset\_x-extra\_length/2,offset\_y\*i))]  
 )  
 placement\_specs.append(i3.Place("ref",(offset\_x-extra\_length/2, -offset\_y)))  
 placement\_specs.append(i3.Place("BP", (-extra\_length/4, -offset\_y-500)))  
  
 electrical\_specs = []  
 wire\_template1 = i3.ElectricalWireTemplate()  
 wire\_template2 = i3.ElectricalWireTemplate()  
 wire\_template1.Layout(width=5.0, layer = i3.TECH.PPLAYER.M1.DRW)  
 wire\_template2.Layout(width=5.0, layer=i3.TECH.PPLAYER.M2.DRW)  
 for i in range(len(self.heater\_lengths) - 1):  
 electrical\_specs.extend([  
 i3.ConnectElectrical(  
 f"heater\_{i + 1}:elec\_1", f"BP:elec\_BP\_1\_{2\*i+1}",  
 start\_angle = -90, end\_angle = 90,  
 trace\_template=wire\_template1,  
 control\_points = [  
 i3.VIA(  
 (i % 2 \* offset\_x-3.125, offset\_y \* i - 30), trace\_template=wire\_template2,  
 layout=Contacts.VIA()  
 ),  
 i3.VIA(  
 (-140 - 10 \* i, offset\_y \* i - 30), trace\_template=wire\_template1,  
 layout=Contacts.VIA()  
 ),  
 i3.VIA(  
 (-140 - 10 \* i, -offset\_y - 400 - 10 \* i), trace\_template=wire\_template2,  
 layout=Contacts.VIA()  
 ),  
 i3.VIA(  
 (-extra\_length / 4 + 190 \* (2 \* i), -offset\_y - 400 - 10 \* i),  
 trace\_template=wire\_template1,  
 layout=Contacts.VIA()  
 ),  
 ],  
 ),  
 i3.ConnectElectrical(  
 f"heater\_{i + 1}:elec\_2", f"BP:elec\_BP\_1\_{2 \* i + 2}",  
 start\_angle=-90, end\_angle=90,  
 control\_points=[i3.VIA((i%2\*offset\_x+3.125+self.heater\_lengths[i+1],offset\_y\*i-36), trace\_template= wire\_template2,  
 layout= Contacts.VIA()),  
 i3.VIA((1650 + 10 \* (i + 1), offset\_y \* i - 36),  
 trace\_template=wire\_template1,  
 direction\_out=i3.SOUTH,  
 layout=Contacts.VIA()),  
 i3.VIA((1650 + 10 \* (i + 1), -offset\_y - 350 - 10 \* i),  
 trace\_template=wire\_template2,  
 layout=Contacts.VIA()),  
  
 i3.VIA((-extra\_length / 4 + 190 \* (2 \* i + 1), -offset\_y - 350 - 10 \* i),  
 trace\_template=wire\_template1,  
 layout=Contacts.VIA()),  
 ],  
 trace\_template=wire\_template1  
 )  
 ])  
 return placement\_specs+electrical\_specs  
  
 class Layout(i3.Circuit.Layout):  
  
 def \_generate\_elements(self, elems):  
 super(Metal\_heater\_loss\_sweep.Layout,self).\_generate\_elements(elems)  
 elems += i3.PolygonText(layer=i3.TECH.PPLAYER.FETCH.CLD,  
 transformation=i3.Rotation(rotation=90),  
 coordinate=(100, 380),  
 text=self.name,  
 alignment=(i3.TEXT.ALIGN.CENTER, i3.TEXT.ALIGN.BOTTOM),  
 font=i3.TEXT.FONT.DEFAULT,  
 height=20,)  
 elems += i3.PolygonText(layer=i3.TECH.PPLAYER.FETCH.CLD,  
 transformation=i3.Rotation(rotation=90),  
 coordinate=(100, 350),  
 text=str(lengths),  
 alignment=(i3.TEXT.ALIGN.CENTER, i3.TEXT.ALIGN.BOTTOM),  
 font=i3.TEXT.FONT.DEFAULT,  
 height=15, )  
 return elems

这段代码定义了一个名为 **Metal\_heater\_loss\_sweep** 的 i3.Circuit 类，主要用于创建和布局金属加热器损耗测试结构。

**主要结构**

**类定义**：

* 继承自 **i3.Circuit**，表示这是一个电路设计
* 包含内部 **Layout** 类用于处理布局细节

**属性定义**：

* **\_name\_prefix**：设置类名前缀
* 多个属性定义如 **ref**, **heater\_lengths**, **bp**, **heater** 等
* 每个属性都有默认值方法（**\_default\_xxx**）

**主要功能**：

* 创建一系列不同长度的金属加热器结构
* 为每个加热器提供光学波导参考
* 添加键合焊盘(Bondpad)用于电气连接
* 自动布局所有组件
* 创建电气连接路径

**关键方法**

**\_default\_insts()**：

* 实例化参考波导、加热器和光学波导
* 为每个加热器长度创建实例

**\_default\_specs()**：

* 定义组件放置位置
* 创建复杂的电气连接路径，包括多个通孔(VIA)和控制点
* 处理奇数/偶数索引组件的不同布局

**Layout.\_generate\_elements()**：

* 添加文本标签显示设计名称和加热器长度列表
* 使用特定技术层进行标注

该代码运行结果如下：

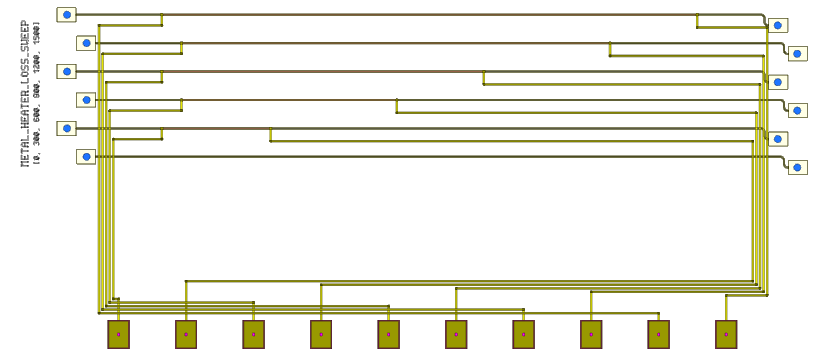


图3-11-2 Thermal Phase Shifter 损耗提取线路设计

## 3.12 Phase Shifter DoE程序设计

Phase Shifter DoE设计同样是用于提取Phase Shifter 调制效率、损耗等参数。

其中调制效率提取设计代码如下：

class PS\_in\_MZI\_C(i3.Circuit):  
 \_name\_prefix = "PN\_in\_MZI\_C"  
  
 gc = i3.ChildCellProperty(doc="used grating coupler")  
 PS = i3.ChildCellProperty(doc="used heaters")  
 trace\_template = i3.ChildCellProperty(doc="used trace template")  
 ref = i3.ChildCellProperty(doc="reference waveguide")  
 mmi = i3.ChildCellProperty(doc="used power splitter")  
 BP = i3.ChildCellProperty(doc="used bondpads")  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_PS(self):  
 return pdk.Fixed\_PS\_PN\_TE\_1550()  
  
 def \_default\_trace\_template(self):  
 return pdk.SWG450()  
  
 def \_default\_mmi(self):  
 return pdk.Fixed\_M1X2\_TE\_1550()  
  
 def \_default\_ref(self):  
 return Reference\_straight\_waveguide(name = self.name + "\_ref")  
  
 def \_default\_BP(self):  
 return PadFrame(no\_pads\_per\_row = 2)  
 def \_default\_insts(self):  
 insts = {  
 "gc1":self.gc,  
 "gc2":self.gc,  
 "ps":self.PS,  
 "ref":self.ref,  
 "sp":self.mmi,  
 "cb":self.mmi,  
 "bp":self.BP  
 }  
 return insts  
  
 def \_default\_specs(self):  
  
 optical\_spec = []  
 electrical\_spec = []  
 io = self.gc.Layout().ports["vertical\_io"].position  
 placement\_spec = [  
 i3.Place("gc1", position=(-20 - io.x, 00)),  
 i3.Place("gc2", position=(2000 + io.x - 20, -30), angle=180),  
 ]  
 placement\_spec.extend([  
 # i3.Place("gc1",(0,0)),  
 # i3.Place("gc2", (2000, 0),angle=180),  
 i3.Place("sp",(20,0)),  
 i3.Place("cb", (1900, 0),angle=180),  
 i3.Place("ps", (800, 30)),  
 i3.Place("ref", (0, -100)),  
 i3.Place("bp", (900, -150)),  
 ])  
 optical\_spec.extend([  
 i3.ConnectManhattan("gc1:wg","sp:in\_1"),  
 i3.ConnectManhattan("gc2:wg", "cb:in\_1"),  
 i3.ConnectManhattan("ps:opt\_in","sp:out\_1", trace\_template = self.trace\_template),  
 i3.ConnectManhattan("ps:opt\_out", "cb:out\_2",trace\_template = self.trace\_template),  
 i3.ConnectManhattan("sp:out\_2", "cb:out\_1",control\_points = [i3.H(-80)],trace\_template = self.trace\_template),  
 ])  
 wire\_template1 = i3.ElectricalWireTemplate()  
 wire\_template2 = i3.ElectricalWireTemplate()  
 wire\_template1.Layout(width=5.0, layer=i3.TECH.PPLAYER.M1.DRW)  
 wire\_template2.Layout(width=5.0, layer=i3.TECH.PPLAYER.M2.DRW)  
 size\_info = self.insts["ps"].Layout().size\_info()  
 electrical\_spec.extend([  
 i3.ConnectElectrical("ps:s\_elec\_n1", "bp:elec\_BP\_1\_1", trace\_template=wire\_template1,  
 start\_angle=180, end\_angle=90,  
 control\_points=[  
 i3.H(30 + size\_info.south - 10), i3.V(700),  
 i3.H(-85)  
 ],  
 ),  
 i3.ConnectElectrical("ps:s\_elec\_n2", "bp:elec\_BP\_1\_1", trace\_template=wire\_template1,  
 start\_angle=0, end\_angle=90,  
 control\_points=[  
 i3.H(30 + size\_info.south - 10), i3.V(700),  
 i3.H(-85)  
 ],  
 ),  
 i3.ConnectElectrical("ps:g\_elec\_p1", "bp:elec\_BP\_1\_2", trace\_template=wire\_template1,  
 start\_angle=180, end\_angle=90,  
 control\_points=[  
 i3.H(30 + size\_info.north + 10), i3.V(1350),  
 i3.H(-65)  
 ],  
 ),  
 i3.ConnectElectrical("ps:g\_elec\_p2", "bp:elec\_BP\_1\_2", trace\_template=wire\_template1,  
 start\_angle=0, end\_angle=90,  
 control\_points=[  
 i3.H(30 + size\_info.north + 10), i3.V(1350),  
 i3.H(-65)  
 ],  
 ),  
  
 ])  
 return placement\_spec+optical\_spec+electrical\_spec  
  
 class Layout(i3.Circuit.Layout):  
  
 def \_generate\_elements(self, elems):  
 super(PS\_in\_MZI\_C.Layout,self).\_generate\_elements(elems)  
 elems += i3.Rectangle(layer=i3.TECH.PPLAYER.FETCH.CLD,  
 center=(60, 0),  
 box\_size=(5, 5))  
  
 elems += i3.Rectangle(layer=i3.TECH.PPLAYER.FETCH.CLD,  
 center=(1860, 0),  
 box\_size=(5, 5))  
 elems += i3.Rectangle(layer=i3.TECH.PPLAYER.DEVREC,  
 center=(60, 0),  
 box\_size=(5, 5))  
  
 elems += i3.Rectangle(layer=i3.TECH.PPLAYER.DEVREC,  
 center=(1860, 0),  
 box\_size=(5, 5))  
 return elems

这段代码定义了一个名为 PS\_in\_MZI\_C 的 i3.Circuit 类，实现了一个包含相位调制器(Phase Shifter, PS)的马赫-曾德尔干涉仪(MZI)结构。

**主要结构**

**类定义**：

* 继承自 i3.Circuit，表示这是一个电路设计
* 包含内部 Layout 类用于处理布局细节

**属性定义**：

**\_name\_prefix**：设置类名前缀

* 多个组件属性如 **gc**(光栅耦合器)、**PS**(相位调制器)、**mmi**(多模干涉耦合器)等
* 每个属性都有默认值方法（**\_default\_xxx**）

**光学部分**：

* 创建完整的MZI光路，包含：
* 两个光栅耦合器(输入/输出)
* 两个MMI耦合器(分束器和合束器)
* 相位调制器(位于一个臂上)

参考波导

**电气部分**：

* 为相位调制器提供电气连接
* 包含键合焊盘用于外部连接

**布局特点**：

* 使用Manhattan连接方式(直角拐弯)连接光学组件
* 为电气连接定义了两层金属布线模板
* 在关键位置添加了标记矩形

**关键方法**

**\_default\_insts()**：

实例化所有组件：光栅耦合器、相位调制器、MMI、参考波导和键合焊盘

**\_default\_specs()**：

* 定义组件放置位置
* 创建光学连接路径
* 建立电气连接路径，包括控制点和角度定义
* 使用不同金属层进行布线

**Layout.\_generate\_elements()**：

* 在分束器和合束器位置添加标记矩形
* 使用特定技术层(CLD和DEVREC)进行标注

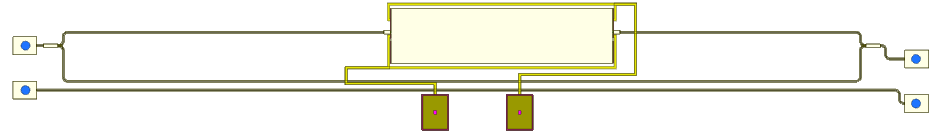


图3-12-1 Phase Shifter 调制效率提取线路设计

接下来是Phase Shifter 调制效率提取线路设计，其设计代码如下：

class PN\_loss\_sweep(i3.Circuit):  
 \_name\_prefix = "PN\_loss\_sweep"  
 Cascaded\_PN = i3.DefinitionProperty(doc = "get the cascaded PN junctions")  
 cascaded\_number = i3.ListProperty(doc="define the cascaded number")  
 gc = i3.ChildCellProperty(doc = "used grating coupler")  
 PS = i3.ChildCellProperty(doc="define the used PN junctions")  
 bp = i3.ChildCellProperty(doc = "define the bondpads")  
 ref = i3.ChildCellProperty(doc = "reference")  
 def \_default\_name(self):  
 return f"Cascaded\_junction\_loss\_sweep\_C"  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_bp(self):  
 return PadFrame(no\_pads\_per\_row = len(self.cascaded\_number)\*2)  
  
 def \_default\_PS(self):  
 return pdk.Fixed\_PS\_PN\_TE\_1550()  
 def \_default\_Cascaded\_PN(self):  
 return Cascaded\_PN\_C  
  
 def \_default\_cascaded\_number(self):  
 return [1,2,3]  
  
 def \_default\_ref(self):  
 return Reference\_straight\_waveguide(name = self.name + "\_ref",bend = True)  
 def \_default\_insts(self):  
 insts = {}  
 insts["bp"] = self.bp  
 insts["ref"] = self.ref  
 for i, j in enumerate(self.cascaded\_number):  
 insts[f"PS{i}"]=self.Cascaded\_PN(PS = self.PS, cascaded\_number =j, name = self.name + f"\_PS{i}")  
 insts[f"gc{i}\_1"] = self.gc  
 insts[f"gc{i}\_2"] = self.gc  
 return insts  
  
 def \_default\_specs(self):  
  
 placement\_specs = []  
 optical\_spec = []  
 electrical\_spec = []  
 placement\_specs.append(  
 i3.Place("bp", position= (480,-150))  
 )  
 placement\_specs.append(  
 i3.Place("ref", position=(0, -150))  
 )  
 wire\_template1 = i3.ElectricalWireTemplate()  
 wire\_template2 = i3.ElectricalWireTemplate()  
 wire\_template1.Layout(width=5.0, layer=i3.TECH.PPLAYER.M1.DRW)  
 wire\_template2.Layout(width=5.0, layer=i3.TECH.PPLAYER.M2.DRW)  
  
 for i, j in enumerate(self.cascaded\_number):  
 size\_info = self.insts[f"PS{i}"].Layout().size\_info()  
 length = size\_info.east - size\_info.west  
 io = self.gc.Layout().ports["vertical\_io"].position  
 placement\_specs.append(  
 i3.Place(f"PS{i}", position= (-length/2+1000, 150\*i))  
 )  
 placement\_specs.append(  
 i3.Place(f"gc{i}\_1", position=(-20-io.x, 150 \* i))  
 )  
 placement\_specs.append(  
 i3.Place(f"gc{i}\_2", position=(2000+io.x-20, 150\* i-30),angle = 180)  
 )  
 optical\_spec.extend([  
 i3.ConnectManhattan(f"gc{i}\_1:wg",f"PS{i}:PS1\_opt\_in"),  
 i3.ConnectManhattan(f"gc{i}\_2:wg", f"PS{i}:PS{i+1}\_opt\_out")  
 ])  
 for m in range(j):  
 electrical\_spec.extend([  
 i3.ConnectElectrical(f"PS{i}:PS{m+1}\_s\_elec\_n1",f"bp:elec\_BP\_1\_{i\*2+1}", trace\_template = wire\_template1,  
 control\_points = [  
 i3.H(150\*i+size\_info.south - 5), i3.V(50- 15\*i), i3.H(-250-15\*i)  
 ], start\_angle=180, end\_angle=90,  
 ),  
 i3.ConnectElectrical(f"PS{i}:PS{m + 1}\_s\_elec\_n2", f"bp:elec\_BP\_1\_{i \* 2 + 1}",  
 trace\_template=wire\_template1,  
 control\_points=[  
 i3.H(150 \* i + size\_info.south - 5), i3.V(50 - 15 \* i),  
 i3.H(-250 - 15 \* i)  
 ], start\_angle=0, end\_angle=90,  
 ),  
 i3.ConnectElectrical(f"PS{i}:PS{m + 1}\_g\_elec\_p1", f"bp:elec\_BP\_1\_{i \* 2 + 2}",  
 trace\_template=wire\_template1,  
 control\_points=[  
 i3.H(150 \* i + size\_info.north + 5), i3.V(1800+ 15 \* i),  
 i3.H(-85 - 10 \* i)  
 ], start\_angle=180, end\_angle=90,  
 ),  
 i3.ConnectElectrical(f"PS{i}:PS{m + 1}\_g\_elec\_p2", f"bp:elec\_BP\_1\_{i \* 2 + 2}",  
 trace\_template=wire\_template1,  
 control\_points=[  
 i3.H(150 \* i + size\_info.north + 5), i3.V(1800 + 15 \* i),  
 i3.H(-85 - 10 \* i)  
 ], start\_angle=0, end\_angle=90,  
 ),  
 ])  
  
 return placement\_specs+optical\_spec+electrical\_spec  
  
 class Layout(i3.Circuit.Layout):  
  
 def \_generate\_elements(self, elems):  
 super(PN\_loss\_sweep.Layout,self).\_generate\_elements(elems)  
  
 elems += i3.PolygonText(layer=i3.TECH.PPLAYER.FETCH.CLD,  
 transformation=i3.Rotation(rotation=0),  
 coordinate=(1600, -80),  
 text=self.name,  
 alignment=(i3.TEXT.ALIGN.CENTER, i3.TEXT.ALIGN.BOTTOM),  
 font=i3.TEXT.FONT.DEFAULT,  
 height=20, )  
 return elems

这段代码定义了一个名为 **PN\_loss\_sweep** 的 i3.Circuit 类，用于创建级联PN结相位调制器的损耗测试结构。

**主要结构**

**类定义**：

* 继承自 i3.Circuit，表示这是一个电路设计
* 包含内部 Layout 类用于处理布局细节
* 使用 \_name\_prefix 设置类名前缀

**关键属性**：

* Cascaded\_PN: 定义级联PN结单元
* cascaded\_number: 定义级联数量的列表(默认为[1,2,3])
* gc: 光栅耦合器(默认TE1550)
* PS: PN结相位调制器
* bp: 键合焊盘
* ref: 参考波导

**主要功能**

**多配置测试**：

* 支持同时测试不同级联数量(1/2/3级)的PN结结构
* 每个配置有独立的光输入输出和电连接

**光学部分**：

* 为每个级联配置创建：
* 输入/输出光栅耦合器
* 级联PN结相位调制器
* 参考波导

**电学部分**：

* 自动计算所需键合焊盘数量(基于级联配置数量)
* 为每个PN结提供电气连接路径
* 使用两层金属布线(M1和M2)

**关键方法**

**\_default\_insts()**：

* 实例化键合焊盘和参考波导
* 为每个级联数量创建：
* 级联PN结相位调制器
* 输入/输出光栅耦合器

**\_default\_specs()**：

* 定义组件放置位置(垂直排列不同配置)
* 创建光学连接(光栅耦合器到PN结)
* 建立电气连接路径，包括：
* N型电极连接
* P型电极连接
* 多层金属布线控制点

**Layout.\_generate\_elements()**：

* 添加设计名称文本标签
* 使用FETCH.CLD层进行标注

该代码运行结果如下：

图示

AI 生成的内容可能不正确。

图3-12-2 Phase Shifter 损耗提取线路设计

## 3.13 MZM DoE程序设计

MZM DoE设计主要用于提取MZM调制效率、带宽等参数。

其提取线路设计代码如下：

class MZM\_modulator\_test(i3.Circuit):  
 \_name\_prefix = "\_modulator\_test"  
  
 gc = i3.ChildCellProperty(doc="used\_grating\_coupler")  
 MZM = i3.ChildCellProperty(doc="used\_MZM\_unit")  
 bp = i3.ChildCellProperty(doc="Bondpad Frame")  
 num\_pad = i3.IntProperty(default=3, doc="the number of pads ")  
  
 def \_default\_bp(self):  
 return Bondpad\_Frame.PadFrame(no\_pads\_per\_row=self.num\_pad)  
  
 def \_default\_name(self):  
 return f"Test\_for\_{self.MZM.name}"  
  
 def \_default\_MZM(self):  
 return pdk.Fixed\_MZM\_TE\_GS\_1550()  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_insts(self):  
 insts = {  
 "gc1": self.gc,  
 "gc2": self.gc,  
 "gc3": self.gc,  
 "gc4": self.gc,  
 "mzm": self.MZM,  
 "bp": self.bp,  
 }  
 return insts  
  
 def \_default\_specs(self):  
 mzm\_size = self.MZM.Layout().size\_info()  
 length = mzm\_size.east - mzm\_size.west  
 width = mzm\_size.north - mzm\_size.south  
  
 placement\_spec = [  
 i3.Place("mzm", position=(0, 0)),  
 i3.Place("bp", position=(length / 2, -width / 2 - 100)),  
 i3.Place("gc1", position=(length + 0, mzm\_size.north + 20), angle=-90),  
 i3.Place("gc2", position=(length + 0, mzm\_size.south - 20), angle=90),  
 i3.Place("gc3", position=(length + 50, mzm\_size.north + 20), angle=-90),  
 i3.Place("gc4", position=(length + 50, mzm\_size.south - 20), angle=90),  
 ]  
 optical\_spec = [  
 i3.ConnectManhattan("mzm:opt\_in", "gc1:wg"),  
 i3.ConnectManhattan("mzm:opt\_out", "gc2:wg"),  
 i3.ConnectManhattan("gc3:wg", "gc4:wg"),  
 ]  
 if self.num\_pad == 2:  
 electrical\_spec = [  
 i3.ConnectElectrical("bp:elec\_BP\_1\_1", "mzm:elec\_mh1", start\_angle=90, end\_angle=-90,  
 control\_points=[i3.H(-20, relative\_to="mzm@S")],  
 ),  
 i3.ConnectElectrical("bp:elec\_BP\_1\_2", "mzm:elec\_mh2", start\_angle=90, end\_angle=-90,  
 control\_points=[i3.H(-30, relative\_to="mzm@S")],  
 ),  
 ]  
 else:  
 electrical\_spec = [  
 i3.ConnectElectrical("bp:elec\_BP\_1\_1", "mzm:bias\_p", start\_angle=90, end\_angle=-90,  
 control\_points=[i3.H(-20, relative\_to="mzm@S")],  
 ),  
 i3.ConnectElectrical("bp:elec\_BP\_1\_2", "mzm:elec\_mh1", start\_angle=90, end\_angle=-90,  
 control\_points=[i3.H(-20, relative\_to="mzm@S")],  
 ),  
 i3.ConnectElectrical("bp:elec\_BP\_1\_3", "mzm:elec\_mh2", start\_angle=90, end\_angle=-90,  
 control\_points=[i3.H(-30, relative\_to="mzm@S")],  
 ),  
 ]  
  
 return placement\_spec + optical\_spec + electrical\_spec  
  
 class Layout(i3.Circuit.Layout):  
  
 def \_generate\_elements(self, elems):  
 mzm\_size = self.MZM.size\_info()  
 length = mzm\_size.east - mzm\_size.west  
 elems += i3.PolygonText(layer=i3.TECH.PPLAYER.FETCH.CLD,  
 transformation=i3.Rotation(rotation=90),  
 coordinate=(0, -length - 100),  
 text=self.name,  
 alignment=(i3.TEXT.ALIGN.CENTER, i3.TEXT.ALIGN.BOTTOM),  
 font=i3.TEXT.FONT.DEFAULT,  
 height=20, )  
 return elems

这段代码定义了一个用于测试马赫-曾德尔调制器(MZM)的电路模块。代码模块功能如下：

1. 类定义与基本属性

class MZM\_modulator\_test(i3.Circuit):

\_name\_prefix = "\_modulator\_test"

定义了一个名为 MZM\_modulator\_test 的类，继承自 i3.Circuit

设置了名称前缀为 "\_modulator\_test"

**2. 属性定义**

gc = i3.ChildCellProperty(doc="used\_grating\_coupler")

MZM = i3.ChildCellProperty(doc="used\_MZM\_unit")

bp = i3.ChildCellProperty(doc="Bondpad Frame")

num\_pad = i3.IntProperty(default=3, doc="the number of pads ")

* 定义了4个属性：
* **gc**: 光栅耦合器(grating coupler)子单元
* **MZM**: 马赫-曾德尔调制器子单元
* **bp**: 焊盘框架(Bondpad Frame)
* **num\_pad**: 焊盘数量，默认为3

**3. 默认值设置方法**

def \_default\_bp(self):

return Bondpad\_Frame.PadFrame(no\_pads\_per\_row=self.num\_pad)

def \_default\_name(self):

return f"Test\_for\_{self.MZM.name}"

def \_default\_MZM(self):

return pdk.Fixed\_MZM\_TE\_1550()

def \_default\_gc(self):

return pdk.Fixed\_GC\_TE\_1550()

* 设置了各属性的默认值：
  + **bp**: 使用指定焊盘数量的焊盘框架
  + **name**: 基于MZM名称生成测试模块名称
  + **MZM**: 使用PDK中固定的TE偏振1550nm MZM
  + **gc**: 使用PDK中固定的TE偏振1550nm光栅耦合器

**4. 实例定义**

def \_default\_insts(self):

insts = {

"gc1": self.gc,

"gc2": self.gc,

"gc3": self.gc,

"gc4": self.gc,

"mzm": self.MZM,

"bp": self.bp,

}

return insts

* 定义了6个子实例：
* 4个光栅耦合器(gc1-gc4)
* 1个MZM调制器(mzm)
* 1个焊盘框架(bp)

**5. 布局规范**

def \_default\_specs(self):

mzm\_size = self.MZM.Layout().size\_info()

length = mzm\_size.east - mzm\_size.west

width = mzm\_size.north - mzm\_size.south

placement\_spec = [

i3.Place("mzm", position=(0, 0)),

i3.Place("bp", position=(length / 2, -width / 2 - 100)),

i3.Place("gc1", position=(length + 0, mzm\_size.north + 20), angle=-90),

i3.Place("gc2", position=(length + 0, mzm\_size.south - 20), angle=90),

i3.Place("gc3", position=(length + 50, mzm\_size.north + 20), angle=-90),

i3.Place("gc4", position=(length + 50, mzm\_size.south - 20), angle=90),

]

...

* 计算MZM的尺寸信息
* 定义各实例的放置位置和角度：
  + MZM放置在原点
  + 焊盘框架放置在MZM下方
  + 光栅耦合器放置在MZM右侧

**6. 连接规范**

optical\_spec = [

i3.ConnectManhattan("mzm:opt\_in", "gc1:wg"),

i3.ConnectManhattan("mzm:opt\_out", "gc2:wg"),

i3.ConnectManhattan("gc3:wg", "gc4:wg"),

]

* 定义光路连接：
* MZM的输入端口连接到gc1
* MZM的输出端口连接到gc2
* gc3和gc4相互连接(可能用于参考或测试)

if self.num\_pad == 2:

electrical\_spec = [...]

else:

electrical\_spec = [...]

