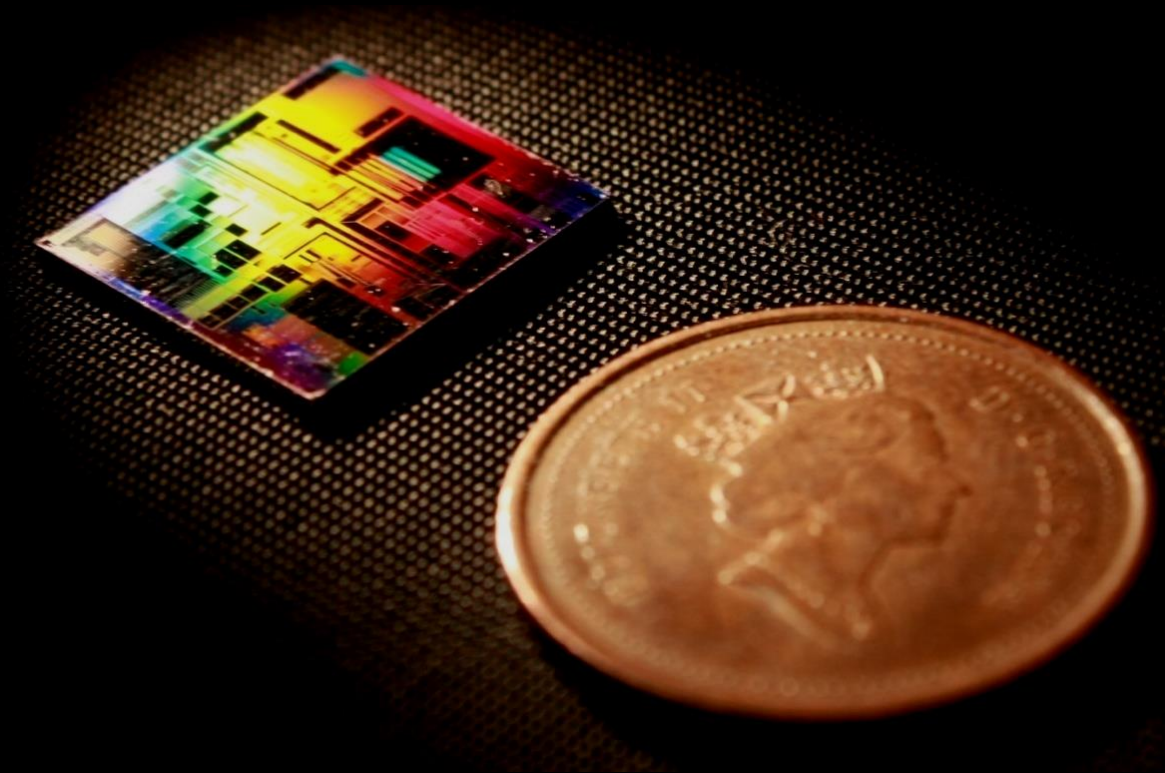


Klayout: Parameterized Cells Scripting

2019 SiEPIC Passive Silicon Photonics Workshop, Vancouver, Canada



Mustafa Hammood,
The University of British Columbia, Vancouver, Canada



a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

MiNa Microsystems and
Nanotechnology Group
Photonics Research Group



Electrical and
Computer
Engineering

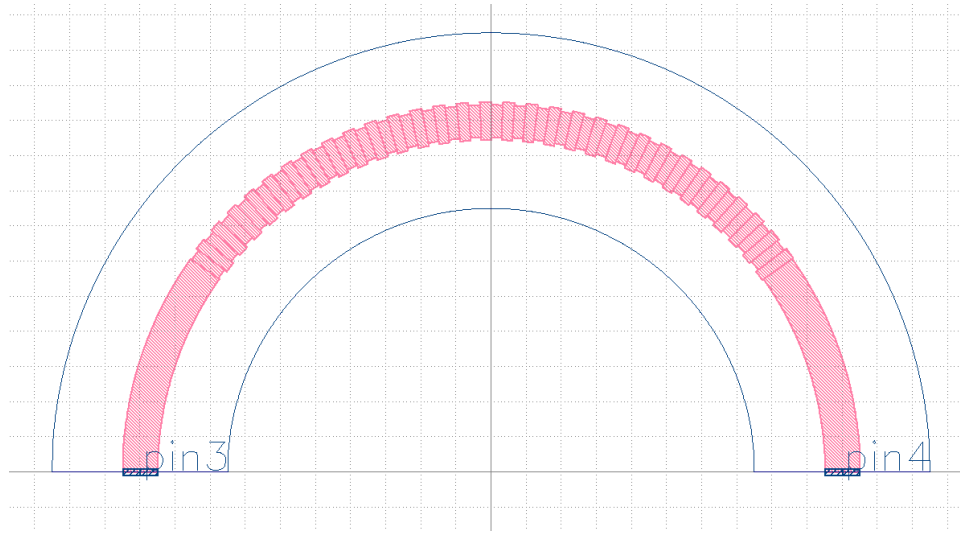
Outline

- What is a parameterized cell?
- How to setup and start creating scripts
- Define components library
- Make your first parameterized cell



Pcell?

