PDK Generator Documentation

For users of PDK-Generator, see section Quick Start Guide

Background

Silicon photonic integrated circuits are conventionally designed like electronic circuits. Just like in electronic circuits, capacitors, resistors, inductors, etc., manipulate the flow of electrons, in silicon photonics integrated circuits, the flow of light is manipulated in waveguides using switches, couplers, filters, etc. These components are often designed manually, over a long process of design, simulation, layout, etc, requiring substantial manual intervention from the designer throughout the design process. It would thus be desirable to automate the design of such circuits and foster their reuse across multiple technology generations, to shorten time-to-market of new products and to free designers from performing repetitive tasks.

We are working on an integrated framework for the development of generators of silicon photonic circuits. Generators are parameterized design procedures that produce schematics and correct layouts optimized to meet a set of input specifications. This can be done by implementing interfaces to integrate all steps of the design flow into a single environment and by providing helper classes – at the schematic and the layout level – to aid the designer in developing parameterized and technology-independent circuit generators. Such a framework will simplify and help codify common tasks in the design flow including technology characterization, schematic and testbench translation, simulator interfacing, layout creation and verification. It will also address the labor-intensive task of circuit layout, by providing template-based extensible layout generators for different styles of circuits. The framework can be developed by capturing the design intent that goes into the manual design of the components, using a python-driven approach. Being able to drive all the design, simulation and layout using such an automatic software framework will lead to a new, rapid way of designing the photonic components. Such components can then be quickly tweaked for specific applications.

What is a PDK / Technology? Ch. 10 of Lukas' Book:
https://qdot-nexus.phas.ubc.ca:25683/s/xd4EnfKRnpAW3SF#pdfviewer

JSTQE 2019 paper about SiEPICfab

Overview

Goals

- ❏ Automate building silicon photonics Process Design Kits (PDKs) and Libraries.
- ❏ Build a PDK and Library of components in the **SiEPICfab-Grouse** process

Key Components

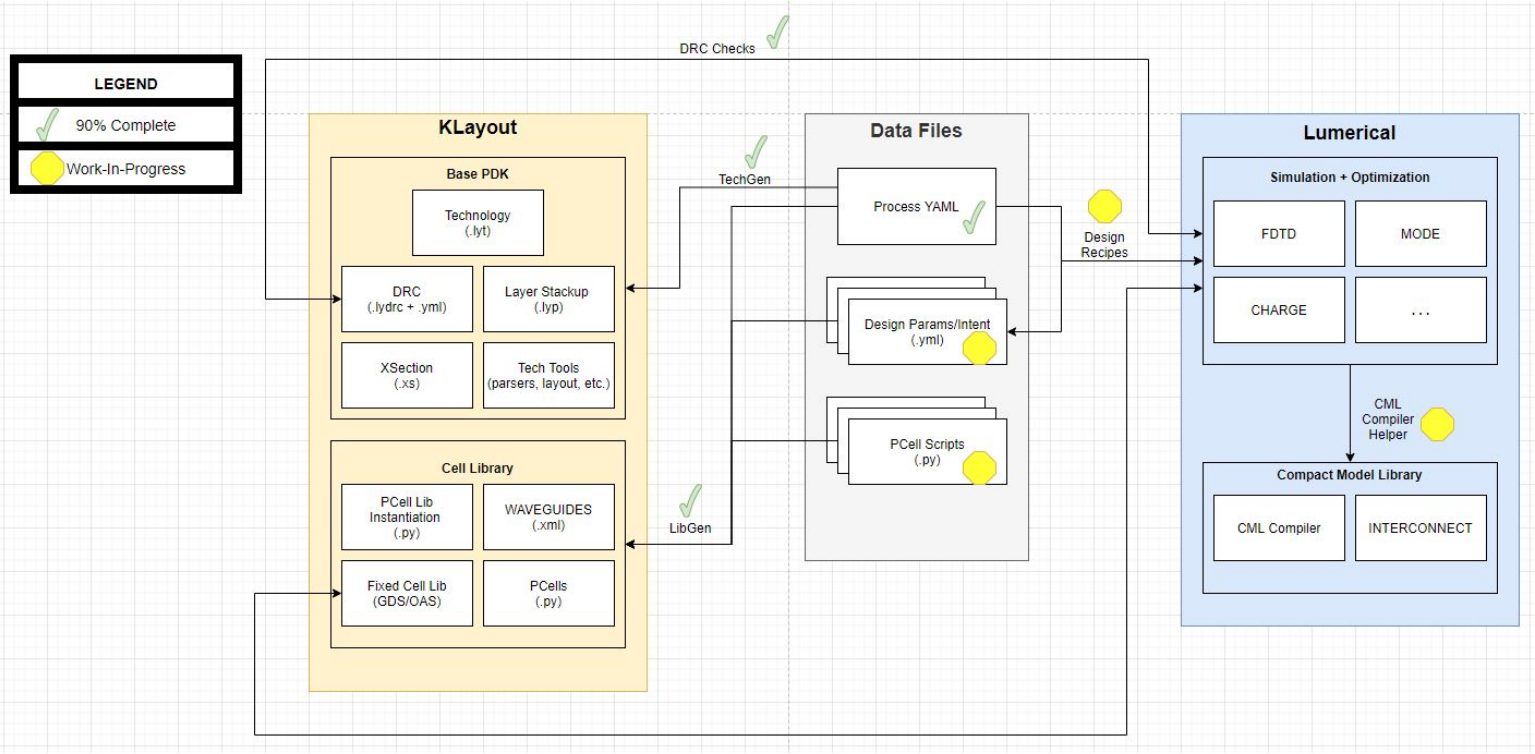
To automate building of PDKs and Libraries, key parts of the PDK and Libraries need to be generated/used:

Component	Description	Examples
PDK Documentation	Each PDK requires documentation about installing/updating the PDK, using the PDK, design rules and guidelines.	PDFs, READMEs (.md)
Layout Tools/Software	The PDK needs to be generated for a particular layout software tool.	KLayout, Mentor Graphics Pyxis, SiEPIC-Tools
Design Rule Checking (DRC)	Every layout needs to undergo design rule checking to ensure manufacturability	KLayout DRC, Calibre DRC
Layout vs. Schematic	Checking between schematic and layout to ensure functionality matches physical realization.	
Component Simulation/Optimization	Simulation/Optimization software is required to develop components	Lumerical FDTD, MODE
Circuit Simulation/Optimization	Simulation/Optimization software is required to develop circuits	Lumerical INTERCONNECT
Lithography Simulation	Simulation software required to simulate lithography inaccuracies and tolerances to ensure components still meet design requirements under certain corner cases.	
Component Documentation	Documentation about components generated in a library showing key performance parameters, limitations, and usage	PDFs, READMEs (.md)
Fixed Cell Libraries	Libraries that contain fixed layouts (not parameterized) and are used as is.	GDS, OAS
Parameterized Cell (PCell) Libraries	Libraries that contain parameterized layouts that are changed by varying design parameters	Python scripts
Compact Model Libraries	Libraries that contain compact models used for circuit simulation or Monte Carlo simulation.	Lumerical (.cml)

Block Diagram

The following shows the desired ecosystem for PDK-Generator. Progress shown is updated as of **Sept. 2, 2020.**

- KLayout is the chosen layout software along with DRC checking.
- Lumerical is the chosen simulation software necessary for circuit and component development.
- SiEPIC-Tools is a KLayout package that is used for developing cell layouts.
- YAML is the primary file format for storing configuration data (i.e. for TechGen or Design Recipes).
- Python is the chosen development language for PDK-Generator



Process YAML (Progress: 90%)

Purpose: Configuration file that encapsulates all process/foundry parameters, including DRC, layer stack or layer properties, and cross-section.

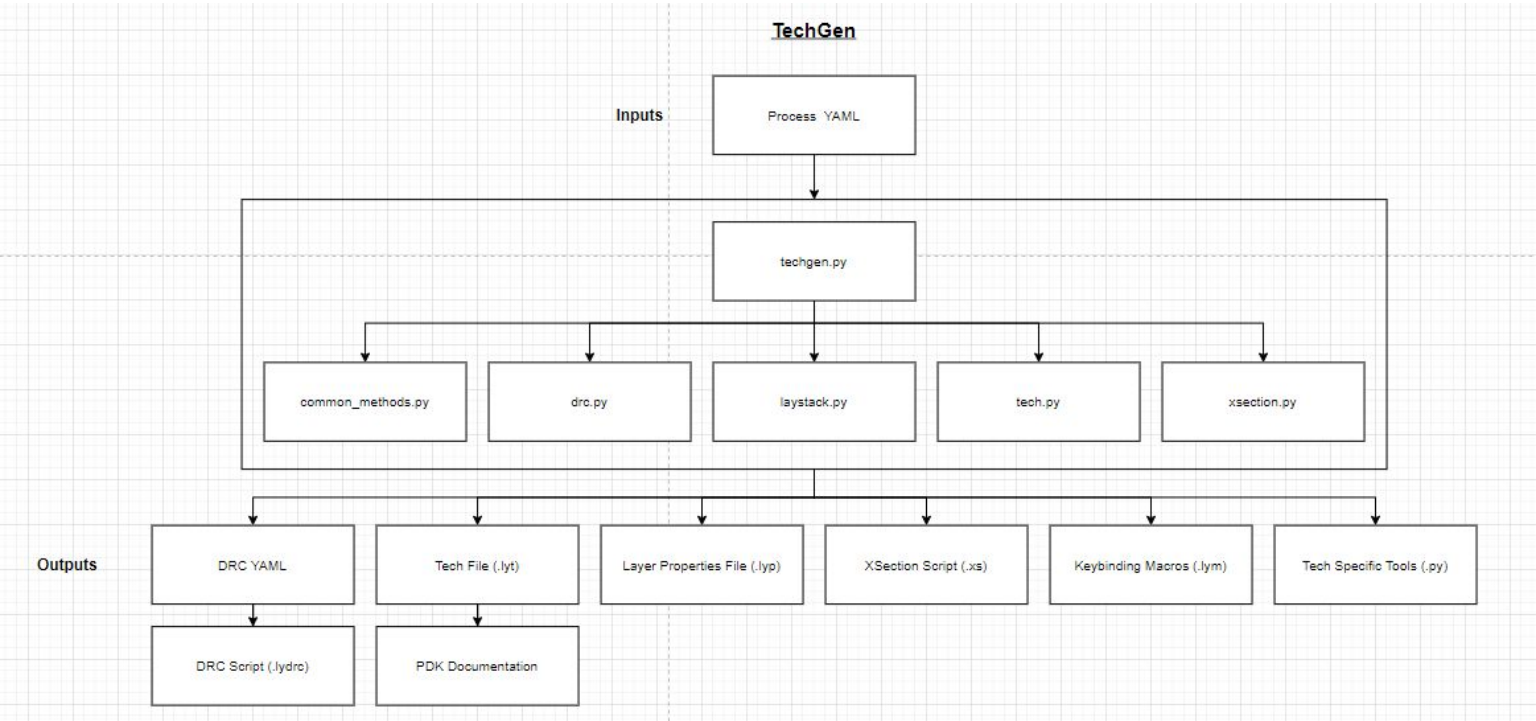
Desired Functionality:

- ✓ Capture .lyt (tech), .lyp (layer properties), .xs (cross section), .lydrc (DRC) params
- ✓ Capture process/foundry parameters (i.e. min feature size = 60 nm, mask = negative)
- ❏ Transferable to other foundries/processes
- ❏ Automate checking of Process YAML for correct formatting, duplicate fields, etc.

