Sam SiN Gratings

19/10/2023

Imports & Organisation

In [ ]:
```
import os
import math
import numpy as np
import klayout.db as db

from pathlib import Path
from IPython.display import Markdown, display, Image

root = Path().absolute()
out_path = Path(f'{root}/Data')
```

Voyager Settings

- We start by setting up the writefield and chip size for the voyager.
- These remain pretty constant during the exposure, so we can set them as variables at the top.
- The writefield size I will use is 500 μm, and the chip size is 15 x 15mm.

In [ ]:
```
writefield_height = 1000
writefield_width = 1000
chip_height = 15000
chip_width = 15000
```

- We assume that the database layout will be in μm, so the above numbers are set to these units.
- Later we ensure that the database units are set accordingly.

Database Setup

- It still makes sense to use one database file with several objects within it.
- It makes sense for as long as it keeps the file size down to a minimum.
- Here is where we set the database units to μm

In [ ]:
```
layout = db.Layout()
layout.dbu = 0.001
sample_identifier = 'AN9'
top_cell = layout.create_cell(f'{sample_identifier}')
```

Functions

- To keep things simple, we can write some of the grating generation as functions.
- Simple things such as making a grating, or a nanohole array.

In [ ]:
```
def makes_bars_cell(layout: object,
                    layer: object,
                    period: float,
                    fill_factor: float,
                    bar_height: float,
                    bar_identifier: str) -> object:
    """
    Create bar cell with the given dimensions.

    Parameters
    ---------
    layout, layer: object
        Database and layer objects from KLayout.
    period, fill_factor, bar_height: float
        Period, fill factor, and height of the bar.
    bar_identifier: string
        Bar cell name.

    Returns
    -------
    bar_cell: object
        Bar cell.

    See Also
    --------
    None

    Notes
    -----
    None

    Example
    -------
    None

    """
    bar_cell = layout.create_cell(f'{bar_identifier}')
    bar_origin = db.DPoint(0, 0)
    bar = db.DBox(
        bar_origin,
        (bar_origin + db.DVector(period * fill_factor, bar_height)))
    bar_cell.shapes(layer).insert(bar)
    return bar_cell


def makes_hole_cell(layout: object,
                    layer: object,
                    radius: float,
                    number_of_vertices: int,
                    hole_identifier: str) -> object:
    """
    Create hole cell with given dimensions.

    Parameters
    ----------
    layout, layer: object
```

```
        Database and layer objects from KLayout.
    radius: float
        Hole radius in database units.
    number_of_vertices: int
        Number of polygon vertices.
    hole_identifier: string
        Hole cell name.

    Returns
    -------
    hole_cell: object
        Hole cell.

    See Also
    --------
    makes_bars_cell

    Notes
    -----
    None

    Example
    -------
    None

    """
    hole_cell = layout.create_cell(f'{hole_identifier}')
    hole = db.DPolygon.ellipse(
        db.DBox(0 - radius, 0 - radius, 0 + radius, 0 + radius),
        number_of_vertices)
    hole_cell.shapes(layer).insert(hole)
    return hole_cell


def generates_texts(layout: object,
                    layer_index: int,
                    dose: float,
                    text_identifier: str,
                    text_string: str,
                    text_magnification: int):
    """
    Generate text string cell in KLayout.

    Parameters
    ---------
    layout: object
        Database object from KLayout.
    layer_index, text_magnification: int
        KLayout layer number and text size.
    dose: float
        Text layer dose.
    text_identifier, text_string: string
        Text cell name and text to write.

    Returns
    -------
    text_cell: object
        Text cell.

    See Also
```

```
    --------
    makes_bars_cell

    Notes
    -----
    None

    Example
    -------
    None

    """
    text_layer = layout.layer(layer_index, dose)
    text_cell = layout.create_cell(f'{text_identifier}')
    generator = db.TextGenerator.default_generator()
    region = generator.text(
        text_string,
        layout.dbu,
        text_magnification)
    text_cell.shapes(text_layer).insert(region)
    return text_cell


def makes_gratings_cell(layout: object,
                        grating_bar: object,
                        period: float,
                        grating_length: float,
                        grating_identifier: str) -> object:
    """
    Create a grating cell with the given bar cell, period, and grating length.

    Parameters
    ----------
    layout, grating_bar: object
        Database and cell objects from KLayout.
    period, grating_length: float
        Period or the grating and grating length in database units.
    grating_identifier: string
        Grating cell name.

    Returns
    -------
    grating_cell: object
        Grating cell.

    See Also
    --------
    makes_bars_cell

    Notes
    -----
    This makes a 1D grating with a period in X.

    Example
    -------
    None

    """
    grating_cell = layout.create_cell(f'{grating_identifier}')
    x_vector = db.DVector(period, 0)
```

```
    num_x = math.floor(grating_length / period)
    y_vector = db.DVector()