- A method to generate the layout of a parameterized device, contains:
 - Physical device parameters
 - Ports definitions
 - Device region definition
 - Compact models (?)



Instance Properties

Cell hi?

Cell Library

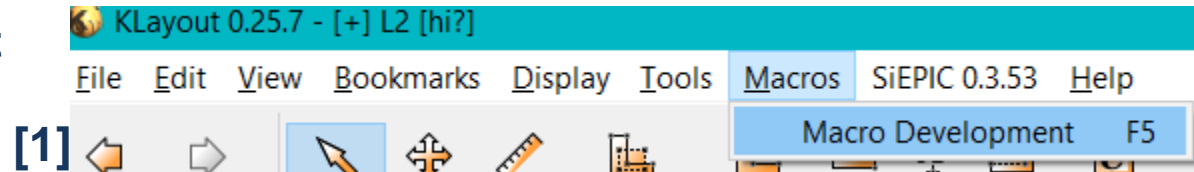
Geometry

PCell parameters

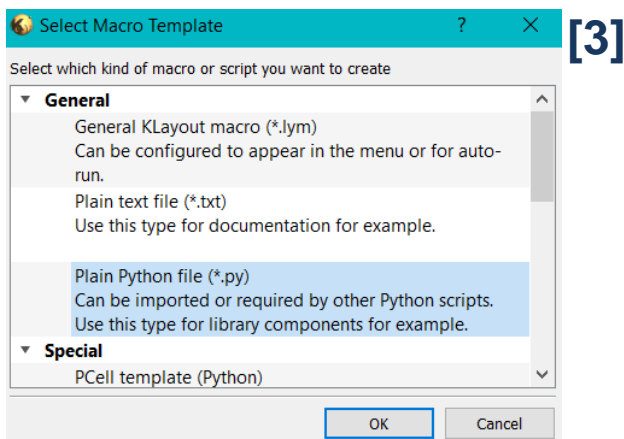
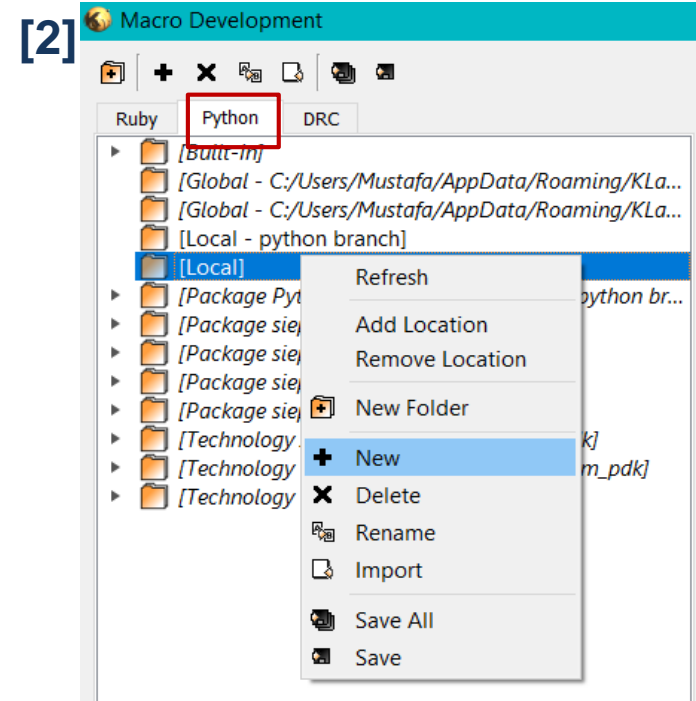
Si Layer	Si - 1/0
Si Gratings Layer	Si - 1/0
Radius (um)	5
Width (um)	0.5
Gratings Period (nm)	318
Corrugation Width (um)	0.04
N (number of corrugations)	30
Bus-to-straight bend radius	5
PinRec Layer	PinRec - 1/10
DevRec Layer	DevRec - 68/0

Setup

- Open Macro Development



- Create a new Python script in your local directory
- Copy content of my_first_script.py



