NIL Masters 19/10/2023

Targets

- NIL master substrates for Tim's project and general development of the process.
- Correct the wrongdoings of the previous failures with generating structures and increase writing time.
- The required gratings are:
  - Positive 1D gratings.
  - Negative 1D gratings
  - Positive 2D gratings
  - Chirped gratings.
  - Surface profilometry marks.
- Following the recent failures using the voyager system, with exposures for Sam and Tim both having issues, this notebook will also aim to fix those issues.
  - Improve write time.
  - Reduce file size.
  - Better quality of write.

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 = 500
writefield_width = 500
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 = 'AM4-7'
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.
- The first thing we can do is set up the chip outline, this layer will not have anything written on it, but it will allow us to see where on the chip our patterns will appear.

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)
```

- In the future, this can easily be a function, because the chip outline will always be the same.

Gratings Dictionary

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

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

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.

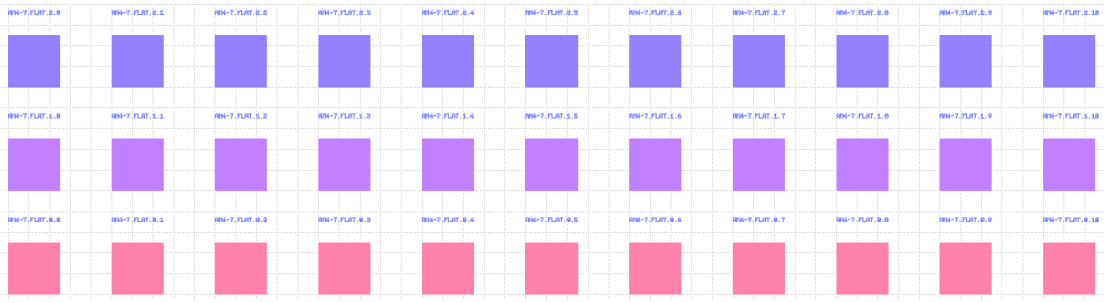
In [ ]:
```
dose_factors = np.arange(1, 2.1, 0.5)
periods = range(400, 501, 10)
fill_factor = 0.7
grating_spacing = 500
text_magnification = 60
text_spacing = 100
```

- 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.
- When we build the dose row, we instead use the DTrans(DVector) method to space them out.
- This mildly increases the file size, but should not be close to that of the previous iteration.

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, period in enumerate(periods):
        grating_period = period / 1000  # scale to database units
        grating_identifier = f'{sample_identifier}.flat.{i}.{j}'
        gratings_dictionary.update(
            {grating_identifier: f'df{dose}_p{period}_ff{fill_factor}'})
        bar_cell = makes_bars_cell(
            layout=layout,
            layer=layer,
            period=grating_period,
            fill_factor=fill_factor,
            bar_height=writefield_height,
```

```
            bar_identifier=f'FlatBar_df{dose}_p{period}_ff{fill_factor}')
        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{fill_factor}'))
        text_cell = generates_texts(
            layout=layout,
            layer_index=len(dose_factors) + 1,
            dose=2000,
            text_identifier=f'FlatText_df{dose}_p{period}_ff{fill_factor}',
            text_string=grating_identifier,
            text_magnification=text_magnification)
        dose_cell.insert(
            db.DCellInstArray(
                grating_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        j * (writefield_width + grating_spacing),
                        0))))
        dose_text_cell.insert(
            db.DCellInstArray(
                text_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        j * (writefield_width + grating_spacing),
                        writefield_height + text_spacing))))
    flat_cell.insert(
        db.DCellInstArray(
            dose_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    0,
                    i * (writefield_height + grating_spacing)))))
    flat_cell.insert(
        db.DCellInstArray(
            dose_text_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    0,
                    i * (writefield_height + grating_spacing)))))
```

- Now we should have a top cell, chip outline, and flat gratings in a singular cell.
- The flat gratings cell looks like this:



- And now we just rinse and repeat for the negative gratings.

Negative Gratings

- Negative gratings for the stamp material is exactly the same but with the opposite fill factor.