    num_y = 1
    grating_cell.insert(
        db.DCellInstArray(
            grating_bar.cell_index(),
            db.DTrans(),
            x_vector,
            y_vector,
            num_x,
            num_y))
    return grating_cell


def makes_nanohole_cell(layout: object,
                        hole_cell: object,
                        period_x: float,
                        period_y: float,
                        grating_length: float,
                        grating_height: float,
                        grating_identifier: str) -> object:
    """
    Create a nanohole cell with the given hole cell, period, and grating length.

    Parameters
    ----------
    layout, grating_bar: object
        Database and cell objects from KLayout.
    period_x, period_y, grating_length, grating_height: float
        Period of the grating and grating length in database units.
    grating_identifier: string
        Grating cell name.

    Returns
    -------
    grating_cell: object
        Grating cell.

    See Also
    --------
    makes_hole_cell

    Notes
    -----
    This makes a 2D grating with a period in X and Y.

    Example
    -------
    None

    """
    grating_cell = layout.create_cell(f'{grating_identifier}')
    x_vector = db.DVector(period_x, 0)
    num_x = math.floor(grating_length / period_x)
    y_vector = db.DVector(0, period_y)
    num_y = math.floor(grating_height / period_y)
    grating_cell.insert(
        db.DCellInstArray(
            hole_cell.cell_index(),
            db.DTrans(),
```

```
            x_vector,
            y_vector,
            num_x,
            num_y))
    return grating_cell


def chirped_coordinates(periods: list,
                        heights: list,
                        writefield_width: float) -> list:
    """
    Parameters
    ----------
    periods, heights: list
        Grating periods at the extreme left, central, and extreme right. Grating
        heights at the same values.

    Returns
    -------
    x_coords, y_coords: list
        List of x and y coordinates for the chirped bars.

    See Also
    --------
    None

    Notes
    -----
    None

    Example
    -------
    None

    """
    number_periods = math.floor(writefield_width / max(periods))
    x_central = []
    for period in periods:
        temporary = np.arange(
            - number_periods / 2 * period,
            (number_periods / 2 * period) + period,
            period)
        x_central.append(list(temporary))
    x_cen = list(
        np.asarray(x_central).reshape(len(periods), -2).transpose())
    x_coords = []
    y_coords = []
    for x_c in x_cen:
        x_c = list(x_c)
        temporary_x = []
        temporary_y = []
        for x, p, h in zip(x_c, periods, heights):
            temporary_x.append(x - (p / 2))
            temporary_y.append(h)
        for x, p, h in zip(reversed(x_c), reversed(periods), reversed(heights)):
            temporary_x.append(x - (p / 2) + (p * (1 - ff)))
            temporary_y.append(h)
        x_coords.append(temporary_x)
        y_coords.append(temporary_y)
    return x_coords, y_coords
```

```
def make_chirpedgrating(layout: object,
                        layer: object,
                        periods: list,
                        heights: list,
                        writefield_width: float,
                        grating_identifier: str) -> object:
    """
    Parameters
    ----------
    layout, layer: object
        Database and layer objected from KLayout.
    periods, heights: list
        Grating periods and heights for grating extremes.
    writefield_width: float
        Writefield width in database units.
    grating_identifier: str
        Grating cell name

    Returns
    -------
    grating_cell: object
        Grating cell.

    See Also
    --------
    chirped_coordinates

    Notes
    -----
    None

    Example
    -------
    None

    """
    grating_cell = layout.create_cell(f'{grating_identifier}')
    x_coordinates, y_coordinates = chirped_coordinates(
        periods=periods,
        heights=heights,
        writefield_width=writefield_width)
    for x, y in zip(x_coordinates, y_coordinates):
        points = []
        for point in zip(x, y):
            points.append(db.DPoint(*point))
        grating_cell.shapes(layer).insert(db.DPolygon(points))
    return grating_cell
```

Database Design

- We know the periods, fill factors, etc for this chip from the previous design, and I will keep things constant again for now.

Gratings Dictionary

- Add the grating dictionaries at the top of the database.

In [ ]:
```
gratings_dictionary = {}
```

Chip Outline

- Create the zero dose chip outline layer.

In [ ]:
```
chip_cell = layout.create_cell('Chip_Outline')
layer_index = 0
layer = layout.layer(layer_index, 0)
```

- Now we draw the objects based on the above chip outline dimensions.

In [ ]:
```
chip_bottom = db.DBox(
    db.DPoint(0, 0),
    (db.DPoint(0, 0) + db.DVector(chip_width, - 100)))
chip_left = db.DBox(
    db.DPoint(0, 0),
    (db.DPoint(0, 0) + db.DVector(-100, chip_height)))
chip_right = db.DBox(
    db.DPoint(chip_width, 0),
    (db.DPoint(chip_width, 0) + db.DVector(100, chip_height)))
chip_top = db.DBox(
    db.DPoint(0, chip_height),
    (db.DPoint(0, chip_height) + db.DVector(chip_width, 100)))
```

- Finally, add the chip outline boxes to the chip_cell.