* 根据焊盘数量定义不同的电路连接：
  + 2焊盘时：连接两个金属焊盘
  + 3焊盘时：连接偏置和两个金属焊盘

**7. Layout类**

class Layout(i3.Circuit.Layout):

def \_generate\_elements(self, elems):

mzm\_size = self.MZM.size\_info()

length = mzm\_size.east - mzm\_size.west

elems += i3.PolygonText(...)

return elems

定义了布局生成方法

添加了包含模块名称的文本标注

这个模块是一个完整的MZM测试结构，包含：

1. 一个MZM调制器核心
2. 四个光栅耦合器(两个用于MZM输入输出，两个相互连接)
3. 一个焊盘框架(2或3个焊盘)
4. 所有必要的光学和电学连接
5. 自动布局和标注功能

该模块可以方便地用于测试不同MZM设计的性能，通过参数化设计支持不同焊盘数量的配置。代码运行结果如下：

图表, 瀑布图

AI 生成的内容可能不正确。

图3-13-1 MZM 调制效率与带宽提取线路设计

## 3.14 RM DoE程序设计

RM DoE设计主要同样是用于提取RM调制效率、带宽等参数。

其提取线路设计代码如下：

class AD\_ring\_mod\_test(i3.Circuit):  
 \_name\_prefix = "Add\_drop\_ring\_modulator\_test"  
  
 gc = i3.ChildCellProperty(doc = "used grating coupler")  
 rm = i3.ChildCellProperty(doc = "used ring modulator")  
 bp = i3.ChildCellProperty(doc = "used bondpad frame")  
  
 def \_default\_name(self):  
 return f"test\_for\_{self.rm.name}"  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
  
 def \_default\_rm(self):  
 return pdk.Fixed\_RM\_AD\_TE\_1550()  
  
 def \_default\_bp(self):  
 return PadFrame(no\_pads\_per\_row = 2, name = self.name + "\_bp")  
  
 def \_default\_insts(self):  
 insts = {  
 "gc1": self.gc,  
 "gc2": self.gc,  
 "gc5": self.gc,  
 "gc6": self.gc,  
 "rm": self.rm,  
 "bp": self.bp,  
  
 "gc3": self.gc,  
 "gc4": self.gc,  
  
 }  
 return insts  
  
 def \_default\_specs(self):  
 placement\_spec = []  
 optical\_spec = []  
 electrical\_spec = []  
  
 placement\_spec.extend([  
 i3.Place("gc1",(0,0)),  
 i3.Place("gc2", (900, 0),angle=180),  
 i3.Place("gc3", (0, -80)),  
 i3.Place("gc4", (900, -80), angle=180),  
 i3.Place("gc5", (750, 40)),  
 i3.Place("gc6", (900, 80), angle=180),  
  
 i3.Place("rm", (350,8.155)),  
 i3.Place("bp", (200, -52.5)),  
 ])  
 optical\_spec.extend([  
 i3.ConnectManhattan("gc1:wg","rm:opt\_in"),  
 i3.ConnectManhattan("gc2:wg", "rm:opt\_out"),  
 i3.ConnectManhattan("gc3:wg", "gc4:wg"),  
 i3.ConnectManhattan("gc5:wg", "rm:opt\_in\_2"),  
 i3.ConnectManhattan("gc6:wg", "rm:opt\_out\_2"),  
 ])  
  
 wire\_template1 = i3.ElectricalWireTemplate()  
 wire\_template2 = i3.ElectricalWireTemplate()  
 wire\_template1.Layout(width=5.0, layer=i3.TECH.PPLAYER.M1.DRW)  
 wire\_template2.Layout(width=5.0, layer=i3.TECH.PPLAYER.M2.DRW)  
  
 electrical\_spec.extend([  
 i3.ConnectElectrical("rm:elec\_mh1","bp:elec\_BP\_1\_1",trace\_template = wire\_template1,  
 start\_angle=180, end\_angle=90,  
 ),  
 i3.ConnectElectrical("rm:elec\_mh2", "bp:elec\_BP\_1\_2", trace\_template=wire\_template1,  
 start\_angle=180, end\_angle=90,  
 )  
 ])  
 return placement\_spec+optical\_spec+electrical\_spec  
 class Layout(i3.Circuit.Layout):  
  
 def \_default\_bp(self):  
 temp = self.cell.bp.get\_default\_view(self)  
 temp.set(pitch\_x = 100)  
 return temp  
  
 def \_generate\_elements(self, elems):  
 elems += i3.PolygonText(i3.TECH.PPLAYER.FETCH.CLD,  
 transformation=i3.Rotation(rotation=0),  
 coordinate=(600, -50),  
 text=self.name,  
 alignment=(i3.TEXT.ALIGN.CENTER, i3.TEXT.ALIGN.BOTTOM),  
 font=i3.TEXT.FONT.DEFAULT,  
 height=20, )  
 return elems

该程序主要内容如下：

1. 模块概述

AD\_ring\_mod\_test是一个用于测试"Add-Drop"型环形调制器(Ring Modulator)的电路模块。该模块提供了完整的测试环境，包括光路连接、电学连接和必要的测试结构，便于对环形调制器进行性能评估和验证。

2. 模块结构与功能

2.1 类定义与基本属性

class AD\_ring\_mod\_test(i3.Circuit):

\_name\_prefix = "Add\_drop\_ring\_modulator\_test"

类名：AD\_ring\_mod\_test

继承自：i3.Circuit (IPKISS电路基类)

名称前缀："Add\_drop\_ring\_modulator\_test"

2.2 属性定义

gc = i3.ChildCellProperty(doc = "used grating coupler")

rm = i3.ChildCellProperty(doc = "used ring modulator")

bp = i3.ChildCellProperty(doc = "used bondpad frame")

gc: 光栅耦合器(grating coupler)子单元属性

rm: 环形调制器(ring modulator)子单元属性

bp: 焊盘框架(bondpad frame)子单元属性

2.3 默认值设置方法

def \_default\_name(self):

return f"test\_for\_{self.rm.name}"

def \_default\_gc(self):

return pdk.Fixed\_GC\_TE\_1550()

def \_default\_rm(self):

return pdk.Fixed\_RM\_AD\_TE\_1550()

def \_default\_bp(self):

return PadFrame(no\_pads\_per\_row = 2, name = self.name + "\_bp")

名称默认值：基于环形调制器名称生成测试模块名称

光栅耦合器默认值：使用PDK中固定的TE偏振1550nm光栅耦合器

环形调制器默认值：使用PDK中固定的TE偏振1550nm Add-Drop型环形调制器

焊盘框架默认值：创建包含2个焊盘的焊盘框架，名称附加"\_bp"后缀

2.4 实例定义

def \_default\_insts(self):

insts = {

"gc1": self.gc,

"gc2": self.gc,

"gc5": self.gc,

"gc6": self.gc,

"rm": self.rm,

"bp": self.bp,

"gc3": self.gc,

"gc4": self.gc,

}

return insts

定义了6个光栅耦合器实例、1个环形调制器实例和1个焊盘框架实例：

gc1, gc2: 主光路输入输出

gc5, gc6: Add-Drop端口输入输出

gc3, gc4: 参考光路(相互连接)

rm: 环形调制器核心

bp: 电学测试焊盘框架

2.5 布局与连接规范

2.5.1 放置规范

7/

精确放置所有光栅耦合器、环形调制器和焊盘框架

主光路光栅耦合器(gc1, gc2)水平放置

Add-Drop端口光栅耦合器(gc5, gc6)位于上方

参考光路光栅耦合器(gc3, gc4)位于下方

环形调制器居中放置

焊盘框架位于左侧下方

2.5.2 光学连接规范

optical\_spec.extend([

i3.ConnectManhattan("gc1:wg","rm:opt\_in"),

i3.ConnectManhattan("gc2:wg", "rm:opt\_out"),

i3.ConnectManhattan("gc3:wg", "gc4:wg"),

i3.ConnectManhattan("gc5:wg", "rm:opt\_in\_2"),

i3.ConnectManhattan("gc6:wg", "rm:opt\_out\_2"),

])

主光路：gc1→环形调制器输入→环形调制器输出→gc2

参考光路：gc3↔gc4(直通连接)

Add-Drop光路：gc5→环形调制器第二输入→环形调制器第二输出→gc6

2.5.3 电学连接规范

wire\_template1 = i3.ElectricalWireTemplate()

wire\_template2 = i3.ElectricalWireTemplate()

wire\_template1.Layout(width=5.0, layer=i3.TECH.PPLAYER.M1.DRW)

wire\_template2.Layout(width=5.0, layer=i3.TECH.PPLAYER.M2.DRW)

electrical\_spec.extend([

i3.ConnectElectrical("rm:elec\_mh1","bp:elec\_BP\_1\_1",trace\_template = wire\_template1,

start\_angle=180, end\_angle=90,

),

i3.ConnectElectrical("rm:elec\_mh2", "bp:elec\_BP\_1\_2", trace\_template=wire\_template1,

start\_angle=180, end\_angle=90,

)

])

定义两种金属层的电学走线模板(宽度5.0μm)

连接环形调制器的两个金属焊盘到焊盘框架

指定连接起始和结束角度(180°→90°)

2.6 Layout类

class Layout(i3.Circuit.Layout):

def \_default\_bp(self):

temp = self.cell.bp.get\_default\_view(self)

temp.set(pitch\_x = 100)

return temp

def \_generate\_elements(self, elems):

elems += i3.PolygonText(i3.TECH.PPLAYER.FETCH.CLD,

transformation=i3.Rotation(rotation=0),

coordinate=(600, -50),

text=self.name,

alignment=(i3.TEXT.ALIGN.CENTER, i3.TEXT.ALIGN.BOTTOM),

font=i3.TEXT.FONT.DEFAULT,

height=20, )

return elems

焊盘框架布局：设置焊盘x方向间距为100μm

标注生成：在指定位置(600, -50)添加模块名称文本标注

使用FETCH.CLD层

字体高度20μm

居中对齐

3. 设计特点

完整测试环境：提供主光路、Add-Drop光路和参考光路，全面测试环形调制器性能

参数化设计：通过属性定义关键组件，便于替换不同实现

精确布局：所有组件位置和连接关系明确定义

电光分离：光学和电学连接分别定义，清晰明了

自动标注：自动生成包含模块名称的标注，便于识别

该程序运行结果如下：

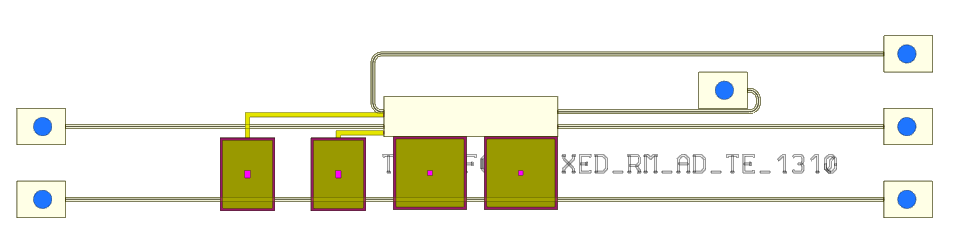


图3-14-1 RM 调制效率与带宽提取线路设计

## 3.15 PhotoDetector DoE程序设计

PhotoDetector DoE设计主要同样是用于提取相应度、带宽以及暗电流等参数。

其提取线路设计代码如下：

class PD\_tests(i3.Circuit):  
 \_name\_prefix = "PD\_test"  
  
 gc = i3.ChildCellProperty(doc = "used grating coupler")  
 pd = i3.ChildCellProperty(doc = "used PD")  
  
 def \_default\_name(self):  
 return f"Test\_for\_{self.pd.name}"  
  
 def \_default\_gc(self):  
 return pdk.Fixed\_GC\_TE\_1550()  
 def \_default\_pd(self):  
 return pdk.Fixed\_GPD\_1550()  
  
 def \_default\_insts(self):  
 optical\_ports = self.pd.Layout().ports.optical\_ports  
 number\_of\_ports = len(optical\_ports)  
 insts = {"pd": self.pd}  
 for i in range(number\_of\_ports):  
 insts[f"gc{i}"] = self.gc  
 return insts  
 def \_default\_specs(self):  
 placement\_spec = [i3.Place("pd",position=(0,0))]  
 optical\_ports = self.pd.Layout().ports.optical\_ports  
 number\_of\_ports = len(optical\_ports)  
 optical\_spec = []  
 for i in range(number\_of\_ports):  
 optical\_ports\_position = optical\_ports[i].position  
 placement\_spec.extend([  
 i3.Place(f"gc{i}", position=(optical\_ports\_position.x - 300, optical\_ports\_position.y))  
 ])  
 optical\_spec.extend(([  
 i3.ConnectManhattan(f"gc{i}:wg",f"pd:{optical\_ports[i].name}")  
 ]))  
  
 return placement\_spec+optical\_spec

1. 模块概述

PD\_tests是一个用于测试光电探测器(Photodetector, PD)的电路模块。该模块提供了灵活的光路连接结构，能够自动适配不同端口数量的光电探测器，便于进行性能测试和验证。

2. 模块结构与功能

2.1 类定义与基本属性

class PD\_tests(i3.Circuit):

\_name\_prefix = "PD\_test"

类名：PD\_tests

继承自：i3.Circuit (IPKISS电路基类)

名称前缀："PD\_test"

2.2 属性定义

gc = i3.ChildCellProperty(doc = "used grating coupler")

pd = i3.ChildCellProperty(doc = "used PD")

gc：光栅耦合器(grating coupler)子单元属性

pd：光电探测器(photodetector)子单元属性

2.3 默认值设置方法

def \_default\_name(self):

return f"Test\_for\_{self.pd.name}"

def \_default\_gc(self):

return pdk.Fixed\_GC\_TE\_1550()

def \_default\_pd(self):

return pdk.Fixed\_GPD\_1550()

名称默认值：基于光电探测器名称生成测试模块名称

光栅耦合器默认值：使用PDK中固定的TE偏振1550nm光栅耦合器

光电探测器默认值：使用PDK中固定的1550nm光电探测器

2.4 实例定义

def \_default\_insts(self):

optical\_ports = self.pd.Layout().ports.optical\_ports

number\_of\_ports = len(optical\_ports)

insts = {"pd": self.pd}

for i in range(number\_of\_ports):

insts[f"gc{i}"] = self.gc

return insts

动态实例创建：

* 自动检测光电探测器的光学端口数量
* 为每个光学端口创建对应的光栅耦合器实例
* 实例命名规则：gc0, gc1, ..., gcN (N=端口数-1)
* 始终包含一个光电探测器实例pd

2.5 布局与连接规范

2.5.1 放置规范

def \_default\_specs(self):

placement\_spec = [i3.Place("pd",position=(0,0))]

optical\_ports = self.pd.Layout().ports.optical\_ports

number\_of\_ports = len(optical\_ports)

for i in range(number\_of\_ports):

optical\_ports\_position = optical\_ports[i].position

placement\_spec.extend([

i3.Place(f"gc{i}", position=(optical\_ports\_position.x - 300, optical\_ports\_position.y))

])

**光电探测器放置**：固定在坐标原点(0,0)

**光栅耦合器放置**：

根据每个光学端口的位置动态确定

统一向左偏移300μm (x-300)

保持y坐标与对应端口一致

**2.5.2 光学连接规范**

optical\_spec = []

for i in range(number\_of\_ports):

optical\_spec.extend(([

i3.ConnectManhattan(f"gc{i}:wg",f"pd:{optical\_ports[i].name}")

]))

**自动连接**：每个光栅耦合器连接到对应的光电探测器光学端口

**连接方式**：使用曼哈顿式(直角)连接

**端口命名**：动态获取光电探测器端口名称

**2.6 输出规范**

return placement\_spec+optical\_spec

合并放置规范和光学连接规范

返回完整的电路规范列表

3. 设计特点

**自动适配性**：动态检测光电探测器端口数量，自动创建对应数量的光栅耦合器和连接

**标准化接口**：兼容不同端口数的光电探测器，提高模块复用性

**智能布局**：基于端口位置自动计算光栅耦合器位置，确保连接最短化

**命名规范**：系统化的实例命名规则(gc0, gc1,...)便于识别和管理

**工艺兼容**：使用PDK标准组件，确保工艺兼容性

该代码运行结果如下：

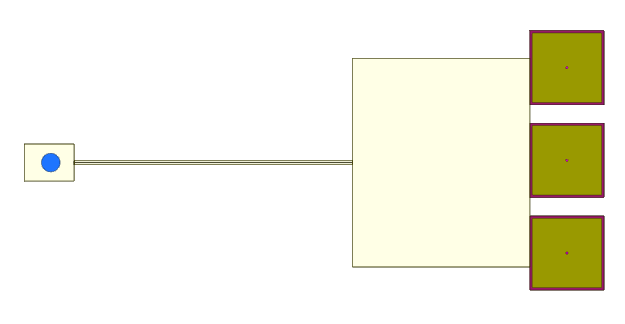


图3-15-1 PhotoDetector 响应度、带宽以及暗电流提取线路设计

# 4 紧凑模型程序设计

在光子集成电路设计领域，IPKISS 紧凑模型发挥着关键作用。IPKISS 中的紧凑模型以CompactModel作为基类，其本质是通过 S 矩阵或一组微分方程，对器件行为进行描述。这种描述方式能够被电路模拟器 CAPHE 所解读，进而实现电路仿真。​

以 1x2 MMI（多模干涉器）这一典型器件为例，其完整的模型涵盖布局视图 Layout、网表视图 Netlist 以及行为模型视图 CircuitModel。在行为模型层面，需要输入诸如中心波长、端口传输谱、端口发射谱等关键参数。其中，中心波长决定了器件运行的核心波长范围；端口传输谱和发射谱则与紧凑模型中的相关参数紧密关联，例如 “transmission”（传输系数）用于表征器件传输特性随波长的变化情况。紧凑模型除了继承行为模型的参数外，还包含仿真的光学端口，像 “i3.OpticalTerm (name = "in1")”“i3.OpticalTerm (name = "out1")”“i3.OpticalTerm (name = "out2")” 等，这些端口明确了光信号的输入输出路径。同时，散射矩阵也是紧凑模型的重要组成部分，它全面刻画了器件各端口间光信号的传输与反射关系。

接下来，将分节介绍项目第二阶段涉及的所有器件的紧凑模型。

## 4.1 Grating Coupler 紧凑模型程序设计

Grating Coupler 紧凑模型程序如下：

class GratingCoupler(CompactModel):  
 parameters = ["gc\_data", "polarisation"]  
 terms = [  
 OpticalTerm(name="vertical\_io", n\_modes=2),  
 OpticalTerm(name="wg", n\_modes=2),  
 ]  
  
 def calculate\_smatrix(parameters, env, S):  
 if env.temperature == 20:  
 peak\_wavelength = parameters.gc\_data["20"][0]  
 peak\_il = parameters.gc\_data["20"][1]  
 bw\_1db = parameters.gc\_data["20"][2]  
 elif env.temperature == 25:  
 peak\_wavelength = parameters.gc\_data["25"][0]  
 peak\_il = parameters.gc\_data["25"][1]  
 bw\_1db = parameters.gc\_data["25"][2]  
  
 peak\_transmission = 10.0 \*\* (-0.1 \* peak\_il)  
 gc\_factor = (2.0 / bw\_1db) \* np.sqrt(0.1)  
 delta\_wavelength = peak\_wavelength - env.wavelength  
 loss = np.sqrt(  
 peak\_transmission \* (10.0 \*\* (-((gc\_factor \* delta\_wavelength) \*\* 2.0)))  
 )  
 if parameters.polarisation == "TE":  
 # for TE polarisation  
 S["vertical\_io:0", "wg:0"] = S["wg:0", "vertical\_io:0"] = loss  
 elif parameters.polarisation == "TM":  
 # for TM polarisation  
 S["vertical\_io:1", "wg:1"] = S["wg:1", "vertical\_io:1"] = loss  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 if env.temperature == 20:  
 peak\_wavelength = parameters.gc\_data["20"][0]  
 peak\_il = parameters.gc\_data["20"][1]  
 bw\_1db = parameters.gc\_data["20"][2]  
 elif env.temperature == 25:  
 peak\_wavelength = parameters.gc\_data["25"][0]  
 peak\_il = parameters.gc\_data["25"][1]  
 bw\_1db = parameters.gc\_data["25"][2]  
 peak\_transmission = 10.0 \*\* (-0.1 \* peak\_il)  
 gc\_factor = (2.0 / bw\_1db) \* np.sqrt(0.1)  
 delta\_wavelength = peak\_wavelength - env.wavelength  
 loss = np.sqrt(  
 peak\_transmission \* (10.0 \*\* (-((gc\_factor \* delta\_wavelength) \*\* 2.0)))  
 )  
 if parameters.polarisation == "TE":  
 # for TE polarisation  
 output\_signals["vertical\_io:0"] = loss \* input\_signals["wg:0"]  
 output\_signals["wg:0"] = loss \* input\_signals["vertical\_io:0"]  
 elif parameters.polarisation == "TM":  
 # for TM polarisation  
 output\_signals["vertical\_io:1"] = loss \* input\_signals["wg:1"]  
 output\_signals["wg:1"] = loss \* input\_signals["vertical\_io:1"]

Grating Coupler 的紧凑模型程序主要包含以下四个部分：

1、模型参数：

parameters = ["gc\_data", "polarisation"]

模型参数是建模的基础，Grating Coupler的模型参数主要包括测试数据“gc\_data”和“polarisation”，gc\_data包含Grating Coupler的峰值损耗、中心波长以及1dB带宽等参数。而“polarisation”是为了让该模型同时适用于TE型和TM型的光栅耦合器而设立。

2、模型端口

terms = [  
 OpticalTerm(name="vertical\_io", n\_modes=2),  
 OpticalTerm(name="wg", n\_modes=2),  
]

在 IPKISS 紧凑模型中，模型端口是实现光信号交互的关键节点，对整个光子集成电路的信号传输起着决定性作用。上述代码中定义的terms列表包含两个OpticalTerm对象，分别代表不同功能的光学端口。

OpticalTerm(name="vertical\_io", n\_modes=2)定义了名为 “vertical\_io” 的光学端口，其中n\_modes=2表示该端口支持两个模式的光信号传输。“vertical\_io” 端口名称暗示其可能用于垂直方向的光输入输出，在一些需要实现光信号垂直耦合或连接垂直方向光器件的场景中，这个端口能将特定模式的光信号引入或引出光子集成电路，实现芯片与外部光器件在垂直维度上的信号交互。

3、频域模型：

def calculate\_smatrix(parameters, env, S):  
 if env.temperature == 20:  
 peak\_wavelength = parameters.gc\_data["20"][0]  
 peak\_il = parameters.gc\_data["20"][1]  
 bw\_1db = parameters.gc\_data["20"][2]  
 elif env.temperature == 25:  
 peak\_wavelength = parameters.gc\_data["25"][0]  
 peak\_il = parameters.gc\_data["25"][1]  
 bw\_1db = parameters.gc\_data["25"][2]  
  
 peak\_transmission = 10.0 \*\* (-0.1 \* peak\_il)  
 gc\_factor = (2.0 / bw\_1db) \* np.sqrt(0.1)  
delta\_wavelength = peak\_wavelength - env.wavelength  
 loss = np.sqrt(  
 peak\_transmission \* (10.0 \*\* (-((gc\_factor \* delta\_wavelength) \*\* 2.0)))  
 )  
 if parameters.polarisation == "TE":  
 # for TE polarisation  
 S["vertical\_io:0", "wg:0"] = S["wg:0", "vertical\_io:0"] = loss  
 elif parameters.polarisation == "TM":  
 # for TM polarisation  
 S["vertical\_io:1", "wg:1"] = S["wg:1", "vertical\_io:1"] = loss

该代码定义的calculate\_smatrix函数是一个基于 Python 的光栅耦合器频域模型，利用散射矩阵（S 矩阵）来描述光栅耦合器在不同条件下的光学特性。以下是对这个模型的详细介绍：

**温度相关参数读取**：模型首先根据环境温度env.temperature读取光栅耦合器的相关参数，包括峰值波长peak\_wavelength、峰值插入损耗peak\_il和 1dB 带宽bw\_1db。这些参数存储在parameters.gc\_data字典中，不同温度对应不同的参数值，体现了光栅耦合器特性对温度的敏感性。

**关键参数计算**：

根据峰值插入损耗peak\_il计算峰值传输率peak\_transmission，公式为peak\_transmission = 10.0 \*\* (-0.1 \* peak\_il) ，这是基于插入损耗与传输率的对数转换关系，将插入损耗从 dB 单位转换为线性单位的传输率。

计算光栅耦合器因子gc\_factor，公式为gc\_factor = (2.0 / bw\_1db) \* np.sqrt(0.1) ，该因子与 1dB 带宽相关，用于后续计算损耗随波长的变化。

**损耗计算**：计算当前波长env.wavelength与峰值波长peak\_wavelength的差值delta\_wavelength，并通过高斯函数形式的公式计算损耗loss：

loss = np.sqrt(

peak\_transmission \* (10.0 \*\* (-((gc\_factor \* delta\_wavelength) \*\* 2.0)))

)

该公式表明损耗与峰值传输率成正比，且随波长偏移量delta\_wavelength的变化呈高斯分布衰减，体现了光栅耦合器在不同波长下的传输损耗特性。

**散射矩阵赋值**：根据光的偏振态parameters.polarisation（TE 或 TM），将计算得到的损耗loss赋值给散射矩阵S中对应的端口连接项。例如，当偏振态为 TE 时，设置S["vertical\_io:0", "wg:0"]和S["wg:0", "vertical\_io:0"]为损耗值，表示从vertical\_io端口 0 到wg端口 0 以及反向的光传输损耗；TM 偏振态同理。通过这种方式，散射矩阵完整描述了光栅耦合器在不同偏振态下，两个端口之间的光信号传输损耗关系。

4、时域模型

def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 if env.temperature == 20:  
 peak\_wavelength = parameters.gc\_data["20"][0]  
 peak\_il = parameters.gc\_data["20"][1]  
 bw\_1db = parameters.gc\_data["20"][2]  
 elif env.temperature == 25:  
 peak\_wavelength = parameters.gc\_data["25"][0]  
 peak\_il = parameters.gc\_data["25"][1]  
 bw\_1db = parameters.gc\_data["25"][2]  
 peak\_transmission = 10.0 \*\* (-0.1 \* peak\_il)  
 gc\_factor = (2.0 / bw\_1db) \* np.sqrt(0.1)  
 delta\_wavelength = peak\_wavelength - env.wavelength  
 loss = np.sqrt(  
 peak\_transmission \* (10.0 \*\* (-((gc\_factor \* delta\_wavelength) \*\* 2.0)))  
 )  
 if parameters.polarisation == "TE":  
 # for TE polarisation  
 output\_signals["vertical\_io:0"] = loss \* input\_signals["wg:0"]  
 output\_signals["wg:0"] = loss \* input\_signals["vertical\_io:0"]  
 elif parameters.polarisation == "TM":  
 # for TM polarisation  
 output\_signals["vertical\_io:1"] = loss \* input\_signals["wg:1"]  
 output\_signals["wg:1"] = loss \* input\_signals["vertical\_io:1"]

与频域模型不同的是，时域模型还需要有输入信号：output\_signals，再基于端口之间的能量传递关系就可以计算输出信号：

if parameters.polarisation == "TE":  
 # for TE polarisation  
 output\_signals["vertical\_io:0"] = loss \* input\_signals["wg:0"]  
 output\_signals["wg:0"] = loss \* input\_signals["vertical\_io:0"]  
elif parameters.polarisation == "TM":  
 # for TM polarisation  
 output\_signals["vertical\_io:1"] = loss \* input\_signals["wg:1"]  
 output\_signals["wg:1"] = loss \* input\_signals["vertical\_io:1"]

## 4.2波导紧凑模型程序设计

波导的紧凑模型程序如下：

class WaveguideCompactModel(CompactModel):  
 parameters = ["wg\_data", "length"]  
 terms = [OpticalTerm(name="in", n\_modes=2), OpticalTerm(name="out", n\_modes=2)]  
  
 def calculate\_smatrix(parameters, env, S):  
 center\_wavelength = parameters.wg\_data[0]  
 alpha\_dBcm = parameters.wg\_data[1]  
 neff0 = parameters.wg\_data[2]  
 ng0 = parameters.wg\_data[3]  
 omega0 = 2.0 \* pi \* c\_mu / center\_wavelength  
 omega = 2.0 \* pi \* c\_mu / env.wavelength  
 neff = neff0 + (ng0 - neff0) \* ((omega / omega0) - 1.0)  
 phase = (2.0 \* pi \* neff \* parameters.length) / env.wavelength  
 loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.length) / 20.0)  
  
 S["in:0", "out:0"] = S["out:0", "in:0"] = loss \* np.exp(1j \* phase)  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 center\_wavelength = parameters.wg\_data[0]  
 alpha\_dBcm = parameters.wg\_data[1]  
 neff0 = parameters.wg\_data[2]  
 ng0 = parameters.wg\_date[3]  
  
 omega0 = 2.0 \* pi \* c\_mu / center\_wavelength  
 omega = 2.0 \* pi \* c\_mu / env.wavelength  
 neff = neff0 + (ng0 - neff0) \* ((omega / omega0) - 1.0)  
 ng = neff0 + (ng0 - neff0) \* ((2.0 \* omega / omega0) - 1.0)  
 phase = (2.0 \* pi \* neff \* parameters.length) / env.wavelength  
 loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.length) / 20.0)  
 t\_delay = ng \* parameters.length / c\_mu  
 trans = loss \* np.exp(1j \* phase)  
  
 output\_signals["in:0"] = trans \* input\_signals["out:0", t - t\_delay]  
 output\_signals["out:0"] = trans \* input\_signals["in:0", t - t\_delay]

Waveguide 的紧凑模型程序主要包含以下四个部分：

1、模型参数：

parameters = ["wg\_data", "length"]

Waveguide 的模型参数主要包括波导特性数据 “wg\_data” 和波导长度 “length”。其中 wg\_data 包含波导的中心波长、每厘米的衰减系数（单位为 dB）、有效折射率以及群折射率等关键参数，这些参数共同决定了波导的光学传输特性；而 length 则明确了所建模波导的物理长度，是计算传输相位和损耗的重要依据。

2、模型端口

terms = [OpticalTerm (name="in", n\_modes=2), OpticalTerm (name="out", n\_modes=2)]

在 IPKISS 紧凑模型中，模型端口是实现光信号交互的关键节点，对整个光子集成电路的信号传输起着决定性作用。上述代码中定义的 terms 列表包含两个 OpticalTerm 对象，分别代表波导的输入和输出光学端口。  
OpticalTerm (name="in", n\_modes=2) 定义了名为 “in” 的光学端口，n\_modes=2 表示该端口支持两个模式的光信号传输，主要用于接收输入到波导的光信号；OpticalTerm (name="out", n\_modes=2) 定义了名为 “out” 的光学端口，同样支持两个模式的光信号传输，用于输出经过波导传输后的光信号。这两个端口共同构成了光信号在波导中传输的路径起点和终点。

3、频域模型：

def calculate\_smatrix (parameters, env, S):  
center\_wavelength = parameters.wg\_data [0]  
alpha\_dBcm = parameters.wg\_data [1]  
neff0 = parameters.wg\_data [2]  
ng0 = parameters.wg\_data [3]  
omega0 = 2.0 \* pi \* c\_mu /center\_wavelength  
omega = 2.0 \* pi \* c\_mu /env.wavelength  
neff = neff0 + (ng0 - neff0) \* ((omega /omega0) - 1.0)  
phase = (2.0 \* pi \* neff \* parameters.length) /env.wavelength  
loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.length) / 20.0)

S["in:0", "out:0"] = S["out:0", "in:0"] = loss \* np.exp(1j \* phase)

代码定义的 calculate\_smatrix 函数是一个基于 Python 的波导频域模型，利用散射矩阵（S 矩阵）来描述波导在不同条件下的光学特性。以下是对这个模型的详细介绍：

参数读取与计算：  
从 parameters.wg\_data 中读取中心波长 center\_wavelength、每厘米衰减系数 alpha\_dBcm、有效折射率 neff 和群折射率 ng。  
根据中心波长计算中心角频率 omega0，公式为 omega0 = 2.0 \* pi \* c\_mu /center\_wavelength；根据环境中的当前波长计算角频率 omega，公式为 omega = 2.0 \* pi \* c\_mu /env.wavelength。  
计算有效折射率 neff，通过公式 neff = neff0 + (ng0 - neff0) \* ((omega /omega0) - 1.0) 实现，该式考虑了角频率变化对有效折射率的影响。

相位与损耗计算：  
计算光信号在波导中传输的相位 phase，公式为 phase = (2.0 \* pi \* neff \* parameters.length) /env.wavelength，其与有效折射率、波导长度以及当前波长相关，反映了光信号传输后的相位变化。  
计算损耗系数 loss，公式为 loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.length) / 20.0)，将每厘米的衰减系数转换为考虑波导长度的线性损耗系数。

散射矩阵赋值：将计算得到的损耗与相位结合，得到复数传输系数 loss \* np.exp (1j \* phase)，并将其赋值给散射矩阵 S 中 “in:0” 到 “out:0” 以及 “out:0” 到 “in:0” 的项，表明光信号在波导的这两个端口之间双向传输时，既存在能量损耗，也会产生相位变化。

1. 时域模型

def calculate\_signals (parameters, env, output\_signals, y, t, input\_signals):  
center\_wavelength = parameters.wg\_data [0]  
alpha\_dBcm = parameters.wg\_data [1]  
neff0 = parameters.wg\_data [2]  
ng0 = parameters.wg\_data [3] # 修正了原代码中的拼写错误 wg\_date 为 wg\_data

omega0 = 2.0 \* pi \* c\_mu / center\_wavelength  
omega = 2.0 \* pi \* c\_mu / env.wavelength  
neff = neff0 + (ng0 - neff0) \* ((omega / omega0) - 1.0)  
ng = neff0 + (ng0 - neff0) \* ((2.0 \* omega / omega0) - 1.0)  
phase = (2.0 \* pi \* neff \* parameters.length) / env.wavelength  
loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.length) / 20.0)  
t\_delay = ng \* parameters.length / c\_mu  
trans = loss \* np.exp(1j \* phase)

output\_signals["in:0"] = trans \* input\_signals["out:0", t - t\_delay]  
output\_signals["out:0"] = trans \* input\_signals["in:0", t - t\_delay]

与频域模型不同的是，时域模型需要输入信号 input\_signals，再基于端口之间的能量传递关系计算输出信号 output\_signals，具体过程如下：

额外参数计算：在频域模型参数计算的基础上，进一步计算群折射率 ng，公式为 ng = neff0 + (ng0 - neff0) \* ((2.0 \* omega /omega0) - 1.0)；计算光信号在波导中的传输延迟 t\_delay，公式为 t\_delay = ng \* parameters.length/c\_mu，该延迟与群折射率、波导长度以及光速相关。  
传输系数计算：得到包含损耗和相位信息的传输系数 trans = loss \* np.exp (1j \* phase)。

输出信号计算：  
对于 “in:0” 端口的输出信号，它等于传输系数 trans 与 “out:0” 端口在 t - t\_delay 时刻的输入信号相乘，即 output\_signals ["in:0"] = trans \* input\_signals ["out:0", t - t\_delay]，体现了光信号从 “out:0” 端口传输到 “in:0” 端口时，经过一定延迟后的信号变化。  
同理，“out:0” 端口的输出信号为传输系数 trans 与 “in:0” 端口在 t - t\_delay 时刻的输入信号相乘，即 output\_signals ["out:0"] = trans \* input\_signals ["in:0", t - t\_delay]，描述了光信号从 “in:0” 端口传输到 “out:0” 端口时的延迟和信号变化情况。  
通过这种方式，时域模型完整呈现了波导在时域下输入信号与输出信号之间的关系，包括信号的幅度衰减、相位变化和传输延迟。