- This is so that when the stamp is coated, the fill factor is 0.7 as we want, rather than the 0.3 we get when we take the inverse of a normal grating.
- Simply repeat the process here.

In [ ]:
```
dose_factors = np.arange(1, 2.1, 0.5)
periods = range(400, 501, 10)
fill_factor = 0.3
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.

In [ ]:
```
layer_index = 1
text_layer = layout.layer(len(dose_factors) + 1, 2000)
negative_cell = layout.create_cell('Negative_Gratings')
for i, dose in enumerate(dose_factors):
    layer = layout.layer(layer_index + i, dose * 1000)
    dose_cell = layout.create_cell(f'Neg_df{dose}')
    dose_text_cell = layout.create_cell(f'NegText_df{dose}')
    for j, period in enumerate(periods):
        grating_period = period / 1000  # scale to database units
        grating_identifier = f'{sample_identifier}.neg.{i}.{j}'
        gratings_dictionary.update(
            {grating_identifier: f'df{dose}_p{period}_ff{fill_factor}'})
        bar_cell = makes_bars_cell(
            layout=layout,
            layer=layer,
            period=grating_period,
            fill_factor=fill_factor,
            bar_height=writefield_height,
            bar_identifier=f'NegBar_df{dose}_p{period}_ff{fill_factor}')
        grating_cell = makes_gratings_cell(
            layout=layout,
            grating_bar=bar_cell,
            period=grating_period,
            grating_length=writefield_width,
            grating_identifier=(
                f'NegGrating_df{dose}_p{period}_ff{fill_factor}'))
        text_cell = generates_texts(
            layout=layout,
            layer_index=len(dose_factors) + 1,
            dose=2000,
            text_identifier=f'NegText_df{dose}_p{period}_ff{fill_factor}',
            text_string=grating_identifier,
            text_magnification=text_magnification)
        dose_cell.insert(
            db.DCellInstArray(
                grating_cell.cell_index(),
```

```
                db.DTrans(
                    db.DVector(
                        j * (writefield_width + grating_spacing),
                        0))))
        dose_text_cell.insert(
            db.DCellInstArray(
                text_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        j * (writefield_width + grating_spacing),
                        writefield_height + text_spacing))))
    negative_cell.insert(
        db.DCellInstArray(
            dose_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    0,
                    i * (writefield_height + grating_spacing)))))
    negative_cell.insert(
        db.DCellInstArray(
            dose_text_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    0,
                    i * (writefield_height + grating_spacing)))))
```

- Now we should have a top cell, chip outline, flat gratings, and negative gratings in a singular cell.
- The negative gratings cell looks like this:

- And now we just need to figure out the nanohole gratings.

Nanoholes

- We can use the same principle as generating the gratings from bars with the nanoholes.
- The setup is different, and we require a y-vector too, but in principle this should be easier than expected.

In [ ]:
```
dose_factors = np.arange(1, 2.1, 0.5)
periods = range(400, 501, 10)
feature_size = 200
hole_radius = feature_size / 2
radius = hole_radius / 1000
num_vertices = 16
grating_spacing = 500
```

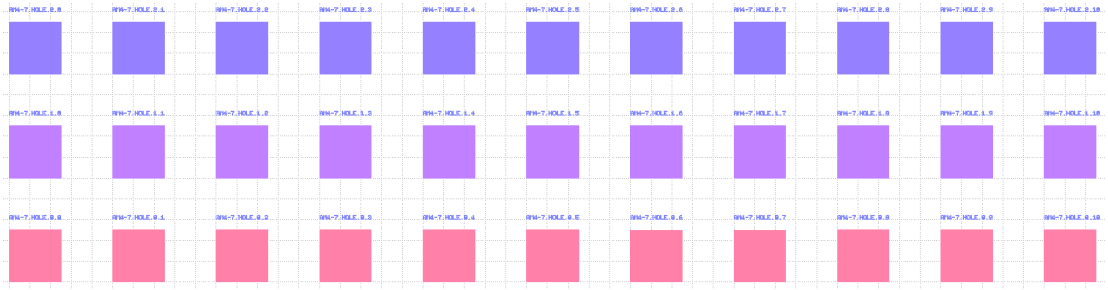
```
text_magnification = 60
text_spacing = 100
```

- 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.