TechGen (Progress: 90%)

Purpose: Generate base PDK files and folder structure based on Process YAML

Desired Functionality:



- ❏ Generate the following from the Process YAML
  - ✓ DRC YAML
  - ✓ DRC script (Ruby macro .lydrc) (see https://www.klayout.de/doc-qt5/manual/drc.html for more info)
  - ✓ Tech File (XML formatted .lyt file) (see https://www.klayout.de/doc-qt5/about/technology_manager.html for more info)
  - ✓ Layer Properties file (XML formatted .lyp file) (see https://www.klayout.de/lyp_format.html for more info)
  - ✓ XSection Script (Ruby script .xs file) (see https://github.com/klayoutmatthias/xsection for more info)
  - ❏ XSection Step-by-Step Script
  - ✓ Keybind macros (.lym). These scripts are used to bind keyboard shortcuts to macros that perform some action in KLayout (i.e. “Shift + X” will perform a cross section check) (see https://www.klayout.de/doc/about/macro_editor.html for more info)
  - ✓ Tech specific tools (.py). This may include python scripts that assist with layout, parsers for data, etc.
  - ❏ PDK Documentation, including fabrication cross section diagram
- ✓ Generate folder structure
- ❏ Automate script validation to verify outputs of TechGen are functional. Ex. Add script that runs generated .lydrc script against known layout to check whether DRC works as expected.
- ❏ Develop unit testing

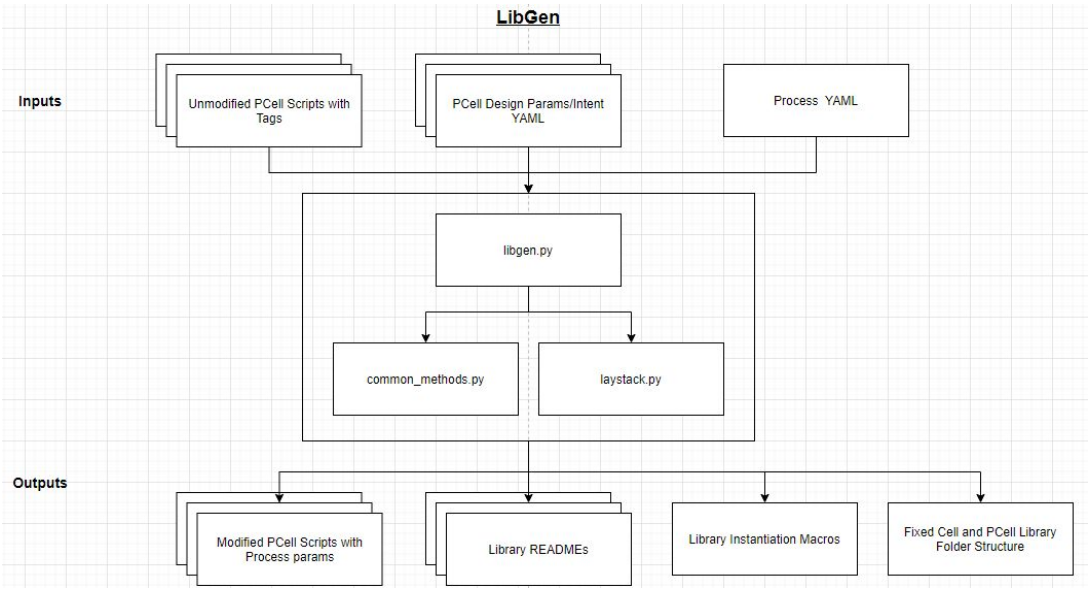
Code Structure

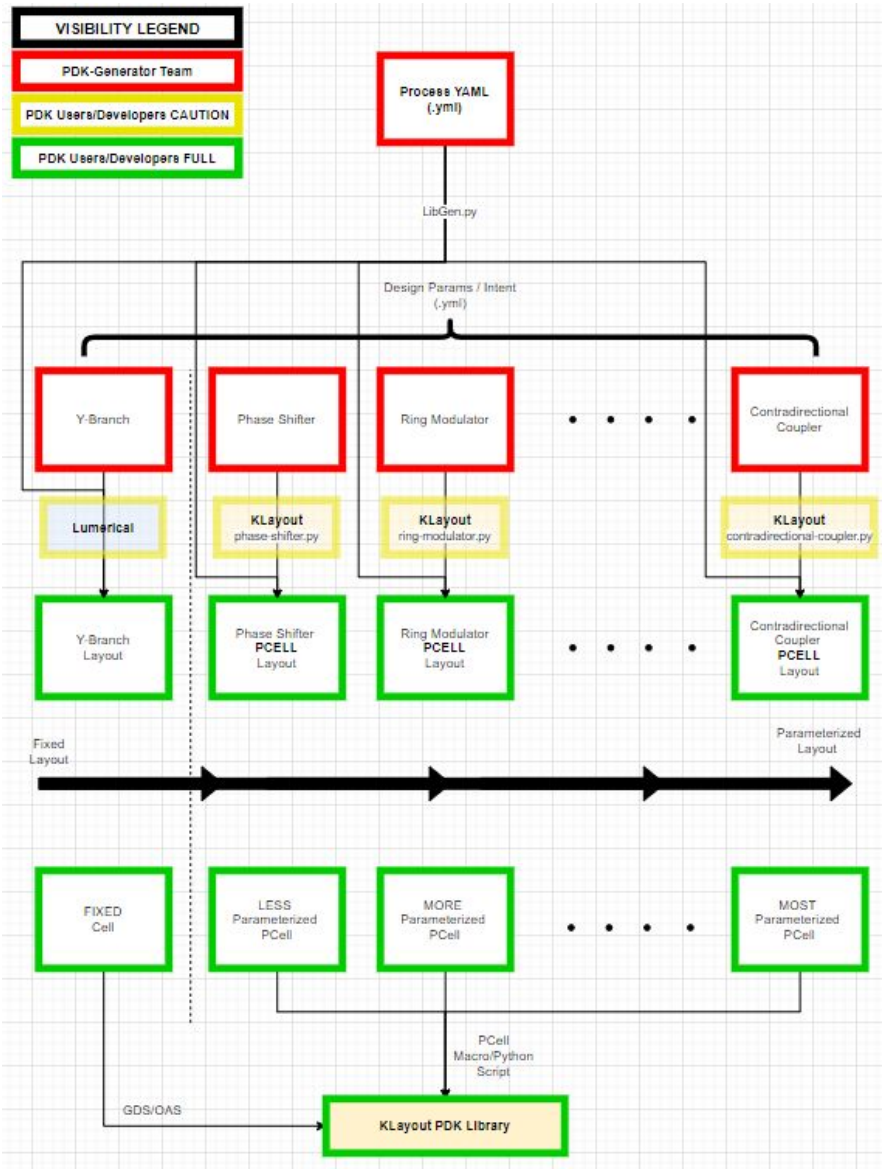
techgen.py: Main TechGen functionality “generate_technology()” which encapsulates TechGen
common_methods.py: Common methods used across TechGen and LibGen (i.e. prettify XML string)
drc.py: DRC class with DRC params and methods to generate the .lydrc script
laystack.py: Layer properties class with layer properties params and methods to generate the .lyp file
tech.py: Tech class with technology params and methods to generate .lyt file
xsection.py: XSection class with XSection params and methods to generate .xs file

LibGen (Progress: 80%)

Purpose: Generate fixed cell and PCell libraries along with READMEs and folder structure

Desired Functionality:



