
ALUVIA Documentation

Release 1.0

Luceda Photonics

Sep 21, 2023

PDK Documentation

1	Changelog	2
2	Document information	3
3	Reference design flow	3
3.1	Circuit layout and routing	4
3.2	Component design	4
3.3	Library management	4
4	Getting Started	5
4.1	Extracting the PDK	5
4.2	Using the PDK with IPKISS	5
5	Technology settings	7
5.1	Layers	7
6	PDK Components	7
6.1	StripWaveguide	7
6.2	StripWaveguide_369	9
6.3	StripWaveguide_405	11
6.4	TaperPCell	13
6.5	SBend	15
6.6	EulerBendPcell	17
6.7	ArcPathPcell	19
6.8	SquareSpiralPcell	21
6.9	SquareSpiralEulerPcell	21
6.10	SpiralPcell	22
6.11	SpiralEulerPcell	22
6.12	YBranchPcell	23
6.13	DirectionalCouplerPcell	25
6.14	RingRes_1x1Pcell	27
6.15	RingRes_2x2Pcell	29
6.16	MMI_1x2Pcell	31
6.17	MMI_2x2Pcell	33
6.18	MMI_nxmPcell	35
6.19	LinearEdgeTaperPcell	37
6.20	Template5000x5000	39

6.21	Template10000x10000	40
6.22	HeaterWaveguide	41
6.23	MetalWire	43
6.24	DCTaper	45
6.25	DCBend	47
6.26	DCPad	49
6.27	DCPadArray	51
6.28	WG_TMPL	53
6.29	HeaterTemplate	53
6.30	MetalWireTemplate	54
Index		55



This document explains how to use the Luceda Process Design Kit (PDK) for ALUVIA (version 1.0).

The PDK can be used from [IPKISS](#), providing the following functionality:

- definition and layout of circuits using ALUVIA's predefined library of cells (building blocks), which is entirely available within IPKISS,
- waveguide routing using the waveguide types defined,
- definition of custom devices using defined layers,
- simulation of custom devices using IPKISS' built-in solvers and Luceda Links to 3rd party tools such as Dassault Systemes Simulia CST Studio Suite ® and Ansys Lumerical FDTD,
- circuit editing from Siemens L-Edit using [Luceda Link for Siemens EDA](#).

This document explains how to get started with the PDK, lists the cells that are available and documents the technology for custom design.

1 Changelog

This changelog shows the changes in the Luceda PDK for ALUVIA.

1.0

- Updated Heaters with Heater Crosstalk region
- Added Virtual Fabrication (Vfab)
- Added an .iclib file (library file recognized by IPKISS Canvas) with symbol drawings

0.1

- Initial version of the PDK

2 Document information

License

The use of this PDK is subject to the “ALUVIA User Non-Disclosure Agreement”.

Usage and responsibility

The use of this PDK is restricted to the design of photonic integrated circuits in the ALUVIA platform, fabricated by ALUVIA (*Foundry*).

- All relevant documentation about the ALUVIA platform, including process description and design rules, can be found in the *ALUVIA Design Manual* as provided by the Foundry. The user is advised to carefully read the *ALUVIA Design Manual*.
- It is the user’s responsibility to verify that the design, made on the basis of this Process Design Kit, meets the Foundry’s design rules.
- In case of doubt, the user should consider the Foundry’s design rule manual as the correct reference (as validated by the Foundry).
- Under no circumstances does LUCEDA represent or warrant that operation of the photonic integrated circuits by the Foundry, designed on the basis of this Process Design Kit, will be error-free or will accord to the user specifications, and hereby disclaims all liability on account thereof.

3 Reference design flow

This chapter describes the reference design flow for the Luceda PDK for ALUVIA, with its functionalities and tooling.

Several types of design actions and design intents are considered:

- Photonic IC circuit design based on pre-defined building blocks and subcircuits
- Photonic IC component design and exploration based on technology information
- Library management

The following tools are used in the reference design flow:

Function	Tool	Versions supported
Circuit layout	L-edit	2019.x, 2020.x
Waveguide routing	IPKISS	3.9.1
Subcircuit layout	IPKISS	3.9.1
	L-Edit	2019.x, 2020.x
Component layout	IPKISS	3.9.1
Component simulation	IPKISS	3.9.1
	CST Studio Suite (FDTD)	2017, 2018
	FDTD Solutions (FDTD)	

Product licenses

- IPKISS is a product of Luceda Photonics. Circuit design and waveguide routing from L-Edit requires a Luceda Link for Siemens EDA license.
- L-Edit and Calibre are products of Mentor Graphics (Tanner), a Siemens company.

- CST Studio suite is a product of CST - Dassault Systèmes.
- FDTD Solutions is a product of Lumerical.

Software licenses have to be procured from the separate vendors.

3.1 Circuit layout and routing

Circuit layout is performed from within L-Edit or from IPKISS python script, depending on the user's preferences.

- Using Luced Link for Siemens EDA, PDK pcells as well as customer-defined cells are available within the L-Edit environment. From the L-Edit GUI, components can be instantiated using drag-and-drop, components can be connected conveniently with waveguides and metal wiring can be done.
- Using IPKISS, the same PDK pcells as well as customer-defined cells are available and can be laid out into a circuit using Python scripting. IPKISS provides features for semi-automated waveguide routing including fanout bundles and other convenience functions.

3.2 Component design

Component and subcircuit layout

User-defined components and subcircuits can be created in IPKISS python script and made available in the L-Edit design environment. IPKISS provides basic building block definition based on technology layers, as well as powerful macros for waveguide routing, device placement, fanouts, and so forth.

Component simulation

Optical (electromagnetic) simulation of devices can be managed with IPKISS. IPKISS provides a built-in 2D mode solver and propagation tool. For more accurate simulations, CST Studio Suite as well as Lumerical FDTD Solutions can be driven in a semi-automated and reproducible way. The simulation results can directly be imported in IPKISS and used as a compact model in the circuit simulation (Caphe).

Device exploration

For the creation of device exploration masks, IPKISS provides features for creating component parameter sweeps and stacking on chip.

3.3 Library management

The customer can create his/her own library of components using the functionalities and tools described above. IPKISS allows to keep a unified description of building blocks with layout, compact model and device simulation recipes. The components can then be used from within IPKISS (python scripts) as well as L-Edit.

4 Getting Started

This section explains how to load the Luceda PDK for ALUVIA and basics of how to use it.

If you are unfamiliar with Luceda IPKISS, please have a look at the IPKISS documentation and samples. The documentation and samples are available from the IPKISS Control Center.

The latest IPKISS documentation is also available online at <http://docs.lucedaphotonics.com>.

4.1 Extracting the PDK

The PDK is distributed as a zip file, which you can extract with the unzip utility of your choice (unzip, WinZip, 7-zip, ...).

For the purpose of this manual, we assume you have unzipped “aluvia.x.x_buildnr.zip” as *C:\designs\pdk\aluvia*.

You can inspect the directory structure of the PDK:

- docs/html/: Contains this file
- docs/examples/: Example IPKISS python scripts
- ipkiss/: IPKISS technology, cells and functions (Python)
- ipkiss/aluvia/aluvia.iclib: Symbol library for the PDK (IPKISS Canvas)
- openaccess/: Design libraries
- openaccess/aluvia/: OpenAccess database for the PDK (L-Edit)
- techfiles/: Tech files (layers, display) for various tools
- techfiles/klayout/: Layer settings for KLayout GDSII visualization

4.2 Using the PDK with IPKISS

Setting up the editor environment

You can follow the instructions given in our documentation http://docs.lucedaphotonics.com/tutorials/environment_setup/create_project_pdk.html.

Examples

The PDK comes bundled with a series of examples for use in IPKISS, in the *<path_to_pdk>\docs\examples*. These are executable Python files. The examples are a good starting point to learn how to design with this PDK. Below is a summary of the shipped examples:

Example name	Description
example_bondpad.py	Visualize all the bondpads that are available in the ALUVIA technology.
example_heater.py	Visualize heater Pcell that is available in the ALUVIA technology.
example_layers.py	Visualize all the layers that are available in the ALUVIA technology.
example_passives.py	Visualize all the Passive devices that are available in the ALUVIA technology.
example_vfab.py	Visualize the Virtual Fabrication available in the PDK.
example_waveguide.py	Visualize the waveguides and transitions that are available in the PDK.

5 Technology settings

The PDK contains layer definitions and virtual fabrication settings. Refer to the ALUVIA design rule manual.

5.1 Layers

The following table gives an overview of the available layers and how they are named in Ipkiss. See the ALUVIA design rules manual for further information about the layers defined by ALUVIA.