In [ ]:
```
layer_index = 1
text_layer = layout.layer(len(dose_factors) + 1, 2000)
nanohole_cell = layout.create_cell('Nanohole_Gratings')
for i, dose in enumerate(dose_factors):
    layer = layout.layer(layer_index + i, dose * 1000)
    dose_cell = layout.create_cell(f'Hole_df{dose}')
    dose_text_cell = layout.create_cell(f'HoleText_df{dose}')
    for j, period in enumerate(periods):
        grating_period = period / 1000  # scale to database units
        grating_identifier = f'{sample_identifier}.hole.{i}.{j}'
        gratings_dictionary.update(
            {grating_identifier: f'df{dose}_p{period}_ff{fill_factor}'})
        hole_cell = makes_hole_cell(
            layout=layout,
            layer=layer,
            radius=radius,
            number_of_vertices=num_vertices,
            hole_identifier=f'Hole_df{dose}_p{period}_ff{fill_factor}')
        grating_cell = makes_nanohole_cell(
            layout=layout,
            hole_cell=hole_cell,
            period_x=grating_period,
            period_y=grating_period,
            grating_length=writefield_width,
            grating_height=writefield_height,
            grating_identifier=(
                f'HoleGrating_df{dose}_p{period}_ff{fill_factor}'))
        text_cell = generates_texts(
            layout=layout,
            layer_index=len(dose_factors) + 1,
            dose=2000,
            text_identifier=f'HoleText_df{dose}_p{period}_ff{fill_factor}',
            text_string=grating_identifier,
            text_magnification=text_magnification)
        dose_cell.insert(
            db.DCellInstArray(
                grating_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        j * (writefield_width + grating_spacing),
                        0))))
        dose_text_cell.insert(
            db.DCellInstArray(
                text_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        j * (writefield_width + grating_spacing),
                        writefield_height + text_spacing))))
    nanohole_cell.insert(
        db.DCellInstArray(
            dose_cell.cell_index(),
```

```
            db.DTrans(
                db.DVector(
                    0,
                    i * (writefield_height + grating_spacing)))))
    nanohole_cell.insert(
        db.DCellInstArray(
            dose_text_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    0,
                    i * (writefield_height + grating_spacing)))))
```

- Now we should have a top cell, chip outline, flat gratings, negative gratings, and nanohole gratings in a singular cell.
- The nanohole gratings cell looks like this:



- And now we just need to figure out the nanohole gratings.

Chirped

- The setup of the chirped gratings is still quite alien to me, I am still using Sam's code with little-to-no reference knowledge.
- I will use the same principle as before to setup the chirped gratings.
- The chirped grating is a period chirp, so we need to set at least 3 periods:
  - The period at one edge
  - The middle period
  - The period at the other edge
- The rest of the parameters we will keep constant.

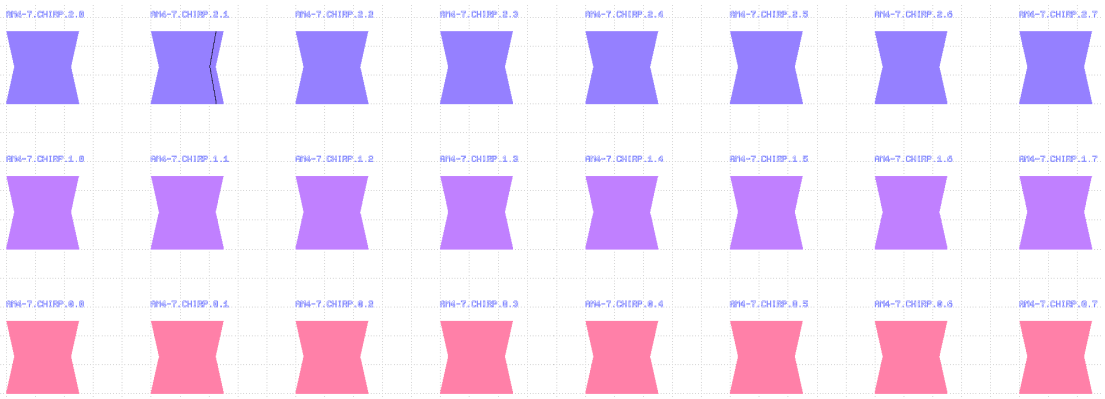
In [ ]:
```
period_range = [450, 350, 450]
fill_factors = np.arange(0.1, 0.9, 0.1)
grating_spacing = 500
text_spacing = 100
text_magnification = 60
dose_factors = np.arange(1, 2.1, 0.5)
heights = [0, writefield_height / 2, writefield_height]
```

- Following Sam's guidelines, the generation of chirped gratings is done differently to the way I have generated the flat gratings and the nanohole arrays.