## 4.3 Crossing紧凑模型程序设计

Crossing的紧凑模型程序如下：

class CrossingCompactModel(CompactModel):  
 parameters = [  
 "insertion\_loss",  
 "cross\_talk",  
 ] # Note that the unit is dB and it is a positive value.  
 terms = [  
 OpticalTerm(name="WEST", n\_modes=2),  
 OpticalTerm(name="NORTH", n\_modes=2),  
 OpticalTerm(name="EAST", n\_modes=2),  
 OpticalTerm(name="SOUTH", n\_modes=2),  
 ]  
  
 def calculate\_smatrix(parameters, env, S):  
 loss = 10.0 \*\* (-0.05 \* parameters.insertion\_loss)  
 ct = 10.0 \*\* (-0.05 \* parameters.cross\_talk)  
 S["WEST:0", "EAST:0"] = S["EAST:0", "WEST:0"] = loss  
 S["NORTH:0", "SOUTH:0"] = S["SOUTH:0", "NORTH:0"] = loss  
 S["WEST:0", "NORTH:0"] = S["NORTH:0", "WEST:0"] = S["WEST:0", "SOUTH:0"] = S[  
 "SOUTH:0", "WEST:0"  
 ] = ct  
 S["EAST:0", "NORTH:0"] = S["NORTH:0", "EAST:0"] = S["EAST:0", "SOUTH:0"] = S[  
 "SOUTH:0", "EAST:0"  
 ] = ct  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 loss = 10.0 \*\* (-0.05 \* parameters.insertion\_loss)  
 ct = 10.0 \*\* (-0.05 \* parameters.cross\_talk)  
 output\_signals["WEST:0"] = loss \* input\_signals["EAST:0"] + ct \* (  
 input\_signals["NORTH:0"] + input\_signals["SOUTH:0"]  
 )  
 output\_signals["NORTH:0"] = loss \* input\_signals["SOUTH:0"] + ct \* (  
 input\_signals["WEST:0"] + input\_signals["EAST:0"]  
 )  
 output\_signals["EAST:0"] = loss \* input\_signals["WEST:0"] + ct \* (  
 input\_signals["NORTH:0"] + input\_signals["SOUTH:0"]  
 )  
 output\_signals["SOUTH:0"] = loss \* input\_signals["NORTH:0"] + ct \* (  
 input\_signals["WEST:0"] + input\_signals["EAST:0"]  
 )

Crossing 的紧凑模型程序主要包含以下四个部分：

1、模型参数：

parameters = [  
"insertion\_loss",  
"cross\_talk",  
]

Crossing 的模型参数主要包括插入损耗 "insertion\_loss" 和串扰 "cross\_talk"（注意：其单位为 dB，且为正值）。插入损耗描述了光信号在交叉结构中传输时的能量损失，而串扰则表征了不同传输路径之间的信号干扰程度。

2、模型端口

terms = [  
OpticalTerm (name="WEST", n\_modes=2),  
OpticalTerm (name="NORTH", n\_modes=2),  
OpticalTerm (name="EAST", n\_modes=2),  
OpticalTerm (name="SOUTH", n\_modes=2),  
]

在 IPKISS 紧凑模型中，模型端口是实现光信号交互的关键节点，对整个光子集成电路的信号传输起着决定性作用。上述代码中定义的 terms 列表包含四个 OpticalTerm 对象，分别代表不同方向的光学端口。  
OpticalTerm (name="WEST", n\_modes=2) 定义了名为 "WEST" 的光学端口，n\_modes=2 表示该端口支持两个模式的光信号传输。同理，"NORTH"、"EAST"、"SOUTH" 端口分别对应北、东、南方向，这些端口用于在不同方向上实现光信号的输入输出，使交叉结构能够实现光信号在多个方向上的传输与交互。

1. 频域模型：

def calculate\_smatrix (parameters, env, S):  
loss = 10.0 (-0.05 \* parameters.insertion\_loss)  
ct = 10.0 (-0.05 \* parameters.cross\_talk)  
S["WEST:0", "EAST:0"] = S["EAST:0", "WEST:0"] = loss  
S["NORTH:0", "SOUTH:0"] = S["SOUTH:0", "NORTH:0"] = loss  
S["WEST:0", "NORTH:0"] = S["NORTH:0", "WEST:0"] = S["WEST:0", "SOUTH:0"] = S[  
"SOUTH:0", "WEST:0"  
] = ct  
S["EAST:0", "NORTH:0"] = S["NORTH:0", "EAST:0"] = S["EAST:0", "SOUTH:0"] = S[  
"SOUTH:0", "EAST:0"  
] = ct

该代码定义的 calculate\_smatrix 函数是一个基于 Python 的交叉结构频域模型，利用散射矩阵（S 矩阵）来描述交叉结构在不同条件下的光学特性。以下是对这个模型的详细介绍：

关键参数计算：  
根据插入损耗 parameters.insertion\_loss 计算损耗系数 loss，公式为 loss = 10.0 (-0.05 \* parameters.insertion\_loss)，这是将插入损耗从 dB 单位转换为线性单位的损耗系数。根据串扰 parameters.cross\_talk 计算串扰系数 ct，公式为 ct = 10.0 (-0.05 \* parameters.cross\_talk)，实现了串扰从 dB 单位到线性单位的转换。

散射矩阵赋值：散射矩阵 S 中各项值描述了不同端口之间的光信号传输关系。对于直通路径，如 WEST 到 EAST 以及 EAST 到 WEST，还有 NORTH 到 SOUTH 以及 SOUTH 到 NORTH，其传输系数均为 loss，代表了光信号在直通方向上的传输损耗情况。  
对于交叉路径及其他非直通路径，如 WEST 与 NORTH、WEST 与 SOUTH、EAST 与 NORTH、EAST 与 SOUTH 之间的双向传输，其传输系数均为 ct，体现了光信号在这些方向上的串扰大小。  
通过这种方式，散射矩阵完整描述了交叉结构在不同端口之间的光信号传输特性。

4、时域模型

def calculate\_signals (parameters, env, output\_signals, y, t, input\_signals):  
loss = 10.0 **(-0.05 \* parameters.insertion\_loss)  
ct = 10.0** (-0.05 \* parameters.cross\_talk)  
output\_signals["WEST:0"] = loss \* input\_signals["EAST:0"] + ct \* (  
input\_signals["NORTH:0"] + input\_signals["SOUTH:0"]  
)  
output\_signals["NORTH:0"] = loss \* input\_signals["SOUTH:0"] + ct \* (  
input\_signals["WEST:0"] + input\_signals["EAST:0"]  
)  
output\_signals["EAST:0"] = loss \* input\_signals["WEST:0"] + ct \* (  
input\_signals["NORTH:0"] + input\_signals["SOUTH:0"]  
)  
output\_signals["SOUTH:0"] = loss \* input\_signals["NORTH:0"] + ct \* (  
input\_signals["WEST:0"] + input\_signals["EAST:0"]  
)

与频域模型不同的是，时域模型需要输入信号 input\_signals，再基于端口之间的能量传递关系计算输出信号 output\_signals：  
对于 WEST 端口的输出信号，它由 EAST 端口输入信号经过损耗传输后的部分，加上 NORTH 和 SOUTH 端口输入信号经过串扰耦合后的部分组成，即 output\_signals ["WEST:0"] = loss \* input\_signals ["EAST:0"] + ct \* (input\_signals ["NORTH:0"] + input\_signals ["SOUTH:0"])。  
同理，NORTH 端口的输出信号由 SOUTH 端口输入信号的损耗传输部分与 WEST、EAST 端口输入信号的串扰耦合部分构成；EAST 端口的输出信号与 WEST 端口类似，是 WEST 端口输入信号的损耗传输部分与 NORTH、SOUTH 端口输入信号的串扰耦合部分之和；SOUTH 端口的输出信号则由 NORTH 端口输入信号的损耗传输部分与 WEST、EAST 端口输入信号的串扰耦合部分组成。通过这种方式，时域模型清晰地描述了交叉结构在时域下各个端口的输入信号与输出信号之间的关系。

## 4.4 Transition紧凑模型程序设计

Transition的紧凑模型程序如下：

class TransitionCompactModel(CompactModel):  
 parameters = [  
 "insertion\_loss",  
 ] # Note that the unit is dB and it is a positive value.  
 terms = [  
 OpticalTerm(name="in", n\_modes=2),  
 OpticalTerm(name="out", n\_modes=2),  
 ]  
  
 def calculate\_smatrix(parameters, env, S):  
 loss = 10.0 \*\* (-0.05 \* parameters.insertion\_loss)  
 S["in:0", "out:0"] = S["out:0", "in:0"] = loss  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 loss = 10.0 \*\* (-0.05 \* parameters.insertion\_loss)  
 output\_signals["out:0"] = loss \* input\_signals["in:0"]  
 output\_signals["in:0"] = loss \* input\_signals["out:0"]

Transition 的紧凑模型程序主要包含以下四个部分：

1、模型参数：

parameters = ["insertion\_loss"]

Transition 的模型参数主要包括插入损耗 "insertion\_loss"（注意：其单位为 dB，且为正值）。插入损耗描述了光信号在过渡结构中传输时的能量损失，是衡量过渡结构光学性能的关键指标，直接影响光信号在不同波导或器件之间转换时的效率。

2、模型端口

terms = [  
OpticalTerm (name="in", n\_modes=2),  
OpticalTerm (name="out", n\_modes=2),  
]

在 IPKISS 紧凑模型中，模型端口是实现光信号交互的关键节点，对整个光子集成电路的信号传输起着决定性作用。上述代码中定义的 terms 列表包含两个 OpticalTerm 对象，分别代表过渡结构的输入和输出光学端口。  
OpticalTerm (name="in", n\_modes=2) 定义了名为 "in" 的光学端口，n\_modes=2 表示该端口支持两个模式的光信号传输，主要用于接收输入到过渡结构的光信号，是光信号进入过渡结构的入口。OpticalTerm (name="out", n\_modes=2) 定义了名为 "out" 的光学端口，同样支持两个模式的光信号传输，用于输出经过过渡结构转换后的光信号，是光信号离开过渡结构的出口。这两个端口共同实现了光信号在过渡结构中的输入与输出，确保光信号能够在不同器件或波导之间顺利传递。

3、频域模型：

def calculate\_smatrix (parameters, env, S):  
loss = 10.0 \*\* (-0.05 \* parameters.insertion\_loss)  
S ["in:0", "out:0"] = S ["out:0", "in:0"] = loss

该代码定义的 calculate\_smatrix 函数是一个基于 Python 的过渡结构频域模型，利用散射矩阵（S 矩阵）来描述过渡结构在不同条件下的光学特性。以下是对这个模型的详细介绍：

关键参数计算：  
根据插入损耗 parameters.insertion\_loss 计算损耗系数 loss，公式为 loss = 10.0 \*\* (-0.05 \* parameters.insertion\_loss)。这一公式将以 dB 为单位的插入损耗转换为线性单位的损耗系数，直观反映了光信号经过过渡结构后的能量留存比例。

散射矩阵赋值：散射矩阵 S 中各项值描述了不同端口之间的光信号传输关系。  
将计算得到的损耗系数 loss 赋值给散射矩阵 S 中 "in:0" 到 "out:0" 以及 "out:0" 到 "in:0" 的项，表明光信号在过渡结构的这两个端口之间双向传输时，仅存在能量损耗，且损耗大小由插入损耗决定。这种赋值方式清晰体现了过渡结构对光信号的衰减作用，以及光信号在该结构中双向传输特性的一致性。

4、时域模型

def calculate\_signals (parameters, env, output\_signals, y, t, input\_signals):  
loss = 10.0 \*\* (-0.05 \* parameters.insertion\_loss)  
output\_signals ["out:0"] = loss \* input\_signals ["in:0"]  
output\_signals ["in:0"] = loss \* input\_signals ["out:0"]

与频域模型不同的是，时域模型需要输入信号 input\_signals，再基于端口之间的能量传递关系计算输出信号 output\_signals：  
对于 "out:0" 端口的输出信号，它等于损耗系数 loss 与 "in:0" 端口的输入信号相乘，即 output\_signals ["out:0"] = loss \* input\_signals ["in:0"]，表示输入到 "in:0" 端口的光信号经过过渡结构的损耗后，从 "out:0" 端口输出。  
同理，"in:0" 端口的输出信号由损耗系数 loss 与 "out:0" 端口的输入信号相乘得到，即 output\_signals ["in:0"] = loss \* input\_signals ["out:0"]，描述了光信号从 "out:0" 端口输入、经过过渡结构损耗后从 "in:0" 端口输出的过程。  
通过这种方式，时域模型清晰地描述了过渡结构在时域下各个端口的输入信号与输出信号之间的线性衰减关系，体现了光信号在过渡结构中双向传输时的能量损耗特性。

## 4.5 MMI1x2紧凑模型程序设计

MMI1x2的紧凑模型程序如下：

class MMI1x2CompactModel(CompactModel):  
 parameters = ["mmi\_il", "mmi\_pi"]  
 terms = [  
 OpticalTerm(name="in\_1", n\_modes=2),  
 OpticalTerm(name="out\_1", n\_modes=2),  
 OpticalTerm(name="out\_2", n\_modes=2),  
 ]  
  
 def calculate\_smatrix(parameters, env, S):  
 il = parameters.mmi\_il, # insertion loss  
 power\_imb = parameters.mmi\_pi, # power imbalance  
 loss = 10.0 \*\* (-0.05 \* il),  
 in1\_out1 = loss \* np.sqrt(0.5 + power\_imb)  
 in1\_out2 = loss \* np.sqrt(0.5 - power\_imb)  
 S["in\_1:0", "out\_1:0"] = S["out\_1:0", "in\_1:0"] = in1\_out1  
 S["in\_1:0", "out\_2:0"] = S["out\_2:0", "in\_1:0"] = in1\_out2  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 il = parameters.mmi\_il, # insertion loss  
 power\_imb = parameters.mmi\_pi, # power imbalance  
 loss = 10.0 \*\* (-0.05 \* il),  
 in1\_out1 = loss \* np.sqrt(0.5 + power\_imb)  
 in1\_out2 = loss \* np.sqrt(0.5 - power\_imb)  
 output\_signals["in\_1:0"] = in1\_out1 \* input\_signals["out\_1:0"] + in1\_out2 \* input\_signals["out\_1:0"]  
 output\_signals["out\_1:0"] = in1\_out1 \* input\_signals["in\_1:0"]  
 output\_signals["out\_2:0"] = in1\_out2 \* input\_signals["in\_1:0"]

MMI1x2 的紧凑模型程序主要包含以下四个部分：

1、模型参数：

parameters = ["mmi\_il", "mmi\_pi"]

模型参数是建模的基础，MMI1x2 的模型参数主要包括插入损耗 “mmi\_il” 和功率不平衡度 “mmi\_pi”。其中，插入损耗 “mmi\_il” 用于描述光信号在 1x2 多模干涉器（MMI）中传输时的能量损失；功率不平衡度 “mmi\_pi” 则表征了光信号经 MMI 分束后，两个输出端口的功率差异，是衡量分束均匀性的重要指标。

2、模型端口

terms = [  
OpticalTerm (name="in\_1", n\_modes=2),  
OpticalTerm (name="out\_1", n\_modes=2),  
OpticalTerm (name="out\_2", n\_modes=2),  
]

在 IPKISS 紧凑模型中，模型端口是实现光信号交互的关键节点，对整个光子集成电路的信号传输起着决定性作用。上述代码中定义的 terms 列表包含三个 OpticalTerm 对象，分别代表 1x2 MMI 的输入和输出光学端口。  
OpticalTerm (name="in\_1", n\_modes=2) 定义了名为 “in\_1” 的光学端口，n\_modes=2 表示该端口支持两个模式的光信号传输，主要用于接收输入到 1x2 MMI 的光信号，是光信号进入 MMI 器件的入口。OpticalTerm (name="out\_1", n\_modes=2) 和 OpticalTerm (name="out\_2", n\_modes=2) 分别定义了名为 “out\_1” 和 “out\_2” 的光学端口，均支持两个模式的光信号传输，用于输出经过 1x2 MMI 分束后的光信号，是光信号离开 MMI 器件的两个出口。这三个端口共同实现了光信号在 1x2 MMI 中的输入与分束输出功能。

3、频域模型：

def calculate\_smatrix (parameters, env, S):  
il = parameters.mmi\_il, # insertion loss  
power\_imb = parameters.mmi\_pi, # power imbalance  
loss = 10.0 \*\* (-0.05 \* il),  
in1\_out1 = loss \* np.sqrt (0.5 + power\_imb)  
in1\_out2 = loss \* np.sqrt (0.5 - power\_imb)  
S ["in\_1:0", "out\_1:0"] = S ["out\_1:0", "in\_1:0"] = in1\_out1  
S ["in\_1:0", "out\_2:0"] = S ["out\_2:0", "in\_1:0"] = in1\_out2

该代码定义的 calculate\_smatrix 函数是一个基于 Python 的 1x2 MMI 频域模型，利用散射矩阵（S 矩阵）来描述 1x2 MMI 在不同条件下的光学特性。以下是对这个模型的详细介绍：

关键参数计算：

根据插入损耗 “il” 计算损耗系数 “loss”，公式为 loss = 10.0 \*\* (-0.05 \* il)，将以 dB 为单位的插入损耗转换为线性单位的损耗系数。

计算输入端口 “in\_1” 到输出端口 “out\_1” 的传输系数 “in1\_out1”，公式为 in1\_out1 = loss \* np.sqrt (0.5 + power\_imb)，其中 0.5 代表理想均分情况下的功率占比，加上功率不平衡度 “power\_imb” 后，再与损耗系数相乘，得到实际传输系数。

类似地，计算输入端口 “in\_1” 到输出端口 “out\_2” 的传输系数 “in1\_out2”，公式为 in1\_out2 = loss \* np.sqrt (0.5 - power\_imb)，体现了功率不平衡对另一输出端口传输特性的影响。

散射矩阵赋值：散射矩阵 S 中各项值描述了不同端口之间的光信号传输关系。

将 “in1\_out1” 赋值给 S ["in\_1:0", "out\_1:0"] 和 S ["out\_1:0", "in\_1:0"]，表示光信号在 “in\_1” 与 “out\_1” 端口之间双向传输时，遵循该传输系数对应的能量传递规律。

将 “in1\_out2” 赋值给 S ["in\_1:0", "out\_2:0"] 和 S ["out\_2:0", "in\_1:0"]，描述光信号在 “in\_1” 与 “out\_2” 端口之间双向传输的特性。

4、时域模型

def calculate\_signals (parameters, env, output\_signals, y, t, input\_signals):  
il = parameters.mmi\_il, # insertion loss  
power\_imb = parameters.mmi\_pi, # power imbalance  
loss = 10.0 \*\* (-0.05 \* il),  
in1\_out1 = loss \* np.sqrt (0.5 + power\_imb)  
in1\_out2 = loss \* np.sqrt (0.5 - power\_imb)  
output\_signals ["in\_1:0"] = in1\_out1 \* input\_signals ["out\_1:0"] + in1\_out2 \* input\_signals ["out\_1:0"]  
output\_signals ["out\_1:0"] = in1\_out1 \* input\_signals ["in\_1:0"]  
output\_signals ["out\_2:0"] = in1\_out2 \* input\_signals ["in\_1:0"]

与频域模型不同的是，时域模型需要输入信号 input\_signals，再基于端口之间的能量传递关系计算输出信号 output\_signals：

对于输入端口 “in\_1:0” 的输出信号，它由输出端口 “out\_1:0” 的输入信号与 “in1\_out1” 的乘积，加上输出端口 “out\_1:0” 的输入信号与 “in1\_out2” 的乘积组成（原文此处可能存在笔误，推测应为 “out\_2:0” 的输入信号，即 output\_signals ["in\_1:0"] = in1\_out1 \* input\_signals ["out\_1:0"] + in1\_out2 \* input\_signals ["out\_2:0"]），体现了两个输出端口信号反向传输到输入端口的叠加效果。

输出端口 “out\_1:0” 的输出信号等于 “in1\_out1” 与输入端口 “in\_1:0” 的输入信号相乘，即 output\_signals ["out\_1:0"] = in1\_out1 \* input\_signals ["in\_1:0"]，描述了光信号从 “in\_1” 到 “out\_1” 的正向传输。

输出端口 “out\_2:0” 的输出信号等于 “in1\_out2” 与输入端口 “in\_1:0” 的输入信号相乘，即 output\_signals ["out\_2:0"] = in1\_out2 \* input\_signals ["in\_1:0"]，体现了光信号从 “in\_1” 到 “out\_2” 的正向传输特性。

通过这种方式，时域模型清晰地描述了 1x2 MMI 在时域下各个端口的输入信号与输出信号之间的关系，包括分束后的功率分配和损耗影响。

## 4.6 MMI2x2紧凑模型程序设计

MMI2x2的紧凑模型与MMI1x2的紧凑模型类似，其模型参数均为插损和功率不平衡度。不同的是MMI2x2有4个端口：

terms = [  
 OpticalTerm(name="in\_1", n\_modes=2),  
 OpticalTerm(name="in\_2", n\_modes=2),  
 OpticalTerm(name="out\_1", n\_modes=2),  
 OpticalTerm(name="out\_2", n\_modes=2),  
]

此外，相比于MMI1x2，MMI2x2的两个输出端口之间有π的相位差。

in1\_out1 = loss \* np.sqrt(0.5 + power\_imb)  
in1\_out2 = 1j \*loss \* np.sqrt(0.5 - power\_imb)

## 4.7 Ring紧凑模型程序设计

Ring的紧凑模型程序如下：

class RingCompactModel(i3.CompactModel):  
 parameters = [  
 "n\_eff\_coef",  
 "alpha",  
 "gamma",  
 "l\_ring",  
 ]  
  
 terms = [  
 i3.OpticalTerm(name="opt\_0"),  
 i3.OpticalTerm(name="opt\_1"),  
 ]  
  
 states = ["vj", "e\_ccw", "e\_cw"]  
  
 def calculate\_smatrix(parameters, env, S):  
 omega = 2.0 \* pi \* c \* 1e6 / env.wavelength  
 l\_ring = parameters.l\_ring \* 1e-6  
 phase = (omega / c) \* polyval(parameters. n\_eff\_coef omega) \* l\_ring  
 phase\_total = phase  
 loss = parameters.alpha  
 sigma = parameters.gamma  
 through = (sigma - loss \* np.exp(1j \* phase\_total)) / (  
 1.0 - sigma \* loss \* np.exp(1j \* phase\_total)  
 )  
 S["opt\_0", "opt\_1"] = S["opt\_1", "opt\_0"] = through  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 omega = 2.0 \* pi \* c \* 1e6 / env.wavelength  
 l\_ring = parameters.l\_ring \* 1e-6  
 phase = (omega / c) \* polyval(parameters.n\_eff\_coeff, omega) \* l\_ring  
 phase\_total = phase  
 loss = parameters.alpha  
 sigma = parameters.gamma  
 through = (sigma - loss \* np.exp(1j \* phase\_total)) / (  
 1.0 - sigma \* loss \* np.exp(1j \* phase\_total)  
 )  
 output\_signals["opt\_1"] = input\_signals["opt\_0"].through  
 output\_signals["opt\_0"] = input\_signals["opt\_1"].through

Ring 的紧凑模型程序主要包含以下四个部分：

1、模型参数

parameters = [

"n\_eff\_coef",

"alpha",

"gamma",

"l\_ring",

]

环形谐振器（Ring）紧凑模型包含四个关键参数：

* **n\_eff\_coef**：环波导的有效折射率多项式系数，用于描述有效折射率随频率变化的色散特性。
* **alpha**：单程传播损耗（幅度因子），表示光信号在环内传播时的衰减程度。
* **gamma**：耦合系数（sigma），描述总线波导与环波导之间的光功率耦合强度。
* **l\_ring**：环的物理长度（单位 μm），影响谐振波长位置和谐振间隔。

这些参数共同决定了环形谐振器的相位响应、损耗特性以及耦合效率，是建立精确传输模型的基础。

2、模型端口

terms = [

i3.OpticalTerm(name="opt\_0"),

i3.OpticalTerm(name="opt\_1"),

]

该模型包含两个光学端口 opt\_0 和 opt\_1，分别代表环形谐振器的输入端和输出端。不同于 MMI1x2 具有一个输入两个输出的结构，环形谐振器在总线波导中只表现为一个输入端和一个输出端，而环本身通过耦合区与总线波导交换能量，从而在透射端口（through port）表现出频率选择性。

3、频域模型

def calculate\_smatrix(parameters, env, S):

omega = 2.0 \* pi \* c \* 1e6 / env.wavelength

l\_ring = parameters.l\_ring \* 1e-6

phase = (omega / c) \* polyval(parameters.n\_eff\_coef, omega) \* l\_ring

phase\_total = phase

loss = parameters.alpha

sigma = parameters.gamma

through = (sigma - loss \* np.exp(1j \* phase\_total)) / (

1.0 - sigma \* loss \* np.exp(1j \* phase\_total)

)

S["opt\_0", "opt\_1"] = S["opt\_1", "opt\_0"] = through

频域模型基于散射矩阵（S 矩阵）描述环形谐振器的透射特性。

首先计算角频率 omega，并根据环长 l\_ring 和有效折射率多项式 n\_eff\_coef 计算光在环中传播的相位延迟 phase。

loss 表示单程传播损耗，sigma 表示耦合系数。

透射系数 through 根据环谐振的干涉原理计算，公式考虑了损耗和耦合共同作用下的复幅度传输比。

最后，将透射系数赋值给 S 矩阵中的两个端口互连项 S["opt\_0","opt\_1"] 和 S["opt\_1","opt\_0"]，表示输入端和输出端的双向传输特性。

4、时域模型

def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):

omega = 2.0 \* pi \* c \* 1e6 / env.wavelength

l\_ring = parameters.l\_ring \* 1e-6

phase = (omega / c) \* polyval(parameters.n\_eff\_coef, omega) \* l\_ring

phase\_total = phase

loss = parameters.alpha

sigma = parameters.gamma

through = (sigma - loss \* np.exp(1j \* phase\_total)) / (

1.0 - sigma \* loss \* np.exp(1j \* phase\_total)

)

output\_signals["opt\_1"] = input\_signals["opt\_0"] \* through

output\_signals["opt\_0"] = input\_signals["opt\_1"] \* through

时域模型根据输入信号 input\_signals，利用与频域模型相同的透射系数公式计算输出信号 output\_signals。

对于输入端 opt\_0，其信号经过透射系数 through 转换后出现在输出端 opt\_1；

对于输入端 opt\_1，信号同样按 through 传输至输出端 opt\_0。  
该过程实现了环形谐振器在时域下的信号传输计算，使模型可在时域仿真中使用。

## 4.8 DirectionalCoupler 紧凑模型程序设计

DirectionalCoupler的紧凑模型程序如下：

class DirectionalCouplerCompactModel(CompactModel):  
 """  
 Compact model that takes into account the wavelength dependent coupling of the 50/50 directional coupler.  
 The model does not include phase, delay or loss information.  
 """  
  
 parameters = [  
 "k\_l\_coeff",# 直波导耦合系数  
 "k\_b\_coeff",# 弯曲波导耦合系数  
 "coupler\_length",# 直波导耦合长度  
 ]  
  
 terms = [  
 OpticalTerm(name="opt\_0"),  
 OpticalTerm(name="opt\_1"),  
 OpticalTerm(name="opt\_2"),  
 OpticalTerm(name="opt\_3"),  
 ]  
  
 def calculate\_smatrix(parameters, env, S):  
 k\_l = polyval(parameters.k\_l\_coeff, env.wavelength \* 1000.0)  
 k\_b = polyval(parameters.k\_b\_coeff, env.wavelength \* 1000.0)  
 kappa = np.sin(k\_l \* parameters.coupler\_length + k\_b)  
 tau = np.sqrt(1.0 - kappa\*\*2)  
 S["opt\_2", "opt\_0"] = S["opt\_0", "opt\_2"] = S["opt\_3", "opt\_1"] = S[  
 "opt\_1", "opt\_3"  
 ] = tau  
 S["opt\_2", "opt\_1"] = S["opt\_1", "opt\_2"] = S["opt\_3", "opt\_0"] = S[  
 "opt\_0", "opt\_3"  
 ] = 1j \* kappa  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 k\_l = polyval(parameters.k\_l\_coeff, env.wavelength \* 1000.0)  
 k\_b = polyval(parameters.k\_b\_coeff, env.wavelength \* 1000.0)  
 kappa = np.sin(k\_l \* parameters.coupler\_length + k\_b)  
 tau = np.sqrt(1.0 - kappa\*\*2)  
 output\_signals["opt\_0"] = (  
 1j \* kappa \* input\_signals["opt\_3"] + tau \* input\_signals["opt\_2"]  
 )  
 output\_signals["opt\_1"] = (  
 1j \* kappa \* input\_signals["opt\_2"] + tau \* input\_signals["opt\_3"]  
 )  
 output\_signals["opt\_2"] = (  
 1j \* kappa \* input\_signals["opt\_1"] + tau \* input\_signals["opt\_0"]  
 )  
 output\_signals["opt\_3"] = (  
 1j \* kappa \* input\_signals["opt\_0"] + tau \* input\_signals["opt\_1"]  
 )

Directional Coupler 的紧凑模型程序主要包含以下四个部分：

1、模型参数：

parameters = [  
"k\_l\_coeff", # 直波导耦合系数  
"k\_b\_coeff", # 弯曲波导耦合系数  
"coupler\_length", # 直波导耦合长度  
]

Directional Coupler 的模型参数主要包括直波导耦合系数 “k\_l\_coeff”、弯曲波导耦合系数 “k\_b\_coeff” 和直波导耦合长度 “coupler\_length”。其中，直波导耦合系数和弯曲波导耦合系数用于描述不同波导结构对光信号耦合的影响，且与波长相关；直波导耦合长度则是决定光信号在定向耦合器中耦合程度的关键物理参数，直接影响耦合效率和传输特性。该模型重点考虑了定向耦合器的波长相关耦合特性，但未包含相位、延迟和损耗信息。

2、模型端口

terms = [  
OpticalTerm (name="opt\_0"),  
OpticalTerm (name="opt\_1"),  
OpticalTerm (name="opt\_2"),  
OpticalTerm (name="opt\_3"),  
]

在 IPKISS 紧凑模型中，模型端口是实现光信号交互的关键节点，对整个光子集成电路的信号传输起着决定性作用。上述代码中定义的 terms 列表包含四个 OpticalTerm 对象，分别代表定向耦合器的四个光学端口。  
这四个端口 “opt\_0”“opt\_1”“opt\_2”“opt\_3” 共同构成了定向耦合器光信号的输入输出通道，用于实现光信号在耦合器中的传输、分配与耦合，是光信号在定向耦合器中进行交互的关键节点。

3、频域模型：

def calculate\_smatrix (parameters, env, S):  
k\_l = polyval (parameters.k\_l\_coeff, env.wavelength \* 1000.0)  
k\_b = polyval (parameters.k\_b\_coeff, env.wavelength \* 1000.0)  
kappa = np.sin (k\_l \* parameters.coupler\_length + k\_b)  
tau = np.sqrt (1.0 - kappa\*\*2)  
S ["opt\_2", "opt\_0"] = S ["opt\_0", "opt\_2"] = S ["opt\_3", "opt\_1"] = S [  
"opt\_1", "opt\_3"  
] = tau  
S ["opt\_2", "opt\_1"] = S ["opt\_1", "opt\_2"] = S ["opt\_3", "opt\_0"] = S [  
"opt\_0", "opt\_3"  
] = 1j \* kappa

该代码定义的 calculate\_smatrix 函数是一个基于 Python 的定向耦合器频域模型，利用散射矩阵（S 矩阵）来描述定向耦合器在不同条件下的光学特性，重点体现了其波长相关的耦合特性。以下是对这个模型的详细介绍：

关键参数计算：

利用 polyval 函数，根据直波导耦合系数 “k\_l\_coeff” 和当前波长（转换为纳米单位，即 env.wavelength \* 1000.0）计算直波导的实际耦合系数 “k\_l”。

同理，利用 polyval 函数和弯曲波导耦合系数 “k\_b\_coeff” 计算弯曲波导的实际耦合系数 “k\_b”。

计算耦合系数 “kappa”，公式为 kappa = np.sin (k\_l \* parameters.coupler\_length + k\_b)，其值由直波导耦合系数、耦合长度以及弯曲波导耦合系数共同决定，反映了光信号在定向耦合器中的耦合程度。

计算透射系数 “tau”，公式为 tau = np.sqrt (1.0 - kappa\*\*2)，体现了光信号直接透射的比例，与耦合系数满足能量守恒关系（tau² + kappa² = 1）。

散射矩阵赋值：散射矩阵 S 中各项值描述了不同端口之间的光信号传输关系。

将透射系数 “tau” 赋值给 S ["opt\_2", "opt\_0"]、S ["opt\_0", "opt\_2"]、S ["opt\_3", "opt\_1"] 和 S ["opt\_1", "opt\_3"]，表示光信号在这些端口对之间双向传输时，遵循透射系数对应的能量传递规律，即光信号从 “opt\_0” 到 “opt\_2”、“opt\_2” 到 “opt\_0”、“opt\_1” 到 “opt\_3”、“opt\_3” 到 “opt\_1” 的传输主要为直接透射。