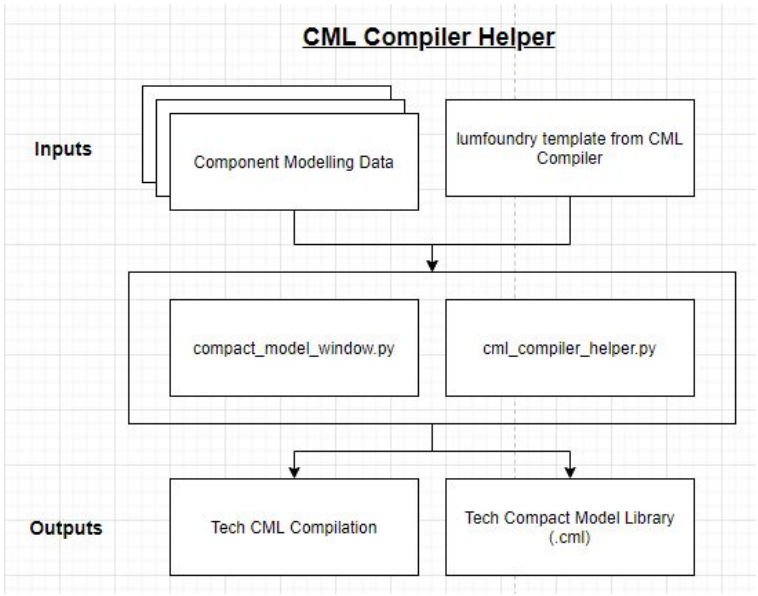


- ✓ Generate GDS and OAS fixed cell library folders with READMEs for mature and dev libraries
- ✓ Generate PCell library folders, modules, READMEs for mature and dev libraries
- ✓ Generate library instantiation macros
- ✓ Replace layers in PCell scripts with Process layers
- ❑ Generate cell documentation
- ❑ Replace particular design params with DRC params
- ❑ Integrate DRC checking within PCell script
- ❑ Copy fixed cells into generated library while replacing layer information and checking DRC
- ❑ Automate script validation to ensure LibGen outputs are functional. Ex. add script that instantiates all PCells to check whether the generated PCell scripts are working with no errors.
- ❑ Generate cross section (side view + top view) for all cells in library
- ❑ Discuss/Adding pre-processing vs. post-processing for ensuring masking is properly handled (negative vs positive masking)
- ❑ Develop unit testing

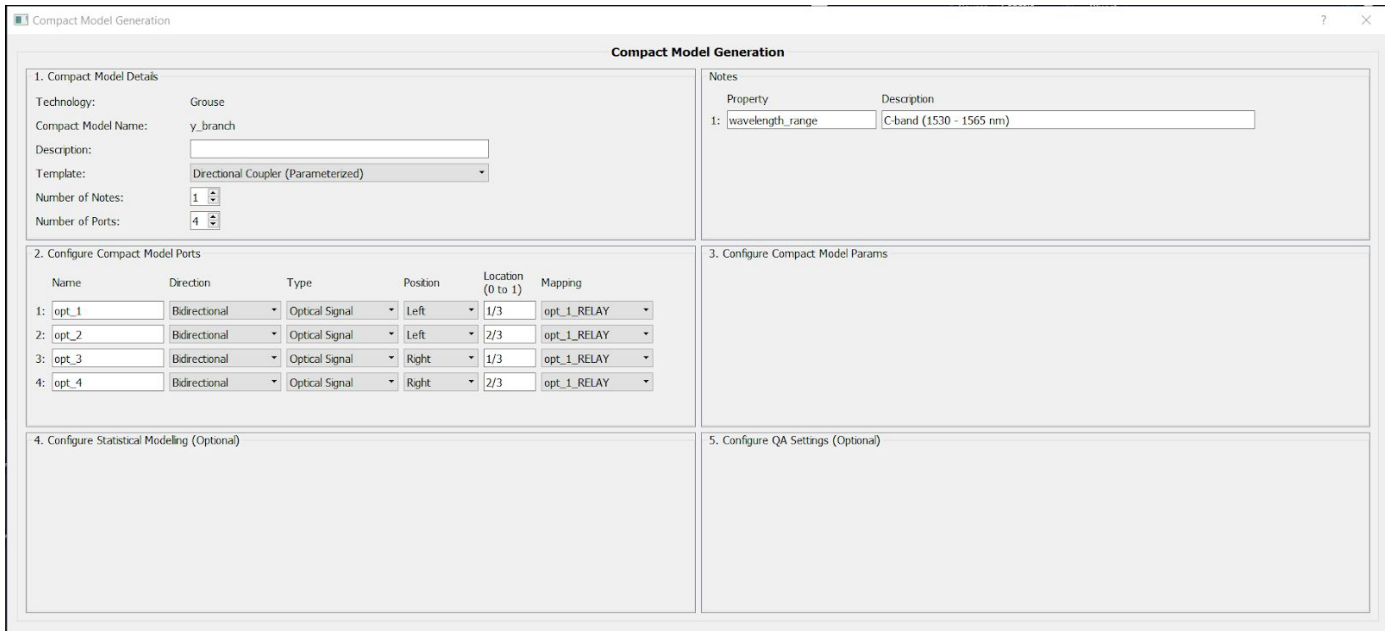
Code Structure

libgen.py: Main script that contains “generate_library()” which encapsulates LibGen.
common_methods.py: Common methods used in TechGen and LibGen (i.e. prettify XML string)
laystack.py: Layer properties class with layer properties params from Process YAML

CML Compiler Helper (Progress: 20%)
Purpose: Assists with generating compact models and integrating compact models into compact model libraries. CML Compiler and INTERCONNECT from Lumerical assist with generating compact models and .cml libraries.
Desired Functionality:



- ✓ Run CML Compiler in python to fit in python development workflow
- ✓ Generate compact model library file (.cml)
- ❑ Assist with compact model template selection
- ❑ Add ability to copy .lsf scripts and replace relevant fields with params specified in compact_model_window GUI shown below



- ❏ Add statistical modeling to compact model generation GUI
- ❏ Add QA settings to compact model generation GUI
- ❏ Add design params to compact model generation GUI

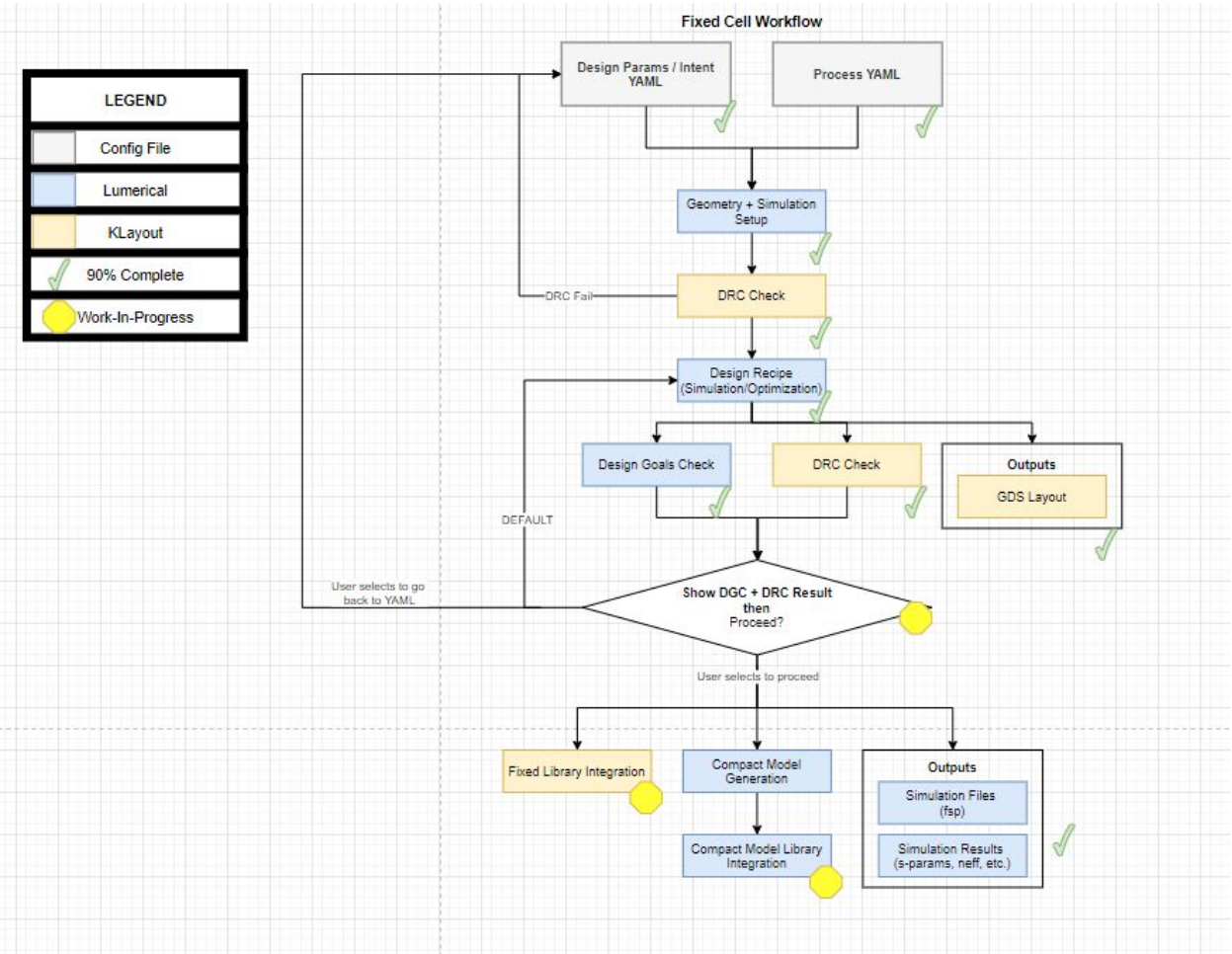
Code Structure

compact_model_window.py: A compact model generation GUI used to assist the designer in generating a compact model.

cml_compiler_helper.py: Runs CML Compiler in python workflow and generates compact model library files (.cml).