- Instead of generating a basic shape and patterning that into a coordinate system, Sam generates the coordinates of the box and then draws a polygon in that space.

In [ ]:
```
layer_index = 1
text_layer = layout.layer(len(dose_factors) + 1, 2000)
chirped_cell = layout.create_cell('Chirped_Gratings')
for i, dose in enumerate(dose_factors):
    layer = layout.layer(layer_index + i, dose * 1000)
    dose_cell = layout.create_cell(f'Chirp_df{dose}')
    dose_text_cell = layout.create_cell(f'ChirpText_df{dose}')
    for j, ff in enumerate(fill_factors):
        grating_periods = [p / 1000 for p in period_range]
        grating_identifier = f'{sample_identifier}.chirp.{i}.{j}'
        gratings_dictionary.update(
            {grating_identifier: f'df{dose}_p{period_range}_ff{ff}'})
        grating_cell = make_chirpedgrating(
            layout=layout,
            layer=layer,
            periods=grating_periods,
            heights=heights,
            writefield_width=writefield_width,
            grating_identifier=f'Chirped_df{dose}_p{period_range}_ff{ff}')
        text_cell = generates_texts(
            layout=layout,
            layer_index=len(dose_factors) + 1,
            dose=2000,
            text_identifier=f'ChirpText_df{dose}_p{period_range}_ff{ff}',
            text_string=grating_identifier,
            text_magnification=text_magnification)
        dose_cell.insert(
            db.DCellInstArray(
                grating_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        j * (writefield_width + grating_spacing),
                        0))))
        dose_text_cell.insert(
            db.DCellInstArray(
                text_cell.cell_index(),
                db.DTrans(
                    db.DVector(
                        j * (writefield_width + grating_spacing),
                        writefield_height + text_spacing))))
    chirped_cell.insert(
        db.DCellInstArray(
            dose_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    writefield_width / 2,
                    i * (writefield_height + grating_spacing)))))
    chirped_cell.insert(
        db.DCellInstArray(
            dose_text_cell.cell_index(),
            db.DTrans(
                db.DVector(
                    0,
                    i * (writefield_height + grating_spacing)))))
```

- Now we should have a top cell, chip outline, flat gratings, negative gratings, nanohole gratings, and chirped gratings in a singular cell.

- There is a mild problem with the chirped grating design, where I can't adjust the period range anymore without breaking the code.
- This is something that I will talk to Sam about at a later date.
- The chirped gratings cell looks like this:



- All that remains is to add the measurement boxes that Tim wants on his chip, and to add them to a position list, or at least figure out the position list.

Measurement Boxes

- This is a super simple one, nothing fancy here.
- Just draw several boxes and space them out.
- Something on the order of magnitude that can be measured by the Dektak.