将 1j \* kappa 赋值给 S ["opt\_2", "opt\_1"]、S ["opt\_1", "opt\_2"]、S ["opt\_3", "opt\_0"] 和 S ["opt\_0", "opt\_3"]，描述光信号在这些端口对之间的耦合传输特性，其中虚数单位 “j” 体现了耦合过程中的相位变化，即光信号从 “opt\_1” 到 “opt\_2”、“opt\_2” 到 “opt\_1”、“opt\_0” 到 “opt\_3”、“opt\_3” 到 “opt\_0” 的传输主要为耦合传输。

4、时域模型

def calculate\_signals (parameters, env, output\_signals, y, t, input\_signals):  
k\_l = polyval (parameters.k\_l\_coeff, env.wavelength \* 1000.0)  
k\_b = polyval (parameters.k\_b\_coeff, env.wavelength \* 1000.0)  
kappa = np.sin (k\_l \* parameters.coupler\_length + k\_b)  
tau = np.sqrt (1.0 - kappa\*\*2)  
output\_signals ["opt\_0"] = (  
1j \* kappa \* input\_signals ["opt\_3"] + tau \* input\_signals ["opt\_2"]  
)  
output\_signals ["opt\_1"] = (  
1j \* kappa \* input\_signals ["opt\_2"] + tau \* input\_signals ["opt\_3"]  
)  
output\_signals ["opt\_2"] = (  
1j \* kappa \* input\_signals ["opt\_1"] + tau \* input\_signals ["opt\_0"]  
)  
output\_signals ["opt\_3"] = (  
1j \* kappa \* input\_signals ["opt\_0"] + tau \* input\_signals ["opt\_1"]  
)

与频域模型不同的是，时域模型需要输入信号 input\_signals，再基于端口之间的能量传递关系计算输出信号 output\_signals：

端口 “opt\_0” 的输出信号由端口 “opt\_3” 的输入信号经耦合传输（1j \* kappa 乘以输入信号）与端口 “opt\_2” 的输入信号经透射传输（tau 乘以输入信号）叠加而成，即 output\_signals ["opt\_0"] = 1j \* kappa \* input\_signals ["opt\_3"] + tau \* input\_signals ["opt\_2"]。

端口 “opt\_1” 的输出信号是端口 “opt\_2” 的输入信号经耦合传输与端口 “opt\_3” 的输入信号经透射传输的叠加，即 output\_signals ["opt\_1"] = 1j \* kappa \* input\_signals ["opt\_2"] + tau \* input\_signals ["opt\_3"]。

端口 “opt\_2” 的输出信号由端口 “opt\_1” 的输入信号经耦合传输与端口 “opt\_0” 的输入信号经透射传输叠加得到，即 output\_signals ["opt\_2"] = 1j \* kappa \* input\_signals ["opt\_1"] + tau \* input\_signals ["opt\_0"]。

端口 “opt\_3” 的输出信号是端口 “opt\_0” 的输入信号经耦合传输与端口 “opt\_1” 的输入信号经透射传输的叠加，即 output\_signals ["opt\_3"] = 1j \* kappa \* input\_signals ["opt\_0"] + tau \* input\_signals ["opt\_1"]。

通过这种方式，时域模型清晰地描述了定向耦合器在时域下各个端口的输入信号与输出信号之间的关系，体现了光信号在定向耦合器中的耦合与透射特性。

## 4.9 Thermal phase shifter 紧凑模型程序设计

Thermal phase shifter的紧凑模型程序如下：

class TPSCompactModel(CompactModel):  
 parameters = [  
 "r\_heater",  
 "heater\_offset",  
 "heater\_width",  
 "heater\_length",  
 "heater\_height",  
 "heater\_p", # 电阻率  
 "p\_pi\_length\_coeff", # heater相对于波导的偏移  
 "p\_pi\_width\_coeff",  
 "p\_pi\_offset\_coeff",  
 "wg\_data",  
 "optical\_wg\_length",  
 "vbias",  
 ]  
 terms = [  
 OpticalTerm(name="in", n\_modes=2),  
 OpticalTerm(name="out", n\_modes=2),  
 ElectricalTerm(name="elec\_in"),  
 ElectricalTerm(name="elec\_out"),  
 ]  
  
 def calculate\_smatrix(parameters, env, S):  
 P\_pi = (polyval(parameters.p\_pi\_length\_coeff, parameters.heater\_length)+  
 polyval(parameters.p\_pi\_width\_coeff, parameters.heater\_width))+  
 polyval(parameters.p\_pi\_offset\_coeff, parameters.heater\_offset)  
  
 phase\_heater = pi \* (  
 (parameters.vbias\*\*2.0) / (parameters.r\_heater \* P\_pi)  
 )  
 center\_wavelength = parameters.wg\_data[0]  
 alpha\_dBcm = parameters.wg\_data[1]  
 neff0 = parameters.wg\_data[2]  
 ng0 = parameters.wg\_data[3]  
 wav = env.wavelength  
 phase\_wg = (  
 ((2.0 \* pi) / wav)  
 \* (neff0 + (center\_wavelength / env.wavelength) \* (ng0 - neff0))  
 \* parameters.optical\_wg\_length  
 )  
 loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.optical\_wg\_length) / 20.0)  
 S["in:0", "out:0"] = S["out:0", "in:0"] = loss \* np.exp(  
 1j \* (phase\_wg + phase\_heater)  
 )  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 vbias = input\_signals["elec\_in"].real - input\_signals["elec\_out"].real  
 P\_pi = (polyval(parameters.p\_pi\_length\_coeff, parameters.heater\_length)+  
 polyval(parameters.p\_pi\_width\_coeff, parameters.heater\_width))+  
 polyval(parameters.p\_pi\_offset\_coeff, parameters.heater\_offset)  
  
 phase\_heater = pi \* (  
 (parameters.vbias\*\*2.0) / (parameters.r\_heater \* P\_pi)  
 )  
 center\_wavelength = parameters.wg\_data[0]  
 alpha\_dBcm = parameters.wg\_data[1]  
 neff0 = parameters.wg\_data[2]  
 ng0 = parameters.wg\_data[3]  
 wav = env.wavelength  
 phase\_wg = (  
 ((2.0 \* pi) / wav)  
 \* (neff0 + (center\_wavelength / env.wavelength) \* (ng0 - neff0))  
 \* parameters.optical\_wg\_length  
 )  
 loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.optical\_wg\_length) / 20.0)  
 t\_delay = ng0 \* parameters.optical\_wg\_length / c\_mu  
 trans = loss \* np.exp(1j \* (phase\_wg + phase\_heater))  
 output\_signals["in:0"] = trans \* input\_signals["out:0", t - t\_delay]  
 output\_signals["out:0"] = trans \* input\_signals["in:0", t - t\_delay]  
 output\_signals["elec\_in"] = vbias / parameters.r\_heater  
 output\_signals["elec\_out"] = -vbias / parameters.r\_heater

Thermal phase shifter 的紧凑模型程序主要包含以下四个部分：

1、模型参数

parameters = [

"r\_heater",

"heater\_offset",

"heater\_width",

"heater\_length",

"heater\_height",

"heater\_p", # 电阻率

"p\_pi\_length\_coeff", # heater相对于波导的偏移

"p\_pi\_width\_coeff",

"p\_pi\_offset\_coeff",

"wg\_data",

"optical\_wg\_length",

"vbias",

]

该热光相移器紧凑模型包含以下主要参数：

* **r\_heater**：加热器的电阻（Ω），由加热器的几何尺寸和材料电阻率决定，是热功率计算的核心参数。
* **heater\_offset**：加热器与波导中心的横向偏移量（μm），影响热量耦合效率。
* **heater\_width / heater\_length / heater\_height**：加热器的几何尺寸（单位 μm），决定其电阻和散热特性。
* **heater\_p**：加热器材料的电阻率（Ω·m），结合几何尺寸计算加热器电阻。
* **p\_pi\_length\_coeff / p\_pi\_width\_coeff / p\_pi\_offset\_coeff**：与 π 相移功耗 PπP\_{\pi}Pπ​ 相关的经验系数，用多项式拟合 heater 长度、宽度和偏移对热光效率的影响。
* **wg\_data**：波导参数列表，依次包含中心波长（μm）、传播损耗（dB/cm）、有效折射率 neffn\_{\mathrm{eff}}neff​、群折射率 ngn\_gng​。
* **optical\_wg\_length**：波导光程长度（μm），用于计算传播相位。
* **vbias**：加热器两端的偏置电压（V）。

2、模型端口

terms = [

OpticalTerm(name="in", n\_modes=2),

OpticalTerm(name="out", n\_modes=2),

ElectricalTerm(name="elec\_in"),

ElectricalTerm(name="elec\_out"),

]

热光相移器模型包含两类端口：

* **光学端口**：in 和 out，均支持两个模式的光信号传输，分别代表波导的输入端和输出端。
* **电学端口**：elec\_in 和 elec\_out，用于给加热器施加电压以产生焦耳热。

光学端口负责信号透射，电学端口驱动热光效应，两者在模型中通过热致相位调制相互耦合。

3、频域模型

def calculate\_smatrix(parameters, env, S):

P\_pi = (polyval(parameters.p\_pi\_length\_coeff, parameters.heater\_length) +

polyval(parameters.p\_pi\_width\_coeff, parameters.heater\_width) +

polyval(parameters.p\_pi\_offset\_coeff, parameters.heater\_offset))

phase\_heater = pi \* ((parameters.vbias\*\*2.0) / (parameters.r\_heater \* P\_pi))

center\_wavelength = parameters.wg\_data[0]

alpha\_dBcm = parameters.wg\_data[1]

neff0 = parameters.wg\_data[2]

ng0 = parameters.wg\_data[3]

wav = env.wavelength

phase\_wg = ((2.0 \* pi) / wav) \* \

(neff0 + (center\_wavelength / env.wavelength) \* (ng0 - neff0)) \* \

parameters.optical\_wg\_length

loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.optical\_wg\_length) / 20.0)

S["in:0", "out:0"] = S["out:0", "in:0"] = loss \* np.exp(1j \* (phase\_wg + phase\_heater))

频域模型使用散射矩阵 SSS 描述热光相移器在稳态下的传输特性：

* **P\_π 计算**：利用多项式拟合参数计算 π 相移所需功率，综合 heater 长度、宽度和偏移对热光效率的影响。
* **热光相位**ϕheater​：由焦耳热功率 Vbias2/RheaterV与 Pπ之比乘以 π 得到。
* **波导传播相位** ϕwg ​：考虑有效折射率 neff与群折射率 ng的色散关系计算得到。
* **光学损耗**：由波导传播损耗 α dB/cm和长度换算为线性衰减系数。
* **S 矩阵赋值**：透射端口的复传输系数为 loss⋅ej(ϕwg+ϕheater)体现了热光相移叠加在波导固有相位上的效果。

4、时域模型

def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):

vbias = input\_signals["elec\_in"].real - input\_signals["elec\_out"].real

P\_pi = (polyval(parameters.p\_pi\_length\_coeff, parameters.heater\_length) +

polyval(parameters.p\_pi\_width\_coeff, parameters.heater\_width) +

polyval(parameters.p\_pi\_offset\_coeff, parameters.heater\_offset))

phase\_heater = pi \* ((vbias\*\*2.0) / (parameters.r\_heater \* P\_pi))

center\_wavelength = parameters.wg\_data[0]

alpha\_dBcm = parameters.wg\_data[1]

neff0 = parameters.wg\_data[2]

ng0 = parameters.wg\_data[3]

wav = env.wavelength

phase\_wg = ((2.0 \* pi) / wav) \* \

(neff0 + (center\_wavelength / env.wavelength) \* (ng0 - neff0)) \* \

parameters.optical\_wg\_length

loss = 10.0 \*\* (-(alpha\_dBcm \* 1e-4 \* parameters.optical\_wg\_length) / 20.0)

t\_delay = ng0 \* parameters.optical\_wg\_length / c\_mu

trans = loss \* np.exp(1j \* (phase\_wg + phase\_heater))

output\_signals["in:0"] = trans \* input\_signals["out:0", t - t\_delay]

output\_signals["out:0"] = trans \* input\_signals["in:0", t - t\_delay]

output\_signals["elec\_in"] = vbias / parameters.r\_heater

output\_signals["elec\_out"] = -vbias / parameters.r\_heater

时域模型在瞬态仿真中计算信号传输：

* 电压 Vbias来自 elec\_in 与 elec\_out 端口的差值，用于实时计算加热器功率。
* 计算热光相位 ϕheater和波导相位 ϕwg与频域模型相同。
* 引入群延迟 tdelay=ngL/c来模拟光信号传播时间。
* 光学端口信号按复传输系数 trans 延迟传输，实现双向透射计算。
* 电学端口输出电流，根据欧姆定律 I=V/R计算，保证电光联合仿真的一致性。

## 4.10 PN Phase shifter 紧凑模型程序设计

PN Phase shifter的紧凑模型程序如下：

class CompactModel\_PN\_phase\_shifter(i3.CompactModel):  
 parameters = [  
 "Propagation\_loss\_at\_0V\_dB\_per\_cm",  
 "Propagation\_loss\_at\_m2V\_dB\_per\_cm",  
 "VpiL\_at\_0V\_V\_mul\_cm",  
 "VpiL\_at\_m2V\_V\_mul\_cm",  
 "VpiL\_2p0Vpp\_V\_mul\_cm",  
 "R\_Ohm\_mul\_um",  
 "C\_at\_0V\_fF\_per\_um",  
 "C\_at\_m2V\_fF\_per\_um",  
 "v\_bias\_dc",  
 "mod\_length",  
 ]  
  
 terms = [  
 i3.OpticalTerm(name="opt\_0"),  
 i3.OpticalTerm(name="opt\_1"),  
 i3.ElectricalTerm(name="anode\_0"),  
 i3.ElectricalTerm(name="cathode\_0"),  
 ]  
  
 states = ["vj"]  
  
 def calculate\_smatrix(parameters, env, S):  
 a\_il = 0.5 \* (parameters.Propagation\_loss\_at\_0V\_dB\_per\_cm - parameters.Propagation\_loss\_at\_m2V\_dB\_per\_cm)  
 il = a\_il \* parameters.v\_bias\_dc + parameters.Propagation\_loss\_at\_0V\_dB\_per\_cm  
 il\_tot = il \* parameters.mod\_length \* 1e-4  
 loss = 10.0 \*\* (il\_tot / 20.0)  
 a\_vpi = 0.5 \* (parameters.VpiL\_at\_0V\_V\_mul\_cm - parameters.VpiL\_at\_m2V\_V\_mul\_cm)  
 vpi = a\_vpi \* parameters.v\_bias\_dc + parameters.VpiL\_at\_0V\_V\_mul\_cm  
 vpi\_tot = vpi / (parameters.mod\_length \* 1e-4)  
 phase = np.exp(1j \* pi \* parameters.v\_bias\_dc / vpi\_tot)  
 S["opt\_0", "opt\_1"] = S["opt\_1", "opt\_0"] = loss \* phase  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 a\_il = 0.5 \* (parameters.Propagation\_loss\_at\_0V\_dB\_per\_cm - parameters.Propagation\_loss\_at\_m2V\_dB\_per\_cm)  
 il = a\_il \* y["vj"] + parameters.Propagation\_loss\_at\_0V\_dB\_per\_cm  
 il\_tot = il \* parameters.mod\_length \* 1e-4  
 loss = 10.0 \*\* (il\_tot / 20.0)  
 vpi\_a = 0.5 \* (parameters.VpiL\_at\_0V\_V\_mul\_cm - parameters.VpiL\_at\_m2V\_V\_mul\_cm)  
 vpi = vpi\_a \* y["vj"] + parameters.VpiL\_at\_0V\_V\_mul\_cm  
 vpi\_tot = vpi / (parameters.mod\_length \* 1e-4)  
 phase = np.exp(1j \* pi \* y["vj"] / vpi\_tot)  
 output\_signals["opt\_0"] = loss \* phase \* input\_signals["opt\_1"]  
 output\_signals["opt\_1"] = loss \* phase \* input\_signals["opt\_0"]  
  
 def calculate\_dydt(parameters, env, dydt, y, t, input\_signals):  
 v\_bias = (input\_signals["anode\_0"] - input\_signals["cathode\_0"]).real  
 capacitance\_a = 0.5 \* (parameters.C\_at\_0V\_fF\_per\_um - parameters.C\_at\_m2V\_fF\_per\_um)  
 capacitance = capacitance\_a \* y["vj"].real + parameters.C\_at\_0V\_fF\_per\_um  
 bw = 1.0 / (2.0 \* pi \* capacitance \* 1e-15 \* parameters.R\_Ohm\_mul\_um \* 1e3)  
 dydt["vj"] = 2.0 \* pi \* bw \* (v\_bias - y["vj"].real)

PN Phase shifter 的紧凑模型程序主要包含以下四个部分：

1、模型参数

parameters = [

"Propagation\_loss\_at\_0V\_dB\_per\_cm",

"Propagation\_loss\_at\_m2V\_dB\_per\_cm",

"VpiL\_at\_0V\_V\_mul\_cm",

"VpiL\_at\_m2V\_V\_mul\_cm",

"VpiL\_2p0Vpp\_V\_mul\_cm",

"R\_Ohm\_mul\_um",

"C\_at\_0V\_fF\_per\_um",

"C\_at\_m2V\_fF\_per\_um",

"v\_bias\_dc",

"mod\_length",

]

PN 相移器（PN Phase Shifter）紧凑模型包含以下关键参数：

* **Propagation\_loss\_at\_0V\_dB\_per\_cm**：偏置电压为 0V 时的传播损耗（dB/cm），反映无调制时的光信号衰减。
* **Propagation\_loss\_at\_m2V\_dB\_per\_cm**：偏置电压为 -2V 时的传播损耗（dB/cm），用于刻画损耗随电压变化的特性。
* **VpiL\_at\_0V\_V\_mul\_cm**：0V 时的 Vπ·L（伏·厘米），表示产生 π 相移所需的电压与器件长度的乘积。
* **VpiL\_at\_m2V\_V\_mul\_cm**：-2V 时的 Vπ·L（伏·厘米），体现 VπL 随偏置变化的行为。
* **VpiL\_2p0Vpp\_V\_mul\_cm**：2.0V 峰峰值驱动时的 Vπ·L，用于高速调制性能评估。
* **R\_Ohm\_mul\_um**：电极等效电阻（Ω·μm），影响 RC 延迟和调制带宽。
* **C\_at\_0V\_fF\_per\_um**：0V 偏置时的电容（fF/μm）。
* **C\_at\_m2V\_fF\_per\_um**：-2V 偏置时的电容（fF/μm）。
* **v\_bias\_dc**：直流偏置电压（V），影响光学损耗与相位延迟。
* **mod\_length**：相移器的有效调制长度（μm），决定整体调制效率。

这些参数共同定义了器件的光学损耗、相位调制效率以及电学带宽特性。

2、模型端口

terms = [

i3.OpticalTerm(name="opt\_0"),

i3.OpticalTerm(name="opt\_1"),

i3.ElectricalTerm(name="anode\_0"),

i3.ElectricalTerm(name="cathode\_0"),

]

该模型包含 **两个光学端口**（opt\_0 和 opt\_1）以及 **两个电学端口**（anode\_0 和 cathode\_0）：

* 光学端口用于输入和输出光信号，PN 相移器本身为直通结构，因此两个光学端口直接连接。
* 电学端口提供调制所需的驱动信号，通过改变 PN 结的折射率来实现相位调制。

3、频域模型

def calculate\_smatrix(parameters, env, S):

a\_il = 0.5 \* (parameters.Propagation\_loss\_at\_0V\_dB\_per\_cm - parameters.Propagation\_loss\_at\_m2V\_dB\_per\_cm)

il = a\_il \* parameters.v\_bias\_dc + parameters.Propagation\_loss\_at\_0V\_dB\_per\_cm

il\_tot = il \* parameters.mod\_length \* 1e-4

loss = 10.0 \*\* (il\_tot / 20.0)

a\_vpi = 0.5 \* (parameters.VpiL\_at\_0V\_V\_mul\_cm - parameters.VpiL\_at\_m2V\_V\_mul\_cm)

vpi = a\_vpi \* parameters.v\_bias\_dc + parameters.VpiL\_at\_0V\_V\_mul\_cm

vpi\_tot = vpi / (parameters.mod\_length \* 1e-4)

phase = np.exp(1j \* pi \* parameters.v\_bias\_dc / vpi\_tot)

S["opt\_0", "opt\_1"] = S["opt\_1", "opt\_0"] = loss \* phase

频域模型使用 **S 矩阵** 描述 PN 相移器在特定直流偏置下的光学传输特性：

* 通过插值（a\_il）计算给定偏置电压下的传播损耗 il，并将其转化为总损耗因子 loss。
* 计算给定偏置下的 VπL 值 vpi，再换算成实际器件的 Vπ vpi\_tot。
* 根据电压与 Vπ 的比值，计算相移量并用复指数形式表示 phase。
* 将复数传输系数 loss \* phase 填入 S 矩阵的互连项，表示两个光学端口之间的双向传输关系。

4、时域模型

def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):

a\_il = 0.5 \* (parameters.Propagation\_loss\_at\_0V\_dB\_per\_cm - parameters.Propagation\_loss\_at\_m2V\_dB\_per\_cm)

il = a\_il \* y["vj"] + parameters.Propagation\_loss\_at\_0V\_dB\_per\_cm

il\_tot = il \* parameters.mod\_length \* 1e-4

loss = 10.0 \*\* (il\_tot / 20.0)

vpi\_a = 0.5 \* (parameters.VpiL\_at\_0V\_V\_mul\_cm - parameters.VpiL\_at\_m2V\_V\_mul\_cm)

vpi = vpi\_a \* y["vj"] + parameters.VpiL\_at\_0V\_V\_mul\_cm

vpi\_tot = vpi / (parameters.mod\_length \* 1e-4)

phase = np.exp(1j \* pi \* y["vj"] / vpi\_tot)

output\_signals["opt\_0"] = loss \* phase \* input\_signals["opt\_1"]

output\_signals["opt\_1"] = loss \* phase \* input\_signals["opt\_0"]

时域模型根据当前结电压 y["vj"] 实时计算器件的光学传输：

* 根据电压计算损耗因子 loss 和相移因子 phase。
* 对输入光信号乘以复数传输系数，得到输出信号。
* 光学端口的信号是双向耦合的，因此 opt\_0 和 opt\_1 都会更新。

## 4.11 MZM 紧凑模型程序设计

在本次建模过程中MZM为层级模型（hierarchical model），即其模型是通过MZM线路中的PN PhaseShifter、Thermal phase shifter，MMI以及波导共同通过仿真引擎决定，在定义MZM时只需加上如下层级模型代码定义即可：

class CircuitModel(i3.CircuitModelView):  
 # 2h. Automatically derive the hierarchical model from the netlist  
 def \_generate\_model(self):  
 return i3.HierarchicalModel.from\_netlistview(self.netlist\_view)

## 4.12 RingModulator紧凑模型程序设计

RingModulator的紧凑模型程序如下：

class CompactModel\_ring\_modulator(i3.CompactModel):  
 parameters = [  
 "centercavelength" # um  
 "n\_eff\_coeff",  
 "dneff",  
 "sigma",  
 "alpha",  
 "dalpha",  
 "v\_bias\_dc", # only for frequcncy domain simulation  
 "bandwidth",  
 "omega\_r\_0V",  
 "omega\_r\_coeff",  
 ]  
  
 terms = [  
 i3.OpticalTerm(name="opt\_0"),  
 i3.OpticalTerm(name="opt\_1"),  
 i3.ElectricalTerm(name="cathode\_0"),  
 i3.ElectricalTerm(name="anode\_0"),  
 ]  
  
 states = ["vj", "e\_ccw", "e\_cw"]  
  
 def calculate\_smatrix(parameters, env, S):  
 neff = polyval(parameters.n\_eff\_coeff, parameters.v\_bias\_dc)  
 phase = (  
 2  
 \* pi  
 / env.wavelength  
 \* (neff - dneff1 \* (env.wavelength - parameters.centercavelength - 0.05))  
 \* 2  
 \* pi  
 \* 11.0  
 \* 1e-6  
 )  
 sigma = parameters.sigma  
 alpha = parameters.alpha - parameters.dalpha \* (  
 env.wavelength - parameters.CenterWavelength - 0.05  
 )  
 through = (sigma - alpha \* np.exp(1j \* phase)) / (  
 1.0 - sigma \* alpha \* np.exp(1j \* phase)  
 )  
 S["opt\_0", "opt\_1"] = S["opt\_1", "opt\_0"] = through  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 kappa = np.sqrt(1.0 - parameters.sigma\*\*2)  
 output\_signals["opt\_1"] = input\_signals["opt\_0"] - 1j \* kappa \* y["e\_ccw"]  
 output\_signals["opt\_0"] = input\_signals["opt\_1"] - 1j \* kappa \* y["e\_cw"]  
  
 def calculate\_dydt(parameters, env, dydt, y, t, input\_signals):  
 v\_bias\_mod = (input\_signals["anode\_0"] - input\_signals["cathode\_0"]).real  
 alpha = parameters.alpha - parameters.dalpha \* (  
 env.wavelength - parameters.CenterWavelength - 0.05  
 )  
 loss = np.sqrt(1.0 - alpha\*\*2)  
 omega\_r = parameters.omega\_r\_0V + polyval(parameters.omega\_r\_coeff, y["vj"])  
 omega = 2.0 \* pi \* c \* 1e6 / env.wavelength  
 delta\_omega = omega\_r - omega  
 kappa = np.sqrt(1.0 - parameters.gamma\*\*2)  
 decay = (kappa\*\*2 ) / 2.0 + (loss\*\*2 ) / 2.0  
 dydt["e\_ccw"] = (1j \* delta\_omega - decay) \* y["e\_ccw"] - 1j \* kappa \* input\_signals["opt\_0"]  
 dydt["e\_cw"] = (1j \* delta\_omega - decay) \* y["e\_cw"] - 1j \* kappa \* input\_signals["opt\_1"]  
 dydt["vj"] = 2.0 \* np.pi \* parameters.bandwidth \* (v\_bias\_mod - y["vj"])

RingModulator 的紧凑模型程序主要包含以下四个部分：

1. 模型参数

这部分定义了微环调制器的核心物理参数，用于描述器件的固有特性和工作条件，具体包括：

centercavelength：微环的中心工作波长（单位 μm），是器件设计的基准波长。

n\_eff\_coeff：有效折射率的多项式系数，通常用于描述有效折射率随电压的变化关系（通过多项式拟合）。

dneff、sigma、alpha、dalpha：分别对应折射率变化量、耦合系数、损耗系数及损耗随波长的变化系数，用于描述光在微环中传输的损耗和耦合特性。

v\_bias\_dc：直流偏置电压，主要用于频域仿真中设定静态工作点。

bandwidth：调制器的带宽参数，影响时域响应速度。

omega\_r\_0V、omega\_r\_coeff：0 偏压下的共振角频率及共振频率随电压变化的多项式系数，用于描述微环共振频率的电压调谐特性。

这些参数共同构成了模型的物理基础，决定了器件的光学和电学特性。

2. 模型端口端口与状态变量定义（terms 和 states）

该模型包含以下端口：

光学端口：opt\_0和opt\_1，分别对应微环的两个光输入 / 输出端口，用于传输光信号。

电学端口：cathode\_0（阴极）和anode\_0（阳极），用于施加调制电压（通过两极间的电压差实现调制）。

状态变量（states）：

vj：结电压，描述微环中实际作用的电压（可能与外部施加电压存在动态差异）。

e\_ccw、e\_cw：分别对应逆时针（counter-clockwise）和顺时针（clockwise）传播的光场振幅，用于描述光在微环内的双向传输特性。

端口是模型与外部信号（光/电）的交互界面，而状态变量则记录了模型的动态演化信息。

3.频域模型（S矩阵）

calculate\_smatrix函数实现了频域模型：

该方法用于计算模型的 S 矩阵（散射矩阵），描述频域下光端口之间的传输关系，核心逻辑包括：

有效折射率计算：通过polyval函数，利用n\_eff\_coeff和直流偏置电压v\_bias\_dc计算有效折射率neff，体现电压对折射率的调制。

相位计算：基于有效折射率、工作波长（env.wavelength）和微环物理尺寸（隐含11.0μm 相关参数），计算光在微环中传输的相位差，同时考虑了波长偏离中心波长时的折射率修正（dneff1相关项）。

传输系数计算：结合耦合系数（sigma）和损耗系数（alpha，含波长修正项），计算光从opt\_0到opt\_1（及反向）的传输系数through，并赋值给 S 矩阵的对应元素（S["opt\_0", "opt\_1"]和S["opt\_1", "opt\_0"]）。

S 矩阵的计算直接反映了器件在频域下的光学传输特性，是静态或准静态仿真的核心。

4.时域模型

这部分描述了模型在时域下的信号响应和状态演化，用于动态仿真：

信号传输（calculate\_signals）：  
定义了光输出信号与输入信号及状态变量的关系。通过耦合系数kappa（由sigma推导，kappa = np.sqrt(1.0 - parameters.sigma\*\*2)），将输出端口（opt\_1、opt\_0）的光信号与输入信号（opt\_0、opt\_1）及环内双向光场（e\_ccw、e\_cw）关联，体现光在环内与端口间的耦合。

状态演化（calculate\_dydt）：  
通过微分方程描述状态变量随时间的变化（dydt），核心包括：

调制电压获取：计算阳极与阴极的电压差v\_bias\_mod，作为实际调制信号。

共振频率调谐：基于状态变量vj和omega\_r\_coeff，计算实时共振角频率omega\_r，体现电压对共振频率的调制。

动态方程构建：分别推导e\_ccw、e\_cw（环内光场）和vj（结电压）的时间导数，考虑了频率失配（delta\_omega）、损耗（decay）、光场耦合等因素，最终形成完整的时域动态模型。

这部分是实现高速调制、瞬态响应等动态仿真的关键，直接描述了器件在时变信号激励下的行为。

## 4.13 PhotoDetector 紧凑模型程序设计

PhotoDetector的紧凑模型程序如下：

class CompactModel\_photodetector(i3.CompactModel):  
  
 parameters = [  
 "Dark\_Current\_at\_m1V\_nA",  
 "Dark\_Current\_at\_m2V\_nA",  
 "Responsivity\_at\_m1V\_A\_per\_W",  
 "Series\_Resistance\_Ohm",  
 "Ideality\_Factor",  
 "bw\_coeff",  
 "temperature",  
 "\_3dB\_BW\_at\_0V\_GHz",  
 "s21\_drop\_at\_50GHz\_at\_0V\_dB",  
 "\_3dB\_BW\_at\_m1V\_GHz",  
 "s21\_drop\_at\_50GHz\_at\_m1V\_dB",  
 "\_3dB\_BW\_at\_m2V\_GHz",  
 "s21\_drop\_at\_50GHz\_at\_m2V\_dB",  
 ]  
  
 terms = [  
 i3.OpticalTerm(name="opt\_0"),  
 i3.OpticalTerm(name="opt\_1"),  
 i3.ElectricalTerm(name="cathode\_0"),  
 i3.ElectricalTerm(name="anode\_0"),  
 ]  
  
 states = ["vj"]  
  
 def calculate\_signals(parameters, env, output\_signals, y, t, input\_signals):  
 rs = parameters.Series\_Resistance\_Ohm  
 vbias = (input\_signals["anode\_0"] - input\_signals["cathode\_0"]).real  
 i\_rs = (vbias - y["vj"].real) / rs  
 output\_signals["anode\_0"] = i\_rs  
 output\_signals["cathode\_0"] = -i\_rs  
  
 def calculate\_dydt(parameters, env, dydt, y, t, input\_signals):  
 v\_bias = (input\_signals["anode\_0"] - input\_signals["cathode\_0"]).real  
 rs = parameters.Series\_Resistance\_Ohm  
 bw = polyval(parameters.bw\_coeff, v\_bias) \* 1.0e9  
 rho = parameters.Responsivity\_at\_m1V\_A\_per\_W  
 id\_factor = parameters.Ideality\_Factor  
 p\_opt = np.abs(input\_signals["opt\_0"] + input\_signals["opt\_1"]) \*\* 2  
 i\_ph = rho \* p\_opt  
 idark\_a = parameters.Dark\_Current\_at\_m1V\_nA - parameters.Dark\_Current\_at\_m2V\_nA  
 idark = idark\_a \* v\_bias + 2.0 \* parameters.Dark\_Current\_at\_m1V\_nA - parameters.Dark\_Current\_at\_m2V\_nA  
 i\_diode = idark \* 1e-9 \* (np.exp((e \* y["vj"].real) / (id\_factor \* k \* parameters.temperature)) - 1)  
 i\_rs = (v\_bias - y["vj"].real) / rs  
 dydt["vj"] = (i\_ph + i\_rs - i\_diode) \* 2.0 \* pi \* rs \* bw