Table 5.1: Layer Table

Layer name	Description	IPKISS (TECH.PPLAYER.)	Layer Number
WG	Al2O3 WaveguideLayer Drawing	WG	1/0
WG_PINREC	Al2O3 WaveguideLayer PinRecognition	WG_PINREC	1/31
DICELINE	DicingLine	DICELINE	41/33
BLEED	Bleed	BLEED	41/34
TEXT	Text	TEXT	41/50
EBEAM_DICELINE	E-beam DicingLine	EBEAM_DICELINE	61/33
EBEAM_TEXT	E-beam Text	EBEAM_TEXT	61/50
HEATER	Heater Drawing	HEATER	53/0
HEATER_CROSSTALK	Heater Crosstalk	HEATER_CROSSTALK	53/60
M1	MetalLayer Drawing	M1	51/0

6 PDK Components

Waveguides

<code>aluvia.all.StripWaveguide</code>	Strip Waveguide
<code>aluvia.all.StripWaveguide_369</code>	Strip Waveguide operating at 369nm wavelength.
<code>aluvia.all.StripWaveguide_405</code>	Strip Waveguide operating at 405nm wavelength.
<code>aluvia.all.TaperPCell</code>	Taper Waveguide.
<code>aluvia.all.SBend</code>	S-Bend Waveguide.
<code>aluvia.all.EulerBendPcell</code>	Euler Bend Waveguide.
<code>aluvia.all.ArcPathPcell</code>	Arc Waveguide

6.1 StripWaveguide

```
class aluvia.all.StripWaveguide(*args, **kwargs)
```

Strip Waveguide

Parameters

trace_template: PCell and _TraceTemplate

Template of the waveguide

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters

The unique name of the pcell

Views

class Layout(*args, **kwargs)

Parameters

width: float and number > 0

Width of the core [um]

shape: Shape

Shape from which the Trace is calculated

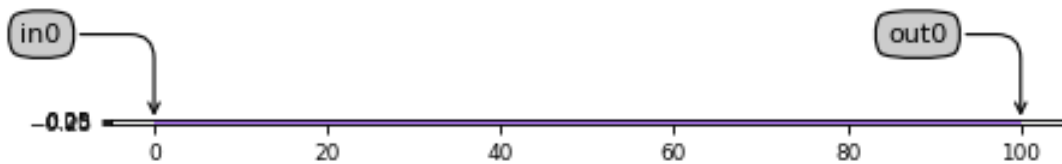
bend_radius: float and number > 0

Bend radius for the auto-generated bends.

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(100.0, 0.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import StripWaveguide

lo = StripWaveguide().Layout()
lo.visualize(annotate=True)
```



6.2 StripWaveguide_369

class aluvia.all.StripWaveguide_369(*args, **kwargs)

Strip Waveguide operating at 369nm wavelength.

Parameters

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters

The unique name of the pcell

Views

class Layout(*args, **kwargs)

Parameters

shape: Shape

Shape from which the Trace is calculated

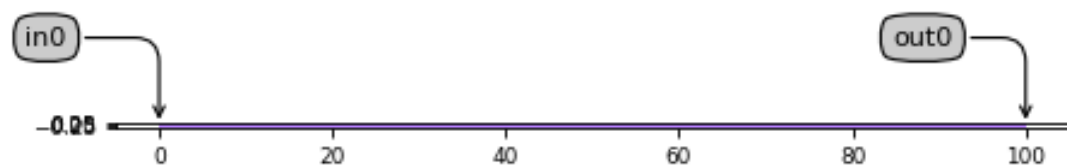
bend_radius: float and number > 0

Bend radius for the auto-generated bends.

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(100.0, 0.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import StripWaveguide_369
```

```
lo = StripWaveguide_369().Layout()
lo.visualize(annotate=True)
```



6.3 StripWaveguide_405

```
class aluvia.all.StripWaveguide_405(*args, **kwargs)
```

Strip Waveguide operating at 405nm wavelength.

Parameters

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters

The unique name of the pcell

Views

```
class Layout(*args, **kwargs)
```

Parameters

shape: Shape

Shape from which the Trace is calculated

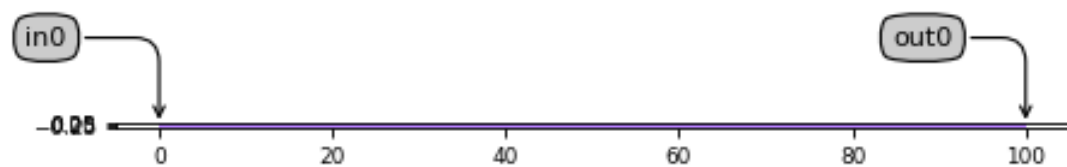
bend_radius: float and number > 0

Bend radius for the auto-generated bends.

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(100.0, 0.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import StripWaveguide_405
```

```
lo = StripWaveguide_405().Layout()
lo.visualize(annotate=True)
```



6.4 TaperPCell

```
class aluvia.all.TaperPCell(*args, **kwargs)
```

Taper Waveguide.

Views

```
class Layout(*args, **kwargs)
```

Parameters

final_width: float and number > 0

Width of the taper at the end.

initial_width: float and number > 0

Width of the taper at the start.

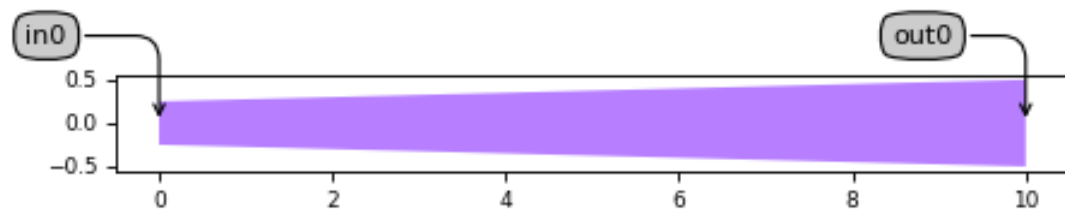
length: float and Real, number and number >= 0

Length of the taper.

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(10.0, 0.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import TaperPCell

lo = TaperPCell().Layout()
lo.visualize(annotate=True)
```



6.5 SBend

```
class aluvia.all.SBend(*args, **kwargs)
    S-Bend Waveguide.
```

Views

```
class Layout(*args, **kwargs)
```

Parameters

height: float

Height of the sbend [um]

length: float and number > 0

Length of the sbend [um]

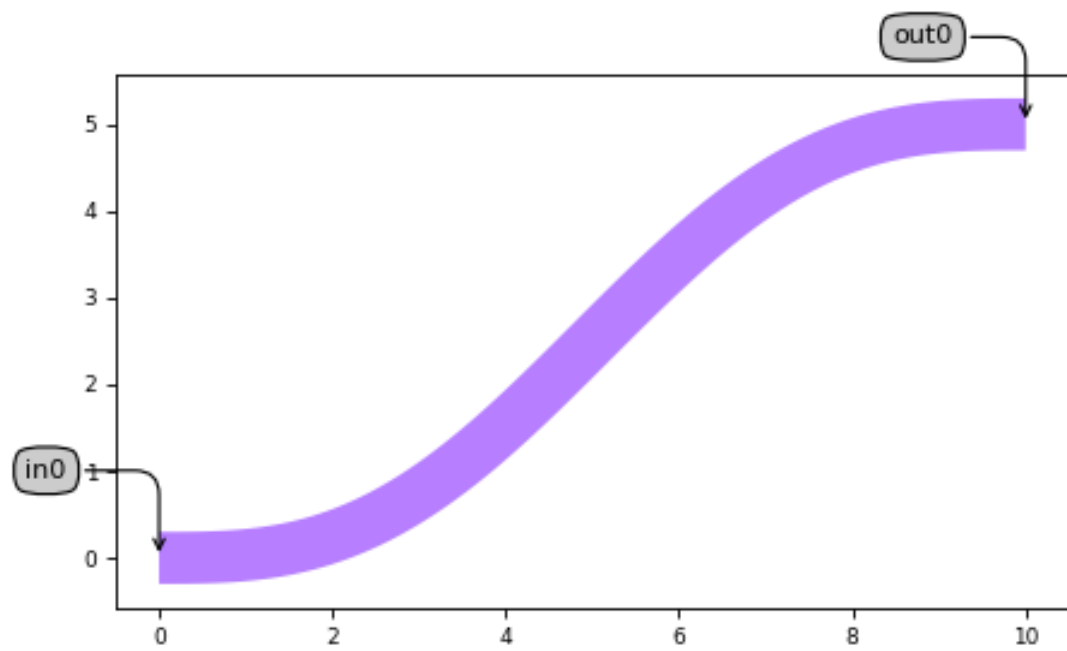
width: float and number > 0

Width of the sbend [um]

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(10.0, 5.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import SBend

lo = SBend().Layout()
lo.visualize(annotate=True)
```



6.6 EulerBendPcell

```
class aluvia.all.EulerBendPcell(*args, **kwargs)
    Euler Bend Waveguide.
```

Views