SIMULATIONS + OPTIMIZATIONS

Fixed Cell (Y-Branch) (Progress 80%)

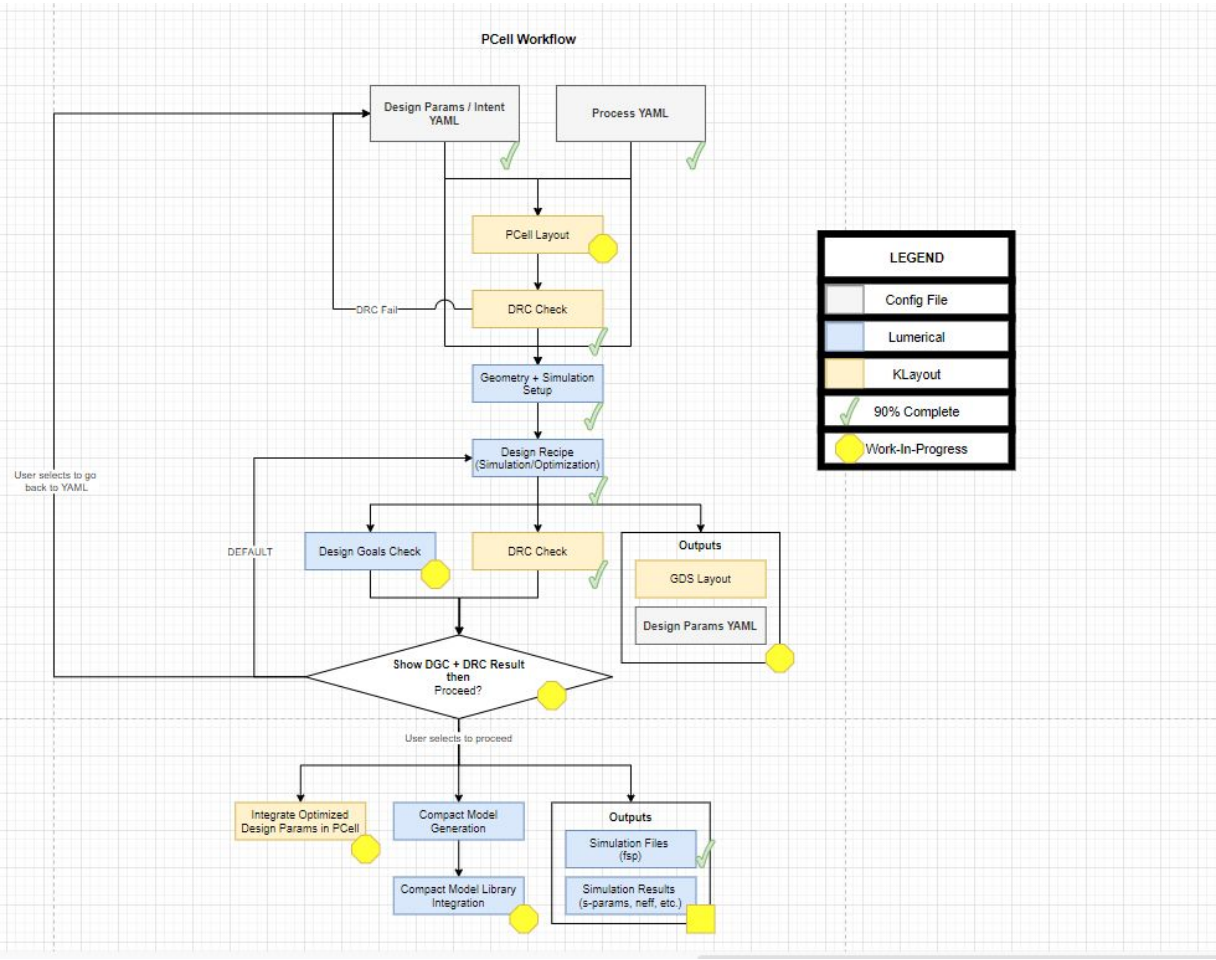


Purpose: To simulate and optimize the design of a Y-branch using the fixed cell design workflow shown above

Desired Functionality:

- ✓ Extract design intent parameters (ie: mode selection, waveguide physical dimensions, spectral ranges) from YAML to Lumerical
- ✓ Compile physical geometry of waveguides and run simulations/optimizations of Y-branch on Lumerical FDTD
- ✓ Perform DRC checks after geometry setup and after final GDS structure is designed
- ✓ Perform design requirement checks after optimizations are completed
- ✓ Generating a finalized GDS layout, Lumerical .fsp file and KLayout .gds file of optimized design
- ✓ Checking for design performance in Lumerical INTERCONNECT
- ❏ Generating GDS layouts for every iteration of Lumerical optimizations
- ❏ Integrating the S-bend physical architecture in the final GDS layout
- ❏ Developing an optional user input so that the user can choose to exit optimizations/simulations or perform a resimulation/reoptimization without YAML modification
- ✓ Developing waveguide physical parameter unit tests for meeting optimization requirements
- ❏ Integrate compact model generation
- ❏ Integrating corner/Monte Carlo Analysis

<u>Parameterized Cell (PSR) (Progress 60-70%)</u>



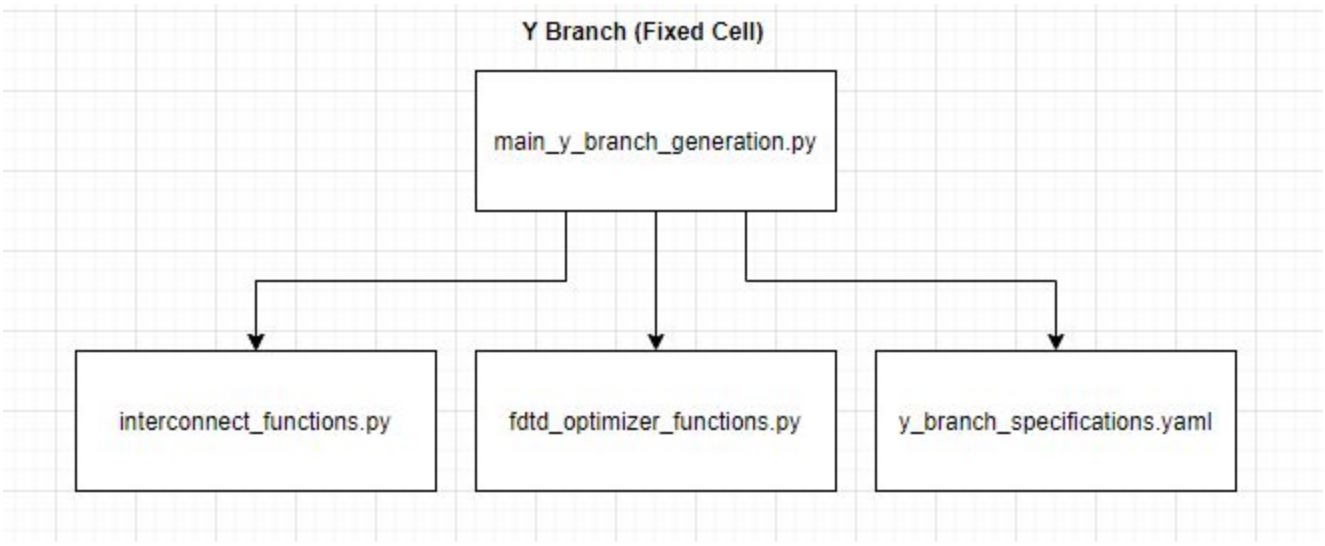
Purpose: To simulate and optimize the design of a PSR using the parameterized cell design workflow shown above

Desired Functionality:

- ❑ Extract design intent parameters (ie: physical dimensions, period, fill factor) from YAML file to KLayout PSR PCell
- ✓ Compile physical geometry of bitaper and adiabatic coupler and running simulations/optimizations of widths and lengths for optimal mode transmission in Lumerical MODE
- ❑ Perform DRC checks after KLayout setup and after final GDS structure is designed
- ❑ Perform design requirement checks after optimizations are completed
- ✓ Generating a finalized GDS layout, Lumerical .fsp file and KLayout .gds file of optimized design
- ❑ Developing an optional user input so that the user can choose to exit optimizations/simulations or perform a resimulation/reoptimization without YAML modification
- ❑ Generate optimized design params YAML
- ❑ Integrate compact model generation
- ❑ Integrating corner/Monte Carlo Analysis
- ✓ PCell Fixes
    - ✓ Combine Hossam's PSR PCell script into PDK-Generator and Grouse
    - ✓ Extending slab taper into silicon waveguide and then extend input/output waveguides
    - ✓ Adding straight waveguide chunk in between taper and coupler
    - ✓ Add high res layer for slab to strip transition
    - ✓ Fix SWG waveguide

Code Structure

Below is the code structure that currently reflects the functionality of both fixed cell and parameterized cell development as of September 2nd, 2020.