PhotoDetector 的紧凑模型程序主要包含以下五个部分：

1. 模型参数

该光电探测器紧凑模型包含以下关键参数：

暗电流参数：

* Dark\_Current\_at\_m1V\_nA：-1V偏置下的暗电流(nA)
* Dark\_Current\_at\_m2V\_nA：-2V偏置下的暗电流(nA)

响应度参数

* Responsivity\_at\_m1V\_A\_per\_W：-1V偏置下的响应度(A/W)，表示单位光功率产生的光电流

电学特性参数

* Series\_Resistance\_Ohm：串联电阻(Ω)
* Ideality\_Factor：理想因子，反映PN结的非理想特性
* \_3dB\_BW\_at\_0V\_GHz：0V偏置下的3dB带宽(GHz)
* \_3dB\_BW\_at\_m1V\_GHz：-1V偏置下的3dB带宽(GHz)
* \_3dB\_BW\_at\_m2V\_GHz：-2V偏置下的3dB带宽(GHz)

频率响应参数

* s21\_drop\_at\_50GHz\_at\_0V\_dB：0V偏置下50GHz频率的S21下降(dB)
* s21\_drop\_at\_50GHz\_at\_m1V\_dB：-1V偏置下50GHz频率的S21下降(dB)
* s21\_drop\_at\_50GHz\_at\_m2V\_dB：-2V偏置下50GHz频率的S21下降(dB)

其他参数

* bw\_coeff：带宽随偏置电压变化的系数
* temperature：工作温度

2. 模型端口

该模型包含以下端口：

* **光学端口**：opt\_0和opt\_1 - 用于光信号的输入
* **电学端口**：
  + cathode\_0和anode\_0 - 用于输出光电流和施加偏置电压

3. 状态变量

vj：结电压，是模型的核心状态变量

4. 信号计算模型

calculate\_signals函数实现了光电探测器的信号计算：

计算偏置电压和串联电阻电流：

vbias = (input\_signals["anode\_0"] - input\_signals["cathode\_0"]).real

i\_rs = (vbias - y["vj"].real) / rs

设置输出电信号：

output\_signals["anode\_0"] = i\_rs

output\_signals["cathode\_0"] = -i\_rs

5. 状态变量微分计算

calculate\_dydt函数实现了状态变量的时间演化：

计算偏置电压和带宽：

v\_bias = (input\_signals["anode\_0"] - input\_signals["cathode\_0"]).real

bw = polyval(parameters.bw\_coeff, v\_bias) \* 1.0e9

计算光电流：

p\_opt = np.abs(input\_signals["opt\_0"] + input\_signals["opt\_1"]) \*\* 2

i\_ph = rho \* p\_opt

计算暗电流：

idark\_a = parameters.Dark\_Current\_at\_m1V\_nA - parameters.Dark\_Current\_at\_m2V\_nA

idark = idark\_a \* v\_bias + 2.0 \* parameters.Dark\_Current\_at\_m1V\_nA - parameters.Dark\_Current\_at\_m2V\_nA

i\_diode = idark \* 1e-9 \* (np.exp((e \* y["vj"].real) / (id\_factor \* k \* parameters.temperature)) - 1)

计算结电压的微分：

dydt["vj"] = (i\_ph + i\_rs - i\_diode) \* 2.0 \* pi \* rs \* bw

# 5 验证程序设计

在本次聚模型精度的验证工作中，我们构建了一套严谨且科学的验证体系。其核心在于通过全方位、深层次地将模型链路仿真结果与验证链路测试结果进行比对，从而实现对模型精度的精准校验。​

具体实施过程如下：首先，在器件模型构建阶段，我们依据设计的参数提取线路，并通过大量的实验数据采集与分析，运用统计学方法对器件的模型参数进行建模，以此为基础搭建起高度精确的器件行为模型。​

紧接着，基于所构建的器件行为模型，我们进一步搭建了一个 6 比特的可调延时线。该延时线的结构设计极为精巧，由 6 个光开关、特定长度与结构的波导以及瘠波导共同构成。其中，光开关作为延时线中的关键控制元件，其内部结构由 MMI2x2（多模干涉 2x2）器件以及热调相移器组合而成。MMI2x2 利用多模干涉原理，实现对光信号的高效分束与合束操作，确保信号在不同路径间的准确传输；热调相移器则通过精确控制温度变化，进而对光信号的相位进行精细调节，以满足延时线对相位控制的严格要求。​

在完成延时线的搭建后，我们运用 ipkiss 的线路仿真引擎 Caphe 对该验证链路的时延和相位特性展开仿真分析。Caphe 引擎基于成熟的电磁场理论以及先进的数值计算方法，能够精确模拟光信号在复杂链路结构中的传输行为。在仿真过程中，充分考虑了光信号在波导中传输时的损耗、色散等物理现象，以及光开关切换过程中的时间延迟与相位变化。通过对大量不同输入信号条件下的仿真计算，生成了详细的时延和相位特性数据。​

与此同时，我们从 cumec 获取了与之对应的测试数据，并组织专业的数据分析团队，运用先进的数据分析工具与统计学方法，将基于 Caphe 引擎仿真得到的时延和相位特性数据与 cumec 提供的测试数据进行全面、细致的对比。通过对比关键指标数值的差异、分析时延和相位变化趋势的一致性以及评估误差范围的合理性，最终对模型的精度做出准确验证，为模型在实际应用中的性能表现提供坚实的数据支撑与科学依据。

## 5.1验证线路版图程序设计

在先进的光通信与光信号处理系统中，6 比特可调延时线扮演着至关重要的角色，而光开关则是该延时线的核心关键组成部分。此光开关采用了精妙的设计架构，其主体由两个 2x2 多模干涉（MMI）器件共同构建成马赫 - 曾德尔干涉仪（MZI）结构。在这一 MZI 结构中，两个 2x2 MMI 器件犹如精密的信号分配与整合枢纽，它们协同工作，精准地引导光信号在不同光路间进行传输与交互。

尤为关键的是，MZI 结构的其中一个臂上集成了热调相移器。热调相移器作为实现光信号相位精确调控的核心元件，利用材料的热光效应这一物理特性来发挥作用。当电流通过热调相移器时，会产生热量，进而改变周围材料的温度。由于材料的折射率会随着温度的变化而改变，光信号在通过该臂时，其相位便会相应地发生精确调整。这种基于热光效应的相位调控方式，具有响应速度快、调控精度高的优势，能够根据系统的实时需求，灵活且精准地改变光信号的相位，从而确保整个 6 比特可调延时线能够稳定、高效地运行，为光通信系统中的信号同步、时延控制以及光信号处理算法的准确执行提供坚实保障 。

### 5.1.1光开关的设计

接下来，基于ipkiss的线路设计函数i3.Circuit来介绍光开关的实现过程。其设计过程分为6步：

1、**导入模块**：

from csip130c import all as pdk

from ipkiss3 import all as i3

这部分代码从 csip130c 模块中导入所有内容并将其命名为 pdk，从 ipkiss3 模块中导入所有内容并将其命名为 i3。这些模块可能包含了用于光子集成电路设计和仿真的各种类、函数和工具。

2、**类定义**：

class Switch(i3.Circuit):

"""This creates a Switch basic unit, made of tunable MZI.Users can specify the length of heater in one of MZI arm，They can also specify the length differences between the arms"""

\_name\_prefix = " Switch "

定义了一个名为 Switch 的类，它继承自 i3.Circuit。类的文档字符串描述了该类的功能：创建一个由可调马赫 - 曾德尔干涉仪（MZI）组成的基本单元，用户可以指定 MZI 一个臂中加热器的长度以及两臂之间的长度差。\_name\_prefix 是一个类属性，用于设置单元名称的前缀。

1. **属性定义**：

dc = i3.ChildCellProperty(doc="Directional coupler in circuit")

heater = i3.ChildCellProperty(doc="Heater in circuit ")

waveguide = i3.ChildCellProperty(doc="Waveguide in circuit ")

wg\_buffer\_dx = i3.PositiveNumberProperty(

default=10.0, doc="Extra non-heated waveguide length around heater"）

heater\_length = i3.LockedProperty(doc="Length of heater in arm")

two\_arm\_length\_difference = i3.NumberProperty(

default=0, doc="The length difference between two arm"

bend\_radius = i3.PositiveNumberProperty(

default=50, doc="The bend radius of the routing waveguide"

)

trace\_template = i3.TraceTemplateProperty(

doc="The trace template for waveguide and heater", locked=True

)

state = i3.IntProperty(

default=0, restriction=i3.RestrictValueList(allowed\_values=[0, 1])

)

vbias = i3.NumberProperty(doc="DC voltage bias on the MZI (for S-matrix only)")

这部分定义了类的多个属性，用于描述电路的不同部分和参数：

dc：定向耦合器，类型为 ChildCellProperty。

heater：加热器，类型为 ChildCellProperty。

waveguide：波导，类型为 ChildCellProperty。

wg\_buffer\_dx：加热器周围非加热波导的额外长度，默认值为 10.0，类型为 PositiveNumberProperty。

heater\_length：臂中加热器的长度，类型为 LockedProperty。

two\_arm\_length\_difference：两臂之间的长度差，默认值为 0，类型为 NumberProperty。

bend\_radius：路由波导的弯曲半径，默认值为 50，类型为 PositiveNumberProperty。

trace\_template：波导和加热器的迹线模板，类型为 TraceTemplateProperty，并且是锁定的。

state：状态，只能取 0 或 1，默认值为 0，类型为 IntProperty。

vbias：MZI 上的直流电压偏置（仅用于 S 矩阵），类型为 NumberProperty。

1. **默认值方法**：

def \_default\_vbias(self):

if self.state == 0:

v = 2.215

elif self.state == 1:

v = 1.28

return v

def \_default\_dc(self):

return pdk.Fixed\_M2X2\_TE\_1550()

def \_default\_heater(self):

ht = pdk.Fixed\_MH\_TE\_1550()

ht.CircuitModel(vbias=self.vbias)

return ht

def \_default\_waveguide(self):

wg = i3.Waveguide(trace\_template=self.trace\_template)

wg.Layout(shape=[(0.0, 0.0), (self.heater\_length, 0.0)])

return wg

def \_default\_heater\_length(self):

if hasattr(self.heater, "heater\_length"):

return self.heater.heater\_length

else:

heater\_size = self.heater.get\_default\_view(i3.LayoutView).size\_info()

return heater\_size.width

def \_default\_trace\_template(self):

return pdk.SWG450()

这些方法用于设置属性的默认值：

\_default\_vbias：根据 state 属性的值返回不同的电压偏置值。

\_default\_dc：返回一个固定的定向耦合器实例。

\_default\_heater：返回一个固定的加热器实例，并设置其电路模型的电压偏置。

\_default\_waveguide：创建一个波导实例，并根据加热器长度设置其形状。

\_default\_heater\_length：尝试获取加热器的长度属性，如果不存在则根据加热器的布局视图获取其宽度。

\_default\_trace\_template：返回一个默认的迹线模板实例。

5、**默认实例和器件放置方法**：

def \_default\_insts(self):

insts = {

"dc\_1": self.dc,

"dc\_2": self.dc,

"ht": self.heater,

"rg": self.waveguide,

}

return insts

def \_default\_specs(self):

r = self.bend\_radius

dx = self.wg\_buffer\_dx

specs = [

i3.Place("dc\_1:in\_1", (0.0, 0.0)),

i3.Place("ht:in", (2.0 \* r + dx, 2.0 \* r), relative\_to="dc\_1:out\_1"),

i3.Place(

"rg:in",

(2.0 \* r + dx, -2.0 \* r - self.two\_arm\_length\_difference / 2),

relative\_to="dc\_1:out\_2",

),

i3.Place("dc\_2:in\_1", (2.0 \* r + dx, -2.0 \* r), relative\_to="ht:out"),

]

return specs

\_default\_insts：返回一个包含电路中各个组件实例的字典。

\_default\_specs：定义了电路中各个组件的放置和连接规格，包括位置和连接路径，使用了 i3.Place 和 i3.ConnectManhattan 方法。

1. **默认暴露端口方法**：

def \_default\_exposed\_ports(self):

exposed\_ports = {

"dc\_1:in\_1": "in2",

"dc\_1:in\_2": "in1",

"dc\_2:out\_1": "out2",

"dc\_2:out\_2": "out1",

"ht:elec\_in": "ht1",

"ht:elec\_out": "ht2",

}

return exposed\_ports

这个方法返回一个字典，定义了电路中暴露的端口及其别名。

基于以上6步，就可以实现如图4-1所示的光开关的设计。

图表

AI 生成的内容可能不正确。

图4-1-1 光开关版图的设计

### 5.1.2可调延时线的设计

可调延时线的原理图如图4-3所示，其由7个光开关以不同长度以及不同类型波导级联而成。为实现该可调延时线的仿真，首先需要在ipkiss中构建版图。

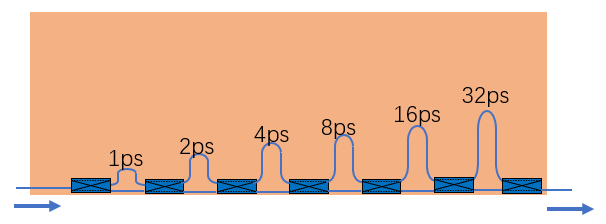


图4-1-2延时线的原理图

我们同样用i3.Circuit的线路设计函数来实现可调延时线的版图。

l\_tops = i3.ListProperty(  
 default=[874.231, 1548.435, 2996.880, 5693.820, 13190.042, 25121.491],  
 doc="List of top lengths",  
)  
l\_bots = i3.ListProperty(  
 default=[200, 200, 300, 300, 959.5, 659.5],  
 doc="List of botom lengths",  
)

代码中分别定义了，延时线上下波导的长度。这个长度与cumec设计的测试线路的长度保持统一。为方便该可调延时线的仿真，同样定义了线路中开光的不同状态:

states = i3.ListProperty(  
 default=[0, 0, 0, 0, 0, 0, 0],  
 doc="List of top lengths",  
)

同时为了使可调延时线的前4个连接波导为条波导，后两个链接波导为瘠波导，代码中也做了相应的设置：

p1 = f"dc\_{cnt}:out2"  
p2 = f"dc\_{cnt + 1}:in2"  
if cnt < 4:  
 specs.append(  
 i3.ConnectManhattan(  
 p1,  
 p2,  
 trace\_template=SWG500(),  
 bend\_radius=self.bend\_radius,  
 )  
 )  
  
else:  
 specs.append(  
 i3.ConnectManhattan(  
 p1,  
 p2,  
 trace\_template=SWG600(),  
 bend\_radius=self.bend\_radius,  
 )  
 )

其中SWG500代表条波导，SWG600代表瘠波导。

最后实现的可调延时线的版图如下：

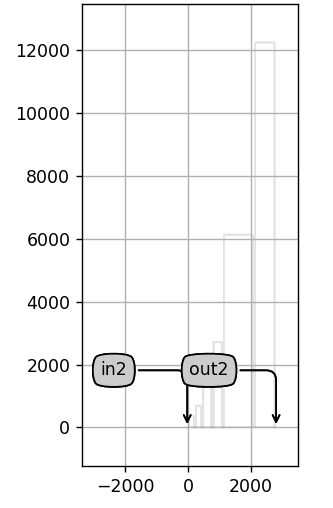


图4-1-3 可调延时线的版图

## 5.2验证线路仿真程序设计

### 5.2.1 光开关仿真程序设计

热调相移器作为光开关中的关键组件，其两端电压的变化会对光开关的性能产生显著影响。当热调相移器两端的电压处于零值时，光开关内部的两个臂不存在初始的相位差。从光学原理角度来看，在这种情况下，光信号在进入光开关后，会均匀地分配到各个输出端口。具体而言，此时的光开关等效于一个 3dB 的分束器，能够将输入光信号平均地分成两部分，分别从不同的输出端口输出，使得两个输出端口的光功率大致相等。

随着热调相移器两端电压逐渐增大，光开关的工作状态开始发生变化，不同的开关状态之间会相互切换。这一过程涉及到复杂的光学物理机制，热调相移器通过改变材料的温度，进而改变材料的折射率，最终实现对光信号相位的精确调控，从而改变光信号在光开关内部的传输路径和分配方式。

例如，当热调相移器两端的电压达到 1.28 V 时，光开关呈现出一种特定的工作状态。在这种状态下，经过详细的仿真分析和理论计算可知，只有 out2 端口有光输出，而其他端口无光信号输出。这种状态被定义为 Cross 状态，在该状态下，光信号在光开关内部发生了特定的干涉和传输路径变化，使得所有光能量都集中输出到了 out2 端口。

同样地，当热调相移器两端的电压升高至 2.215 V 时，光开关又切换到了另一种状态 — Bar 状态。在 Bar 状态下，光开关的特性表现为只有 out1 端口有光输出，out2 端口则没有光信号输出。这一状态的实现同样依赖于热调相移器对光信号相位的精确调控，通过改变电压调整热调相移器的工作状态，进而改变光信号在光开关内部的干涉和传输情况，使得光能量全部集中到 out1 端口输出。

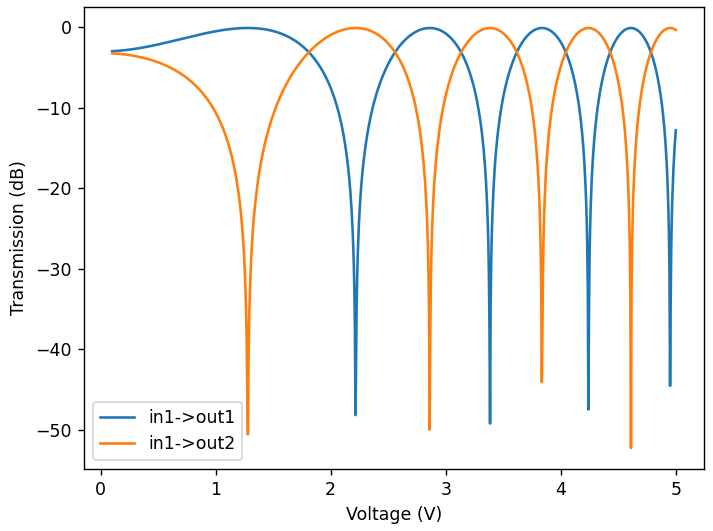


图4-2-1 光开关的仿真结果

因此，在定义光开关属性时，可以用0和1来表征不同的开关状态，方便6比特可调延时线的仿真：

state = i3.IntProperty(  
 default=0, restriction=i3.RestrictValueList(allowed\_values=[0, 1])  
)  
vbias = i3.NumberProperty(doc="DC voltage bias on the MZI (for S-matrix only)")  
  
def \_default\_vbias(self):  
 if self.state == 0:  
 v = 2.215 *# Bar* elif self.state == 1:  
 v = 1.28 *# Cross* return v

### 5.2.2 可调延时线的仿真程序设计

构建好延时线的版图，就可以利用ipkiss的线路仿真功能来验证延时线的性能。

首先可以来验证延时线的时延特性。

为得到不同状态下的时延，首先需要定义仿真参数，如中心波长扫频范围等。

center\_wavelength = 1.55  
dl = 1e-10  
wavelengths = np.linspace(center\_wavelength - dl, center\_wavelength + dl, 10)  
  
domega = (  
 2 \* pi \* c / (wavelengths \* 1e-6) / 1e9  
 - 2 \* pi \* c / (center\_wavelength \* 1e-6) / 1e9  
)

借助 Python 的 itertools.product 函数生成由 0 和 1 组成的所有可能的 7 位二进制序列，可以来仿真该可调延时线的所以状态。

states = [list(comb) for comb in itertools.product([0, 1], repeat=7)]

通过遍历这些状态，可以得到不同状态下的时延结果：

for cnt, state in enumerate(states):  
 circuit = Tuable\_delay\_line(states=state)  
 cm = circuit.CircuitModel()  
 S = cm.get\_smatrix(wavelengths=wavelengths)  
 angle = np.angle(S["out1", "in1"])  
 slope, intercept = np.polyfit(domega, angle, 1)  
 if slope != 0 and slope > 0:  
 delay\_times.append(slope \* 1e3)  
 cnts.append(a)  
 a += 1

通过代码可以看出时延是通过求不同光角频率于相位的斜率得到的:

slope, intercept = np.polyfit(domega, angle, 1)

时延的仿真结果如图4-5所示，不同状态下时延整体表现良好的线性，且于cumec的实测数据也有较高的一致性，仿真结果和实测数据之间的将在下一节分析。

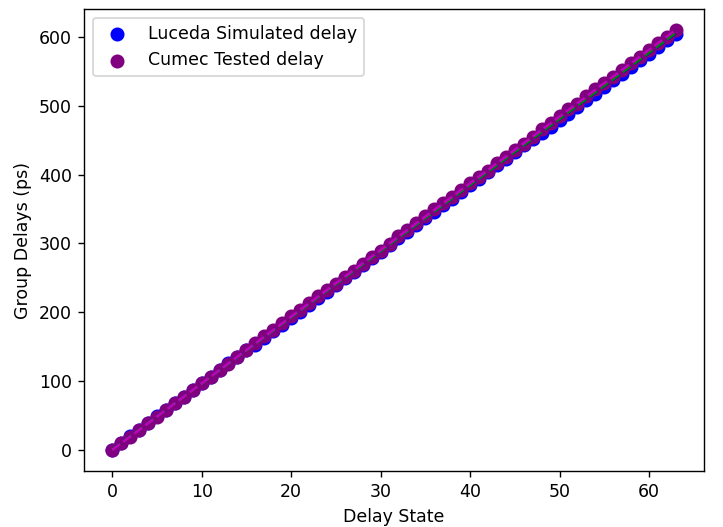


图4-2-2 64总状态下仿真和测试分别得到的时延结果

相位的仿真和时延一样，需要遍历开关所有的状态：

for cnt, state in enumerate(states):  
 circuit = Tuable\_delay\_line(states=state)  
 cm = circuit.CircuitModel()  
 S = cm.get\_smatrix(wavelengths=wavelengths)  
 angle = np.angle(S["out1", "in1"])  
 unwrapped\_angle = np.unwrap(angle)  
 phases.append(unwrapped\_angle)

为使仿真后的相位平滑，需要将将仿真得到的相位数据中因周期性导致的跳变进行修正：

unwrapped\_angle = np.unwrap(angle)

最后，64总状态下仿真和测试分别得到的相位结果如何4-6所示，相位于频率之间呈现应有的线性关系，且于测试结果保持较高的一致性，当时延量较大时，由于误差累计，导致仿真结果与测试结果出现了一定的偏差，但是整体结果还是保持了较高的一致性。

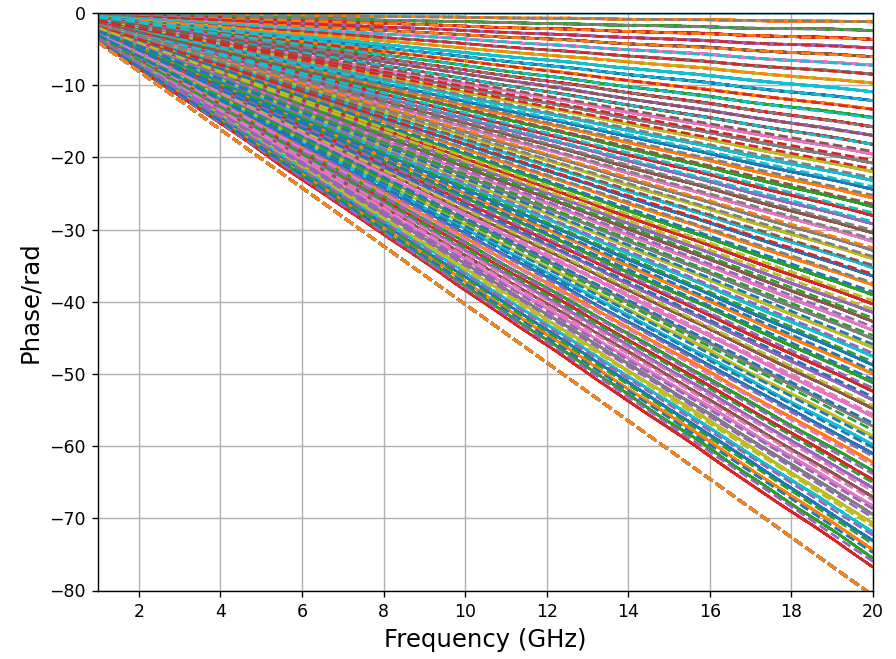


图4-2-3 64总状态下仿真和测试分别得到的相位结果

# 6 测试数据处理程序设计

光芯片技术飞速发展的当下，自动化测试产生的海量数据是衡量芯片性能、保障产品质量的核心依据。而测试数据预处理作为数据处理的首道关键工序，如同精密仪器的校准环节，直接决定着后续分析的效率与结果准确性，对推动硅光芯片技术创新与产业应用意义重大。​

在硅光芯片自动化测试的数据处理流程中，测试数据分类与可视化是两大核心支柱。测试数据分类需围绕芯片特性，基于光功率、波长、插入损耗、偏振相关损耗等专业参数，以及测试环境、测试设备型号等维度，构建针对性分类体系。例如，按光功率波动范围将数据划分为正常区间、临界区间与异常区间，或根据不同测试设备产生的数据进行归类。这种精细化分类，能快速筛除无效数据，精准定位异常测试结果，在数据清洗、特征提取阶段，大幅减少冗余计算，显著提升处理效率，让工程师能够更高效地聚焦关键数据，加速问题排查。​

测试数据可视化则是解锁硅光芯片性能密码的重要手段。通过专业图表，如用折线图呈现芯片在不同温度下的光功率变化趋势，以散点图展示波长与插入损耗的关系，借助热力图直观呈现不同区域芯片的性能分布，可将复杂抽象的测试数据转化为直观图像。比如，在分析一批硅光芯片的偏振相关损耗数据时，利用箱线图能快速识别数据中的离群值，定位性能异常的芯片；通过瀑布图展示不同工艺步骤对芯片损耗的影响，帮助工程师清晰洞察各环节对芯片性能的作用。可视化不仅降低了数据解读难度，还能辅助工程师发现潜在规律，为优化芯片设计、改进制造工艺提供直观的数据支撑，从根源上保障数据处理结果的准确性。​

测试数据分类与可视化相辅相成，共同服务于硅光芯片测试。分类为可视化提供规整的数据基础，让可视化结果更具分析价值；可视化通过直观呈现，反馈分类的合理性，辅助优化分类规则。二者协同运作，为硅光芯片性能评估、质量管控筑牢根基，在推动芯片技术迭代、加速产品量产等方面发挥关键作用。

## 6.1 测试数据分类程序设计

CUMEC发送的自动化测试的数据，保存在C1、C2、C3、C4、O1、O2、O3以及O4这样命名的文件夹中，并分别通过C\_recipe\_position\_C1\_C4.xlsx 和O\_recipe\_position\_O1\_O4.xlsx 这两表格文件标注了不同DUT的位置信息，为了方便数据的自动化处理，我们通过如下分类代码对测试数据进行了分类：

class strip\_mzi\_c:  
 def value(self):  
 return [0, 50, 100, 150]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C1"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class crossing\_c:  
 def value(self):  
 return [-1800, -1750, -1700, -1650]  
  
 def columns\_to\_remove(self):  
 return [2, 3] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C1"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class gc\_c:  
 def value(self):  
 return [-1800, -1750, -1700, -1650]  
  
 def columns\_to\_remove(self):  
 return [0, 1] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C1"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class mmi1x2\_c:  
 def value(self):  
 return [-2400, -2350, -2300, -2250, -2200]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C1"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class mmi2x2\_c:  
 def value(self):  
 return [-2438.225, -2388.225, -2338.225, -2288.225, -2238.225]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C1"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class strip\_spiral\_c:  
 def value(self):  
 return [-3003.655, -2953.655, -2903.655, -2853.655]  
  
 def columns\_to\_remove(self):  
 return [2, 3] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C1"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class strip\_bend\_c:  
 def value(self):  
 return [-3003.655, -2953.655, -2903.655, -2853.655]  
  
 def columns\_to\_remove(self):  
 return [0, 1] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C1"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class tr\_sm\_c:  
 def value(self):  
 return [-4767.674, -4717.674, -4667.674]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C2"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class tr\_sr\_c:  
 def value(self):  
 return [-4767.775, -4717.775, -4667.775]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C2"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class rib\_mzi\_c:  
 def value(self):  
 return [-5707, -5657, -5607, -5557]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C2"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class rib\_bend\_c:  
 def value(self):  
 return [-8181.325, -8131.325, -8081.325, -8031.325]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C2"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class ring\_c:  
 def value(self):  
 return [-9744, -9694, -9644, -9594]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C2"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class mod\_mzi\_c:  
 def value(self):  
 return [0, 50, 100, 150]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C3"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class mod\_bend\_c:  
 def value(self):  
 return [-1200, -1150, -1100, -1050]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C3"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class crossing\_crosstalk\_c:  
 def value(self):  
 return [-1200, -1150, -1100, -1050]  
  
 def columns\_to\_remove(self):  
 return [0] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C3"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class rib\_spiral\_c:  
 def value(self):  
 return [-2367.392, -2317.392, -2267.392, -2217.392]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C3"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class mod\_spiral\_c:  
 def value(self):  
 return [-3572.655, -3522.655, -3472.655, -3422.655]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C3"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class dc\_c:  
 def value(self):  
 return [-3330, -3280, -3230, -3180, -3130]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "C3"  
  
 def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"  
  
  
class strip\_mzi\_o:  
 def value(self):  
 return [0, 50, 100, 150]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O1"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class crossing\_o:  
 def value(self):  
 return [-1800, -1750, -1700, -1650]  
  
 def columns\_to\_remove(self):  
 return [3, 4, 5] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O1"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class gc\_o:  
 def value(self):  
 return [-1800, -1750, -1700, -1650]  
  
 def columns\_to\_remove(self):  
 return [0, 1, 2] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O1"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class mmi1x2\_o:  
 def value(self):  
 return [-2400, -2350, -2300, -2250, -2200]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O2"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class mmi2x2\_o:  
 def value(self):  
 return [-2438.19, -2388.19, -2338.19, -2288.19, -2238.19]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O2"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class strip\_spiral\_o:  
 def value(self):  
 return [-3003.48, -2953.48, -2903.48, -2853.48]  
  
 def columns\_to\_remove(self):  
 return [3, 4, 5] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O2"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class strip\_bend\_o:  
 def value(self):  
 return [-3003.48, -2953.48, -2903.48, -2853.48]  
  
 def columns\_to\_remove(self):  
 return [0, 1, 2] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O2"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class tr\_sm\_o:  
 def value(self):  
 return [-4767.57, -4717.57, -4667.57]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O2"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class tr\_sr\_o:  
 def value(self):  
 return [-4767.67, -4717.67, -4667.67]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O2"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class rib\_mzi\_o:  
 def value(self):  
 return [-5707, -5657, -5607, -5557]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O3"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class rib\_bend\_o:  
 def value(self):  
 return [-8181.325, -8131.325, -8081.325, -8031.325]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O3"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class ring\_o:  
 def value(self):  
 return [-9744, -9694, -9644, -9594]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O3"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class mod\_mzi\_o:  
 def value(self):  
 return [-600, -550, -500, -450]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O4"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class mod\_bend\_o:  
 def value(self):  
 return [-1200, -1150, -1100, -1050]  
  
 def columns\_to\_remove(self):  
 return [1, 2] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O4"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class crossing\_crosstalk\_o:  
 def value(self):  
 return [-1200, -1150, -1100, -1050]  
  
 def columns\_to\_remove(self):  
 return [0] # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O4"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class rib\_spiral\_o:  
 def value(self):  
 return [-2367.392, -2317.392, -2267.392, -2217.392]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O4"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class mod\_spiral\_o:  
 def value(self):  
 return [-3572.655, -3522.655, -3472.655, -3422.655]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O4"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class dc\_o:  
 def value(self):  
 return [-3880, -3830, -3780, -3730, -3680]  
  
 def columns\_to\_remove(self):  
 return None # for AA\*A\*\*AA\*A\*\* distribution  
  
 def sheet\_name(self):  
 return "O4"  
  
 def table\_data(self):  
 return "O\_recipe\_position\_O1\_O4.xlsx"  
  
  
class wafer:  
 def wafer\_map(self):  
 return {  
 "1": (-5, -2),  
 "2": (-4, -2),  
 "3": (-3, -2),  
 "4": (-2, -2),  
 "5": (-1, -2),  
 "6": (0, -1),  
 "7": (-1, -1),  
 "8": (-2, -1),  
 "9": (-3, -1),  
 "10": (-4, -1),  
 "11": (-5, -1),  
 "12": (-6, -1),  
 "13": (-6, 0),  
 "14": (-5, 0),  
 "15": (-4, 0),  
 "16": (-3, 0),  
 "17": (-2, 0),  
 "18": (-1, 0),  
 "19": (0, 0),  
 "20": (0, 1),  
 "21": (-1, 1),  
 "22": (-2, 1),  
 "23": (-3, 1),  
 "24": (-4, 1),  
 "25": (-5, 1),  
 "26": (-6, 1),  
 "27": (-6, 2),  
 "28": (-5, 2),  
 "29": (-4, 2),  
 "30": (-3, 2),  
 "31": (-2, 2),  
 "32": (-1, 2),  
 "33": (0, 2),  
 "34": (0, 3),  
 "35": (-1, 3),  
 "36": (-2, 3),  
 "37": (-3, 3),  
 "38": (-4, 3),  
 "39": (-5, 3),  
 "40": (-6, 3),  
 "41": (-4, 4),  
 "42": (-3, 4),  
 "43": (-2, 4),  
 }  
def get\_all\_classes():  
 classes = []  
 for obj\_name, obj in globals().items():  
 if isinstance(obj, type):  
 classes.append(obj)  
 return classes  
  