In [ ]:
```
measure_cell = layout.create_cell('Measure_Marks')
layer = layout.layer(5, 1000)
bar_cell = layout.create_cell(f'MeasureBar')
bar = db.DBox(
    db.DPoint(0, 0),
    (db.DPoint(0, 0) + db.DVector(500, 1000)))
bar_cell.shapes(layer).insert(bar)
number_repeats = 3
for n in range(0, number_repeats):
    measure_cell.insert(
        db.DCellInstArray(
            bar_cell.cell_index(),
            db.DTrans(db.DVector(n * 1000, 0))))
```

Patterning the Chip

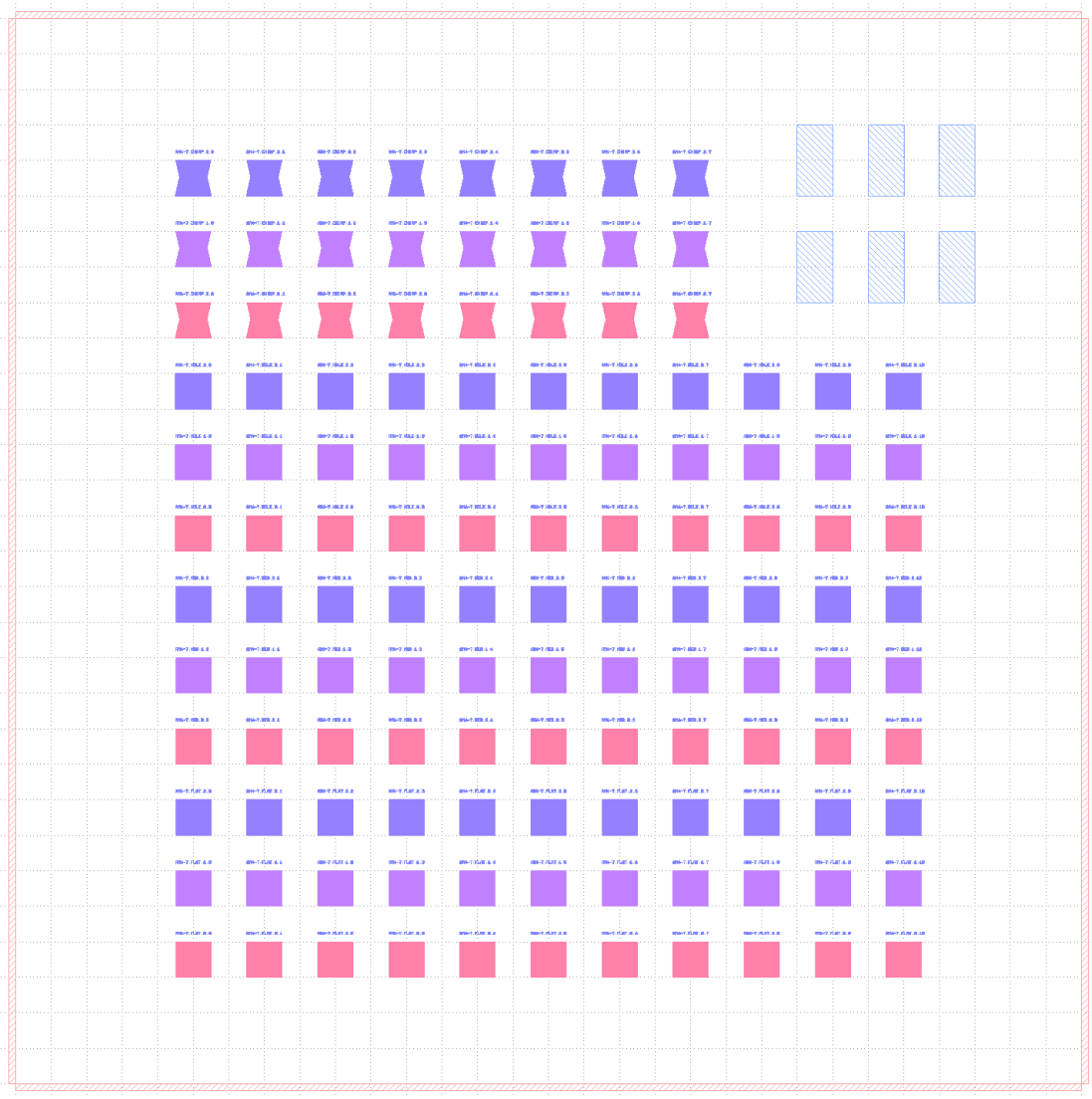
- This bit can be done either within KLayout or on the chip.