Define components library

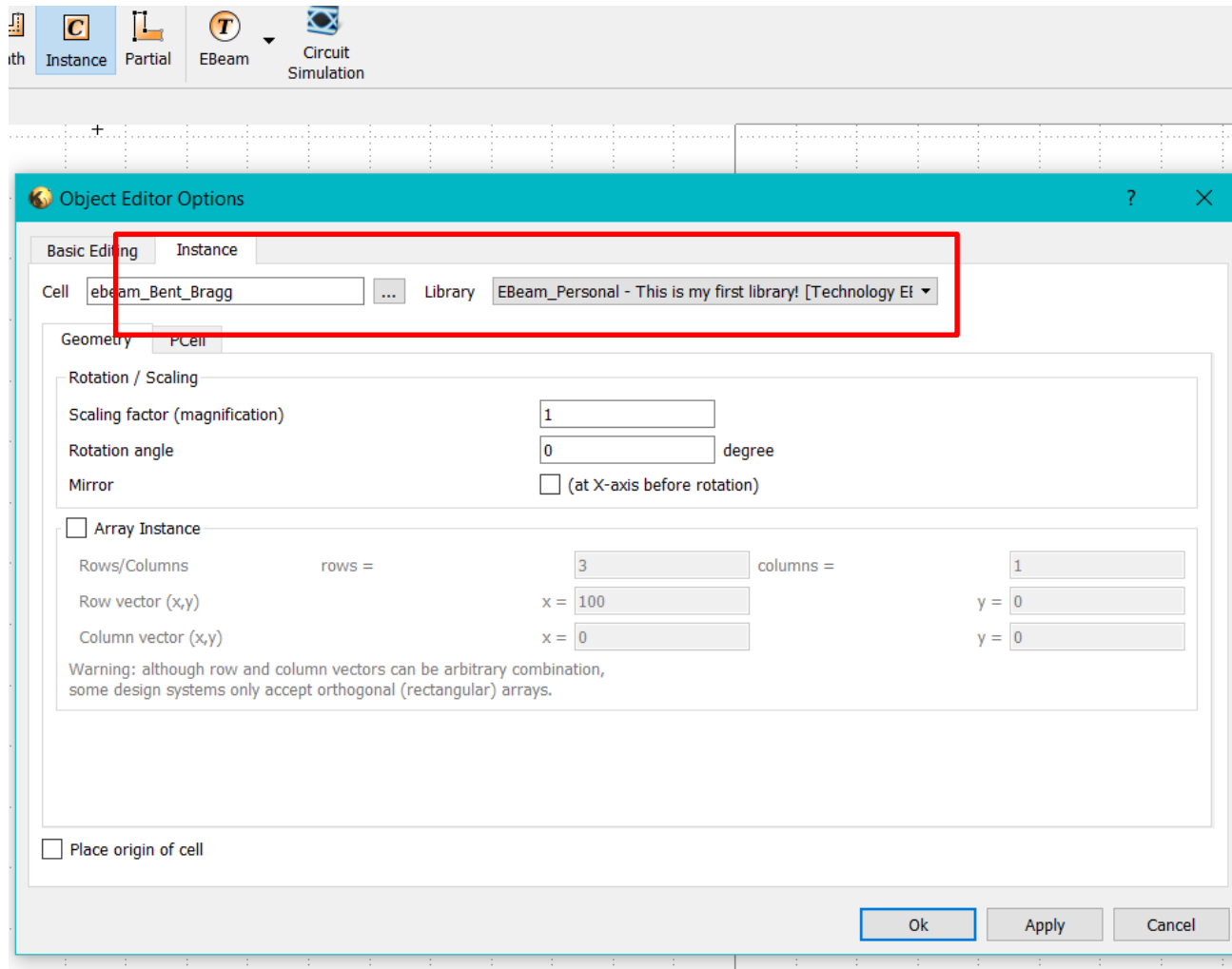
- Call it SiEPIC_Demo
- Built on a technology
 - 'EBeam'
- Register all PCells in the script
 - 'Ebeam_Bent_Bragg'

```

157 class SiEPIC_Demo(Library):
158     """
159     The library where we will put the PCells and GDS into
160     """
161
162     def __init__(self):
163
164         tech_name = 'EBeam_Personal'
165         library = tech_name
166
167         print("Initializing '%s' Library." % library)
168
169         # Set the description
170         # windows only allows for a fixed width, short description
171         self.description = ""
172         # OSX does a resizing:
173         self.description = "This is my first library!"
174
175         # Create the PCell declarations
176         self.layout().register_pcell("ebeam_Bent_Bragg", ebeam_Bent_Bragg())
177
178         # Register the library with the technology name
179         # If a library with that name already existed, it will be replaced then.
180         self.register(library)
181
182         self.technology='EBeam'
183
184         # Instantiate and register the library
185         SiEPIC_Demo()

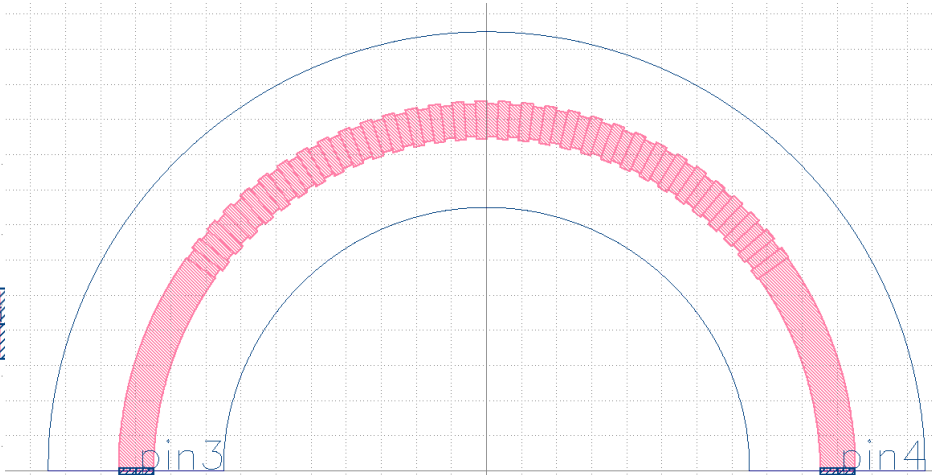
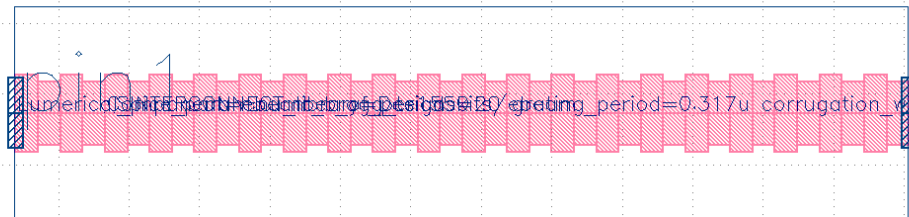
```