  
\_\_all\_\_ = get\_all\_classes()

对于每一个测试单元(DUT)，均定义了四种方法来对测试数据进行定位。

以C波段Strip类型的Spiral为例：

首先定义了 value, value是自动化测试时，光纤对的纵坐标，基于该坐标就可以找到对应的DUT。

def value(self):  
 return [-3003.655, -2953.655, -2903.655, -2853.655]

其次是columns\_to\_remove，在拼版时会有不同的DUT平行摆放，为了区分，所以设置了columns\_to\_remove这个参数。

def columns\_to\_remove(self):  
 return [2, 3]

接下来是sheet\_name, 根据它可以找到DUT在分类表格中对应的sheet。

def sheet\_name(self):  
 return "C1"

最后是table\_data， table\_data为C波段和O波段DUT坐标信息的表格文件。

def table\_data(self):  
 return "C\_recipe\_position\_C1\_C4.xlsx"

除此之外，还有wafer\_map的定义，将8英寸晶圆的43个shot，1到43的编号与shot的坐标进行映射，就可以用来找到对应shot上DUT的数据和用来绘制基于位置分布的Wafer map。

class wafer:  
 def wafer\_map(self):  
 return {…}

## 6.2 测试数据可视化程序设计

测试数据分类好之后，就可以利用python 提供的交互式绘图包：PyQt6，来查看测试数据和初步的数据处理,设计代码如下：

import sys  
import pandas as pd  
import matplotlib.pyplot as plt  
from PyQt6.QtWidgets import (  
 QApplication,  
 QWidget,  
 QVBoxLayout,  
 QHBoxLayout,  
 QComboBox,  
 QLabel,  
 QGridLayout,  
 QPushButton,  
 QSizePolicy,  
 QSpacerItem,  
)  
from matplotlib.backends.backend\_qtagg import FigureCanvasQTAgg as FigureCanvas  
from date\_classification import all as dt  
from common\_file.DUT\_parse import DUT  
import numpy as np  
from scipy.signal import argrelextrema  
from PyQt6.QtGui import QPainter, QPen, QBrush  
from PyQt6.QtCore import Qt  
  
  
class DataPlotApp(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("csip130c test data")  
 self.setGeometry(100, 100, 800, 600)  
  
 # Component list  
 components = [  
 "strip\_mzi\_c",  
 "crossing\_c",  
 "gc\_c",  
 "mmi1x2\_c",  
 "mmi2x2\_c",  
 "strip\_spiral\_c",  
 "strip\_bend\_c",  
 "tr\_sm\_c",  
 "tr\_sr\_c",  
 "rib\_mzi\_c",  
 "rib\_bend\_c",  
 "ring\_c",  
 "mod\_mzi\_c",  
 "mod\_bend\_c",  
 "crossing\_crosstalk\_c",  
 "rib\_spiral\_c",  
 "mod\_spiral\_c",  
 "dc\_c",  
 "strip\_mzi\_o",  
 "crossing\_o",  
 "gc\_o",  
 "mmi1x2\_o",  
 "mmi2x2\_o",  
 "strip\_spiral\_o",  
 "strip\_bend\_o",  
 "tr\_sm\_o",  
 "tr\_sr\_o",  
 "rib\_mzi\_o",  
 "rib\_bend\_o",  
 "ring\_o",  
 "mod\_mzi\_o",  
 "mod\_bend\_o",  
 "crossing\_crosstalk\_o",  
 "rib\_spiral\_o",  
 "mod\_spiral\_o",  
 "dc\_o",  
 ]  
 # Create the main layout  
 main\_layout = QVBoxLayout()  
 # Create the top layout  
 top\_layout = QHBoxLayout()  
 top\_layout.setSpacing(30)  
  
 # Create font  
 font = QLabel().font()  
 font.setPointSize(10)  
 font.setBold(True)  
 font.setFamily("Arial")  
  
 # Create combo boxes  
 self.comboBox = QComboBox()  
 self.comboBox.addItems(components)  
 self.comboBox\_a = QComboBox()  
 self.comboBox\_a.addItems([str(i) for i in range(5)])  
 self.comboBox\_dut\_number = QComboBox()  
 self.comboBox\_dut\_number.addItems([str(i) for i in range(3)])  
 self.comboBox\_order = QComboBox()  
 self.comboBox\_order.addItems(  
 [str(i) for i in [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200]]  
 )  
  
 # Add components to the top layout  
 top\_layout.addWidget(QLabel("Component:", font=font))  
 top\_layout.addWidget(self.comboBox)  
 top\_layout.addWidget(QLabel("A:", font=font))  
 top\_layout.addWidget(self.comboBox\_a)  
 top\_layout.addWidget(QLabel("DUT Number:", font=font))  
 top\_layout.addWidget(self.comboBox\_dut\_number)  
 top\_layout.addWidget(QLabel("Order:", font=font))  
 top\_layout.addWidget(self.comboBox\_order)  
  
 main\_layout.addLayout(top\_layout)  
  
 # Create the plotting area  
 self.figure, self.ax = plt.subplots()  
 self.canvas = FigureCanvas(self.figure)  
  
 # 设置绘图区域的大小策略  
 sizePolicy = QSizePolicy(QSizePolicy.Policy.Fixed, QSizePolicy.Policy.Fixed)  
 self.canvas.setSizePolicy(sizePolicy)  
 self.canvas.setFixedSize(  
 4 \* 100, 3 \* 100  
 ) # 这里将英寸转换为像素，假设 1 英寸 = 100 像素  
  
 # 防止布局管理器调整绘图区域大小  
 self.canvas.setMinimumSize(self.canvas.size())  
 self.canvas.setMaximumSize(self.canvas.size())  
  
 # Create wafer ID buttons  
 self.wafer\_buttons = []  
 wafer\_grid\_layout = QGridLayout()  
 button\_size = 30  
 custom\_positions = [  
 dt.wafer().wafer\_map().get(f"{i + 1}", []) for i in range(43)  
 ]  
 min\_x = min(pos[0] for pos in custom\_positions)  
 min\_y = min(pos[1] for pos in custom\_positions)  
 custom\_positions = [(x - min\_x, y - min\_y) for x, y in custom\_positions]  
  
 for i, (row, col) in enumerate(custom\_positions):  
 button = QPushButton(str(i + 1))  
 button.clicked.connect(lambda \_, idx=i + 1: self.set\_wafer\_id(idx))  
 button.setFixedSize(button\_size, button\_size)  
 self.wafer\_buttons.append(button)  
 wafer\_grid\_layout.addWidget(button, row, col)  
 wafer\_grid\_layout.setHorizontalSpacing(15)  
 wafer\_grid\_layout.setVerticalSpacing(0)  
  
 # Combine the plotting area and button layout  
 canvas\_and\_buttons\_layout = QHBoxLayout()  
 canvas\_and\_buttons\_layout.addWidget(self.canvas)  
 canvas\_and\_buttons\_layout.addLayout(wafer\_grid\_layout)  
  
 main\_layout.addLayout(canvas\_and\_buttons\_layout)  
  
 self.setLayout(main\_layout)  
  
 # Connect signals to slots  
 self.comboBox.currentIndexChanged.connect(self.plot\_data)  
 self.comboBox\_a.currentIndexChanged.connect(self.plot\_data)  
 self.comboBox\_dut\_number.currentIndexChanged.connect(self.plot\_data)  
 self.comboBox\_order.currentIndexChanged.connect(self.plot\_data)  
  
 # Initial wafer ID  
 self.current\_wafer\_id = 1  
 self.wafer\_buttons[0].setStyleSheet("background-color: lightblue")  
  
 # Initial plot  
 self.plot\_data()  
  
 def set\_wafer\_id(self, wafer\_id):  
 self.current\_wafer\_id = wafer\_id  
 for i, button in enumerate(self.wafer\_buttons):  
 button.setStyleSheet(  
 "background-color: lightblue" if i == wafer\_id - 1 else ""  
 )  
 self.plot\_data()  
  
 def find\_local\_maxima(self, x, y, order=10):  
 max\_indices = argrelextrema(y, np.greater, order=order)[0]  
 return x[max\_indices], y[max\_indices]  
  
 def plot\_data(self):  
 selected\_component = self.comboBox.currentText()  
 a = int(self.comboBox\_a.currentText())  
 dut\_number = int(self.comboBox\_dut\_number.currentText())  
 order = int(self.comboBox\_order.currentText())  
 wafer\_id = self.current\_wafer\_id  
  
 component\_map = {  
 "strip\_mzi\_c": dt.strip\_mzi\_c,  
 "crossing\_c": dt.crossing\_c,  
 "gc\_c": dt.gc\_c,  
 "mmi1x2\_c": dt.mmi1x2\_c,  
 "mmi2x2\_c": dt.mmi2x2\_c,  
 "strip\_spiral\_c": dt.strip\_spiral\_c,  
 "strip\_bend\_c": dt.strip\_bend\_c,  
 "tr\_sm\_c": dt.tr\_sm\_c,  
 "tr\_sr\_c": dt.tr\_sr\_c,  
 "rib\_mzi\_c": dt.rib\_mzi\_c,  
 "rib\_bend\_c": dt.rib\_bend\_c,  
 "ring\_c": dt.ring\_c,  
 "mod\_mzi\_c": dt.mod\_mzi\_c,  
 "mod\_bend\_c": dt.mod\_bend\_c,  
 "crossing\_crosstalk\_c": dt.crossing\_crosstalk\_c,  
 "rib\_spiral\_c": dt.rib\_spiral\_c,  
 "mod\_spiral\_c": dt.mod\_spiral\_c,  
 "dc\_c": dt.dc\_c,  
 "strip\_mzi\_o": dt.strip\_mzi\_o,  
 "crossing\_o": dt.crossing\_o,  
 "gc\_o": dt.gc\_o,  
 "mmi1x2\_o": dt.mmi1x2\_o,  
 "mmi2x2\_o": dt.mmi2x2\_o,  
 "strip\_spiral\_o": dt.strip\_spiral\_o,  
 "strip\_bend\_o": dt.strip\_bend\_o,  
 "tr\_sm\_o": dt.tr\_sm\_o,  
 "tr\_sr\_o": dt.tr\_sr\_o,  
 "rib\_mzi\_o": dt.rib\_mzi\_o,  
 "rib\_bend\_o": dt.rib\_bend\_o,  
 "ring\_o": dt.ring\_o,  
 "mod\_mzi\_o": dt.mod\_mzi\_o,  
 "mod\_bend\_o": dt.mod\_bend\_o,  
 "crossing\_crosstalk\_o": dt.crossing\_crosstalk\_o,  
 "rib\_spiral\_o": dt.rib\_spiral\_o,  
 "mod\_spiral\_o": dt.mod\_spiral\_o,  
 "dc\_o": dt.dc\_o,  
 }  
  
 data = pd.DataFrame()  
 if selected\_component in component\_map:  
 data = self.get\_data(  
 component\_map[selected\_component], wafer\_id, a, dut\_number  
 )  
  
 self.\_draw\_plot(data, order)  
  
 def \_draw\_plot(self, data, order):  
 if not data.empty:  
 x = data["X"].to\_numpy()  
 y = data["Y"].to\_numpy()  
 max\_x, max\_y = self.find\_local\_maxima(x, y, order)  
 degree = 4  
 popt1 = np.polyfit(max\_x, max\_y, degree)  
 polynomial = np.poly1d(popt1)  
 y\_fit = polynomial(x)  
 max\_value\_index = np.argmax(y\_fit)  
 max\_value = y\_fit[max\_value\_index]  
 max\_position = x[max\_value\_index]  
  
 # 设置固定大小，防止自动调整  
 self.figure.set\_size\_inches(4, 3) # 固定绘图区域大小  
 self.ax.clear()  
 self.ax.plot(x, y, marker="o", linestyle="-", color="black", markersize=2)  
 self.ax.plot(  
 max\_x, max\_y, marker="o", color="red", markersize=4, linestyle=""  
 )  
 self.ax.plot(x, y\_fit, linestyle="-", color="blue", markersize=4)  
  
 self.ax.set\_xlabel("Wavelength (nm)", fontsize=10, fontfamily="Arial")  
 self.ax.set\_ylabel("Optical Power (dBm)", fontsize=10, fontfamily="Arial")  
 self.ax.tick\_params(axis="x", labelsize=8)  
 self.ax.tick\_params(axis="y", labelsize=8)  
  
 # 固定坐标轴范围，避免自动缩放  
 x\_min, x\_max = np.min(x), np.max(x)  
 y\_min, y\_max = np.min(y), np.max(y)  
 self.ax.set\_xlim(  
 x\_min - 0.1 \* (x\_max - x\_min), x\_max + 0.1 \* (x\_max - x\_min)  
 )  
 self.ax.set\_ylim(  
 y\_min - 0.1 \* (y\_max - y\_min), y\_max + 0.1 \* (y\_max - y\_min)  
 )  
  
 # 在图下方添加两个框，使用固定的图形位置  
 textstr1 = f"Center Wavelength: {max\_position:.2f} nm"  
 textstr2 = f"Peak Loss: {max\_value:.2f} dB"  
  
 props = dict(boxstyle="round", facecolor="wheat", alpha=0.5)  
  
 # 确保文本框不被移动  
 bottom\_margin = 0.35  
 y\_text = -bottom\_margin + 0.05  
 y\_text\_2 = -bottom\_margin - 0.15  
  
 x\_text = 0.05  
 self.ax.text(  
 x\_text,  
 y\_text,  
 textstr1,  
 transform=self.ax.transAxes,  
 fontsize=10,  
 verticalalignment="top",  
 bbox=props,  
 )  
  
 self.ax.text(  
 x\_text,  
 y\_text\_2,  
 textstr2,  
 transform=self.ax.transAxes,  
 fontsize=10,  
 verticalalignment="top",  
 bbox=props,  
 )  
  
 else:  
 self.ax.set\_title("No Data Available")  
  
 # 强制调整边距并应用  
 self.figure.subplots\_adjust(left=0.15, bottom=0.35, right=0.85, top=0.9)  
  
 # 使用 tight\_layout 确保边距的调整  
 self.figure.tight\_layout(pad=1.0)  
  
 # 强制更新绘图区域  
 self.figure.canvas.draw\_idle()  
  
 self.canvas.draw()  
  
 def paintEvent(self, event):  
 super().paintEvent(event)  
 painter = QPainter(self)  
 painter.setRenderHint(QPainter.RenderHint.Antialiasing)  
  
 pen = QPen(Qt.GlobalColor.black)  
 pen.setWidth(1)  
 painter.setPen(pen)  
  
 wafer\_grid\_x, wafer\_grid\_y = 611, 354 # 晶圆按钮区域的左上角坐标  
 radius = 345 # 晶圆的半径  
  
 # 创建一个画刷来填充圆形  
 brush = QBrush(Qt.GlobalColor.darkGray)  
 painter.setBrush(brush)  
  
 painter.drawEllipse(  
 wafer\_grid\_x - radius // 2, wafer\_grid\_y - radius // 2, radius, radius  
 )  
  
 def get\_data(self, dut\_type, wafer\_id, a, dut\_number):  
 dut = DUT(dut=dut\_type(), a=a, wafer\_id=wafer\_id, data\_number=dut\_number)  
 data = dut.load\_data()  
 return pd.DataFrame({"X": data[0], "Y": data[1]})  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 mainWin = DataPlotApp()  
 mainWin.show()  
 sys.exit(app.exec())

这段 Python 代码基于 PyQt6 和 matplotlib，实现了一个用于硅光芯片测试数据可视化的应用程序。以下是对各模块功能的简要介绍：

**1、导入模块**：引入程序运行所需的各类库，包括系统交互、数据处理、绘图、图形界面构建、信号处理等相关模块。

**2、DataPlotApp类定义：**

初始化方法：创建图形界面窗口，设置标题、初始大小；定义组件列表并添加下拉框；创建绘图区域；生成晶圆 ID 按钮并布局；连接下拉框和按钮的信号与槽函数；初始化晶圆 ID 和初始绘图。

set\_wafer\_id方法：更新当前选中的晶圆 ID，并改变按钮样式，触发数据重新绘制。

find\_local\_maxima方法：使用scipy.signal.argrelextrema函数，根据给定的阶数找到数据中的局部最大值及其对应的横坐标。

plot\_data方法：获取用户在下拉框中选择的参数，根据选择的组件从映射中获取数据，调用\_draw\_plot方法绘制数据。

\_draw\_plot方法：对获取到的数据进行处理和绘图，包括绘制原始数据、局部最大值、拟合曲线；设置坐标轴标签、刻度、范围；添加文本框展示中心波长和峰值损耗；调整绘图布局并更新绘图区域。

paintEvent方法：重写绘图事件，在界面上绘制一个灰色圆形作为背景装饰。

get\_data方法：根据传入的组件类型、晶圆 ID、参数a和 DUT 编号，加载对应的数据并转换为pandas.DataFrame格式返回。

**3、主程序入口**：创建应用程序实例，实例化DataPlotApp类并显示窗口，启动应用程序的事件循环 。

该程序运行结果如下：

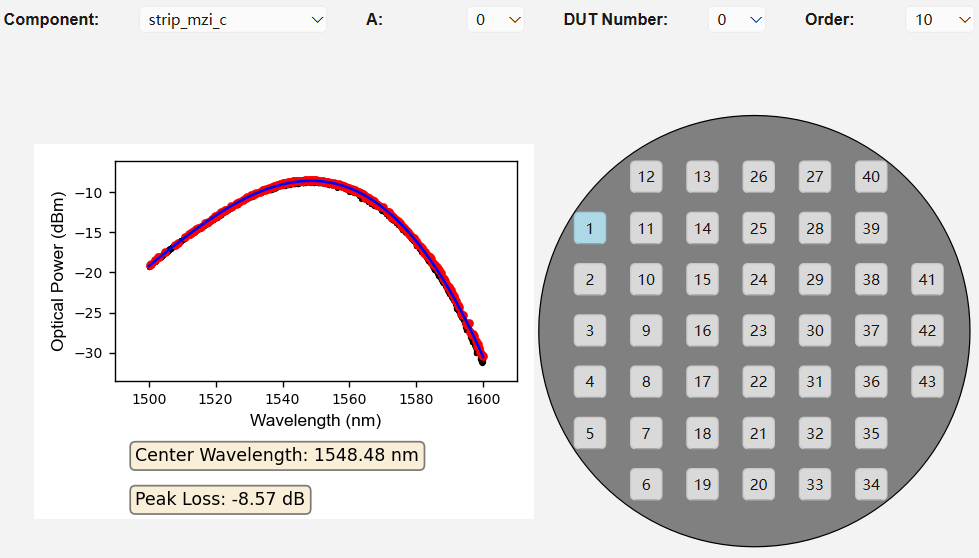


图 5-1-1 测试数据可视化程序运行结果

测试数据可视化程序会生成图5-1-1所示的绘图交互式界面，在Component中可以选择查看所需DUT的测试结果，A和DUT Number可以查看，同一个shot第DUT Number组DUT中的第A个光栅对的测试结果，Order是极大值拟合所选点的步长数，主要是为了适配不同DUT极大值的拟合，此外绘图面板上还有shot的分布，通过点击shot的编号，就可以实时查看对应shot上DUT的测试结果。

# 7模型参数提取程序设计

在项目的第二阶段，器件模型参数的精准提取是保障系统性能的关键环节。从无源器件来看，波导传输损耗源于介质吸收、模式泄露等因素，直接影响光信号的传输距离与信噪比；光栅耦合器的插入损耗关系到芯片与外部光路的耦合效率，损耗过大可能造成信号大幅衰减。此外，波导的有效折射率与光波传播模式紧密相连，影响着光场分布特性，群折射率则决定光信号的群速度，关乎脉冲传输的色散特性，这两个参数对波导器件的设计和性能优化意义重大。

有源器件如马赫 - 曾德尔调制器（MZM）和谐振调制器（RM），其工作性能高度依赖带宽与调制效率。带宽决定了器件对高频电信号的响应能力，直接限制光通信系统的数据传输速率；调制效率反映了电信号转换为光信号调制深度的能力，与系统功耗、调制灵敏度息息相关。鉴于这些参数在器件性能评估和系统设计中的核心地位，本节将系统梳理无源与有源器件模型参数提取的程序设计逻辑，从数据采集、算法优化到程序实现进行全方位解析，为项目的精确建模与性能提升筑牢技术根基。

## 7.1 损耗与带宽提取程序设计

对于无源器件损耗与带宽的提取主要分为两种，一种是光栅耦合器，另外一种是损耗很小，需要多级联结构测试的器件，如Crosing,Transition等的损耗与带宽提取。

光栅耦合器损耗与带宽提取的程序，如下：

import matplotlib.pyplot as plt  
from date\_classification import all as dt  
import numpy as np  
import pandas as pd  
import seaborn as sns  
from joblib import Parallel, delayed  
from collections import defaultdict  
from pathlib import Path  
from utiles.math\_functions import scale\_to\_range, calculate\_r2  
  
from common\_file.DUT\_parse import DUT  
  
  
def find\_value(x, arg):  
 return np.polyval(arg[0], x) - arg[1]  
  
  
def GC\_model\_extraction(wafer\_id):  
 distribution = {  
 "center\_wavelength": [],  
 "peak\_loss": [],  
 "1db\_bandwidth": [],  
 "R2": [],  
 }  
 for i in range(4):  
 for j in range(2):  
 gc = DUT(dut=dt.gc\_c(), wafer\_id=wafer\_id + 1, a=i, data\_number=j)  
 gc\_data = gc.load\_data()  
 x = gc\_data[0] / 1000  
 x = scale\_to\_range(x)  
 y = gc\_data[1] # fit in log scale  
  
 fitting\_degree = 4  
 coeff = np.polyfit(x, y, fitting\_degree)  
 y\_fit = np.polyval(coeff, x)  
 r2 = calculate\_r2(y, y\_fit)  
 plt.plot(x, y)  
 plt.plot(x, y\_fit)  
 # plt.show()  
 polynomial = np.poly1d(coeff)  
 derivative = np.polyder(polynomial)  
 critical\_points = np.roots(derivative)  
 second\_derivative = np.polyder(derivative)  
 central\_wavelength = [  
 point  
 for point in critical\_points  
 if np.polyval(second\_derivative, point) < 0  
 ]  
 central\_wavelength = [  
 np.real(x) for x in central\_wavelength if -0.9 <= x <= 0.9  
 ]  
 IL = np.real(np.polyval(coeff, central\_wavelength))  
 db1\_point = IL - 1 # change to linear scale  
 from scipy.optimize import fsolve  
  
 roots = fsolve(  
 find\_value,  
 [central\_wavelength[0] - 0.3, central\_wavelength[0] + 0.3],  
 args=[coeff, db1\_point],  
 )  
 BW1dB = np.abs(roots[0] - roots[1])  
  
 distribution["center\_wavelength"].append(  
 abs(float(central\_wavelength[0])) \* 100 / 2  
 )  
 distribution["peak\_loss"].append(-float(IL) / 2)  
 distribution["1db\_bandwidth"].append(float(BW1dB) \* 100 / 2)  
 distribution["R2"].append(float(r2))  
  
 return distribution  
  
  
def parallel\_process\_and\_dump\_data(wafer\_id\_list, model):  
 distribution = defaultdict(list)  
  
 load\_res = Parallel(n\_jobs=-1)(delayed(model)(m) for m in wafer\_id\_list)  
  
 for result in load\_res:  
 for key, values in result.items():  
 distribution[key].extend(values)  
  
 return dict(distribution)  
  
  
###########load all the data and save the processed results to yaml files###########  
wafer\_id\_list = range(43)  
  
distribution = parallel\_process\_and\_dump\_data(wafer\_id\_list, GC\_model\_extraction)  
  
import yaml  
  
with open("gc\_cband\_TE\_distribution.yaml", "w", encoding="utf-8") as outfile:  
 yaml.dump(distribution, outfile, default\_flow\_style=False, allow\_unicode=True)

该代码中通过四阶多项式来拟合测试得到的光栅耦合器的光谱数据：

fitting\_degree = 4  
coeff = np.polyfit(x, y, fitting\_degree)  
y\_fit = np.polyval(coeff, x)  
r2 = calculate\_r2(y, y\_fit)

并通过求拟合的峰值和fslove求1dB带宽得到损耗与带宽：

IL = np.real(np.polyval(coeff, central\_wavelength))  
db1\_point = IL - 1 # change to linear scale  
from scipy.optimize import fsolve  
  
roots = fsolve(  
 find\_value,  
 [central\_wavelength[0] - 0.3, central\_wavelength[0] + 0.3],  
 args=[coeff, db1\_point],  
)  
BW1dB = np.abs(roots[0] - roots[1])

并将损耗与带宽数据保存在yaml文件中：

import yaml  
with open("gc\_cband\_TE\_distribution.yaml", "w", encoding="utf-8") as outfile:  
 yaml.dump(distribution, outfile, default\_flow\_style=False, allow\_unicode=True)

对于Transition这种单个损耗很小需要多个级联的结构，则采用截距法来得到损耗，其提取程序与光栅耦合器不同的地方在于：

def GC\_model\_extraction(wafer\_id):  
 distribution = {"loss": []}  
 length = ["0", "40", "70"]  
 length\_float = [float(l) for l in length]  
  
 for i in range(2):  
 peak = []  
 for j in range(3):  
 dut = DUT(dut=dt.tr\_sm\_c(), a=j, wafer\_id=wafer\_id + 1, data\_number=i)  
 dut\_data = dut.load\_data()  
  
 x = np.array(dut\_data[0]) / 1000  
 x\_s = scale\_to\_range(x)  
 y = np.array(dut\_data[1])  
  
 fitting\_degree = 4  
 x\_s, y\_env, coeff, r2 = poly\_fit\_get\_envelop(x\_s, y, fitting\_degree)  
 peak.append(max(y\_env))  
 coefficients = np.polyfit(length\_float, peak, 1)  
 slope, intercept = coefficients  
 distribution["loss"].append(float(slope) \* (-1))  
 return distribution

其损耗是通过截距法，即不同数量下的损耗的线性拟合得到：

coefficients = np.polyfit(length\_float, peak, 1)  
slope, intercept = coefficients  
distribution["loss"].append(float(slope) \* (-1))

## 7.2 有效折射率和群折射率提取程序设计

波导有效折射率和群折射率提取的程序设计如下：

import matplotlib.pyplot as plt  
from pathlib import Path  
from date\_classification import all as dt  
from DUT\_parse import DUT  
import numpy as np  
from scipy.signal import savgol\_filter  
from Generic\_MZI\_DOE.utils import get\_data, calculate\_deltal\_l  
from functools import partial  
from joblib import Parallel, delayed  
from scipy.optimize import minimize  
from math\_functions import calculate\_r2  
from collections import defaultdict  
from mzi\_CM\_for\_wg\_extraction import MZI, MZI\_simulation  
from data\_save\_load import load\_hdf5\_to\_dict,save\_dict\_to\_hdf5  
  
  
  
current\_directory = Path(\_\_file\_\_).parent.resolve()  
  
# device\_dict = defaultdict(dict)  
  
def generate\_device\_dict(wafer\_id, i,j):  
 device\_dict = defaultdict(dict)  
 for temp\_1 in range(wafer\_id):  
 for temp\_2 in range(i):  
 for temp\_3 in range(j):  
 device\_dict[f"{temp\_1}\_{temp\_2}\_{temp\_3}"] = {}  
 return device\_dict  
  
def filter\_device\_dict(device\_dict, j =2 ):  
 device\_dict\_new = defaultdict(dict)  
  
 for keys, values in device\_dict.items():  
 if int(keys.split("\_")[2]) == j:  
 if values["r2"][0] > 0.98:  
 continue  
 else:  
 device\_dict\_new[keys] = {}  
 return device\_dict\_new  
  
def get\_device\_distribution(device\_dict, j = 2):  
 distribution = defaultdict(list)  
 for keys, values in device\_dict.items():  
 if int(keys.split("\_")[2]) == j:  
 for keys2,values2 in values.items():  
 # if values["r2"][0] < 0.98:  
 # continue  
 distribution[keys2].append(values2)  
 else:  
 continue  
 return distribution  
  
  
def model\_function(x,paras,model):  
  
 n\_g, n\_eff,p1,p2,p3,p4,p5 = paras  
 p = [p1,p2,p3,p4,p5]  
 gc = np.polyval(p,x)  
 gc = 10\*\*(gc/10)  
 return MZI\_simulation(x,n\_g, n\_eff,model)\*gc  
  
def objective\_function(paras, x\_data, y\_data, model):  
  
 residuals = y\_data - model\_function(x\_data,paras, model)  
 return np.sum(residuals \*\* 2)  
  
def objective\_function2(paras, x\_data, y\_data, model):  
  
 r2 = calculate\_r2(y\_data,model\_function(x\_data,paras, model))  
 return 10\*np.log10(1-r2)  
  
  
def WG\_data\_extraction(device\_name = "0\_0\_0"):  
 wg\_parameters = defaultdict(list)  
 [wafer\_id, i, j] = device\_name.split("\_")  
 wafer\_id = int(wafer\_id)  
 i = int(i)  
 j = int(j)  
 wafer\_id = wafer\_id + 1  
 dut = dt.rib\_mzi\_070()  
  
 dut\_mzi = DUT(dut=dut,a=j,wafer\_id = wafer\_id,data\_number=i)  
 dut\_data = dut\_mzi.load\_data()  
 x = dut\_data[0]/1000  
 y\_log = np.array(dut\_data[1].tolist())  
 y\_log=savgol\_filter(y\_log,50,3)  
 y = 10\*\*(y\_log/10)  
 m = [16, 15, 110]  
 # plt.plot(x,y, label = f"{m[j]}\_order\_MZI")  
 # plt.show()  
 data\_path = r"..\Si\_1550nm\_TE0\_neff\_ng\_for\_different\_width.txt"  
 data = get\_data(data\_path)  
 dl = calculate\_deltal\_l(data, m[j], 1.55, int(dut.\_\_class\_\_.\_\_name\_\_.split("\_")[2]) / 100)  
 mzi = MZI(path\_difference=dl[1])  
 mzi\_lo = mzi.Layout()  
 mzi\_lo.dependencies()  
 mzi\_cm = mzi.CircuitModel()  
  
 # local\_minima\_indices = argrelextrema(y\_log, np.greater, order=10)[0]  
 # x\_minima = x[local\_minima\_indices]  
 # y\_minima = y\_log[local\_minima\_indices]  
 # gc = np.polyfit(x\_minima, y\_minima, 4)  
 gc = [-4.53284895e+04, 2.24928235e+05, -3.98478505e+05, 2.89368539e+05, -6.71541631e+04]  
 # print(gc)  
 # gc\_fit = np.polyval(gc, x)  
 # gc\_fit = 10 \*\* (gc\_fit / 10)  
 # plt.plot(x, y)  
 # plt.plot(x, gc\_fit, "--")  
 # plt.show()  
 # continue  
 bounds = [(3.60, 3.63), (2.425, 2.455), (-np.inf, np.inf), (-np.inf, np.inf), (-np.inf, np.inf), (-np.inf, np.inf),  
 (-np.inf, np.inf)]  
 res = minimize(partial(objective\_function2, model=mzi\_cm), [3.62, 2.445, gc[0], gc[1], gc[2], gc[3], gc[4]],  
 args=(x, y), bounds=bounds  
 # ,constraints=constraints  
 )  
 print(res.x)  
 print(res.fun)  
 r2 = 1 - 10 \*\* (res.fun / 10)  
 wg\_parameters[f"res"] = [float(res.x[0]), float(res.x[1])]  
 # wg\_parameters[f"fitting\_residual"] = [res.fun]  
 wg\_parameters[f"r2"] = [float(r2)]  
 wg\_parameters[f"gc"] = [float(res.x[2]), float(res.x[3]), float(res.x[4]),  
 float(res.x[5]), float(res.x[6])]  
 device\_dict = {device\_name: wg\_parameters}  
 return device\_dict  
  
def parallel\_process\_and\_dump\_data(device\_dict\_new,model):  
 load\_res = Parallel(n\_jobs=-1)(delayed(model)(m) for m in device\_dict\_new)  
 for temp in load\_res:  
 device\_dict\_new.update(temp)  
 return device\_dict\_new  
  