- Given that the overall file size before this part is just under 2500 KB, I am going to pattern the chip and load one singular file into the voyager system.
- The system can handle the individual database elements too, so having a "position list" is not a bad thing either.
- We start by adding the cells to the top cell.
- The cell names are:
  - chip_cell
  - flat_cell

- negative_cell
- nanohole_cell
- chirped_cell

In [ ]:
```
x_shift = 2250
y_shift = 1500
transform_0 = db.ICplxTrans(1, 0, False, 0, 0)
transform_1 = db.ICplxTrans(1, 0, False, 0 + x_shift, y_shift + 0)
transform_2 = db.ICplxTrans(
    1, 0, False, 0 + x_shift, y_shift + 6 * grating_spacing)
transform_3 = db.ICplxTrans(
    1, 0, False, 0 + x_shift, y_shift + 12 * grating_spacing)
transform_4 = db.ICplxTrans(
    1, 0, False, x_shift, y_shift + 18 * grating_spacing)
transform_5 = db.ICplxTrans(
    1, 0, False, 11000, y_shift + 500 + 18 * grating_spacing)
transform_6 = db.ICplxTrans(
    1, 0, False, 11000, y_shift + 2000 + 18 * grating_spacing)
top_cell.insert(
    db.DCellInstArray(
        chip_cell.cell_index(),
        transform_0))
top_cell.insert(
    db.DCellInstArray(
        flat_cell.cell_index(),
        transform_1))
top_cell.insert(
    db.DCellInstArray(
        negative_cell.cell_index(),
        transform_2))
top_cell.insert(
    db.DCellInstArray(
        nanohole_cell.cell_index(),
        transform_3))
top_cell.insert(
    db.DCellInstArray(
        chirped_cell.cell_index(),
        transform_4))
top_cell.insert(
    db.DCellInstArray(
        measure_cell.cell_index(),
        transform_5))
top_cell.insert(
    db.DCellInstArray(
        measure_cell.cell_index(),
        transform_6))
```

Out[ ]:
```
cell_index=375 r0 11000000,12500000
```

- Now the chip is complete and looks like this:



Grating Keys

- Just the grating keys to output, so that each of the grating numbers on the chips can be easily identified.
- To reduce the writing time on the chip, this can simply be printed here.

In [ ]:
```
display(
    Markdown(
        '\n'.join(
            [
                f'{key}: {value}\n'
                for key, value
                in gratings_dictionary.items()])))
```

AM4-7.flat.0.0: df1.0_p400_ff0.7

AM4-7.flat.0.1: df1.0_p410_ff0.7

AM4-7.flat.0.2: df1.0_p420_ff0.7