Define components library



Pcell: Objective

- We want to turn Bragg gratings from straight to bent over an Arc
- Parameters:
 - Radius
 - Waveguide width
 - Bragg period
 - Bragg corrugation width
 - Number of Bragg periods
 - Layers



Pcell: Format

- `__init__`: initialize technology and GUI Pcell parameters
- `Display_text_impl`: Description of the Pcell, appears when selected
- `produce`: Actual implementation of the Pcell (polygon drawing definition) and ports definition, and device recognition

```

16 class ebeam_Bent_Bragg(PCellDeclarationHelper):
17
18     def __init__(self):
19         # initialize the super class and technology
20         super(ebeam_Bent_Bragg, self).__init__()
21         TECHNOLOGY = get_technology_by_name('EBeam')
22
23         # declare the PCell parameters
24         self.param("silayer", self.TypeLayer, "Si Layer", default = TECHNOLOGY["Waveguide"])
25         self.param("width", self.TypeDouble, "Width (um)", default = 0.5)
26
27     def display_text_impl(self):
28         # Provide a descriptive text for the cell
29         return "Description"
30
31     def produce(self, layout, layers, parameters, cell):
32
33         # Draw the PCell polygons
34
35         # Create the device pins, as short paths:
36
37         # Create the device recognition layer -- make it 1 * wg_width away from the waveguides.
38

```


Pcell: Objective

- We want to turn Bragg gratings from straight to bent over an Arc
- Parameters:
 - Radius
 - Waveguide width
 - Bragg period
 - Bragg corrugation width
 - Number of Bragg periods
 - Layers

Pcell: Script Parameters GUI and tech

- Method: `__init__`

```
def __init__(self):
```

```
    # Important: initialize the super class
```

```
    super(ebeam_Bent_Bragg, self).__init__()
```

```
    TECHNOLOGY = get_technology_by_name('EBeam')
```

```
    # declare the parameters
```

```
    self.param("silayer", self.TypeLayer, "Si Layer", default = TECHNOLOGY['Waveguide'])
```

```
    self.param("silayer_gratings", self.TypeLayer, "Si Gratings Layer", default = TECHNOLOGY['31_Si_p6nm'])
```

```
    self.param("radius", self.TypeDouble, "Radius (um)", default = 25)
```

```
    self.param("width", self.TypeDouble, "Width (um)", default = 0.5)
```

```
    self.param("period", self.TypeDouble, "Gratings Period (nm)", default = 318)
```

```
    self.param("deltaW", self.TypeDouble, "Corrugation Width (um)", default = 0.04)
```

```
    self.param("gamma", self.TypeDouble, "N (number of corrugations)", default = 135)
```