#  
# device\_dict = generate\_device\_dict(43,6,3)  
# # print(device\_dict)  
# # device\_dict = load\_hdf5\_to\_dict("rib\_070.h5")  
# device\_dict\_new = filter\_device\_dict(device\_dict,conditions=0)  
#  
# device\_dict\_new = parallel\_process\_and\_dump\_data(device\_dict\_new,WG\_data\_extraction)  
#  
# device\_dict.update(device\_dict\_new)  
# save\_dict\_to\_hdf5(device\_dict,"rib\_070.h5")  
  
device\_dict = load\_hdf5\_to\_dict("rib\_070.h5")  
distribution = get\_device\_distribution(device\_dict)  
print(distribution["res"])  
x = []  
y = []  
z = []  
  
for i, j in enumerate(distribution["res"]) :  
 x.append(j[1])  
 y.append(j[0])  
 z.append(distribution["r2"][i])  
  
  
  
print(len(x))  
fig = plt.figure()  
ax = fig.add\_subplot(projection='3d')  
ax.scatter(x, y, z)  
# plt.plot(x\_shift,y,"o")  
# plt.show()  
# print(distribution)  
values = {"r2":[1]}  
device\_dict\_new = filter\_device\_dict(device\_dict)  
  
device\_dict\_new = parallel\_process\_and\_dump\_data(device\_dict\_new,WG\_data\_extraction)  
device\_dict.update(device\_dict\_new)  
print(device\_dict\_new)  
plt.show()  
save\_dict\_to\_hdf5(device\_dict,"rib\_070.h5")  
print("done")

这段代码主要用于基于测试的 MZI（马赫 - 曾德尔干涉仪）结构提取波导的有效折射率和群折射率。它通过对 MZI 光学响应数据的建模和拟合，实现了波导参数的自动化提取，并对结果进行可视化分析。

**主要模块功能分析**

**1. 数据结构初始化与管理模块**

from collections import defaultdict

from data\_save\_load import load\_hdf5\_to\_dict, save\_dict\_to\_hdf5

def generate\_device\_dict(wafer\_id, i, j):

device\_dict = defaultdict(dict)

for temp\_1 in range(wafer\_id):

for temp\_2 in range(i):

for temp\_3 in range(j):

device\_dict[f"{temp\_1}\_{temp\_2}\_{temp\_3}"] = {}

return device\_dict

def filter\_device\_dict(device\_dict, j=2):

device\_dict\_new = defaultdict(dict)

for keys, values in device\_dict.items():

if int(keys.split("\_")[2]) == j:

if values["r2"][0] > 0.98:

continue

else:

device\_dict\_new[keys] = {}

return device\_dict\_new

def get\_device\_distribution(device\_dict, j=2):

distribution = defaultdict(list)

for keys, values in device\_dict.items():

if int(keys.split("\_")[2]) == j:

for keys2, values2 in values.items():

distribution[keys2].append(values2)

return distribution

功能：

generate\_device\_dict：生成设备字典，用于存储不同晶圆、不同位置的 MZI 器件数据

filter\_device\_dict：根据 R² 拟合优度过滤设备数据，保留拟合效果不佳的数据点

get\_device\_distribution：提取特定类型设备的参数分布

**2. 模型与优化模块**

from scipy.optimize import minimize

from math\_functions import calculate\_r2

from mzi\_CM\_for\_wg\_extraction import MZI, MZI\_simulation

def model\_function(x, paras, model):

n\_g, n\_eff, p1, p2, p3, p4, p5 = paras

p = [p1, p2, p3, p4, p5]

gc = np.polyval(p, x)

gc = 10\*\*(gc/10)

return MZI\_simulation(x, n\_g, n\_eff, model)\*gc

def objective\_function(paras, x\_data, y\_data, model):

residuals = y\_data - model\_function(x\_data, paras, model)

return np.sum(residuals \*\* 2)

def objective\_function2(paras, x\_data, y\_data, model):

r2 = calculate\_r2(y\_data, model\_function(x\_data, paras, model))

return 10\*np.log10(1-r2)

**功能**：

model\_function：定义 MZI 光学响应模型，结合有效折射率、群折射率和高斯耦合参数

objective\_function：基于残差平方和的优化目标函数

objective\_function2：基于 R² 拟合优度的优化目标函数

**3. 波导参数提取核心模块**

from date\_classification import all as dt

from DUT\_parse import DUT

import numpy as np

from scipy.signal import savgol\_filter

from Generic\_MZI\_DOE.utils import get\_data, calculate\_deltal\_l

def WG\_data\_extraction(device\_name="0\_0\_0"):

wg\_parameters = defaultdict(list)

[wafer\_id, i, j] = device\_name.split("\_")

wafer\_id = int(wafer\_id) + 1

i = int(i)

j = int(j)

# 加载器件数据

dut = dt.rib\_mzi\_070()

dut\_mzi = DUT(dut=dut, a=j, wafer\_id=wafer\_id, data\_number=i)

dut\_data = dut\_mzi.load\_data()

x = dut\_data[0]/1000

y\_log = np.array(dut\_data[1].tolist())

y\_log = savgol\_filter(y\_log, 50, 3)

y = 10\*\*(y\_log/10)

# 计算MZI路径差

data\_path = r"..\Si\_1550nm\_TE0\_neff\_ng\_for\_different\_width.txt"

data = get\_data(data\_path)

dl = calculate\_deltal\_l(data, m[j], 1.55, int(dut.\_\_class\_\_.\_\_name\_\_.split("\_")[2]) / 100)

mzi = MZI(path\_difference=dl[1])

mzi\_lo = mzi.Layout()

mzi\_lo.dependencies()

mzi\_cm = mzi.CircuitModel()

# 初始参数设置

gc = [-4.53284895e+04, 2.24928235e+05, -3.98478505e+05, 2.89368539e+05, -6.71541631e+04]

bounds = [(3.60, 3.63), (2.425, 2.455), (-np.inf, np.inf), (-np.inf, np.inf), (-np.inf, np.inf), (-np.inf, np.inf), (-np.inf, np.inf)]

# 优化拟合

res = minimize(partial(objective\_function2, model=mzi\_cm), [3.62, 2.445, gc[0], gc[1], gc[2], gc[3], gc[4]],

args=(x, y), bounds=bounds)

# 提取结果

r2 = 1 - 10 \*\* (res.fun / 10)

wg\_parameters[f"res"] = [float(res.x[0]), float(res.x[1])] # n\_g, n\_eff

wg\_parameters[f"r2"] = [float(r2)]

wg\_parameters[f"gc"] = [float(res.x[2]), float(res.x[3]), float(res.x[4]), float(res.x[5]), float(res.x[6])]

return {device\_name: wg\_parameters}

**功能**：

加载特定 MZI 器件的测试数据

对数据进行预处理（如平滑滤波）

构建 MZI 电路模型

通过优化拟合提取波导的有效折射率 (n\_eff) 和群折射率 (n\_g)

返回拟合结果和评估指标

**4. 并行处理与结果可视化模块**

from joblib import Parallel, delayed

import matplotlib.pyplot as plt

def parallel\_process\_and\_dump\_data(device\_dict\_new, model):

load\_res = Parallel(n\_jobs=-1)(delayed(model)(m) for m in device\_dict\_new)

for temp in load\_res:

device\_dict\_new.update(temp)

return device\_dict\_new

# 主程序部分

device\_dict = load\_hdf5\_to\_dict("rib\_070.h5")

distribution = get\_device\_distribution(device\_dict)

# 提取参数用于可视化

x = []

y = []

z = []

for i, j in enumerate(distribution["res"]):

x.append(j[1]) # n\_eff

y.append(j[0]) # n\_g

z.append(distribution["r2"][i])

# 3D可视化

fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

ax.scatter(x, y, z)

plt.show()

# 处理并保存数据

device\_dict\_new = filter\_device\_dict(device\_dict)

device\_dict\_new = parallel\_process\_and\_dump\_data(device\_dict\_new, WG\_data\_extraction)

device\_dict.update(device\_dict\_new)

save\_dict\_to\_hdf5(device\_dict, "rib\_070.h5")

功能：

parallel\_process\_and\_dump\_data：并行处理多个器件数据，提高处理效率

3D 散点图可视化有效折射率、群折射率与拟合优度之间的关系

数据的加载、更新与保存操作

这种参数提取方法对于光子集成电路的设计和制造具有重要意义，可以帮助工程师准确了解波导特性，优化器件性能。

## 7.3 有源器件带宽与调制效率提取程序设计

有源器件调制效提取的代码设计如下：

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from scipy.optimize import root  
  
# 读取数据  
data = pd.read\_csv("MZM\_MZI\_E\_C\_Start\_0\_Stop\_16.txt", sep="\t")  
  
# 提取 Voltage(V) 和 Optical power(dBm) 列  
voltage = data["Voltage(V)"]  
optical\_power = data["Optical power(dBm)"]  
  
# 找到最大值点  
max\_index = optical\_power.idxmax()  
max\_value = optical\_power[max\_index]  
min\_index = optical\_power.idxmin()  
min\_value = optical\_power[min\_index]  
  
# 截取 0 到 10V 的数据  
mask = (voltage >= 1) & (voltage <= 10)  
voltage\_cut = voltage[mask]  
optical\_power\_cut = optical\_power[mask]  
  
# 用最大值点对截取的数据做归一化  
optical\_power\_normalized = optical\_power\_cut - max\_value  
optical\_power\_normalized\_percentage = 10 \*\* (optical\_power\_normalized / 10)  
# I=(0.5+0.5\*exp(i\*fai))^2  
# I=0.25\*(1+exp(i\*fai))^2  
# I=0.25\*[(1+cos(fai))^2+sin(fai)^2]  
# I=0.25\*[1+cos(fai)^2+2\*cos(fai)+sin(fai)^2]  
# I=0.25\*[1+1+2\*cos(fai)]  
# I=[1+cos(fai)]/2  
# cos(fai)=2\*I-1  
phase = np.arccos(optical\_power\_normalized\_percentage \* 2 - 1) / np.pi  
  
# 进行 3 阶多项式拟合  
coefficients = np.polyfit(voltage\_cut, phase, 3)  
poly = np.poly1d(coefficients)  
  
print(coefficients)  
# 生成用于绘制拟合曲线的 x 值，范围从 0 到 20V  
x\_fit = np.linspace(0, 20, 100)  
y\_fit = poly(x\_fit)  
  
# 找到横坐标为 0 的纵坐标点  
y0 = poly(0)  
  
  
# 找到纵坐标减小 1 对应的横坐标点  
def find\_x(y\_target, poly\_func):  
 def eq(x):  
 return poly\_func(x) - y\_target  
  
 result = root(eq, 0)  
 return result.x[0] if result.success else None  
  
  
x1 = find\_x(y0 - 1, poly)  
  
# 绘制关系图  
plt.figure(figsize=(6, 4))  
plt.plot(voltage\_cut, phase, marker="o", linestyle="-", label='Original Data')  
plt.plot(x\_fit, y\_fit, linestyle="--", color='red', label='3th Order Polynomial Fit')  
plt.plot(0, y0, 'ro', label='Point at x=0')  
plt.plot(x1, y0 - 1, 'go', label='Point with y decreased by 1 $\pi$')  
  
# 画两个点横轴和纵轴的距离及虚线  
if x1 is not None:  
 # 画水平和垂直虚线  
 plt.plot([0, x1], [y0, y0], 'k--', alpha=0.5)  
 plt.plot([x1, x1], [y0 - 1, y0], 'k--', alpha=0.5)  
 plt.plot([0, 0], [y0 - 1, y0], 'k--', alpha=0.5)  
  
 # 标注横向距离  
 horizontal\_distance = x1 - 0  
 y\_text\_level = y0 - 0.05  
 plt.text((0 + x1) / 2, y\_text\_level, f'{horizontal\_distance:.2f} V', ha='center', va='center',  
 backgroundcolor='white')  
  
 # 标注纵向距离  
 vertical\_distance = y0 - (y0 - 1)  
 x\_text\_pos = x1 + 0.1  
 plt.text(x\_text\_pos, (y0 + y0 - 1) / 2, f'{vertical\_distance:.2f} $\pi$', ha='left', va='center',  
 backgroundcolor='white')  
  
plt.title("Voltage vs Phase")  
plt.xlabel("Voltage (V)")  
plt.ylabel("Phase ($\pi$)")  
plt.xlim(0, 20)  
plt.grid(True)  
plt.legend()  
plt.show()

这段代码主要完成了数据的读取和初步处理：

从文本文件中读取电压和光功率数据

找出光功率的最大值和最小值点

截取 1V 到 10V 之间的数据用于后续分析

**数据归一化与相位计算**

# 用最大值点对截取的数据做归一化

optical\_power\_normalized = optical\_power\_cut - max\_value

optical\_power\_normalized\_percentage = 10 \*\* (optical\_power\_normalized / 10)

# I=(0.5+0.5\*exp(i\*fai))^2

# I=0.25\*(1+exp(i\*fai))^2

# I=0.25\*[(1+cos(fai))^2+sin(fai)^2]

# I=0.25\*[1+cos(fai)^2+2\*cos(fai)+sin(fai)^2]

# I=0.25\*[1+1+2\*cos(fai)]

# I=[1+cos(fai)]/2

# cos(fai)=2\*I-1

phase = np.arccos(optical\_power\_normalized\_percentage \* 2 - 1) / np.pi

此部分实现了数据的归一化和相位计算：

将光功率数据相对于最大值进行归一化

利用光学强度与相位的理论关系，从归一化光功率计算出对应的相位值

相位值以 π 为单位进行表示

**多项式拟合与关键参数计算**

# 进行 3 阶多项式拟合

coefficients = np.polyfit(voltage\_cut, phase, 3)

poly = np.poly1d(coefficients)

print(coefficients)

# 生成用于绘制拟合曲线的 x 值，范围从 0 到 20V

x\_fit = np.linspace(0, 20, 100)

y\_fit = poly(x\_fit)

y0 = poly(0)

def find\_x(y\_target, poly\_func):

def eq(x):

return poly\_func(x) - y\_target

result = root(eq, 0)

return result.x[0] if result.success else None

x1 = find\_x(y0 - 1, poly)

该部分完成了多项式拟合和关键参数的计算：

对电压和相位数据进行 3 阶多项式拟合

得到拟合多项式的系数并打印

计算拟合曲线上横坐标为 0 时的相位值

定义函数求解相位减小 π 时对应的电压值，这一电压值是计算调制效率的关键参数

该代码运行结果如下：

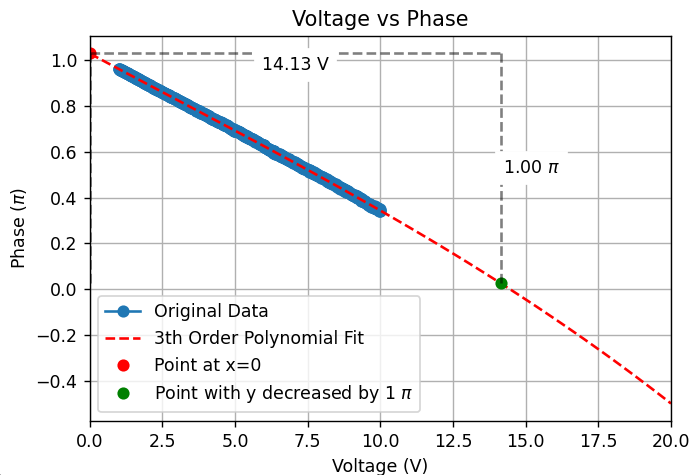


图 6-3-1 调制效率提取程序运行结果

接下来是调制器带宽提取程序，基于调制器测得的S21数据，通过如下代码，可以直接提取其带宽：

import skrf as rf  
import matplotlib.pyplot as plt  
import numpy as np  
  
# 导入S2P文件  
filename = "FIXED\_MZM\_TE\_GSG\_B2\_1550.s2p" # 替换为你的S2P文件路径  
network = rf.Network(filename)  
  
# 获取频率和S21数据  
frequencies = network.frequency.f\_scaled / 1e9 # 转换为GHz  
# 提取S21的幅度  
s21\_mag = np.abs(network.s[:, 1, 0])  
s21\_mag\_db = rf.mag\_2\_db(s21\_mag) # S21幅度（dB）  
  
# 找到S21的最大值  
max\_s21 = np.max(s21\_mag\_db)  
print(f"S21的最大值: {max\_s21:.2f} dB")  
  
# 找到S21下降3dB的阈值  
threshold = max\_s21 - 3  
  
# 找到3dB带宽的起始和结束频率  
mask = s21\_mag\_db >= threshold  
freq\_low = frequencies[mask][0]  
freq\_high = frequencies[mask][-1]  
bandwidth\_3db = freq\_high - freq\_low

该代码运行结果如下：

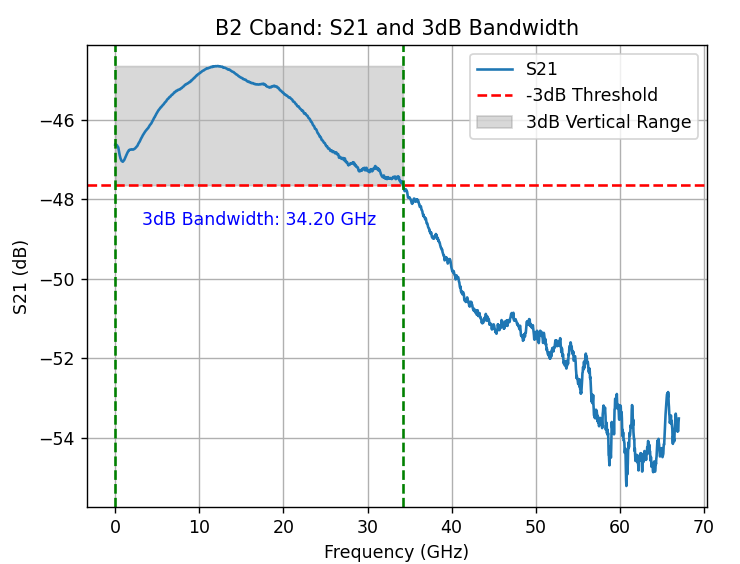


图6-3-2 调制器带宽提取程序运行结果

# 8 统计学建模程序设计

在项目第二阶段，统计学建模作为核心技术手段，深度挖掘硅光芯片测试数据的潜在价值。基于样品数量的统计学建模，通过科学计算样本量与置信区间的关系，构建高效抽样模型，保障数据统计的可靠性与代表性；基于位置分布的统计学建模，结合晶圆上芯片的物理坐标，运用空间自相关、克里金插值等算法，精准分析芯片性能在空间维度的分布规律与异常波动；基于工艺波动的统计学建模，则聚焦生产流程中的关键工艺参数，借助时间序列分析、多元回归等方法，量化工艺变化对芯片性能的影响机制，预测性能波动趋势。这三类统计学建模从不同维度为硅光芯片的质量优化与工艺改进提供数据支撑，接下来将分节介绍这三类建模的程序设计过程。

## 8.1 基于样品数量的统计学建模程序设计

基于样品数量的统计学建模是将晶圆上同一DUT的所有测试数据进行处理，而得到模型参数的统计学分布过程，以光栅耦合器为例, 其基于数量的统计学建模程序设计如下：

import matplotlib.pyplot as plt  
from date\_classification import all as dt  
import numpy as np  
import pandas as pd  
import seaborn as sns  
from joblib import Parallel, delayed  
from collections import defaultdict  
from pathlib import Path  
from utiles.math\_functions import scale\_to\_range, calculate\_r2  
from utiles.data\_save\_load import save\_dict\_to\_hdf5, load\_hdf5\_to\_dict, dump\_sampled\_data\_to\_gmm\_dist  
from common\_file.DUT\_parse import DUT  
  
  
def find\_value(x, arg):  
 return np.polyval(arg[0], x) - arg[1]  
  
  
def GC\_model\_extraction(wafer\_id):  
 gc\_parameter\_extraction\_poly = defaultdict(list)  
 gc\_parameter\_distribution\_poly = defaultdict(list)  
 gc\_poly\_coeff\_distribution = defaultdict(list)  
 responses = defaultdict(list)  
 for i in range(4):  
 for j in range(2):  
 gc = DUT(dut=dt.gc\_c(), wafer\_id=wafer\_id + 1, a=i, data\_number=j)  
 gc\_data = gc.load\_data()  
 x = gc\_data[0] / 1000  
 x = scale\_to\_range(x)  
 y = gc\_data[1] # fit in log scale  
  
 fitting\_degree = 4  
 coeff = np.polyfit(x, y, fitting\_degree)  
 y\_fit = np.polyval(coeff, x)  
 r2 = calculate\_r2(y, y\_fit)  
  
  
 for k in range(fitting\_degree + 1):  
 gc\_poly\_coeff\_distribution[f"a{k}"].append(float(coeff[k]))  
 gc\_poly\_coeff\_distribution["r2"].append(float(r2))  
  
 plt.plot(x, y)  
 plt.plot(x, y\_fit)  
 # plt.show()  
 polynomial = np.poly1d(coeff)  
 derivative = np.polyder(polynomial)  
 critical\_points = np.roots(derivative)  
 second\_derivative = np.polyder(derivative)  
 central\_wavelength = [point for point in critical\_points if np.polyval(second\_derivative, point) < 0]  
 central\_wavelength = [np.real(x) for x in central\_wavelength if -0.9 <= x <= 0.9]  
 # print(central\_wavelength)  
 IL = np.real(np.polyval(coeff, central\_wavelength))  
 db1\_point = IL - 1 # change to linear scale  
 # print(wafer\_id + 1, i, j, IL,coeff)  
 from scipy.optimize import fsolve  
 roots = fsolve(find\_value, [central\_wavelength[0] - 0.3, central\_wavelength[0] + 0.3],  
 args=[coeff, db1\_point])  
 BW1dB = np.abs(roots[0] - roots[1])  
  
 gc\_parameter\_extraction\_poly[f"gc\_{wafer\_id}\_{i}\_{j}"] = [float(central\_wavelength[0]), -float(IL[0]),  
 float(BW1dB.tolist())]  
 gc\_parameter\_distribution\_poly["peak\_wavelength1"].append(float(central\_wavelength[0]))  
 gc\_parameter\_distribution\_poly["peak\_il1"].append(-float(IL[0]))  
 gc\_parameter\_distribution\_poly["bw\_1db1"].append(float(BW1dB.tolist()))  
 responses[f"gc\_{wafer\_id}\_{i}"] = y  
 return gc\_parameter\_extraction\_poly, gc\_parameter\_distribution\_poly, gc\_poly\_coeff\_distribution, responses  
  
  
def parallel\_process\_and\_dump\_data(wafer\_id\_list, model):  
 parameters = defaultdict(list)  
 distribution = defaultdict(list)  
 coeff = defaultdict(list)  
 response = defaultdict(list)  
 load\_res = Parallel(n\_jobs=-1)(delayed(model)(m) for m in wafer\_id\_list)  
  
 for temp in load\_res:  
 for key, value in temp[0].items():  
 parameters[key].extend(value)  
 for key, value in temp[1].items():  
 distribution[key].extend(value)  
 for key, value in temp[2].items():  
 coeff[key].extend(value)  
 for key, value in temp[3].items():  
 response[key].extend(value)  
  
 return parameters, distribution, coeff, response  
  
  
###########load all the data and save the processed results ###########  
  
# GC\_model\_extraction(1)  
calculate = False  
if calculate:  
 wafer\_id\_list = range(43)  
 parameters, distribution, coeff, responses = parallel\_process\_and\_dump\_data(wafer\_id\_list, GC\_model\_extraction)  
 save\_dict\_to\_hdf5(parameters, 'gc\_cband\_parameters\_extraction.h5')  
 save\_dict\_to\_hdf5(distribution, 'gc\_cband\_parameter\_distribution.h5')  
 save\_dict\_to\_hdf5(coeff, 'gc\_cband\_coeff.h5')  
 save\_dict\_to\_hdf5(responses, 'gc\_cband\_responses.h5')  
  
plot = True  
if plot:  
 ### plot res:  
 responses = load\_hdf5\_to\_dict('gc\_cband\_responses.h5')  
 response = pd.DataFrame(responses)  
 response = response.T  
 response\_mean = response.mean().to\_numpy()  
 response\_cov = response.var().to\_numpy()  
 response\_std = np.sqrt(response\_cov)  
 wl = np.linspace(1500, 1600, 5001)  
 plt.errorbar(wl, response\_mean, yerr=response\_std, fmt='o', capsize=5, alpha=0.01)  
 plt.xlabel('Wavelength (nm)', fontsize=12)  
 plt.ylabel('Optical Power (dBm)', fontsize=12)  
 plt.show()  
 ### load data and do processing  
  
 paras = load\_hdf5\_to\_dict('gc\_cband\_coeff.h5')  
 data = pd.DataFrame(paras)  
  
 dump\_sampled\_data\_to\_gmm\_dist(data, "gc\_cband\_model\_coeff.data")  
  
 # # <editor-fold desc="plot coeff distribution">  
 plt.rcParams["font.family"] = "Arial"  
 plt.rcParams["font.size"] = 12  
  
 # Create the figure with the specified size  
 plt.figure(figsize=(8, 6)) # Adjust the figure size for better spacing  
  
 # Create the subplots  
 plt.subplot(3, 2, 1)  
 sns.histplot(data["a0"], bins=50, kde=True)  
 plt.xlabel('a0')  
 plt.ylabel('Frequency')  
  
 plt.subplot(3, 2, 2)  
 sns.histplot(data["a1"], bins=50, kde=True)  
 plt.xlabel('a1')  
 plt.ylabel('Frequency')  
  
 plt.subplot(3, 2, 3)  
 sns.histplot(data["a2"], bins=50, kde=True)  
 plt.xlabel('a2')  
 plt.ylabel('Frequency')  
  
 plt.subplot(3, 2, 4)  
 sns.histplot(data["a3"], bins=50, kde=True)  
 plt.xlabel('a3')  
 plt.ylabel('Frequency')  
  
 plt.subplot(3, 2, 5)  
 sns.histplot(data["a4"], bins=50, kde=True)  
 plt.xlabel('a4')  
 plt.ylabel('Frequency')  
  
 plt.subplot(3, 2, 6)  
 sns.histplot(data["r2"], bins=50, kde=True)  
 plt.xlabel('r2')  
 plt.ylabel('Frequency')  
  
 # Adjust layout to prevent overlapping and reduce spacing between subplots  
 plt.tight\_layout() # Reduces the space between subplots  
  
 # Show the plot  
 plt.show()  
  
print("done")

这段代码围绕硅光芯片中光栅耦合器（GC）进行基于样品数量的统计学建模，从数据处理、模型参数提取到结果可视化与存储，实现了完整的分析流程。下面分模块为你介绍：

1. **导入模块**：

引入数据处理、绘图、并行计算、文件读写等功能所需的多个库，为后续程序运行提供基础支持。如matplotlib.pyplot用于绘图，numpy和pandas用于数值计算和数据处理，seaborn用于高级可视化，joblib实现并行计算以提升效率。

**2、辅助函数定义：**

find\_value：定义一个函数，用于计算多项式在给定自变量x处的值与目标值arg[1]的差值，供后续根求解使用。

GC\_model\_extraction：针对单个晶圆 ID，对光栅耦合器数据进行处理和参数提取。循环遍历不同参数组合的光栅耦合器数据，进行数据缩放、多项式拟合，计算拟合优度r2；通过求导找到多项式的极值点，确定中心波长、插入损耗（IL）和 1dB 带宽（BW1dB）等关键参数，并将这些参数及多项式系数等结果存储在相应的字典中。

**3、并行处理与数据存储函数：**

parallel\_process\_and\_dump\_data：实现对多个晶圆 ID 的并行处理，调用GC\_model\_extraction函数获取每个晶圆的参数结果，汇总整合不同晶圆的数据；将处理后的数据通过自定义的save\_dict\_to\_hdf5函数保存为 HDF5 格式文件，便于后续使用。

**4、主逻辑部分：**

计算控制：通过calculate变量控制是否执行数据处理和存储操作。若为True，则对 43 个晶圆 ID 并行执行GC\_model\_extraction，并保存处理后的参数、分布、系数和响应数据。

绘图控制：通过plot变量控制是否执行数据可视化操作。若为True，先加载之前保存的响应数据，计算均值和标准差并绘制误差线图；再加载系数数据，使用seaborn绘制各系数和拟合优度r2的分布直方图，直观展示数据的分布特征。

通过将calculate赋值True, 如：

calculate = True

该程序将保存光栅耦合器的模型参数分布：

save\_dict\_to\_hdf5(parameters, 'gc\_cband\_parameters\_extraction.h5')  
save\_dict\_to\_hdf5(distribution, 'gc\_cband\_parameter\_distribution.h5')  
save\_dict\_to\_hdf5(coeff, 'gc\_cband\_coeff.h5')  
save\_dict\_to\_hdf5(responses, 'gc\_cband\_responses.h5')

通过将plot赋值True, 如：

plot = True

该代码将绘制光栅耦合器模型参数统计学分布：

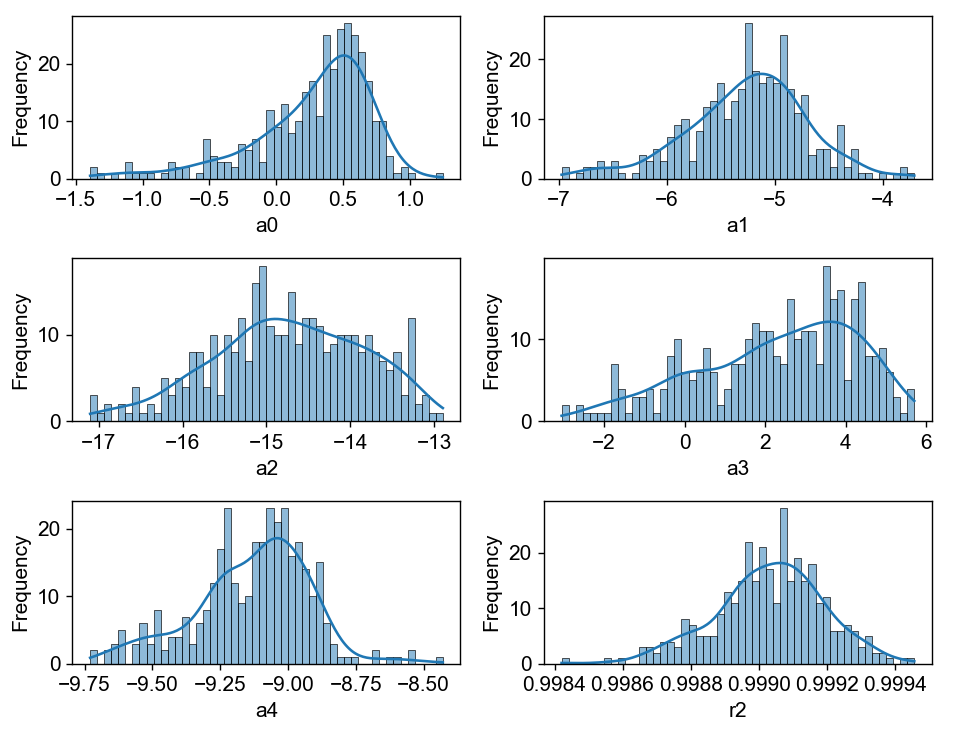


图 7-1-1 光栅耦合器模型参数的统计学分布

得到模型参数的统计学分布之后，就可以基于分布参数进行蒙特卡洛仿真：

import yaml  
import pandas as pd  
from scipy import stats  
import numpy as np  
from matplotlib import pyplot as plt  
  
  
with open("gc\_parameter\_ployfit\_distribution.yml", "r") as stream:  
 try:  
 paras = yaml.safe\_load(stream)  
 except yaml.YAMLError as exc:  
 print(exc)  
  
data = pd.DataFrame(paras)  
dist = stats.gaussian\_kde(data.values.T)  
samples = dist.resample(100)  
  
  
def gc\_model(wavelength\_range, coefficients):  
 polynomial = np.poly1d(coefficients)  
 loss = polynomial(wavelength\_range)  
 return loss  
  
transposed\_data = list(zip(\*samples))  
  
x = np.linspace(-1.0, 1.0, 10000)  
for sample in transposed\_data:  
 y = gc\_model(x, sample)  
 x\_plt = x \* 0.05 + 1.55  
 plt.plot(x\_plt, y)  
 plt.xlabel("Wavelengths [um]")  
 plt.ylabel("Transmission (dB)")  
plt.show()