```
class Layout(*args, **kwargs)
```

Parameters

arc_angle: float
Angle of the Euler bend [degree]

min_radius: float and number > 0
Radius of the bend [um]

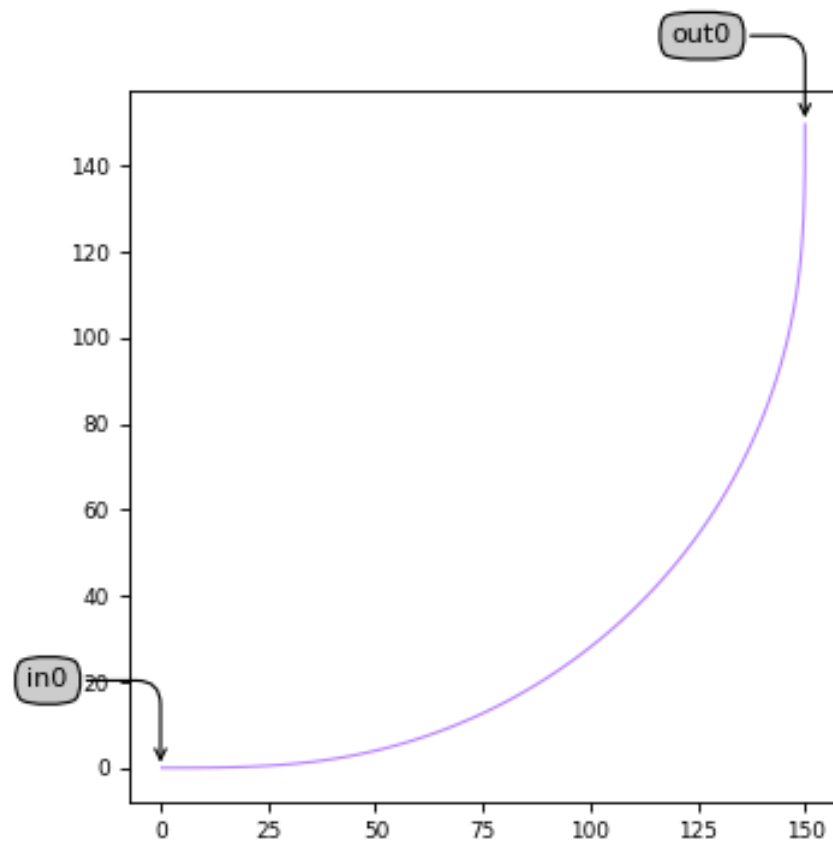
p_factor: float and number > 0
Fraction of the bend having linearly increasing curvature

width: float and number > 0
Width of the Euler bend [um]

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(150.0, 150.0)	90.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import EulerBendPcell

lo = EulerBendPcell().Layout()
lo.visualize(annotate=True)
```



6.7 ArcPathPcell

```
class aluvia.all.ArcPathPcell(*args, **kwargs)
```

Arc Waveguide

Views

```
class Layout(*args, **kwargs)
```

Parameters

arc_angle: float

Angle of the arc in the interval, negative angle to go down and positive to go up [degree]

radius: float and number > 0

Radius of the waveguide [um]

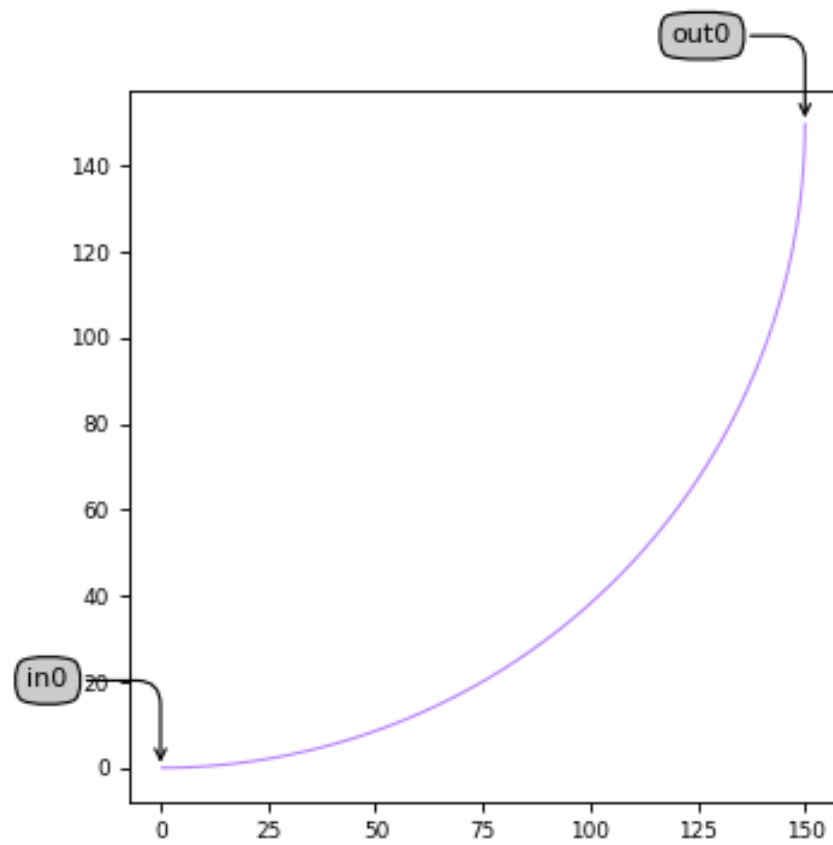
width: float and number > 0

Width of the arc path [um]

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(150.0, 150.0)	90.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import ArcPathPcell

lo = ArcPathPcell().Layout()
lo.visualize(annotate=True)
```



Spirals

<code>aluvia.all.SquareSpiralPcell</code>	Square Spiral.
<code>aluvia.all.SquareSpiralEulerPcell</code>	Square Spiral with Euler arcs.
<code>aluvia.all.SpiralPcell</code>	Circular Spiral.
<code>aluvia.all.SpiralEulerPcell</code>	Circular Spiral with Euler arcs.

6.8 SquareSpiralPcell

```
class aluvia.all.SquareSpiralPcell(*args, **kwargs)
```

Square Spiral.

Views

```
class Layout(*args, **kwargs)
```

Parameters

length: float and number > 0

Total length of the spiral

radius: float and number > 0

Radius of the arcs

spacing: float and number > 0

Spacing b/w the turns of the spiral

width: float and number > 0

Width of the spiral

6.9 SquareSpiralEulerPcell

```
class aluvia.all.SquareSpiralEulerPcell(*args, **kwargs)
```

Square Spiral with Euler arcs.

Views

```
class Layout(*args, **kwargs)
```

Parameters

length: float and number > 0

Total length of the spiral

p_factor: float and number > 0

Fraction of the bend having linearly increasing curvature

radius: float and number > 0

Radius of the arcs

spacing: float and number > 0

Spacing b/w the turns of the spiral

width: float and number > 0
Width of the spiral

6.10 SpiralPcell

class aluvia.all.SpiralPcell(*args, **kwargs)
Circular Spiral.

Views

class Layout(*args, **kwargs)

Parameters

inner_radius: float and number > 0
Inner Radius of the spiral

length: float and number > 0
Total length of the spiral

spacing: float and number > 0
Spacing b/w the turns of the spiral

width: float and number > 0
Width of the spiral

6.11 SpiralEulerPcell

class aluvia.all.SpiralEulerPcell(*args, **kwargs)
Circular Spiral with Euler arcs.

Views

class Layout(*args, **kwargs)

Parameters

inner_radius: float and number > 0
Inner Radius of the spiral

length: float and number > 0
Total length of the spiral

p_factor: float and number > 0
Fraction of the bend having linearly increasing curvature

spacing: float and number > 0
Spacing b/w the turns of the spiral

width: float and number > 0
Width of the spiral

Couplers

<code>aluvia.all.YBranchPcell</code>	YBranch splitter.
<code>aluvia.all.DirectionalcouplerPcell</code>	Directional Coupler.
<code>aluvia.all.RingRes_1x1Pcell</code>	1x1 Ring Resonator.
<code>aluvia.all.RingRes_2x2Pcell</code>	2x2 Ring Resonator.
<code>aluvia.all.MMI_1x2Pcell</code>	1x2 MMI.
<code>aluvia.all.MMI_2x2Pcell</code>	2x2 MMI.
<code>aluvia.all.MMI_nxmPcell</code>	NxM MMI.
<code>aluvia.all.LinearEdgeTaperPcell</code>	Linear tapered waveguide for edge coupling.

6.12 YBranchPcell

class `aluvia.all.YBranchPcell(*args, **kwargs)`

YBranch splitter.

Views

class `Layout(*args, **kwargs)`

Parameters

sbn_length: float and number > 0

Length of the S-bends [um]

sbn_separation: float and number > 0

Separation of the S-bends [um]

tap_length: float and number > 0

Length of the taper [um]

use_taper: (bool, bool_, bool or int)

Use Taper

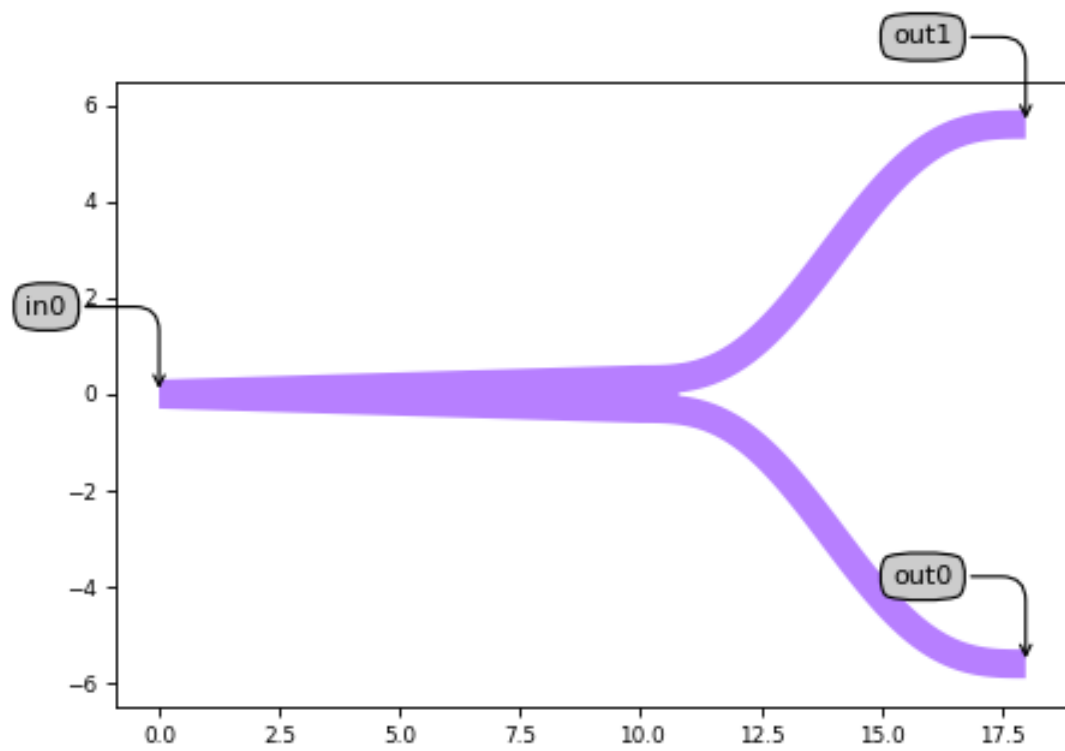
width: float and number > 0

Width of the waveguides [um]

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(18.0, -5.6)	0.0	WG_TMPL	0.0
out1	Optical	(18.0, 5.6)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import YBranchPcell

lo = YBranchPcell().Layout()
lo.visualize(annotate=True)
```