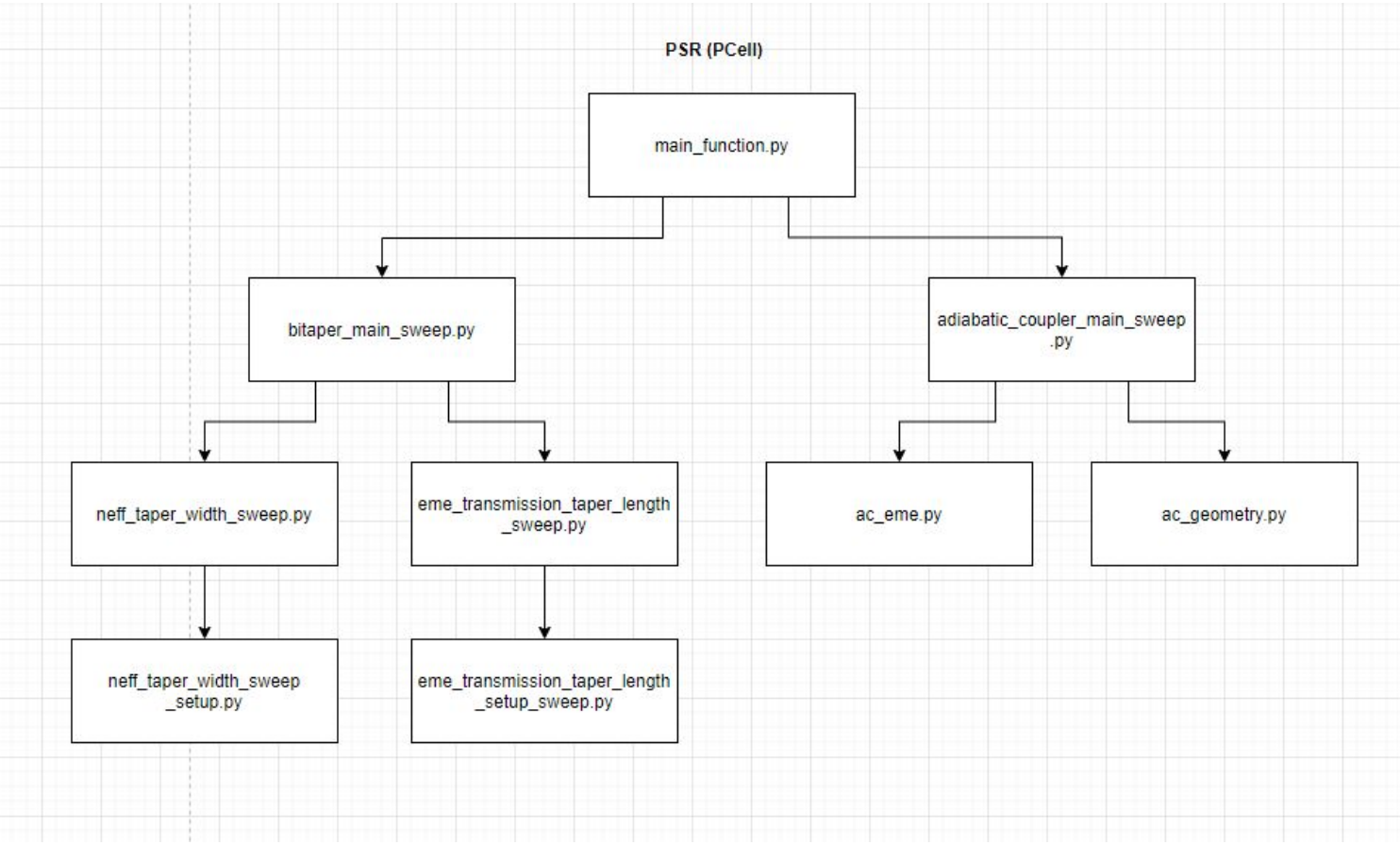


Main_y_branch_generation.py: Main function that generates, simulates and optimizes Y branch design based on design intent YAML file

Interconnect_functions.py: Function library that consists of functions needed to simulate final Y branch design on Lumerical INTERCONNECT

Fdtd_optimizer_functions.py: Function library that consists of functions needed to simulate and optimize potential Y branch designs on Lumerical FDTD

Y_branch_specification.yaml: YAML file that consists of Y branch design intent parameters such as spectral range, mode selection and waveguide physical parameters



Main_function.py: Instantiates the PSR bitaper sweeps and adiabatic coupler sweeps
Bitaper_main_sweep.py: Function that calls the PSR bitaper width and length sweeps
Adiabatic_coupler_main_sweep.py: Function that calls the PSR adiabatic coupler width and length sweeps
Ac_eme.py: Function library that configures the EME settings for the adiabatic coupler sweeps
Ac_geometry.py: Function library that configures the physical geometries and materials for the adiabatic coupler sweeps
Neff_taper_width_sweep.py: Function that commences the PSR bitaper width sweep through interpreting a self generated Effective index vs waveguide width graph and selects the optimal widths for each segment of the bitaper
Eme_transmission_taper_length_sweep.py: Function that commences the PSR bitaper length sweep through running the Lumerical EME solver and selects optimal lengths for each section of the bitaper based on maximum modal transmission
Neff_taper_width_sweep_setup.py: Function library that configures the physical geometries and eigensolver settings of the waveguides needed to generate the Effective index vs waveguide width graph in neff_taper_width_sweep.py
Eme_transmission_taper_length_sweep_setup.py: Function library that configures the physical geometries and eigensolver settings of the bitaper needed to commence the PSR bitaper length sweep

Quick Start Guide

Technology Generation

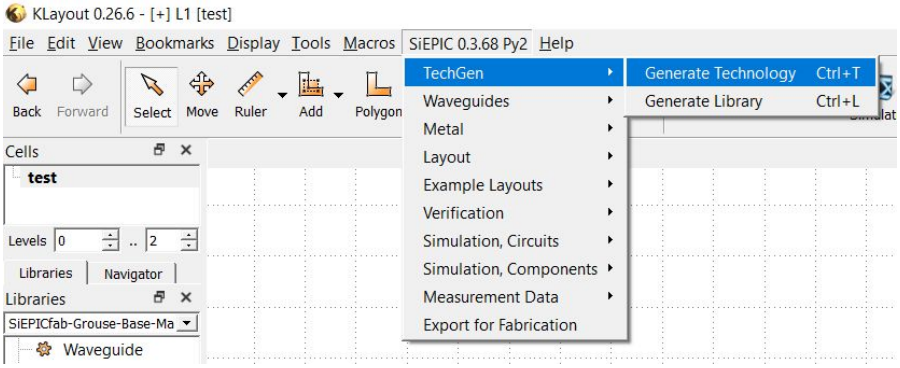
Requirements

- KLayout
- PDK-Generator Repo (Installation instructions can be found on repo)
- SiEPIC-Tools v0.3.69

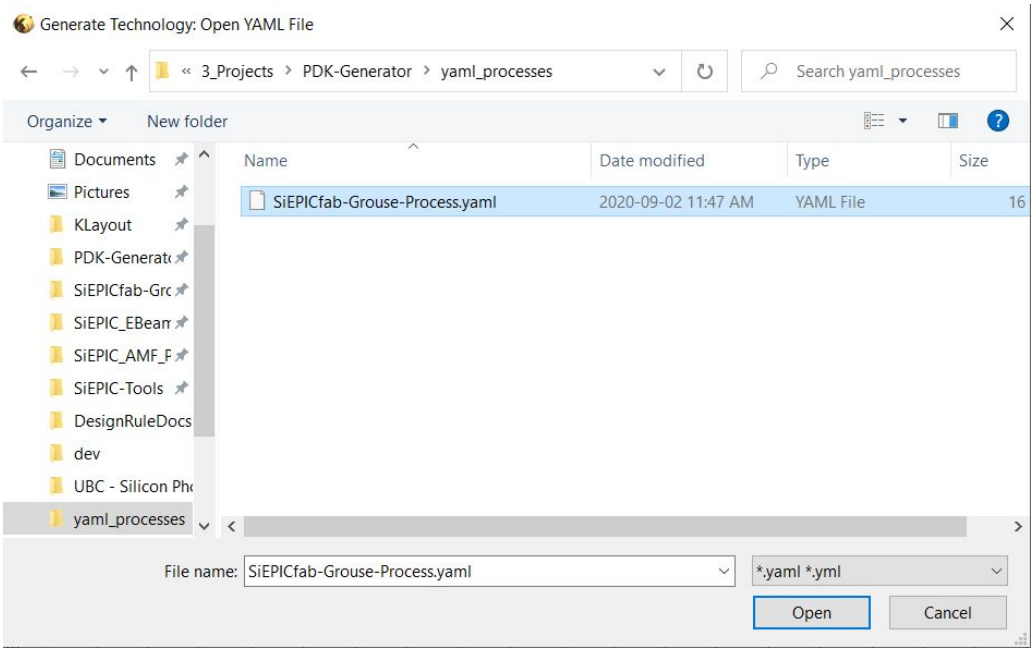
Usage

To generate a PDK and a Library:

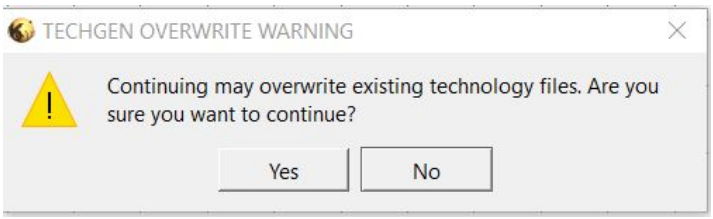
1. Navigate to the SiEPIC menu --> TechGen



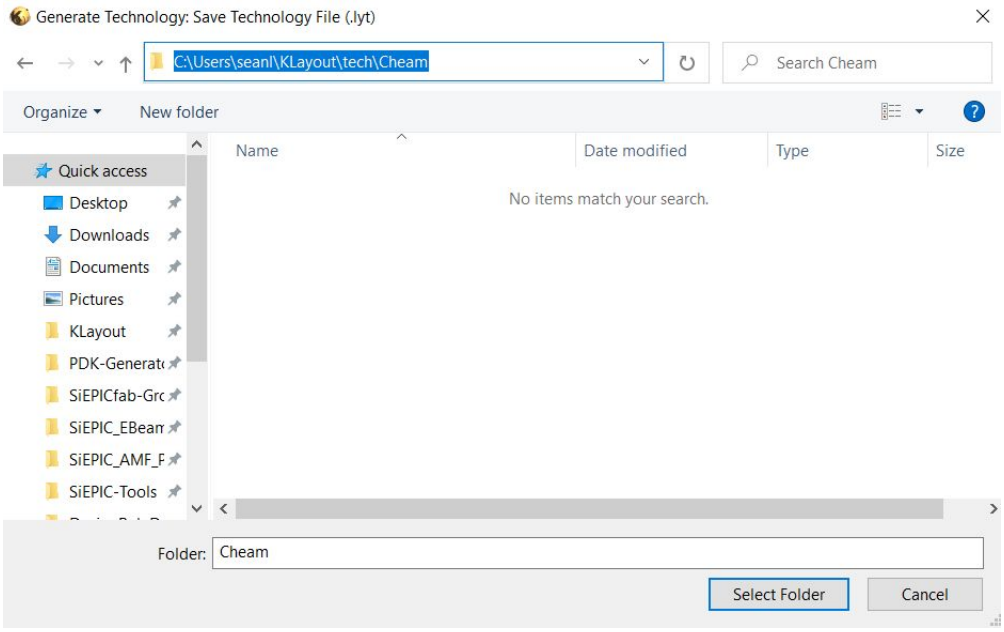
2. After clicking on “Generate Technology”, a window pops up prompting you to select the Process YAML (Process YAMLs can be found in the PDK-Generator repo in the yaml_processes folder). Navigate to the Process YAML and hit “Open”



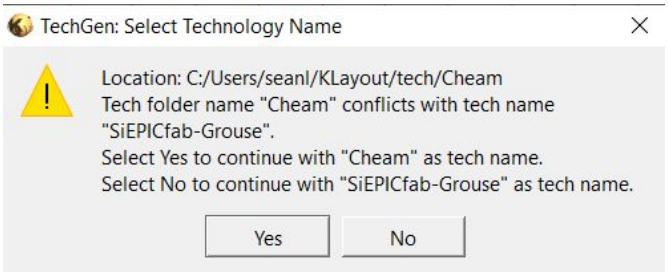
3. An overwrite warning pops up, hit “Yes” to continue



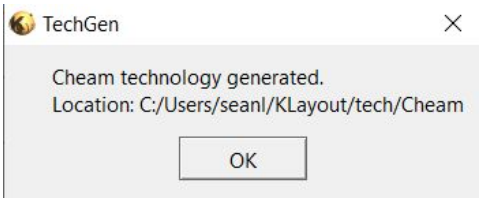
4. Choose a folder where the base PDK will live. Usually, when generating tech files, select a folder in the KLayout --> tech --> <techname>. The following shows an example of generating in a tech called “Cheam”



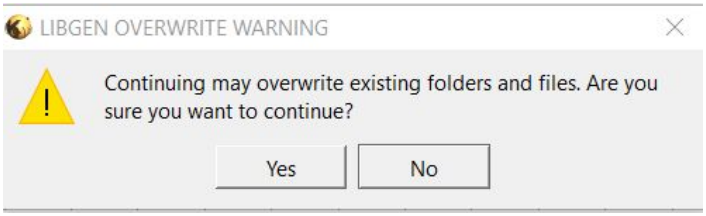
5. Depending on the specified tech name from the YAML file, you may need to choose which tech name to use. **This will affect the folder name and tech file names.** The following chooses “Cheam”.