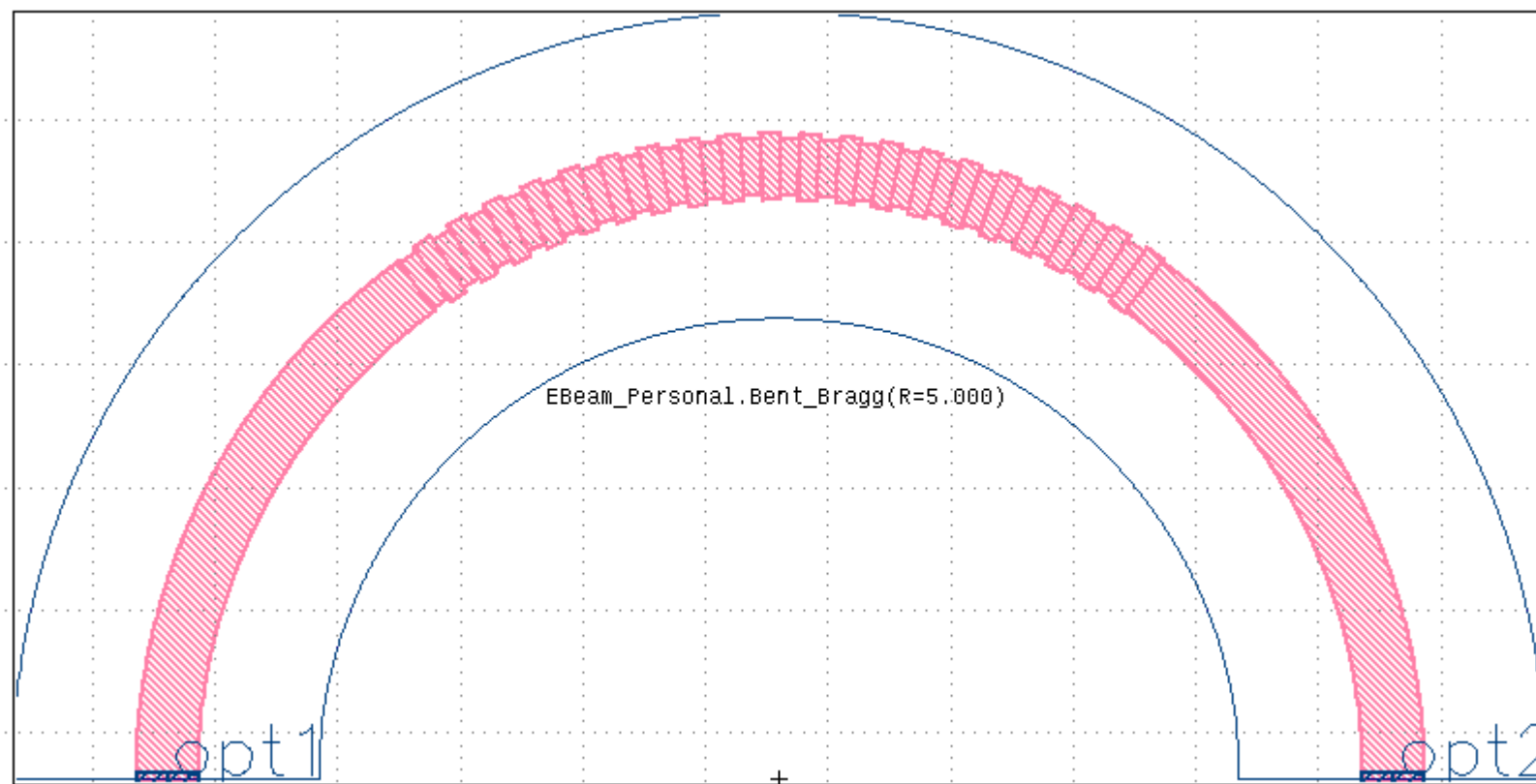
```
    self.param("pinrec", self.TypeLayer, "PinRec Layer", default = TECHNOLOGY['PinRec'])
```

```
    self.param("devrec", self.TypeLayer, "DevRec Layer", default = TECHNOLOGY['DevRec'])
```

Pcell: Description

- Method: `display_text_impl`

```
def display_text_impl(self):
    # Provide a descriptive text for the cell
    return "Bent_Bragg(R=" + ('%.3f' % self.radius) + ")"
```