这段代码实现了基于分布参数的光栅耦合器 (GC) 蒙特卡洛仿真，通过统计建模生成随机样本并计算光学响应。**该**代码通过高斯核密度估计 (Gaussian KDE) 从已有参数分布中采样，然后使用多项式模型模拟不同参数下的光栅耦合器光学响应，最终通过蒙特卡洛方法生成大量随机样本并可视化结果。

代码只要包括以下模块：

**导入模块**：引入必要的库，包括 YAML 文件解析、数据处理、统计分析、数值计算和绘图工具。

**加载参数分布数据**：从 YAML 文件读取预计算的光栅耦合器参数分布数据，该数据可能包含多项式拟合系数的统计特性。

**构建概率密度模型**：使用scipy.stats.gaussian\_kde对参数数据进行核密度估计，构建多维概率密度函数，并从中随机采样 100 组参数。

**定义光学模型**：gc\_model函数实现了光栅耦合器的多项式模型，输入波长范围和多项式系数，输出对应的光学损耗。

**蒙特卡洛仿真**：将采样得到的参数组逐一输入光学模型，计算不同参数下的光谱响应曲线。

**结果可视化**：将所有仿真结果绘制在同一图表上，直观展示光栅耦合器性能的统计分布特性。

**参数分布建模**：使用高斯核密度估计 (KDE) 对多项式系数进行非参数概率密度估计，相比简单的正态分布假设，KDE 能更灵活地拟合复杂的实际数据分布。

**波长归一化处理**：代码中对波长进行了归一化处理 (x \* 0.05 + 1.55)，将归一化坐标 (-1.0~1.0) 转换为实际波长范围 (1.5~1.6μm，对应 C 波段光通信窗口)。

**蒙特卡洛方法**：通过随机采样和模型计算，模拟了光栅耦合器在工艺波动下的性能变化，帮助评估制造过程中的不确定性对器件性能的影响。

**可视化分析**：通过绘制大量随机样本的响应曲线，直观展示了光栅耦合器性能的统计分布，可用于分析中心波长、带宽和插入损耗等关键指标的波动范围。

基于以上代码实现的光栅耦合器蒙特卡洛仿真结果如下：

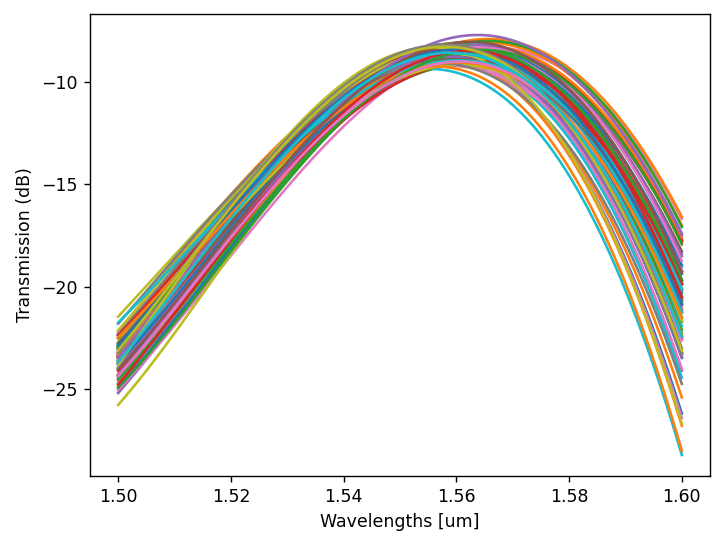


图 7-1-2 光栅耦合器100次采样下蒙特卡洛仿真结果

## 8.2 基于位置分布的统计学建模程序设计

在基于样品数量的统计学建模程序中，同样会根据shot的编号来保存模型参数：

gc\_parameter\_extraction\_poly[f"gc\_{wafer\_id}\_{i}\_{j}"] = [float(central\_wavelength[0]), -float(IL[0]), float(BW1dB.tolist())]

因此基于该方式保存的参数分布，可以来建立基于shou在晶圆上的位置来建立统计学模型，以光栅耦合器为例，其程序设计如下：

import yaml  
import matplotlib.pyplot as plt  
import numpy as np  
from matplotlib.colors import Normalize  
from matplotlib.cm import ScalarMappable  
from date\_classification.all import wafer  
  
  
def plot\_wafer\_maps(yaml\_file="wafer\_parameters\_summary.yaml"):  
 *"""  
 Load wafer data from YAML file and plot wafer maps for three parameters  
 Supports wafer\_id from 0-42 in YAML, mapped to 1-43 in wafer map  
 Displays three parameters side by side horizontally using squares  
 """* try:  
 # Read YAML file  
 with open(yaml\_file, "r") as f:  
 wafer\_data = yaml.safe\_load(f)  
  
 if not wafer\_data:  
 raise ValueError("No data found in YAML file")  
  
 # Create Wafer instance to get coordinate mapping  
 coordinates = wafer().wafer\_map()  
  
 # Map YAML wafer\_id (0-42) to wafer\_map ID (1-43)  
 yaml\_to\_map = {  
 str(yaml\_id): str(map\_id) for yaml\_id, map\_id in enumerate(range(1, 44))  
 }  
  
 # Validate data and convert IDs  
 valid\_data = {}  
 for yaml\_id, data in wafer\_data.items():  
 yaml\_id\_str = str(yaml\_id)  
 if yaml\_id\_str in yaml\_to\_map:  
 map\_id = yaml\_to\_map[yaml\_id\_str]  
 if map\_id in coordinates and all(  
 k in data  
 for k in ["mean\_central\_wavelength", "mean\_peak\_il", "mean\_bw\_1db"]  
 ):  
 valid\_data[map\_id] = data # Use map\_id as key  
  
 if not valid\_data:  
 raise ValueError("No valid wafer data found, please check YAML file format")  
  
 # Create 1x3 subplots (horizontal layout)  
 fig, axes = plt.subplots(1, 3, figsize=(12,5))  
  
 # Define parameter configurations  
 params = [  
 {  
 "name": "Center Wavelength (nm)",  
 "key": "mean\_central\_wavelength",  
 "cmap": "viridis",  
 },  
 {"name": "Insertion Loss (dB)", "key": "mean\_peak\_il", "cmap": "plasma"},  
 {"name": "1dB Bandwidth (nm)", "key": "mean\_bw\_1db", "cmap": "inferno"},  
 ]  
  
 # Draw 8-inch wafer circle (radius = 4.3, approximate scale)  
 wafer\_circle\_props = {  
 "fill": False,  
 "edgecolor": "black",  
 "linestyle": "-", # 改为实线  
 "linewidth": 2,  
 "alpha": 0.8, # 稍微增加不透明度  
 }  
  
 # Plot wafer map for each parameter  
 for i, param in enumerate(params):  
 ax = axes[i]  
  
 # Extract parameter values  
 values = [valid\_data[map\_id][param["key"]] for map\_id in valid\_data]  
 if not values:  
 raise ValueError(f"No valid data for parameter '{param['name']}'")  
  
 vmin, vmax = min(values), max(values)  
  
 # Draw each die (using squares)  
 for map\_id, (x, y) in coordinates.items():  
 if map\_id in valid\_data:  
 value = valid\_data[map\_id][param["key"]]  
  
 # Draw square with separate facecolor and edgecolor  
 square = plt.Rectangle(  
 (x - 0.45, y - 0.45),  
 0.9,  
 0.9,  
 facecolor=plt.get\_cmap(param["cmap"])(  
 Normalize(vmin, vmax)(value)  
 ),  
 edgecolor="black",  
 linewidth=1,  
 )  
 ax.add\_patch(square)  
  
 # Add wafer ID (transparent)  
 ax.text(  
 x, y, map\_id, ha="center", va="center", fontsize=8, alpha=0.2  
 )  
  
 # Add parameter value (bold and opaque)  
 ax.text(  
 x,  
 y,  
 f"{value:.2f}",  
 ha="center",  
 va="center",  
 fontsize=4,  
 fontweight="bold",  
 alpha=1,  
 )  
  
 # Add wafer circle to indicate 8-inch wafer  
 wafer\_circle = plt.Circle((-3, 1), 4.3, \*\*wafer\_circle\_props)  
 ax.add\_patch(wafer\_circle)  
  
 # Set chart properties  
 ax.set\_aspect("equal")  
 ax.set\_title(f"{param['name']}", fontsize=10)  
 ax.set\_xlim(-9, 3)  
 ax.set\_ylim(-5, 7)  
  
 # Hide axes  
 ax.axis("off")  
  
 # Add colorbar (调整大小)  
 sm = ScalarMappable(cmap=param["cmap"], norm=Normalize(vmin, vmax))  
 sm.set\_array([])  
  
 # 使用shrink参数调整colorbar大小  
 cbar = fig.colorbar(sm, ax=ax, pad=0.02, shrink=0.5) # 缩小到原大小的70%  
 cbar.set\_label(param["name"], fontsize=10) # 稍微减小标签字体  
  
 plt.tight\_layout()  
 plt.savefig("wafer\_maps.png", dpi=300, bbox\_inches="tight")  
 plt.show()  
 print(f"Successfully generated wafer maps and saved to wafer\_maps.png")  
  
 except FileNotFoundError:  
 print(f"Error: File {yaml\_file} not found, please check file path")  
 except yaml.YAMLError as e:  
 print(f"YAML parsing error: {e}, please check file format")  
 except ValueError as e:  
 print(f"Data error: {e}")  
 except Exception as e:  
 print(f"Unknown error: {e}, please check data integrity")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 plot\_wafer\_maps("wafer\_parameters\_summary.yaml")

这段代码通过读取 YAML 格式的参数分布数据，将关键参数以晶圆地图形式直观展示。以下是对代码内容的详细解析：

**代码功能概述**

代码通过 matplotlib 库创建晶圆地图可视化，将硅光芯片晶圆上各个芯片的三个关键参数（中心波长、插入损耗和 1dB 带宽）以颜色编码的方式展示在晶圆布局上。这有助于工程师快速识别晶圆上不同位置芯片的性能分布，发现异常区域并分析工艺波动的影响。

**模块解析**

**导入模块**：引入必要的库，包括 YAML 解析、数值计算、绘图工具以及晶圆坐标映射模块。

**主函数定义**：plot\_wafer\_maps函数负责读取数据、处理坐标映射、创建可视化图表并保存结果。

**数据读取与验证**：

从 YAML 文件加载晶圆测试数据，格式要求包含晶圆 ID 及对应参数。

将 YAML 中使用的晶圆 ID（0-42）映射到晶圆坐标系统中的 ID（1-43）。

验证数据完整性，确保每个晶圆包含所需的三个参数。

**可视化准备**：

* 创建 1 行 3 列的子图布局，分别展示三个参数。
* 定义每个参数的显示配置（名称、数据键、颜色映射）。

**晶圆地图绘制**：

* 为每个参数创建颜色映射，根据参数值的范围设置颜色渐变。
* 遍历晶圆上的每个芯片位置，使用正方形表示芯片，颜色深浅表示参数值大小。
* 在正方形上标注晶圆 ID 和具体参数值。

**图表美化与保存**：

* 添加晶圆轮廓圆，标记 8 英寸晶圆的范围。
* 为每个子图添加颜色条，显示参数值与颜色的对应关系。
* 设置合适的布局，保存图像并显示结果。

**错误处理**：捕获并处理可能的文件不存在、YAML 解析错误、数据格式问题等异常情况。

**关键技术细节**

**坐标映射与可视化**：通过date\_classification.all.wafer模块获取晶圆上芯片的物理坐标，将测试数据与物理位置对应。

**颜色编码**：使用不同的颜色映射（viridis、plasma、inferno）为三个参数分别创建直观的颜色表示，参数值通过颜色深浅体现。

**参数标注**：在每个芯片位置同时标注晶圆 ID（半透明小字体）和参数值（不

基于上面代码建立的光栅耦合器基于位置摆放的模型如下：

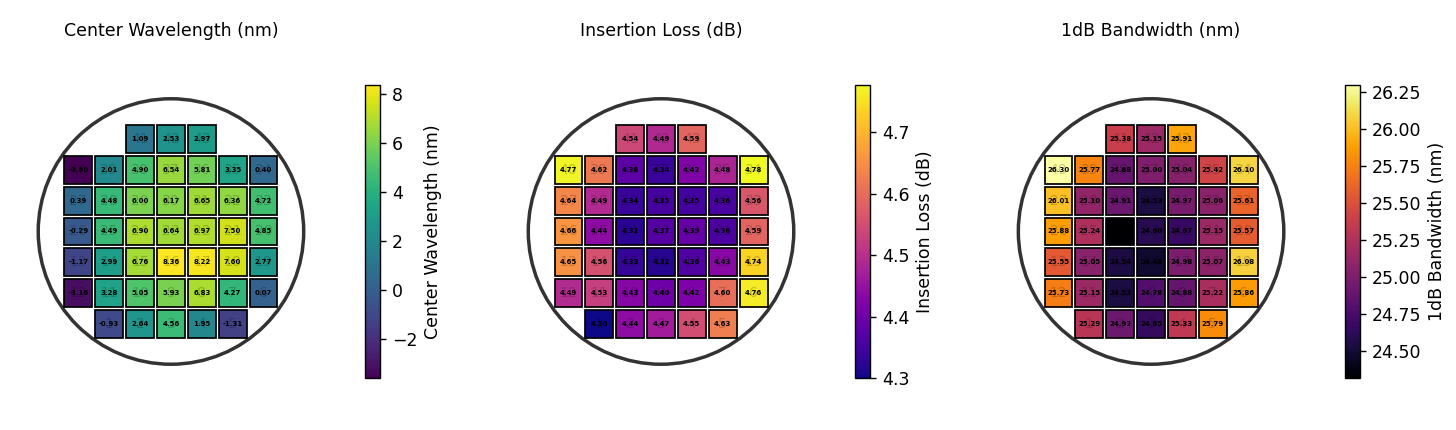


图7-2 光栅耦合器基于位置摆放的统计学模型分布

## 8.3 基于工艺波动的统计学建模程序设计

基于6-2节介绍的波导有效折射率和群折射率提取程序，可以得到波导有效折射率和群折射率的分布，基于该分布可以得到波导厚度、线宽以及倾角的分布。

要想实现该过程，首先需要通过仿真代码建立波导厚度、线宽以及倾角与波导有效折射率与群折射率之间的联系。

波导仿真代码如下:

import numpy as np  
import matplotlib.pyplot as plt  
from shapely.geometry import Polygon  
from shapely.ops import clip\_by\_rect  
from collections import OrderedDict  
from femwell.mesh import mesh\_from\_OrderedDict  
from femwell.maxwell.waveguide import compute\_modes  
from skfem import Basis, ElementTriP0  
import math  
import refractiveindex as ri  
from skfem.io.meshio import from\_meshio  
from femwell.visualization import plot\_domains  
def simulate\_soi\_strip\_waveguide\_with\_etch(  
 bottom\_width=0.5, # Bottom width of the waveguide core (µm)  
 height=0.22, # Waveguide height/thickness (µm)  
 etch\_angle=10.0, # Etching angle from vertical (degrees)  
 core\_resolution=0.02, # Mesh resolution for core (µm)  
 wavelength=1.55, # Wavelength (µm)  
 num\_modes=2, # Number of modes to compute,  
 plot\_mesh=False, # Whether to plot the mesh  
 plot\_modes=False # Whether to plot mode profiles  
):  
 # Step 1: Define geometry with etching angle  
 theta\_rad = math.radians(etch\_angle)  
 top\_offset = height \* math.tan(theta\_rad)  
 top\_width = bottom\_width - 2 \* top\_offset  
  
 # Ensure top\_width is sufficiently large  
 min\_width = core\_resolution \* 2 # At least twice the mesh resolution  
 if top\_width < min\_width:  
 raise ValueError(  
 f"Top width ({top\_width:.6f} µm) is too small with "  
 f"bottom\_width={bottom\_width}, height={height}, etch\_angle={etch\_angle}°. "  
 f"Must be > {min\_width} µm. Reduce etch\_angle or increase bottom\_width."  
 )  
  
 # Define trapezoidal core  
 core = Polygon([  
 (-bottom\_width / 2, 0),  
 (bottom\_width / 2, 0),  
 (top\_width / 2 , height),  
 (-top\_width / 2, height)  
 ])  
 print("Core vertices:", list(core.exterior.coords))  
  
 # Step 2: Assign regions  
 polygons = OrderedDict(  
 core=core,  
 box=clip\_by\_rect(core.buffer(1.0, resolution=4), -np.inf, -np.inf, np.inf, 0),  
 clad=clip\_by\_rect(core.buffer(1.0, resolution=4), -np.inf, 0, np.inf, np.inf),  
 )  
  
 resolutions = {"core": {"resolution": core\_resolution, "distance": 2}}  
  
 mesh = from\_meshio(mesh\_from\_OrderedDict(polygons, resolutions, default\_resolution\_max=0.6))  
  
 # Step 6: Define basis and permittivity  
 basis0 = Basis(mesh, ElementTriP0())  
 epsilon = basis0.zeros(dtype=complex)  
  
 # Material properties  
 si\_data = ri.RefractiveIndexMaterial("main","Si","Li-293K")  
 n\_si = si\_data.get\_refractive\_index(wavelength\*1000) # wavelength in µm  
  
 # Silicon Dioxide (SiO₂) - Fused silica, Malitson (1965)  
 sio2\_data = ri.RefractiveIndexMaterial("main","SiO2", "Malitson")  
 n\_sio2 = sio2\_data.get\_refractive\_index(wavelength\*1000)  
 for subdomain, n in {"core": n\_si, "box": n\_sio2, "clad": n\_sio2}.items():  
 epsilon[basis0.get\_dofs(elements=subdomain)] = n \*\* 2  
  
 # Step 7: Compute modes  
 modes = compute\_modes(  
 basis0,  
 epsilon,  
 wavelength=wavelength,  
 num\_modes=num\_modes  
 )  
  
 # Step 8: Extract effective indices  
 effective\_indices = [mode.n\_eff for mode in modes]  
 print(f"Effective indices: {np.real(effective\_indices)}")  
  
 wavelength\_in\_unit = wavelength\*1e6 # wavelength in meter  
 alpha\_indices = [(2 \* np.pi / wavelength\_in\_unit) \* neff.imag for neff in effective\_indices]  
 loss\_indices = [20 \* np.log10(np.e) \* alpha for alpha in alpha\_indices] #loss dB/m  
 print(f"loss: {loss\_indices} (dB/m)")  
  
 # Step 9: Optional mesh visualization  
 if plot\_mesh:  
 mesh.draw().show()  
 plot\_domains(mesh)  
 plt.show()  
  
 # Step 10: Optional mode visualization  
 if plot\_modes:  
 for i, mode in enumerate(modes):  
 mode.show("I", colorbar=True)  
  
 return effective\_indices, modes  
  
  
# Example usage  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 neffs, modes = simulate\_soi\_strip\_waveguide\_with\_etch(  
 bottom\_width=0.5,  
 height=0.22,  
 etch\_angle=5.0,  
 core\_resolution=0.005,  
 plot\_mesh=True,  
 plot\_modes=True,  
 num\_modes=2  
 )

本程序用于模拟带有蚀刻角度的SOI (Silicon-on-Insulator) strip波导的光学特性，包括计算模式有效折射率和场分布。程序基于Python实现，主要使用了以下库：

* numpy和matplotlib：数值计算和可视化
* shapely：几何形状定义和处理
* femwell：有限元网格生成和模式求解
* refractiveindex：材料折射率数据获取

导入库和模块

import numpy as np

import matplotlib.pyplot as plt

from shapely.geometry import Polygon

from shapely.ops import clip\_by\_rect

from collections import OrderedDict

from femwell.mesh import mesh\_from\_OrderedDict

from femwell.maxwell.waveguide import compute\_modes

from skfem import Basis, ElementTriP0

import math

import refractiveindex as ri

from skfem.io.meshio import from\_meshio

from femwell.visualization import plot\_domains

这部分导入了程序所需的所有库和模块，包括：

* 数值计算和绘图基础库
* 几何处理工具
* 有限元分析相关模块
* 材料折射率数据库接口

主函数定义

def simulate\_soi\_strip\_waveguide\_with\_etch(

bottom\_width=0.5, # 波导底部宽度(µm)

height=0.22, # 波导高度(µm)

etch\_angle=10.0, # 蚀刻角度(度)

core\_resolution=0.02, # 核心网格分辨率(µm)

wavelength=1.55, # 波长(µm)

num\_modes=2, # 计算模式数量

plot\_mesh=False, # 是否绘制网格

plot\_modes=False # 是否绘制模式图

):

主函数simulate\_soi\_strip\_waveguide\_with\_etch定义了波导的几何参数和模拟选项，包括默认值。

几何形状计算

# Step 1: Define geometry with etching angle

theta\_rad = math.radians(etch\_angle)

top\_offset = height \* math.tan(theta\_rad)

top\_width = bottom\_width - 2 \* top\_offset

# Ensure top\_width is sufficiently large

min\_width = core\_resolution \* 2 # At least twice the mesh resolution

if top\_width < min\_width:

raise ValueError(...)

这部分计算考虑蚀刻角度后的波导顶部宽度，并进行有效性检查，确保顶部宽度足够大以保证网格划分质量。

波导核心定义

# Define trapezoidal core

core = Polygon([

(-bottom\_width / 2, 0),

(bottom\_width / 2, 0),

(top\_width / 2 , height),

(-top\_width / 2, height)

])

print("Core vertices:", list(core.exterior.coords))

使用**shapely.Polygon**定义梯形波导核心的几何形状，并打印顶点坐标用于验证。

区域划分和网格生成

# Step 2: Assign regions

polygons = OrderedDict(

core=core,

box=clip\_by\_rect(core.buffer(1.0, resolution=4), -np.inf, -np.inf, np.inf, 0),

clad=clip\_by\_rect(core.buffer(1.0, resolution=4), -np.inf, 0, np.inf, np.inf),

)

resolutions = {"core": {"resolution": core\_resolution, "distance": 2}}

mesh = from\_meshio(mesh\_from\_OrderedDict(polygons, resolutions, default\_resolution\_max=0.6))

定义三个区域：核心(core)、衬底(box)和包层(clad)，并为不同区域设置网格分辨率参数，然后生成有限元网格。

材料属性和介电常数分布

# Step 6: Define basis and permittivity

basis0 = Basis(mesh, ElementTriP0())

epsilon = basis0.zeros(dtype=complex)

# Material properties

si\_data = ri.RefractiveIndexMaterial("main","Si","Li-293K")

n\_si = si\_data.get\_refractive\_index(wavelength\*1000) # wavelength in µm

# Silicon Dioxide (SiO₂) - Fused silica, Malitson (1965)

sio2\_data = ri.RefractiveIndexMaterial("main","SiO2", "Malitson")

n\_sio2 = sio2\_data.get\_refractive\_index(wavelength\*1000)

for subdomain, n in {"core": n\_si, "box": n\_sio2, "clad": n\_sio2}.items():

epsilon[basis0.get\_dofs(elements=subdomain)] = n \*\* 2

创建有限元基函数，初始化介电常数分布，并从折射率数据库中获取硅和二氧化硅在指定波长下的折射率，然后为不同区域分配相应的介电常数值。

模式计算

# Step 7: Compute modes

modes = compute\_modes(

basis0,

epsilon,

wavelength=wavelength,

num\_modes=num\_modes

)

调用**compute\_modes**函数计算波导中的模式。

结果提取和分析

# Step 8: Extract effective indices

effective\_indices = [mode.n\_eff for mode in modes]

print(f"Effective indices: {np.real(effective\_indices)}")

wavelength\_in\_unit = wavelength\*1e6 # wavelength in meter

alpha\_indices = [(2 \* np.pi / wavelength\_in\_unit) \* neff.imag for neff in effective\_indices]

loss\_indices = [20 \* np.log10(np.e) \* alpha for alpha in alpha\_indices] #loss dB/m

print(f"loss: {loss\_indices} (dB/m)")

提取模式的有效折射率并计算相应的传播损耗。

可视化选项

# Step 9: Optional mesh visualization

if plot\_mesh:

mesh.draw().show()

plot\_domains(mesh)

plt.show()

# Step 10: Optional mode visualization

if plot\_modes:

for i, mode in enumerate(modes):

mode.show("I", colorbar=True)

根据输入参数决定是否显示网格和模式场分布。

示例使用

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

neffs, modes = simulate\_soi\_strip\_waveguide\_with\_etch(

bottom\_width=0.5,

height=0.22,

etch\_angle=5.0,

core\_resolution=0.005,

plot\_mesh=True,

plot\_modes=True,

num\_modes=2

)

运行该使用示例，得到的波导的仿真结果：

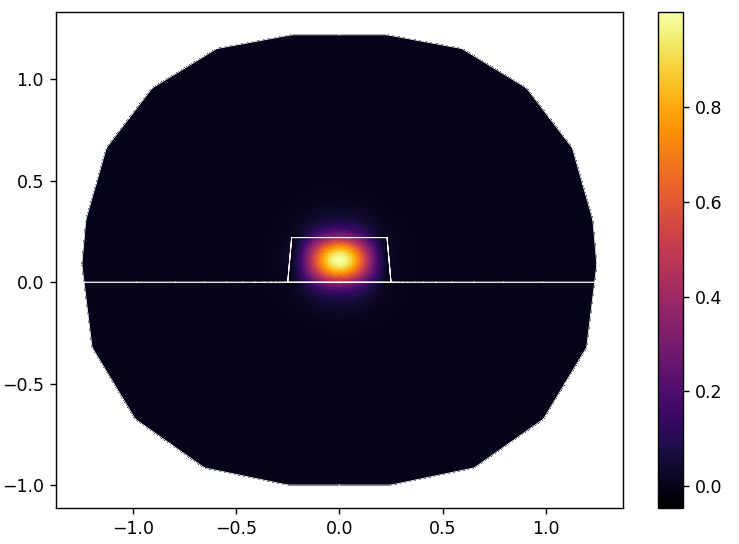


图 7-3-1 Strip波导TE模式仿真结果

图7-3-1为波导中TE模式的方分布，其有效折射率和群折射率分别为：

2.41172193和1.75680383

通过扫描波导宽度和厚度，就可以得到不同宽度与厚度下的波导折射率与群折射率。

import numpy as np  
from Femwell\_strip import simulate\_soi\_strip\_waveguide\_with\_etch  
import itertools  
import time  
from multiprocessing import Pool, cpu\_count  
  
# 假设你的函数比较耗时  
def wg\_size\_sweep(width, height, wavelength, etch\_angle = 5):  
 """一个模拟耗时计算的函数"""  
 # print(f"Starting computation for a={a}, b={b}, c={c}")  
 # time.sleep(0.1) # 模拟计算耗时  
 n\_eff, \_ = simulate\_soi\_strip\_waveguide\_with\_etch(  
 bottom\_width=width,  
 height=height,  
 etch\_angle=etch\_angle,  
 core\_resolution=0.005,  
 plot\_mesh=False,  
 plot\_modes=False,  
 wavelength = wavelength,  
 num\_modes=1)  
 # print(f"Finished computation for a={a}, b={b}, c={c}")  
 return width, height, wavelength, etch\_angle, n\_eff[0].real # 返回输入和输出，方便对照  
  
  
width = np.linspace(0.43,0.46,7)  
height = np.linspace(0.210,0.225,4)  
wavelength = np.linspace(1.549,1.551,3)  
  
# 生成所有参数组合  
param\_combinations = list(itertools.product(width, height, wavelength)) # 需要先转成列表  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 num\_workers = cpu\_count() # 使用所有 CPU 核心  
 print(f"Using {num\_workers} workers for parallel processing.")  
  
 with Pool(processes=num\_workers) as pool:  
 results\_parallel = pool.starmap(wg\_size\_sweep, param\_combinations)  
  
 data = {  
 "width": [float(x[0]) for x in results\_parallel],  
 "height": [float(x[1]) for x in results\_parallel],  
 "wavelength": [float(x[2]) for x in results\_parallel],  
 "angle": [float(x[3]) for x in results\_parallel],  
 "N\_eff": [float(x[4]) for x in results\_parallel]  
 }  
  
 import yaml  
  
 with open("wg\_neff\_sweep\_width\_height\_wavelength\_angle.yml","w") as file:  
 yaml.dump(data,file)  
 print("done")

其中波导宽度与厚度的扫描范围为：

width = np.linspace(0.43,0.46,7)  
height = np.linspace(0.210,0.225,4)

最终的仿真结果保存在: wg\_neff\_sweep\_width\_height\_wavelength\_angle.yml文件中。

基于波导有效折射率和群折射率的分布数据：strip\_neff\_ng\_KDE.data，以及通过参数扫描的到的有效折射率和群折射率与波导宽度与厚度之间的关系：Mapping\_from\_[Ng,Neff]\_to\_[width,height]\_angle\_5.data。就可以得到波导的宽度与厚度分布，其代码设计如下：

import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import pickle  
from scipy\_poly\_model import MultiPolyModel  
from scipy.stats import gaussian\_kde  
from typing import List, Tuple  
  
# 加载KDE模型和映射模型  
with open("strip\_neff\_ng\_KDE.data", "rb") as f:  
 kde = pickle.load(f)  
  
with open("Mapping\_from\_[Ng,Neff]\_to\_[width,height]\_angle\_5.data", "rb") as f:  
 mapping\_model = pickle.load(f)  
  
# 生成样本数据  
samples = kde.resample(10000).T  
neff\_data = samples[:, 0]  
ng\_data = samples[:, 1]  
  
# 使用列表推导式预测宽度和高度  
predictions = [  
 mapping\_model.predict([ng, neff]) for ng, neff in zip(ng\_data, neff\_data)  
]  
widths, heights = zip(\*predictions)  
  
# 转换为numpy数组（如果需要）  
widths = np.array(widths).flatten() \* 1000  
heights = np.array(heights).flatten() \* 1000  
  
plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC"]  
plt.rcParams["axes.unicode\_minus"] = False # 解决负号显示问题  
  
# 创建联合分布图（已设置高度和比例）  
g = sns.jointplot(  
 x=widths,  
 y=heights,  
 kind="scatter",  
 color="skyblue",  
 height=6, # 控制图形高度（英寸）  
 ratio=4, # 控制主图与边缘图的比例  
 space=0.1,  
 xlim=(min(widths) - 1, max(widths) + 1),  
 ylim=(min(heights) - 1, max(heights) + 1),  
 marginal\_kws=dict(bins=100, kde=True), # 设置直方图 bins 数量  
)  
  
# 设置标题（使用 JointGrid 对象的方法）  
# g.fig.suptitle("映射点的联合分布")  
plt.subplots\_adjust(top=1, left=0.12, bottom=0.1) # 调整标题位置  
  
plt.xlabel("width (nm)", fontfamily="Times New Roman", fontsize=14)  
plt.ylabel("height (nm)", fontfamily="Times New Roman", fontsize=14)  
  
plt.title("示例图形", fontsize=16) # 标题字体可能仍受全局设置影响  
plt.grid(True, linestyle="--", alpha=0.7)  
plt.show()

运行该代码的，得到的波导宽度与厚度分布如下：

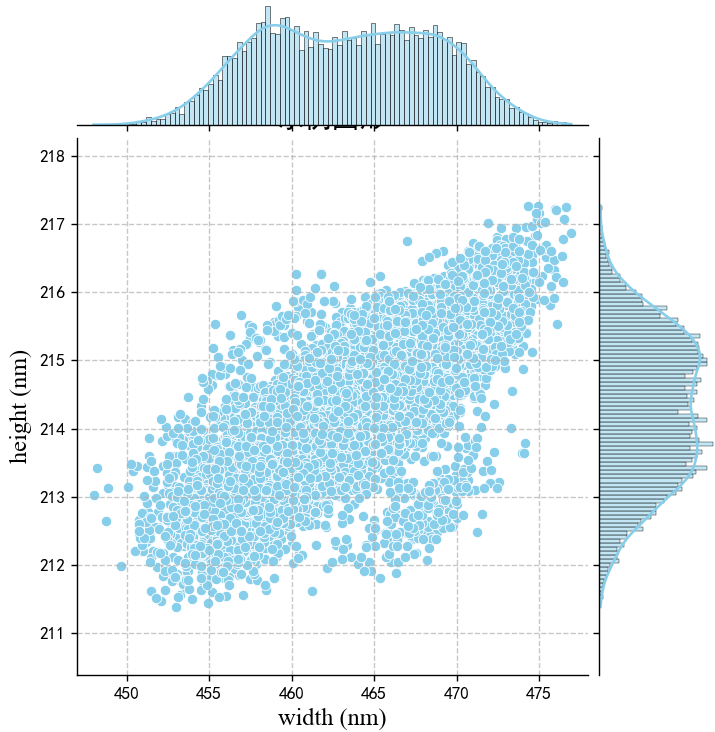


图7-3-2 基于波导有效折射率与群折射率分布得到的波导宽度与厚度分布