AM4-7.flat.0.3: df1.0_p430_ff0.7

AM4-7.flat.0.4: df1.0_p440_ff0.7

AM4-7.flat.0.5: df1.0_p450_ff0.7

AM4-7.flat.0.6: df1.0_p460_ff0.7

AM4-7.flat.0.7: df1.0_p470_ff0.7

AM4-7.flat.0.8: df1.0_p480_ff0.7

AM4-7.flat.0.9: df1.0_p490_ff0.7

AM4-7.flat.0.10: df1.0_p500_ff0.7

AM4-7.flat.1.0: df1.5_p400_ff0.7

AM4-7.flat.1.1: df1.5_p410_ff0.7

AM4-7.flat.1.2: df1.5_p420_ff0.7

AM4-7.flat.1.3: df1.5_p430_ff0.7

AM4-7.flat.1.4: df1.5_p440_ff0.7

AM4-7.flat.1.5: df1.5_p450_ff0.7

AM4-7.flat.1.6: df1.5_p460_ff0.7

AM4-7.flat.1.7: df1.5_p470_ff0.7

AM4-7.flat.1.8: df1.5_p480_ff0.7

AM4-7.flat.1.9: df1.5_p490_ff0.7

AM4-7.flat.1.10: df1.5_p500_ff0.7

AM4-7.flat.2.0: df2.0_p400_ff0.7

AM4-7.flat.2.1: df2.0_p410_ff0.7

AM4-7.flat.2.2: df2.0_p420_ff0.7

AM4-7.flat.2.3: df2.0_p430_ff0.7

AM4-7.flat.2.4: df2.0_p440_ff0.7

AM4-7.flat.2.5: df2.0_p450_ff0.7

AM4-7.flat.2.6: df2.0_p460_ff0.7

AM4-7.flat.2.7: df2.0_p470_ff0.7

AM4-7.flat.2.8: df2.0_p480_ff0.7

AM4-7.flat.2.9: df2.0_p490_ff0.7

AM4-7.flat.2.10: df2.0_p500_ff0.7

AM4-7.neg.0.0: df1.0_p400_ff0.3

AM4-7.neg.0.1: df1.0_p410_ff0.3

AM4-7.neg.0.2: df1.0_p420_ff0.3

AM4-7.neg.0.3: df1.0_p430_ff0.3

AM4-7.neg.0.4: df1.0_p440_ff0.3

AM4-7.neg.0.5: df1.0_p450_ff0.3

AM4-7.neg.0.6: df1.0_p460_ff0.3

AM4-7.neg.0.7: df1.0_p470_ff0.3

AM4-7.neg.0.8: df1.0_p480_ff0.3

AM4-7.neg.0.9: df1.0_p490_ff0.3

AM4-7.neg.0.10: df1.0_p500_ff0.3

AM4-7.neg.1.0: df1.5_p400_ff0.3

AM4-7.neg.1.1: df1.5_p410_ff0.3

AM4-7.neg.1.2: df1.5_p420_ff0.3

AM4-7.neg.1.3: df1.5_p430_ff0.3

AM4-7.neg.1.4: df1.5_p440_ff0.3

AM4-7.neg.1.5: df1.5_p450_ff0.3

AM4-7.neg.1.6: df1.5_p460_ff0.3

AM4-7.neg.1.7: df1.5_p470_ff0.3

AM4-7.neg.1.8: df1.5_p480_ff0.3

AM4-7.neg.1.9: df1.5_p490_ff0.3

AM4-7.neg.1.10: df1.5_p500_ff0.3

AM4-7.neg.2.0: df2.0_p400_ff0.3

AM4-7.neg.2.1: df2.0_p410_ff0.3

AM4-7.neg.2.2: df2.0_p420_ff0.3

AM4-7.neg.2.3: df2.0_p430_ff0.3

AM4-7.neg.2.4: df2.0_p440_ff0.3

AM4-7.neg.2.5: df2.0_p450_ff0.3

AM4-7.neg.2.6: df2.0_p460_ff0.3

AM4-7.neg.2.7: df2.0_p470_ff0.3

AM4-7.neg.2.8: df2.0_p480_ff0.3

AM4-7.neg.2.9: df2.0_p490_ff0.3

AM4-7.neg.2.10: df2.0_p500_ff0.3

AM4-7.hole.0.0: df1.0_p400_ff0.3

AM4-7.hole.0.1: df1.0_p410_ff0.3

AM4-7.hole.0.2: df1.0_p420_ff0.3

AM4-7.hole.0.3: df1.0_p430_ff0.3

AM4-7.hole.0.4: df1.0_p440_ff0.3

AM4-7.hole.0.5: df1.0_p450_ff0.3