Pcell: Produce: fetch layout parameters

- Method: Produce

```
def produce(self, layout, layers, parameters, cell):

    #coerce parameters (make consistent)

    self._layers = layers
    self.cell = cell
    self._param_values = parameters
    self.layout = layout

    # cell: layout cell to place the layout
    # LayerSiN: which layer to use
    # r: radius
    # w: waveguide width
    # length units in dbu

    from math import pi, cos, sin
    from SiEPIC.utils import arc_wg, arc_wg_xy
    from SiEPIC._globals import PIN_LENGTH

    # fetch the parameters
    dbu = self.layout.dbu
    ly = self.layout

    LayerSi = self.silayer
    LayerSiN_gratings = self.silayer_gratings_layer
    LayerSiN = self.silayer_layer
    LayerPinRecN = ly.layer(self.pinrec)
    LayerDevRecN = ly.layer(self.devrec)

    from SiEPIC.extend import to_itype
    w = to_itype(self.width,dbu)
    r = to_itype(self.radius,dbu)
    period = self.period
    deltaW = to_itype(self.deltaW,dbu)
    N = int(self.gamma)
```

Pcell: Produce: Draw layout!

- Method: Produce

Center of everything

$x = 0$

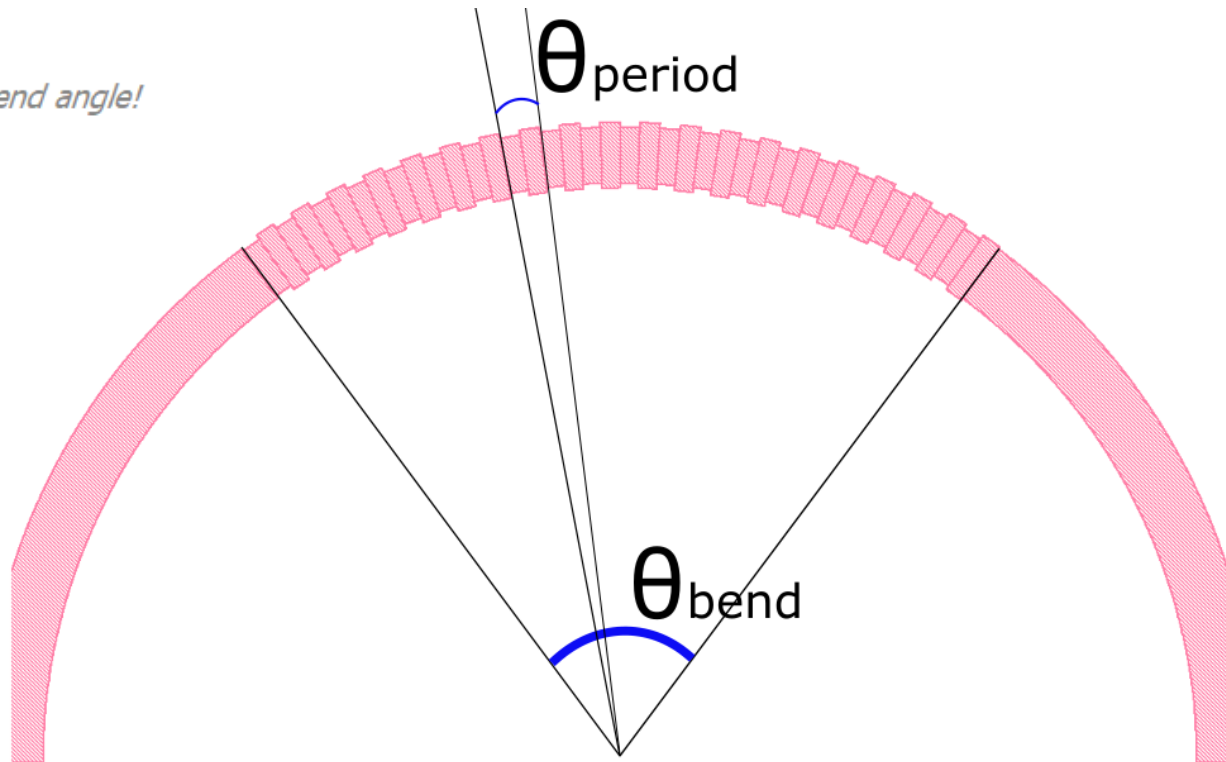
$y = 0$

Angle of Bragg corrugated portion, also bend angle!