6.13 DirectionalCouplerPcell

```
class aluvia.all.DirectionalcouplerPcell(*args, **kwargs)
```

Directional Coupler.

Views

```
class Layout(*args, **kwargs)
```

Parameters

inp_length: float and number > 0

Length of the S-bends [um]

inp_separation: float and number > 0

Separation of the S-bends [um]

length: float and number > 0

Length of the Coupling section [um]

separation: float and number > 0

Gap of the Coupler [um]

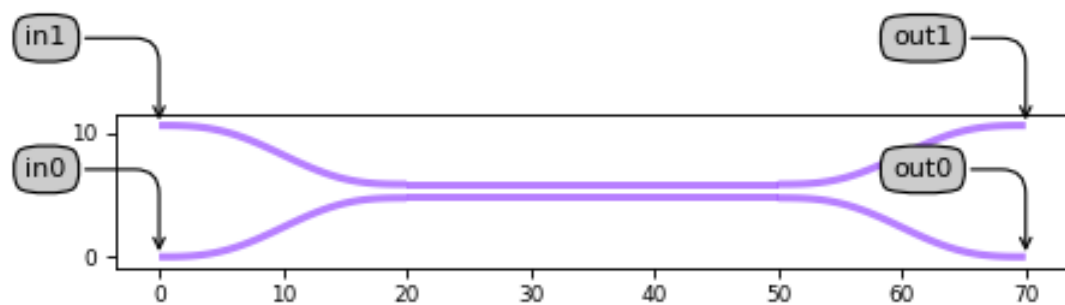
width: float and number > 0

Width of the waveguides

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
in1	Optical	(0.0, 10.6)	180.0	WG_TMPL	0.0
out1	Optical	(70.0, 10.6)	0.0	WG_TMPL	0.0
out0	Optical	(70.0, 0.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import DirectionalcouplerPcell

lo = DirectionalcouplerPcell().Layout()
lo.visualize(annotate=True)
```



6.14 RingRes_1x1Pcell

```
class aluvia.all.RingRes_1x1Pcell(*args, **kwargs)
```

1x1 Ring Resonator.

Views

```
class Layout(*args, **kwargs)
```

Parameters

bend_radius: float and number > 0

Bend radius of the ring

guide_width: float and number > 0

Width of the straight waveguide

path_width: float and number > 0

Width of the ring

separation: float and number > 0

Gap between straight and ring

straight_height: float and Real, number and number >= 0

Height of the straight in the ring

straight_length: float and Real, number and number >= 0

Length of the straight in the ring

total_length: float and number > 0

Total Path Length of the ring

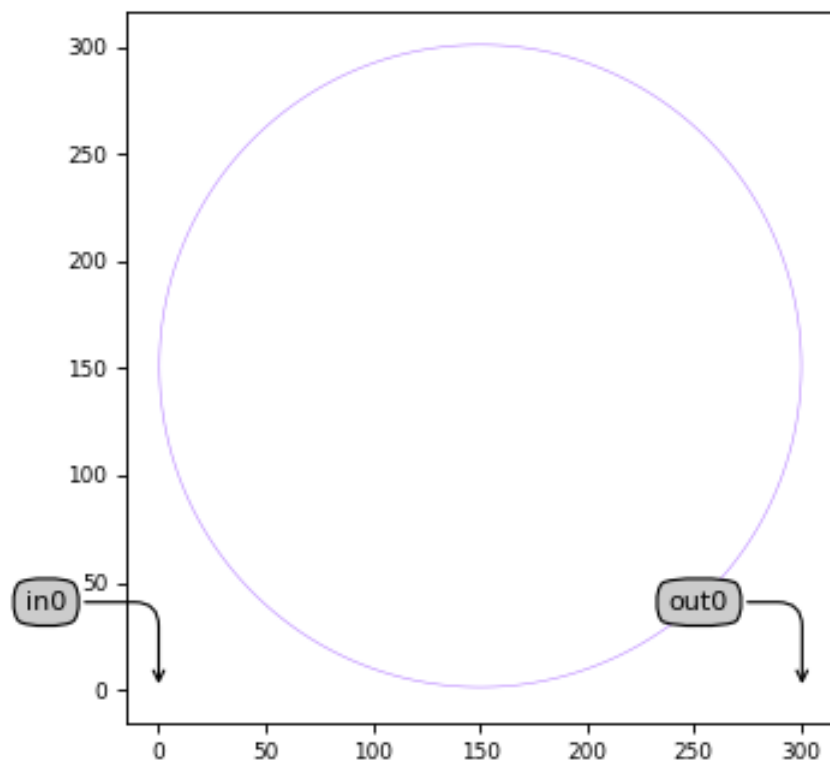
use_total_length: (bool, bool_, bool or int)

Use total_length parameter?

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(300.6, 0.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import RingRes_1x1Pcell

lo = RingRes_1x1Pcell().Layout()
lo.visualize(annotate=True)
```



6.15 RingRes_2x2Pcell

```
class aluvia.all.RingRes_2x2Pcell(*args, **kwargs)
```

2x2 Ring Resonator.

Views

```
class Layout(*args, **kwargs)
```

Parameters

bend_radius: float and number > 0

Bend radius of the ring

guide_width: float and number > 0

Width of the straight waveguide

path_width: float and number > 0

Width of the ring

separation: float and number > 0

Gap between straight and ring

straight_height: float and Real, number and number >= 0

Height of the straight in the ring

straight_length: float and Real, number and number >= 0

Length of the straight in the ring

total_length: float and number > 0

Total Path Length of the ring

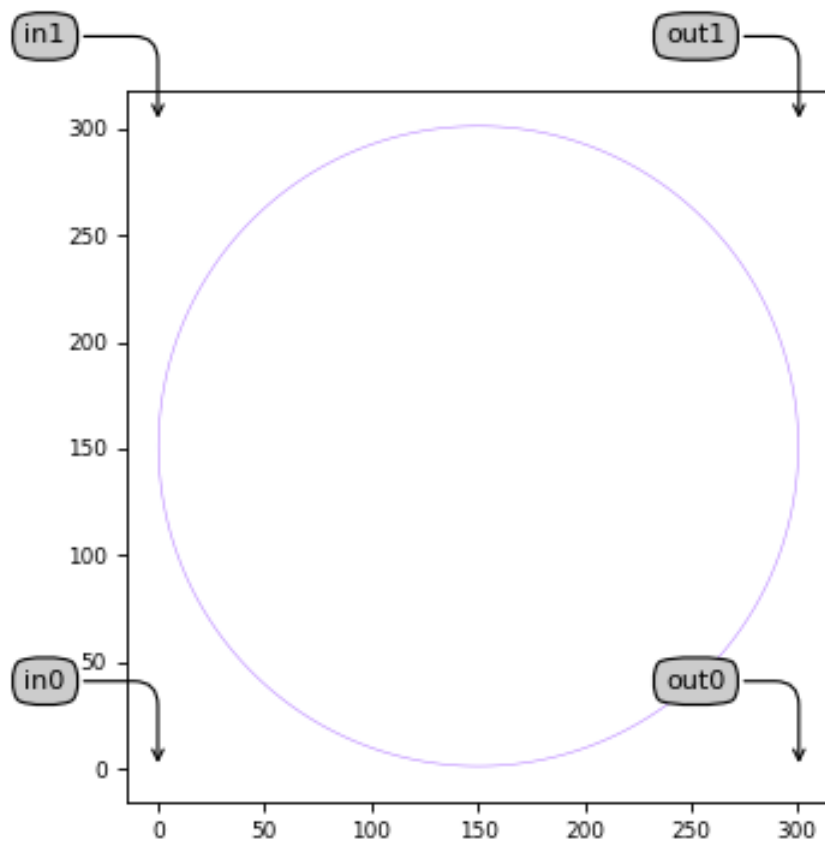
use_total_length: (bool, bool_, bool or int)

Use total_length parameter?