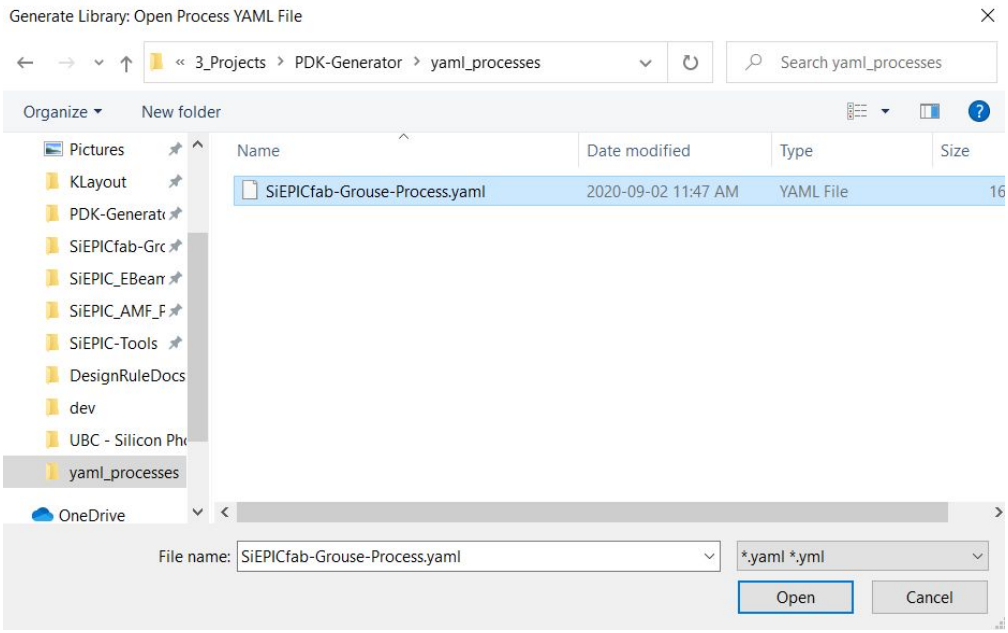
6. After, technology files will be generated and the location will be shown:



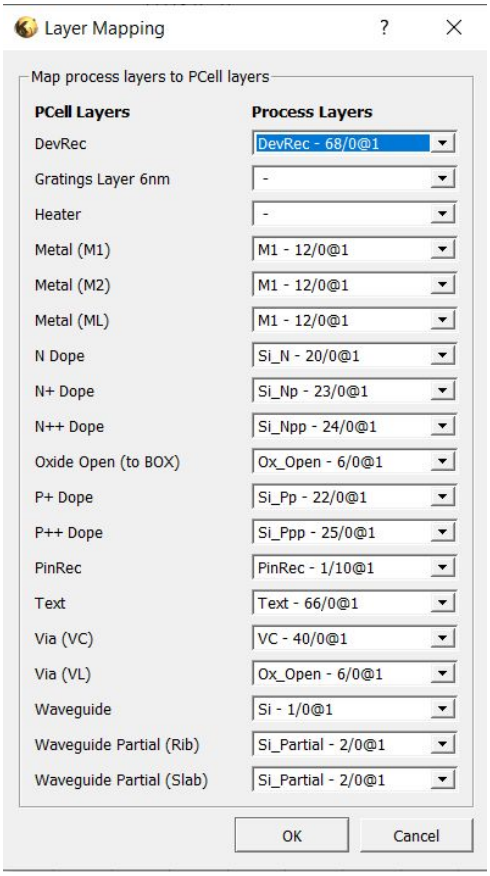
7. You also have the option of generating a library for the generated technology. Clicking “No” will exit from LIBGEN. Let’s assume we want to generate a library, so we click “Yes”.



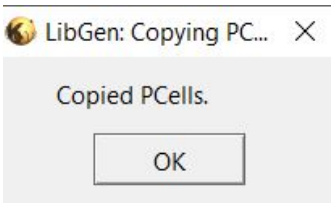
8. You must select the Process YAML again (Process YAMLs can be found in the PDK-Generator repo in the yaml_processes folder)



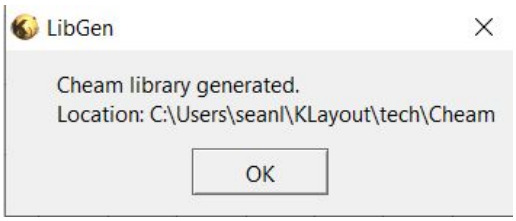
9. A layer mapping window pops up where you must map PCell layers to Process layers. If all layers in the PCell script are supported by the layers defined in the Process, then the PCell script will be copied to the generated library.



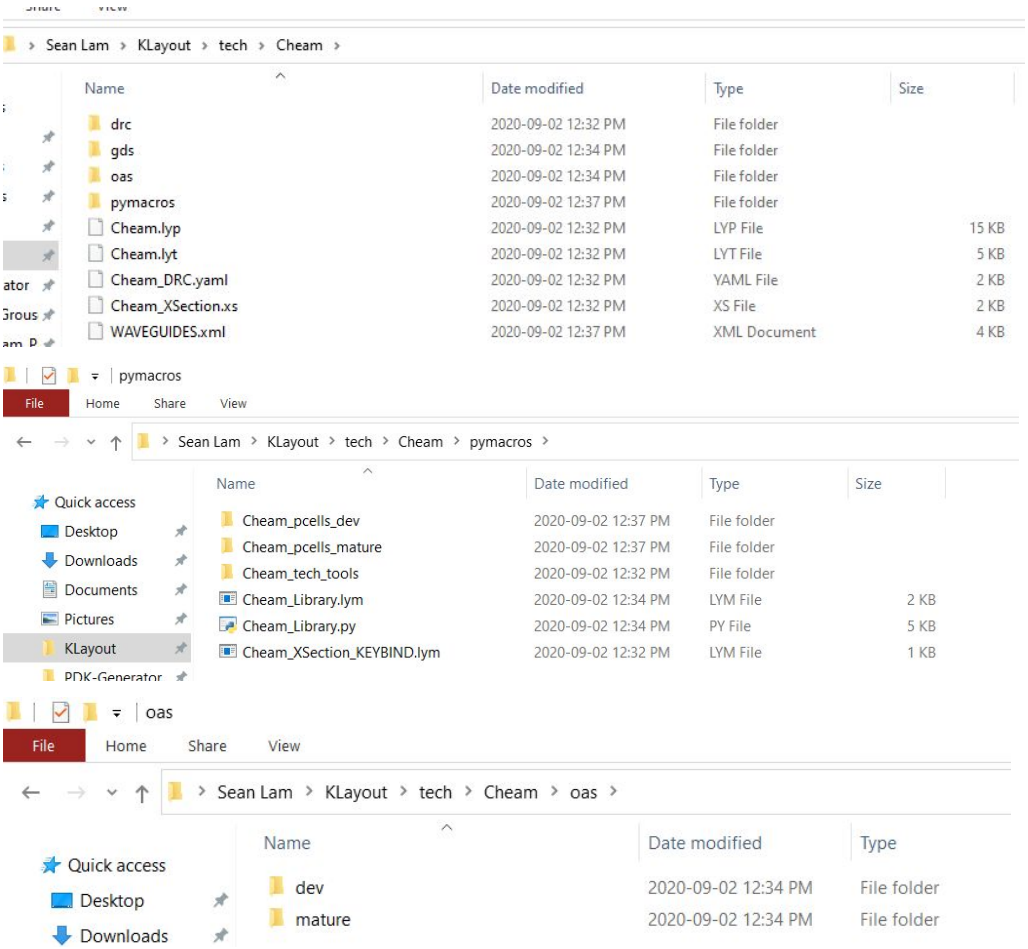
10. Done copying PCells

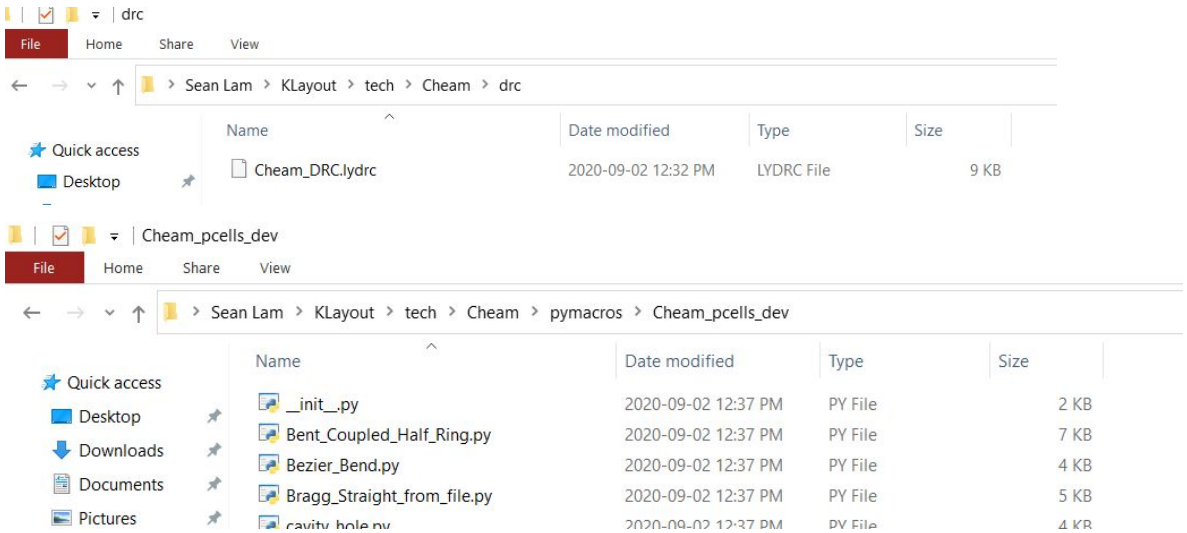


11. In addition to copying PCells, library instantiation scripts are generated and the folder structure is created to support mature and dev PCells and fixed cells.