In [ ]:
```
chip_cell.shapes(layer).insert(chip_bottom)
chip_cell.shapes(layer).insert(chip_left)
chip_cell.shapes(layer).insert(chip_right)
chip_cell.shapes(layer).insert(chip_top)
```

Out[ ]:
```
box (0,15000000;15000000,15100000)
```

Flat Gratings

- Let's start with the flat gratings.
- We are going to use a bunch of different gratings with different periods and different dose factor scalars.
- So we start by setting up those ranges.

In [ ]:
```
dose_factors = np.arange(1, 2.1, 0.5)
periods = range(420, 491, 10)
fill_factors = np.arange(0.6, 0.81, 0.1)
grating_spacing = 500
text_magnification = 60
text_spacing = 100
```

- Now we set up the dose cells.

- The dose cells will contain rows of gratings.
- Remember that we need to scale the dose factor by 1000.
- Note that when are replicating the same shape, then we use a different instance of DCellInstArray.

In [ ]:
```
layer_index = 1
text_layer = layout.layer(len(dose_factors) + 1, 2000)
flat_cell = layout.create_cell('Flat_Gratings')
for i, dose in enumerate(dose_factors):
    layer = layout.layer(layer_index + i, dose * 1000)
    dose_cell = layout.create_cell(f'Flat_df{dose}')
    dose_text_cell = layout.create_cell(f'FlatText_df{dose}')
    for j, ffs in enumerate(fill_factors):
        ff = round(ffs, 1)
        ff_cell = layout.create_cell(f'Flat_df{dose}_ff{ff}')
        ff_text_cell = layout.create_cell(f'FlatText_df{dose}_ff{ff}')
        for k, period in enumerate(periods):
            grating_period = period / 1000  # scale to database units
            grating_identifier = f'{sample_identifier}.flat.{i}.{j}.{k}'
            gratings_dictionary.update(
                {grating_identifier: f'df{dose}_p{period}_ff{ff}'})
            bar_cell = makes_bars_cell(
                layout=layout,
                layer=layer,
                period=grating_period,
                fill_factor=ff,
                bar_height=writefield_height,
                bar_identifier=f'FlatBar_df{dose}_p{period}_ff{ff}')
            grating_cell = makes_gratings_cell(
                layout=layout,
                grating_bar=bar_cell,
                period=grating_period,
                grating_length=writefield_width,
                grating_identifier=(
                    f'FlatGrating_df{dose}_p{period}_ff{ff}'))
            text_cell = generates_texts(
                layout=layout,
                layer_index=len(dose_factors) + 1,
                dose=2000,
                text_identifier=f'FlatText_df{dose}_p{period}_ff{ff}',
                text_string=grating_identifier,
                text_magnification=text_magnification)
            ff_cell.insert(
                db.DCellInstArray(
                    grating_cell.cell_index(),
                    db.DTrans(
                        db.DVector(
                            k * (writefield_width + grating_spacing),
                            0))))
            ff_text_cell.insert(
                db.DCellInstArray(
                    text_cell.cell_index(),
                    db.DTrans(
                        db.DVector(
                            k * (writefield_width + grating_spacing),
                            writefield_height + text_spacing))))
        dose_cell.insert(
            db.DCellInstArray(
```

```
                ff_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        0,
                        j * (writefield_width + grating_spacing)))))
        dose_text_cell.insert(
            db.DCellInstArray(
                ff_text_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        0,
                        j * (writefield_width + grating_spacing)))))
    flat_cell.insert(
        db.DCellInstArray(
            dose_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    0,
                    i * (
                            (writefield_height * len(fill_factors)) +
                            (grating_spacing * len(fill_factors)))))))
    flat_cell.insert(
        db.DCellInstArray(
            dose_text_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    0,
                    i * (
                            (writefield_height * len(fill_factors)) +
                            (grating_spacing * len(fill_factors)))))))
```

Patterning the Chip

- This bit can be done either within KLayout or on the chip.
- We start by adding the cells to the top cell.
- The cell names are:
  - chip_cell
  - flat_cell

In [ ]:
```
x_shift = 1800
y_shift = 1000
transform_0 = db.ICplxTrans(1, 0, False, 0, 0)
transform_1 = db.ICplxTrans(1, 0, False, x_shift, y_shift)
top_cell.insert(
    db.DCellInstArray(
        chip_cell.cell_index(),
        transform_0))
top_cell.insert(
    db.DCellInstArray(
        flat_cell.cell_index(),
        transform_1))
```

Out[ ]:
```
cell_index=2 r0 1800000,1000000
```

Database Write Out

- Finish by writing the database to a file.

In [ ]:
```
layout.write(f'{out_path}/Sam_AN9_231019.gds')
```

Out[ ]:
```
<klayout.dbcore.Layout at 0x18d67f4db60>
```