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(300.6, 0.0)	0.0	WG_TMPL	0.0
in1	Optical	(0.0, 302.2)	180.0	WG_TMPL	0.0
out1	Optical	(300.6, 302.2)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import RingRes_2x2Pcell
```

```
lo = RingRes_2x2Pcell().Layout()
lo.visualize(annotate=True)
```



6.16 MMI_1x2Pcell

```
class aluvia.all.MMI_1x2Pcell(*args, **kwargs)
    1x2 MMI.
```

Views

```
class Layout(*args, **kwargs)
```

Parameters

inp_separation: float and number > 0
Separation b/w output ports

inp_width: float and number > 0
Input width of tapers

mmi_length: float and number > 0
Length of MMI

mmi_separation: float and number > 0
Separation b/w output tapers

mmi_width: float and number > 0
Width of MMI

sbn_length: float and number > 0
Length of sbends at the output

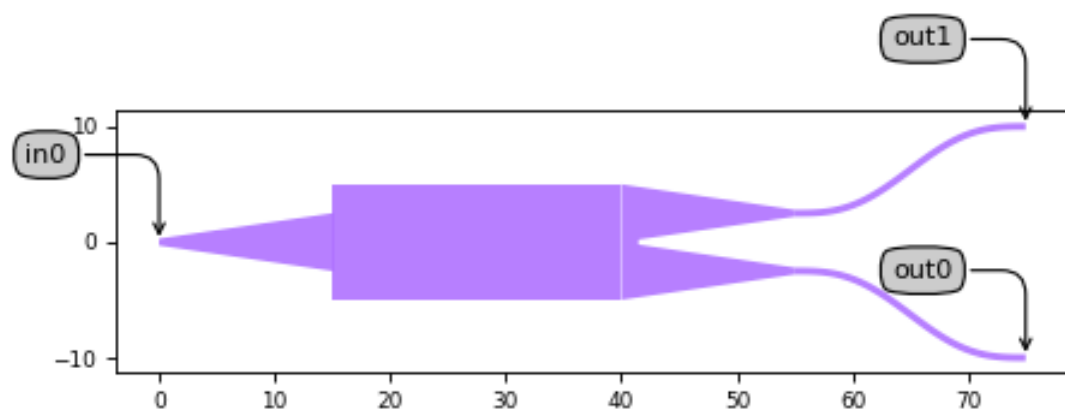
tap_length: float and number > 0
Length of tapers

tap_width_out: float and number > 0
Output width of tapers

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out0	Optical	(75.0, -10.0)	0.0	WG_TMPL	0.0
out1	Optical	(75.0, 10.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import MMI_1x2Pcell

lo = MMI_1x2Pcell().Layout()
lo.visualize(annotate=True)
```



6.17 MMI_2x2Pcell

```
class aluvia.all.MMI_2x2Pcell(*args, **kwargs)
    2x2 MMI.
```

Views

```
class Layout(*args, **kwargs)
```

Parameters

inp_separation: float and number > 0
Separation b/w ports

inp_width: float and number > 0
Input width of tapers

mmi_length: float and number > 0
Length of MMI

mmi_separation: float and number > 0
Separation b/w output tapers

mmi_width: float and number > 0
Width of MMI

sbn_length: float and number > 0
Length of sbends at the output

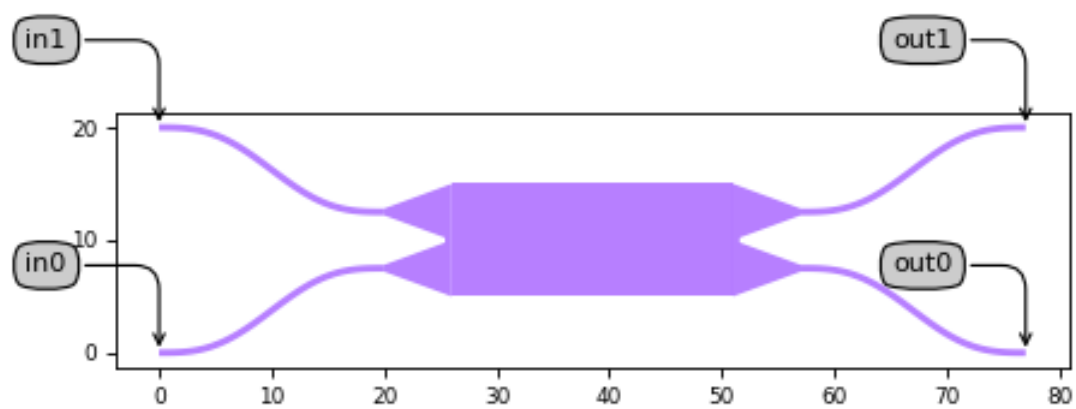
tap_length: float and number > 0
Length of tapers

tap_width_out: float and number > 0
Output width of tapers

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
in1	Optical	(0.0, 20.0)	180.0	WG_TMPL	0.0
out0	Optical	(77.0, 0)	0.0	WG_TMPL	0.0
out1	Optical	(77.0, 20.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import MMI_2x2Pcell

lo = MMI_2x2Pcell().Layout()
lo.visualize(annotate=True)
```



6.18 MMI_nxmPcell

```
class aluvia.all.MMI_nxmPcell(*args, **kwargs)
    NxM MMI.
```

Views

```
class Layout(*args, **kwargs)
```

Parameters

inp_width: float and number > 0
Input waveguide width of tapers

mmi_length: float and number > 0
Length of the MMI section.

mmi_width: float and number > 0
Width of the MMI section.

n_inputs: int and number > 0
The number of inputs for the MMI

n_outputs: int and number > 0
The number of outputs for the MMI

tap_length: float and number > 0
Length of tapers

tap_width_out: float and number > 0
Output width of tapers

waveguide_offset_input: float and Real, number and number >= 0
Offset input with respect to center.

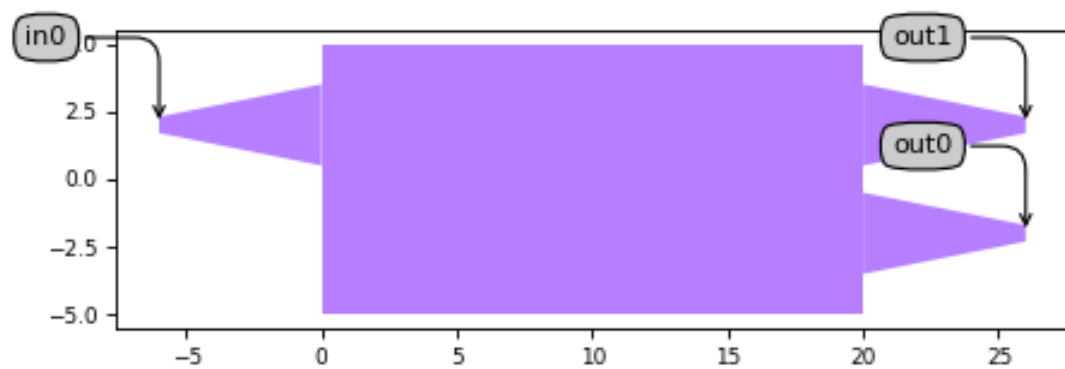
waveguide_spacing_input: float and Real, number and number >= 0
Spacing between the input waveguides.

waveguide_spacing_output: float and number > 0
Spacing between the output waveguides.

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(-6.0, 2.0)	180.0	WG_TMPL	0.0
out0	Optical	(26.0, -2.0)	0.0	WG_TMPL	0.0
out1	Optical	(26.0, 2.0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import MMI_nxmPcell

lo = MMI_nxmPcell().Layout()
lo.visualize(annotate=True)
```



6.19 LinearEdgeTaperPCell

```
class aluvia.all.LinearEdgeTaperPCell(*args, **kwargs)
```

Linear tapered waveguide for edge coupling.