$\text{periodAngle} = (180/\pi) * (\text{period}/2) / r$

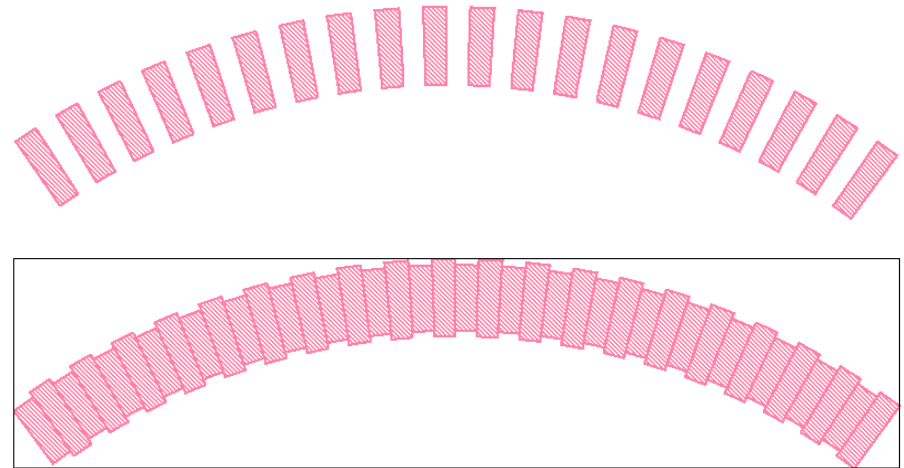
Bend angle

$\text{bendAngle} = (180/\pi) * (N * \text{period}/2) / r$



Pcell: Produce: Draw layout!

- **Method: Produce**
- **Draw corrugations (wide)**
- **Use an arc function (arg_wg_xy)**
 - skip every other period
- **Draw corrugations (narrow)**
- **Use an arc function (arg_wg_xy)**
 - skip every other period

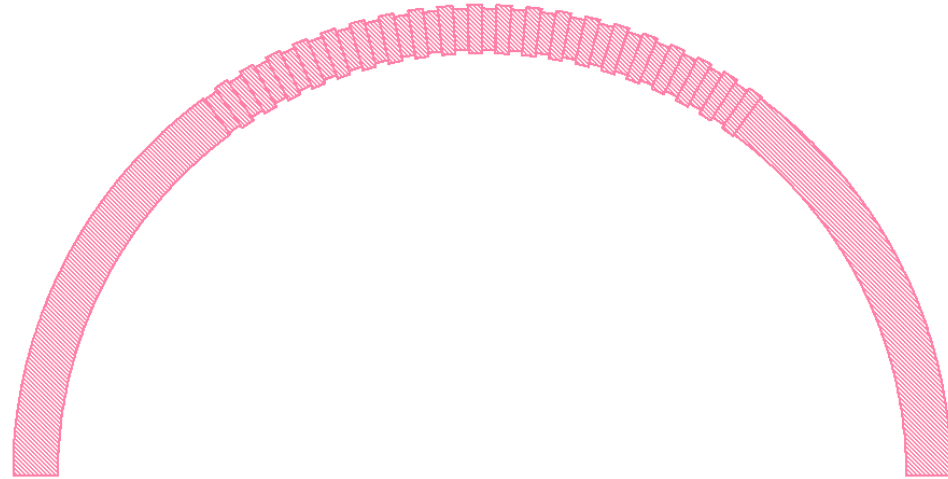


```
# Bragg wide
ii = periodAngle*2
while ii < N+periodAngle*1.5:
    self.cell.shapes(LayerSiN_gratings).insert(arc_wg_xy(x,y, r, w+deltaW, 90+bendAngle-ii, 90+bendAngle-ii-periodAngle))
    ii = ii+periodAngle
    ii = ii+periodAngle
```

```
# Bragg narrow
ii = periodAngle
while ii < N:
    self.cell.shapes(LayerSiN_gratings).insert(arc_wg_xy(x,y, r, w-deltaW, 90+bendAngle-ii, 90+bendAngle-ii-periodAngle))
    ii = ii+periodAngle
    ii = ii+periodAngle
```

Pcell: Produce: Draw layout!