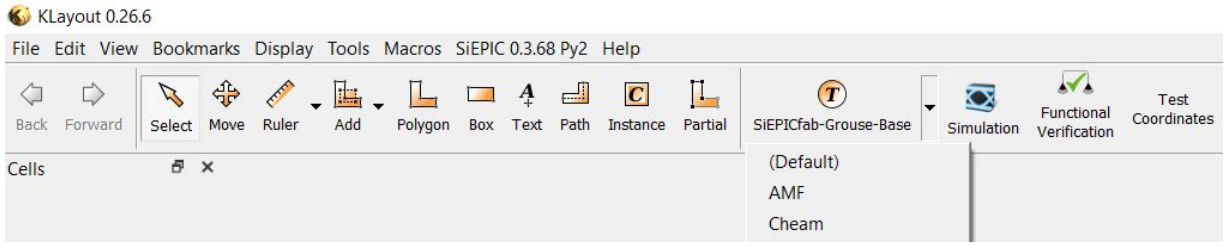


12. This is what the folder structure looks like:



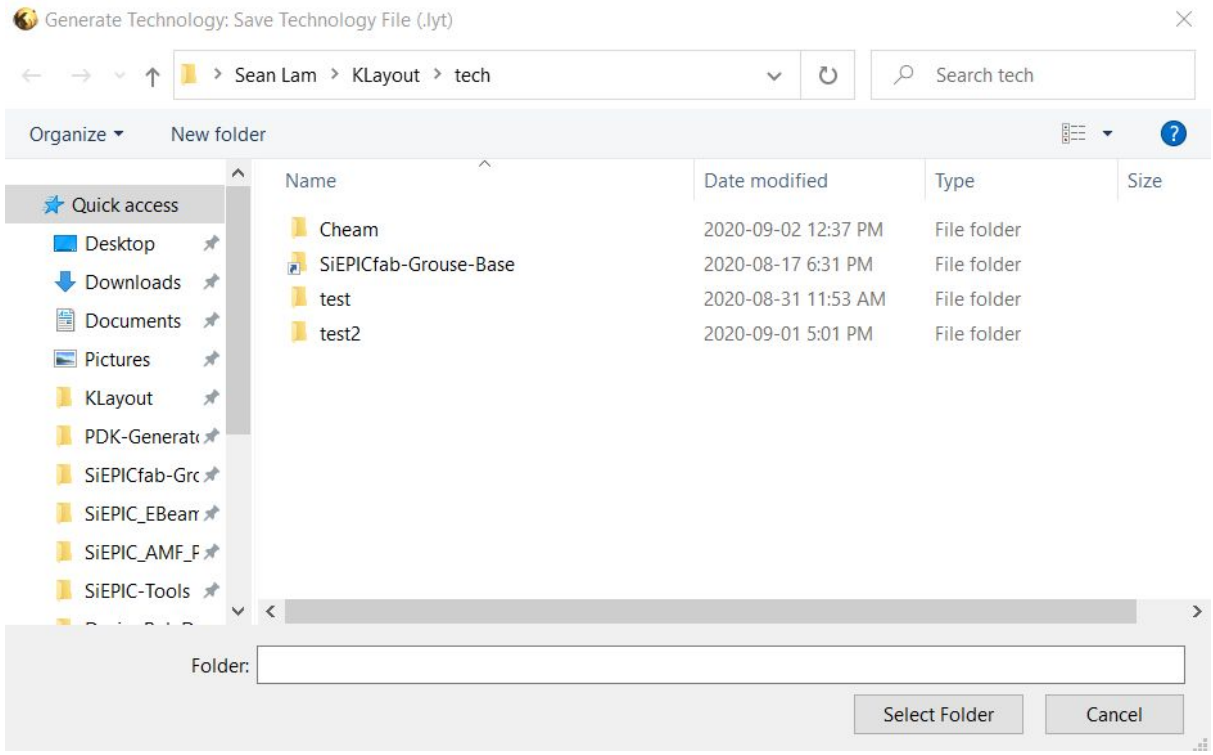


13. Close KLayout then reopen KLayout so that the new technology is registered and can be used:

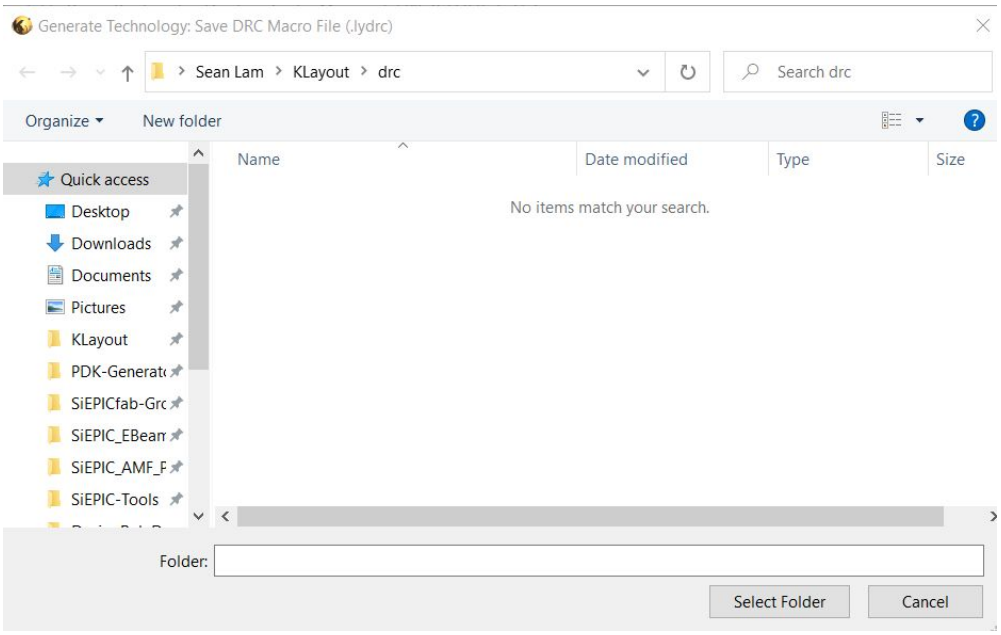


If you want to only generate a single file or a few files, you can do so by:

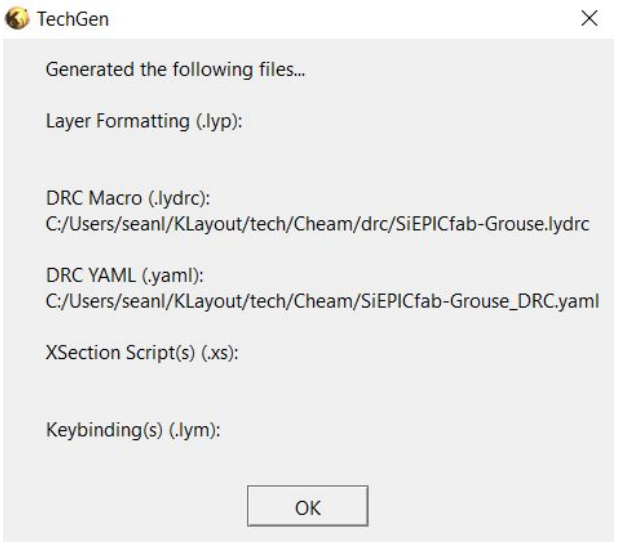
1. Following steps 1 to 3 from above (up to the point where you have to select a folder to save the Technology .lyt file). Then, hit “Cancel”:



2. Depending on the file, you want to save. Click on “Select Folder” or “Save”.



3. By clicking “Cancel” through all of the prompts, no technology/file is generated. Otherwise, if a file was generated, the file location would be shown in the completion dialog.