Views

```
class Layout(*args, **kwargs)
```

Parameters

square_width: float and number > 0

Width of the square box holding edge of taper section.

taper_edge_width: float and number > 0

Edge taper width.

taper_end_width: float and number > 0

Edge end width.

taper_length: float and number > 0

Taper length.

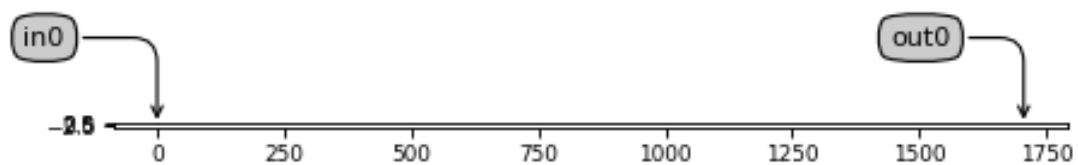
taper_length_fixed_width: float and number > 0

Length of fixed taper edge width.

Name	Type	Position	Angle	Waveguide Template	Inclination
in0	Optical	(0, 0)	0.0		0.0
out0	Optical	(1705.0, 0)	0.0	WG_TMPL	0.0

```
from aluvia import technology
from aluvia.all import LinearEdgeTaperPCell

lo = LinearEdgeTaperPCell().Layout()
lo.visualize(annotate=True)
```



Die Templates

<code>aluvia.all.Template5000x5000</code>	5000x5000 design template.
<code>aluvia.all.Template10000x10000</code>	10000x10000 design template.

6.20 Template5000x5000

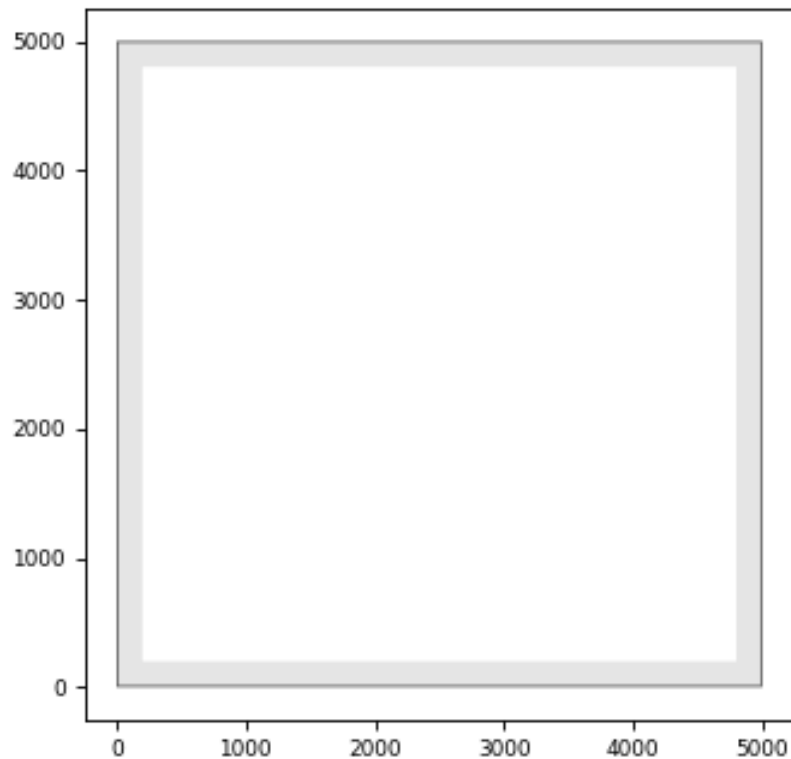
```
class aluvia.all.Template5000x5000(*args, **kwargs)
    5000x5000 design template.
```

Views

```
class Layout(*args, **kwargs)
```

```
from aluvia import technology
from aluvia.all import Template5000x5000

lo = Template5000x5000().Layout()
lo.visualize(annotate=True)
```



6.21 Template10000x10000

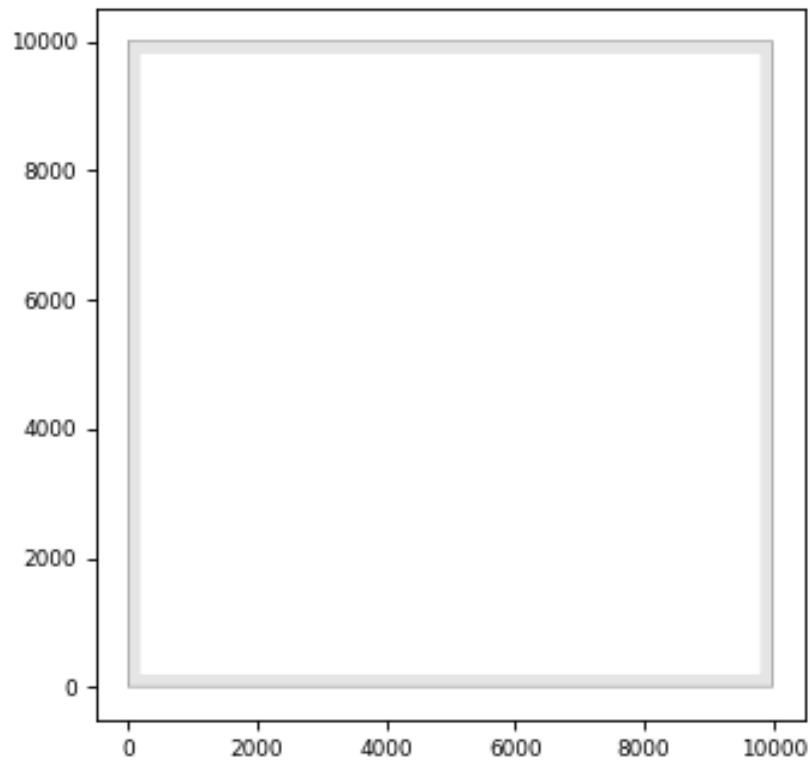
```
class aluvia.all.Template10000x10000(*args, **kwargs)
    10000x10000 design template.
```

Views

```
class Layout(*args, **kwargs)
```

```
from aluvia import technology
from aluvia.all import Template10000x10000

lo = Template10000x10000().Layout()
lo.visualize(annotate=True)
```



Metal

<code>aluvia.all.HeaterWaveguide</code>	Heater waveguide that uses the HeaterTemplate as trace_template and adds electrical ports at the start and at the end.
<code>aluvia.all.MetalWire</code>	Metal wire.
<code>aluvia.all.DCTaper</code>	Metal Taper with 45 degree slope.
<code>aluvia.all.DCBend</code>	A metal bent section that follows the 45 degree Manhattan routing rule.
<code>aluvia.all.DCPad</code>	A metal pad that can be interconnected to the metal lines and contacted through a probe or through wire-bonding.
<code>aluvia.all.DCPadArray</code>	Array of DCPad.

6.22 HeaterWaveguide

class `aluvia.all.HeaterWaveguide(*args, **kwargs)`

Heater waveguide that uses the HeaterTemplate as trace_template and adds electrical ports at the start and at the end.

Parameters

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters
The unique name of the pcell

Views

class `Layout(*args, **kwargs)`

Parameters

core_width: float and Real, number and number ≥ 0
Width of the waveguide [um]

ground_wire_width: float and Real, number and number ≥ 0
Width of the Metal path [um]

heater_offset: float
Offset of the heater line relative to the waveguide centerline [um]

heater_width: float and number > 0
Width of the Heater path [um]

metal_width: float and Real, number and number ≥ 0
Width of the Metal path [um]

wire_width: float and Real, number and number ≥ 0
Width of the Metal path [um]

shape: Shape
Shape from which the Trace is calculated

bend_radius: float and number > 0
Bend radius for the auto-generated bends.