AM4-7.hole.0.6: df1.0_p460_ff0.3

AM4-7.hole.0.7: df1.0_p470_ff0.3

AM4-7.hole.0.8: df1.0_p480_ff0.3

AM4-7.hole.0.9: df1.0_p490_ff0.3

AM4-7.hole.0.10: df1.0_p500_ff0.3

AM4-7.hole.1.0: df1.5_p400_ff0.3

AM4-7.hole.1.1: df1.5_p410_ff0.3

AM4-7.hole.1.2: df1.5_p420_ff0.3

AM4-7.hole.1.3: df1.5_p430_ff0.3

AM4-7.hole.1.4: df1.5_p440_ff0.3

AM4-7.hole.1.5: df1.5_p450_ff0.3

AM4-7.hole.1.6: df1.5_p460_ff0.3

AM4-7.hole.1.7: df1.5_p470_ff0.3

AM4-7.hole.1.8: df1.5_p480_ff0.3

AM4-7.hole.1.9: df1.5_p490_ff0.3

AM4-7.hole.1.10: df1.5_p500_ff0.3

AM4-7.hole.2.0: df2.0_p400_ff0.3

AM4-7.hole.2.1: df2.0_p410_ff0.3

AM4-7.hole.2.2: df2.0_p420_ff0.3

AM4-7.hole.2.3: df2.0_p430_ff0.3

AM4-7.hole.2.4: df2.0_p440_ff0.3

AM4-7.hole.2.5: df2.0_p450_ff0.3

AM4-7.hole.2.6: df2.0_p460_ff0.3

AM4-7.hole.2.7: df2.0_p470_ff0.3

AM4-7.hole.2.8: df2.0_p480_ff0.3

AM4-7.hole.2.9: df2.0_p490_ff0.3

AM4-7.hole.2.10: df2.0_p500_ff0.3

AM4-7.chirp.0.0: df1.0_p[450, 350, 450]_ff0.1

AM4-7.chirp.0.1: df1.0_p[450, 350, 450]_ff0.2

AM4-7.chirp.0.2: df1.0_p[450, 350, 450]_ff0.30000000000000004

AM4-7.chirp.0.3: df1.0_p[450, 350, 450]_ff0.4

AM4-7.chirp.0.4: df1.0_p[450, 350, 450]_ff0.5

AM4-7.chirp.0.5: df1.0_p[450, 350, 450]_ff0.6

AM4-7.chirp.0.6: df1.0_p[450, 350, 450]_ff0.7000000000000001

AM4-7.chirp.0.7: df1.0_p[450, 350, 450]_ff0.8

AM4-7.chirp.1.0: df1.5_p[450, 350, 450]_ff0.1

AM4-7.chirp.1.1: df1.5_p[450, 350, 450]_ff0.2

AM4-7.chirp.1.2: df1.5_p[450, 350, 450]_ff0.30000000000000004

AM4-7.chirp.1.3: df1.5_p[450, 350, 450]_ff0.4

AM4-7.chirp.1.4: df1.5_p[450, 350, 450]_ff0.5

AM4-7.chirp.1.5: df1.5_p[450, 350, 450]_ff0.6

AM4-7.chirp.1.6: df1.5_p[450, 350, 450]_ff0.7000000000000001

AM4-7.chirp.1.7: df1.5_p[450, 350, 450]_ff0.8

AM4-7.chirp.2.0: df2.0_p[450, 350, 450]_ff0.1

AM4-7.chirp.2.1: df2.0_p[450, 350, 450]_ff0.2

AM4-7.chirp.2.2: df2.0_p[450, 350, 450]_ff0.30000000000000004

AM4-7.chirp.2.3: df2.0_p[450, 350, 450]_ff0.4

AM4-7.chirp.2.4: df2.0_p[450, 350, 450]_ff0.5

AM4-7.chirp.2.5: df2.0_p[450, 350, 450]_ff0.6

AM4-7.chirp.2.6: df2.0_p[450, 350, 450]_ff0.7000000000000001

AM4-7.chirp.2.7: df2.0_p[450, 350, 450]_ff0.8

Database Write Out

- Finish by writing out the database file.

In [ ]:
```
layout.write(f'{out_path}/NIL_Masters_231019.gds')
```

Out[ ]:
```
<klayout.dbcore.Layout at 0x1a689452500>
```