- Method: Produce
- Draw bend sections (uncorrugated)
- Use an arc function (arg_wg_xy)

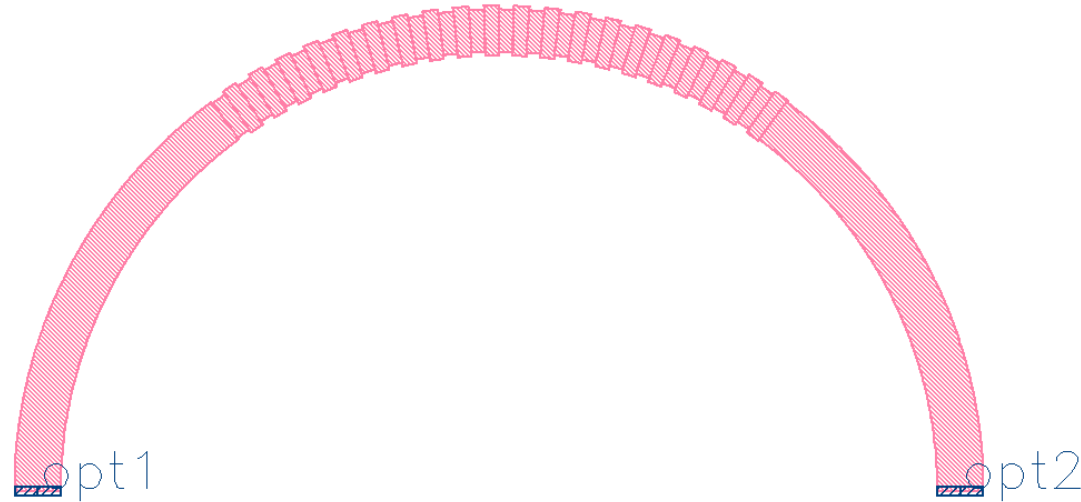


```
# bend non-corrugated left
self.cell.shapes(LayerSiN).insert(arc_wg_xy(x,y, r, w, 180-(90-bendAngle), 180))
```

```
# bend non-corrugated right
self.cell.shapes(LayerSiN).insert(arc_wg_xy(x,y, r, w, 0, 90-bendAngle))
```


Pcell: Produce: Draw layout!

- **Method: Produce**
- **Draw optical input pins**
- **As wide as waveguides width**
- **Be aware of orientation!**
- **This enables component snapping**
- **Used for connectivity/netlist**



bend non-corrugated left

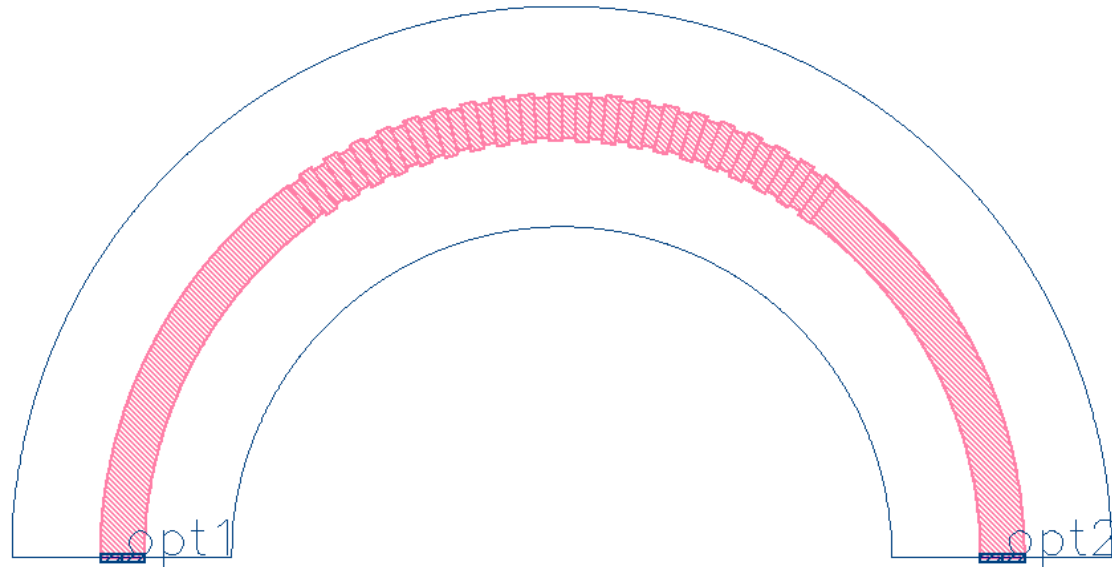
```
self.cell.shapes(LayerSiN).insert(arc_wg_xy(x,y, r, w, 180-(90-bendAngle), 180))
```

bend non-corrugated right

```
self.cell.shapes(LayerSiN).insert(arc_wg_xy(x,y, r, w, 0, 90-bendAngle))
```

Pcell: Produce: Draw layout!

- **Method: Produce**
- **Draw device recognition (bounding) box**
- **Used for device area recognition, prevents from overlapping with other components**



*# Create the device recognition layer -- make it 1 * wg_width away from the waveguides.*

```
self.cell.shapes(LayerDevRecN).insert(arc_wg_xy(x ,y, r, w*5, 0, 180))
```