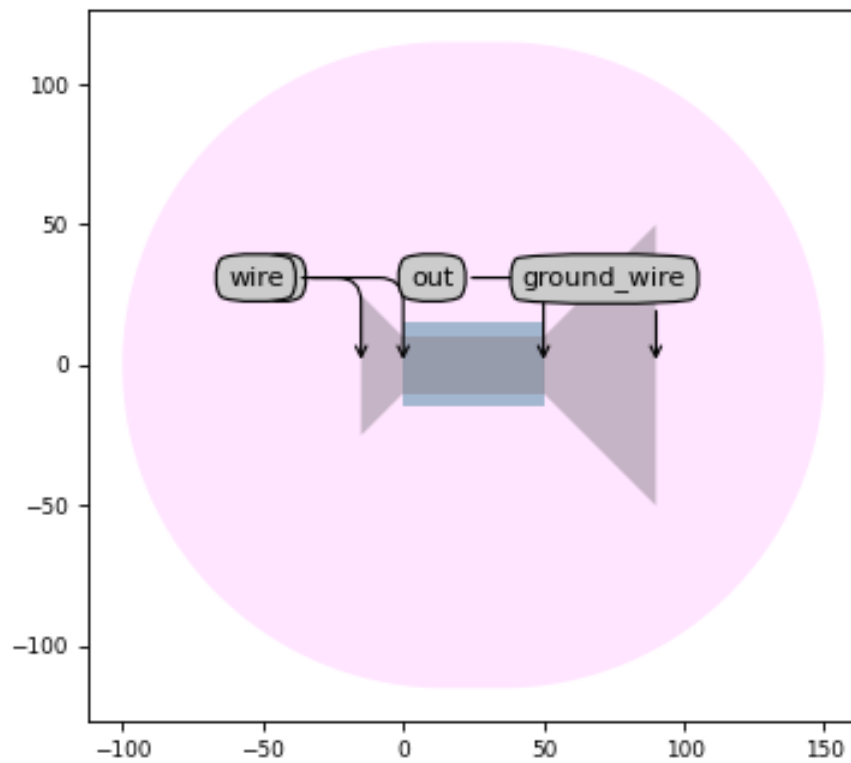
Name	Type	Position	Angle	Waveguide Template	Inclination
in	Optical	(0.0, 0.0)	180.0	WG_TMPL	0.0
out	Optical	(50.0, 0.0)	0.0	WG_TMPL	0.0
wire	Electrical	(-15.0, 0.0)	180.0	MetalWireTemplate	0.0
ground_wire	Electrical	(90.0, 0.0)	0.0	MetalWireTemplate	0.0

```

from aluvia import technology
from aluvia.all import HeaterWaveguide

lo = HeaterWaveguide().Layout()
lo.visualize(annotate=True)

```



6.23 MetalWire

```
class aluvia.all.MetalWire(*args, **kwargs)
```

Metal wire.

Parameters

trace_template: PCell and ElectricalWireTemplate

external_port_names: str

Dictionary for remapping of the port names of the contents to the external ports

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters

The unique name of the pcell

Other Parameters

contents: PCell and _Trace, locked

Views

```
class Layout(*args, **kwargs)
```

Parameters

core_width: float and number > 0

Width of the metal [um]

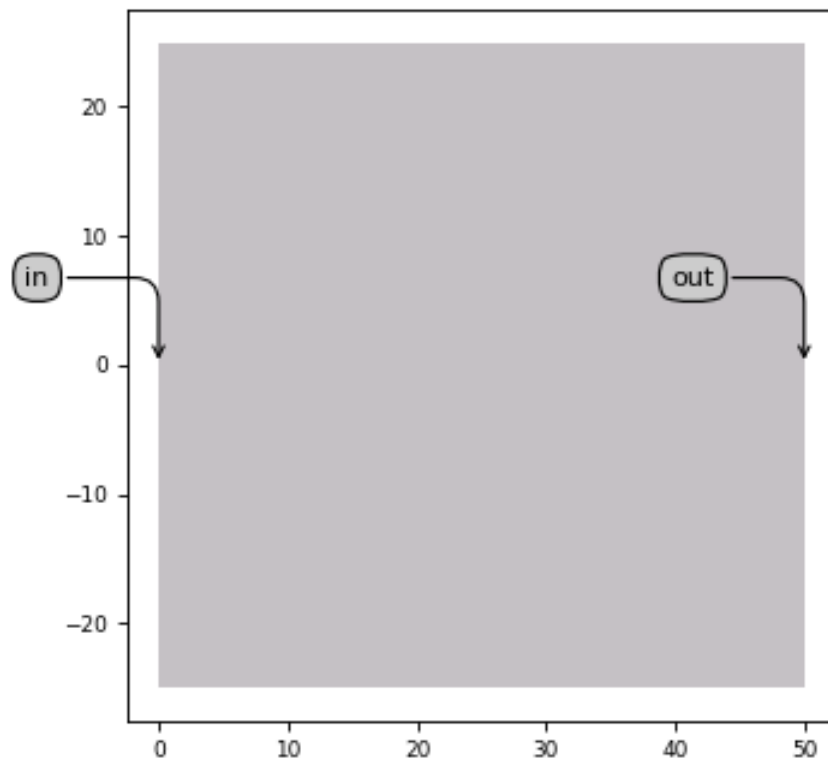
shape: Shape

Shape from which the Trace is calculated

Name	Type	Position	Angle	Waveguide Template	Inclination
in	Electrical	(0.0, 0.0)	180.0	MetalWireTemplate	0.0
out	Electrical	(50.0, -0.0)	0.0	MetalWireTemplate	0.0

```
from aluvia import technology
from aluvia.all import MetalWire

lo = MetalWire().Layout()
lo.visualize(annotate=True)
```



6.24 DCTaper

class aluvia.all.DCTaper(*args, **kwargs)

Metal Taper with 45 degree slope.

Parameters

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters

The unique name of the pcell

Views

class Layout(*args, **kwargs)

Parameters

in_width: float and number > 0

Width of the metal at the input[um]

length: float and Real, number and number >= 0

Length of the taper based on in_width and out_width for a 45 degree angle by default

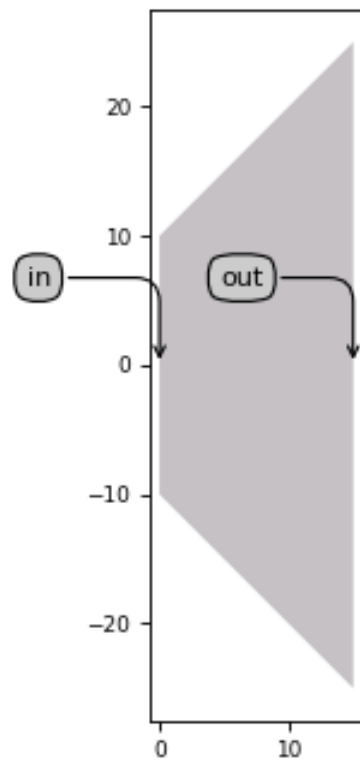
out_width: float and number > 0

Width of the metal at the output[um]

Name	Type	Position	Angle	Waveguide Template	Inclination
in	Electrical	(0.0, 0.0)	180.0	MetalWireTemplate	0.0
out	Electrical	(15.0, 0.0)	0.0	MetalWireTemplate	0.0

```
from aluvia import technology
from aluvia.all import DCTaper
```

```
lo = DCTaper().Layout()
lo.visualize(annotate=True)
```



6.25 DCBend

class aluvia.all.DCBend(*args, **kwargs)

A metal bent section that follows the 45 degree Manhattan routing rule. The radius can be defined and also the orientation (clockwise, counterclockwise).

Parameters

trace_template: PCell and ElectricalWireTemplate

external_port_names: str

Dictionary for remapping of the port names of the contents to the external ports

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters

The unique name of the pcell

Other Parameters

contents: PCell and _Trace, locked

Views

class Layout(*args, **kwargs)

Parameters

bend_radius: float and number > 0

Radius of the bend[um]

orientation: str and String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters and List with value restriction, allowed values: ['clockwise', 'counterclockwise']

Orientation of the bend clockwise/counterclockwise.Default is clockwise

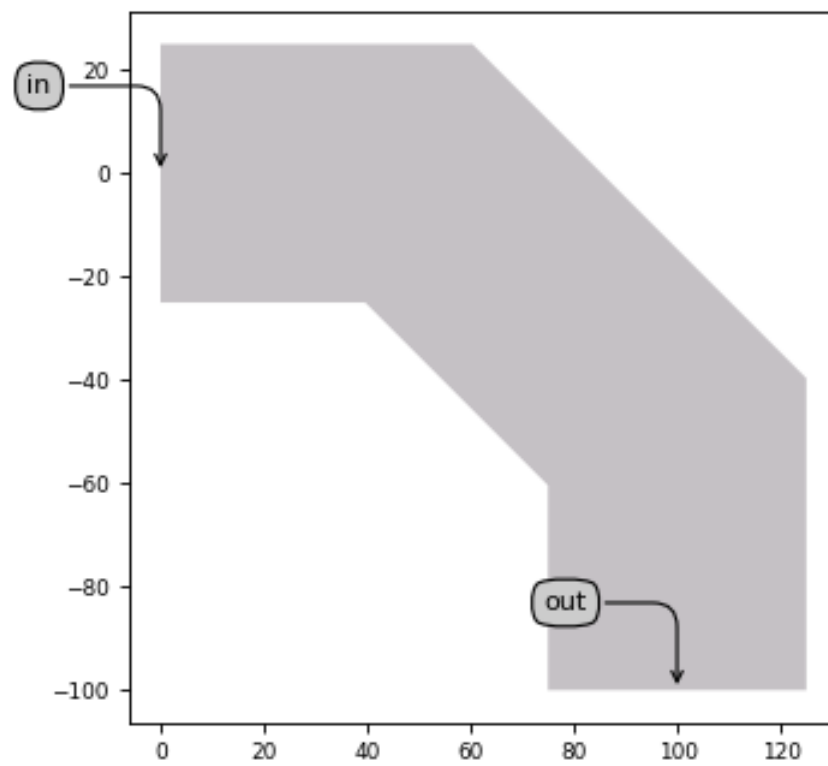
core_width: float and number > 0

Width of the metal [um]

Name	Type	Position	Angle	Waveguide Template	Inclination
in	Electrical	(0.0, 0.0)	180.0	MetalWireTemplate	0.0
out	Electrical	(100.0, 100.0)	- -90.0	MetalWireTemplate	0.0

```
from aluvia import technology
from aluvia.all import DCBend

lo = DCBend().Layout()
lo.visualize(annotate=True)
```



6.26 DCPad

class aluvia.all.DCPad(*args, **kwargs)

A metal pad that can be interconnected to the metal lines and contacted through a probe or through wire-bonding.

Parameters

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters

The unique name of the pcell

Views

class Layout(*args, **kwargs)

Parameters

length: float and number > 0

Pad Length [um]

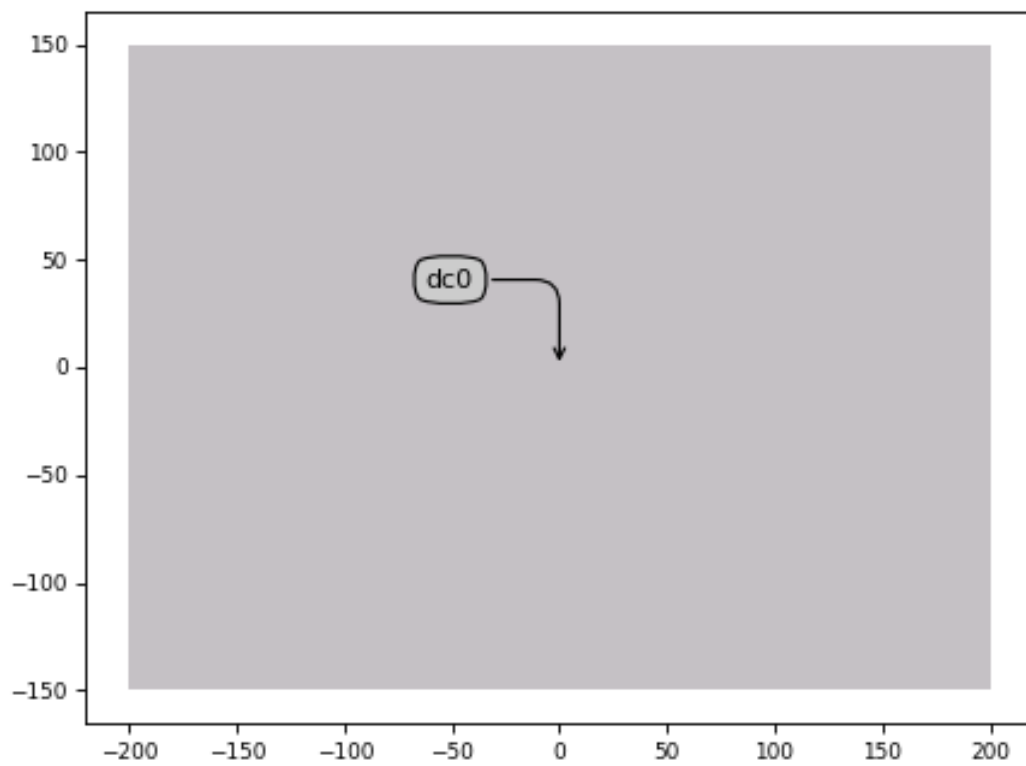
width: float and number > 0

Pad Width [um]

Name	Type	Position	Angle	Waveguide Template	Inclination
dc0	Electrical	(0.0, 0.0)	0.0	MetalWireTemplate	0.0

```
from aluvia import technology
from aluvia.all import DCPad

lo = DCPad().Layout()
lo.visualize(annotate=True)
```



6.27 DCPadArray

```
class aluvia.all.DCPadArray(*args, **kwargs)
```

Array of DCPad.

Parameters

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters
The unique name of the pcell

Views

```
class Layout(*args, **kwargs)
```

Parameters

n_o_pads: tuple2 and Tuple of <class 'int'> and length == 2
number of DCPads in x and y

pad_length: float and number > 0
Pad Length [um]

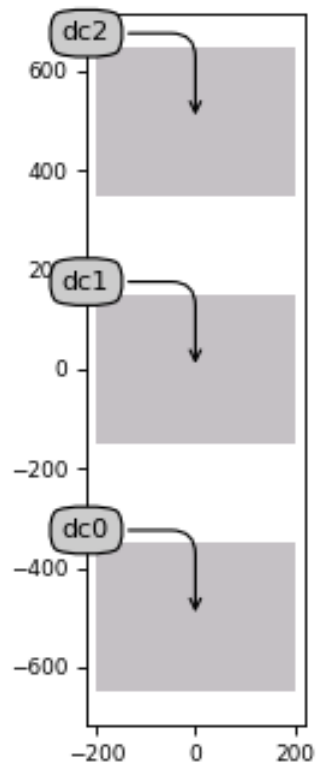
pad_width: float and number > 0
Pad Width [um]

pitch: float and number > 0
Pitch of the DCPads [um]

Name	Type	Position	Angle	Waveguide Template	Inclination
dc0	Electrical	(0.0, -500.0)	0.0	MetalWireTemplate	0.0
dc1	Electrical	(0.0, 0.0)	0.0	MetalWireTemplate	0.0
dc2	Electrical	(0.0, 500.0)	0.0	MetalWireTemplate	0.0

```
from aluvia import technology
from aluvia.all import DCPadArray
```

```
lo = DCPadArray().Layout()
lo.visualize(annotate=True)
```



Trace Templates

<code>aluvia.all.WG_TMPL</code>	Waveguide Template with Al2O3 Core.
<code>aluvia.all.HeaterTemplate</code>	Heater waveguide template which consists of a waveguide template with the heater layer on top.
<code>aluvia.all.MetalWireTemplate</code>	Metal wire template.

6.28 WG_TMPL

class `aluvia.all.WG_TMPL(*args, **kwargs)`

Waveguide Template with Al2O3 Core.

Parameters

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters
The unique name of the pcell

Views

class `Layout(*args, **kwargs)`

Parameters

core_width: float and number > 0
width of the waveguide core

6.29 HeaterTemplate

class `aluvia.all.HeaterTemplate(*args, **kwargs)`

Heater waveguide template which consists of a waveguide template with the heater layer on top. To be used with HeaterWaveguide in order to have ElectricalPorts at the start and at the end.

Parameters

name: String that contains only ISO/IEC 8859-1 (extended ASCII py3) or pure ASCII (py2) characters
The unique name of the pcell

Views

class `Layout(*args, **kwargs)`

Parameters

core_width: float and Real, number and number >= 0
Width of the waveguide [um]

ground_wire_width: float and Real, number and number >= 0
Width of the Metal path [um]

heater_offset: float
Offset of the Heater line relative to the waveguide centerline [um]

heater_width: float and number > 0

Width of the Heater path [um]

metal_width: float and Real, number and number >= 0

Width of the Metal path [um]

wire_width: float and Real, number and number >= 0

Width of the Metal path [um]

6.30 MetalWireTemplate

```
class aluvia.all.MetalWireTemplate(*args, **kwargs)
```

Metal wire template.

Views

```
class Layout(*args, **kwargs)
```

Parameters

core_width: float and number > 0

Width of the metal [um]

Index

A

ArcPathPcell (*class in aluvia.all*), 19

D

DCBend (*class in aluvia.all*), 47

DCPad (*class in aluvia.all*), 49

DCPadArray (*class in aluvia.all*), 51

DCTaper (*class in aluvia.all*), 45

DirectionalCouplerPcell (*class in aluvia.all*), 25

E

EulerBendPcell (*class in aluvia.all*), 17

H

HeaterTemplate (*class in aluvia.all*), 53

HeaterWaveguide (*class in aluvia.all*), 41

L

LinearEdgeTaperPCell (*class in aluvia.all*), 37

M

MetalWire (*class in aluvia.all*), 43

MetalWireTemplate (*class in aluvia.all*), 54

MMI_1x2Pcell (*class in aluvia.all*), 31

MMI_2x2Pcell (*class in aluvia.all*), 33

MMI_nxmPcell (*class in aluvia.all*), 35

R

RingRes_1x1Pcell (*class in aluvia.all*), 27

RingRes_2x2Pcell (*class in aluvia.all*), 29

S

SBend (*class in aluvia.all*), 15

SpiralEulerPcell (*class in aluvia.all*), 22

SpiralPcell (*class in aluvia.all*), 22

SquareSpiralEulerPcell (*class in aluvia.all*), 21

SquareSpiralPcell (*class in aluvia.all*), 21

StripWaveguide (*class in aluvia.all*), 7

StripWaveguide_369 (*class in aluvia.all*), 9

StripWaveguide_405 (*class in aluvia.all*), 11

T

TaperPCell (*class in aluvia.all*), 13

Template10000x10000 (*class in aluvia.all*), 40

Template5000x5000 (*class in aluvia.all*), 39

W

WG_TMPL (*class in aluvia.all*), 53

Y

YBranchPcell (*class in aluvia.all*), 23