Library Generation

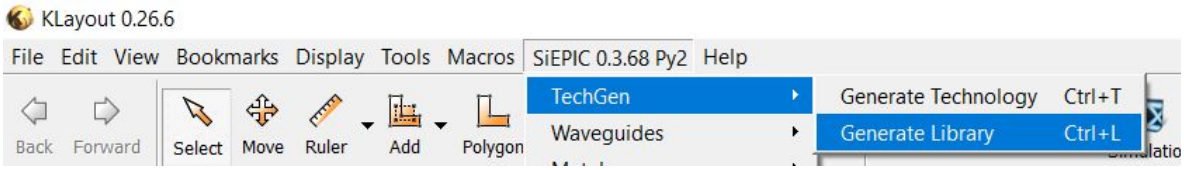
Requirements

- KLayout
- PDK-Generator Repo (Installation instructions can be found on repo)
- SiEPIC-Tools v0.3.69

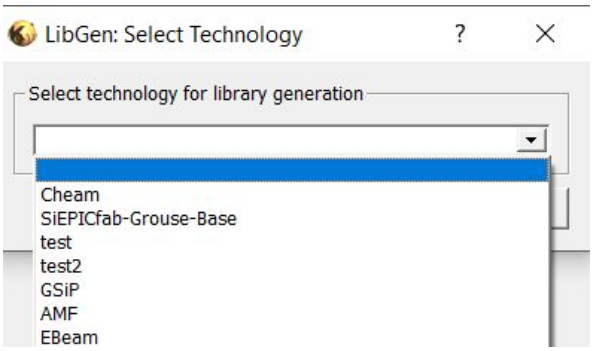
Usage

To generate a library for a given technology:

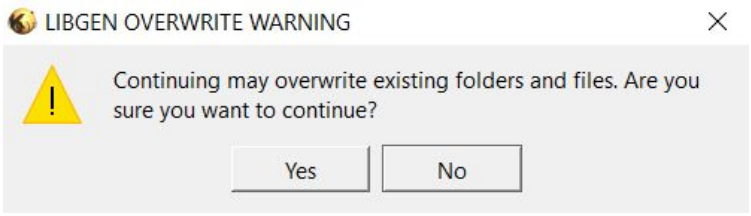
1. Navigate to the SiEPIC menu and hit “Generate Library”



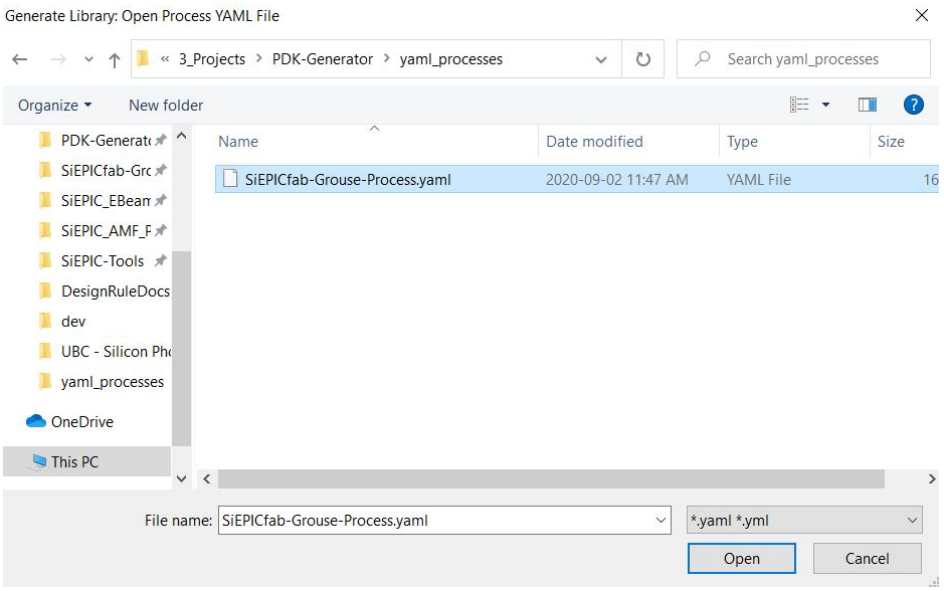
2. Select the technology you want to generate the library for



3. An overwrite warning will show up. Click “Yes” to continue

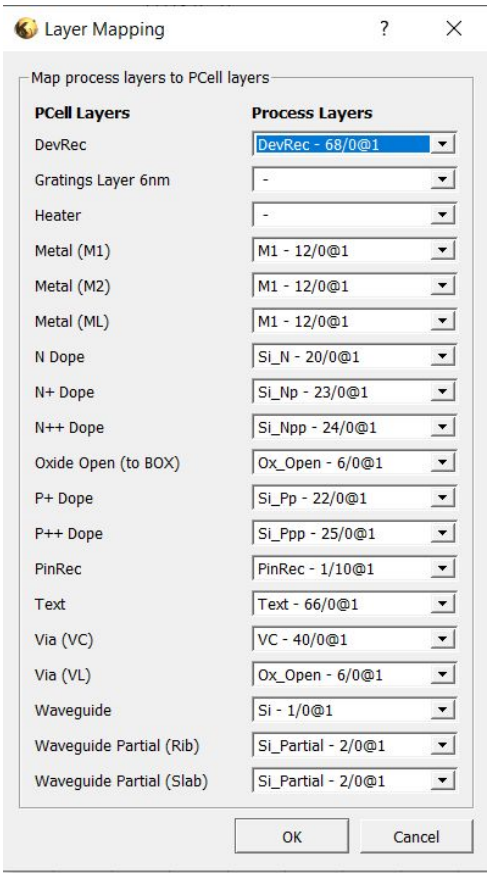


4. A window will pop up for you to find the Process YAML:

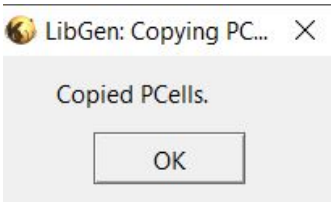


5. A layer mapping window pops up where you must map PCell layers to Process layers. If all layers in the PCell script are supported by the layers defined in the Process, then the PCell script will be copied to the generated

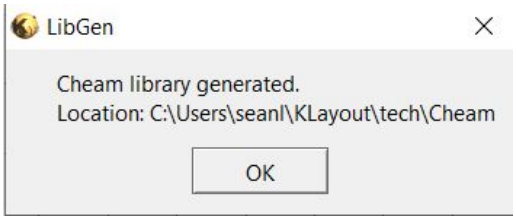
library.



6. Done copying PCells



7. In addition to copying PCells, library instantiation scripts are generated and the folder structure is created to support mature and dev PCells and fixed cells.



Fixed Cell Simulation/Optimization: Y-Branch Inverse Design Process

Requirements:

- All code from inverse_design_y_branch located in the PDK generator repository
- Spyder IDE configured to run main_y_branch_generation.py in an external terminal
    - Top Toolbar -> Run -> Run configuration per file ->Console -> Execute in an external system terminal

Usage:

The following demonstrates the usage of the Y branch code:

1. In the y_branch_specfications.yaml, the following design intent settings can be configured:
    a. Waveguide physical parameters
    b. Layers
    c. Mode selection
    d. Spectral Range and Interpolation points
    e. Maximum optimization iterations

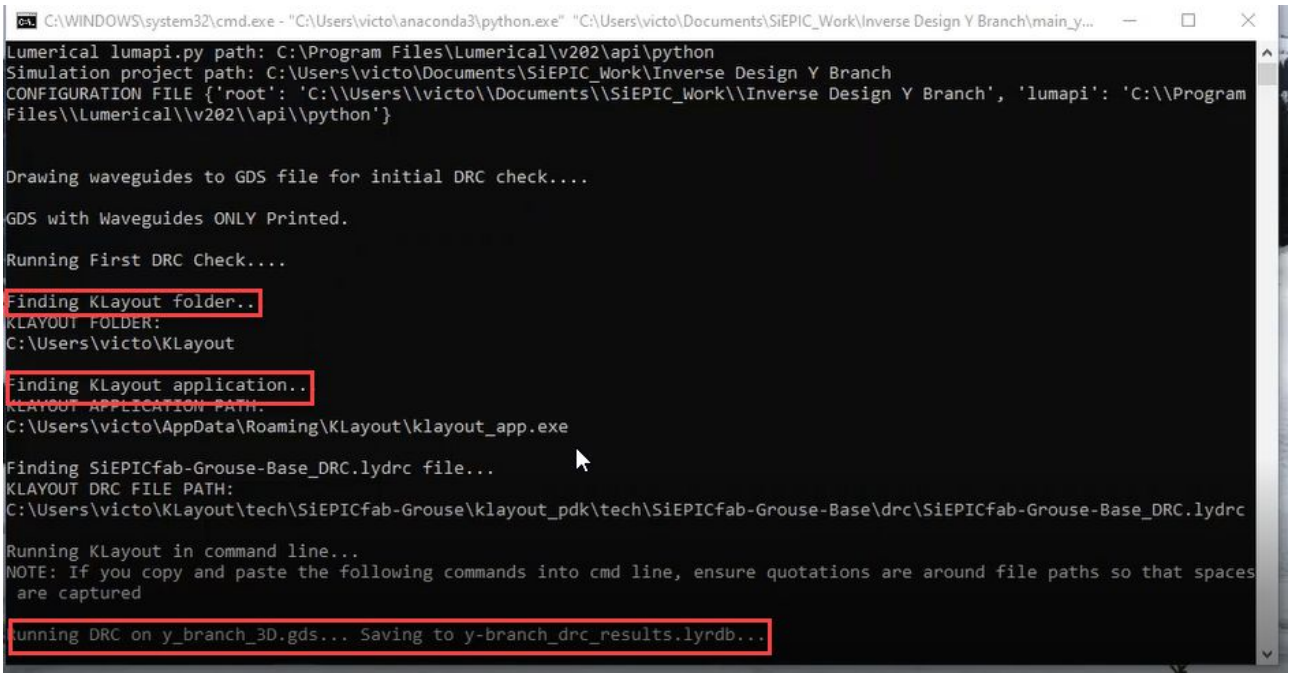
```yaml
# Hypothetical Waveguide Parameters for Y Branch Generation
---
  #Optimization Requirements
  optimization_reqs:

    #Maximum gain through single port (dB)
    - gain_through_single_port: -3.15

 #Component Parameters
  component:
      #Input Waveguide
    - name: "input_wg"
      x_span: 3.0e-6 #waveguide length
      y_span: 0.5e-6 #waveguide width
      z_span: 220.0e-9 #waveguide height
      x: -2.5e-6
      y: 0
      z: 0
      material: "Si: non-dispersive"

      #Output Waveguide (top)
    - name: "output_wg_top"
      x_span: 3.0e-6 #waveguide length
      y_span: 0.5e-6 #waveguide width
      z_span: 220.0e-9 #waveguide height
      x: 2.5e-6
      y: 0.35e-6
      z: 0
      material: "Si: non-dispersive"
```

```yaml
#Layers
#NOTE: Will not be used in the fab-GROUSE run,
#extracting from YAML instead
layers_used:

  - material_type: "Dielectric"
    name: "Si: non-dispersive"
    material_name: "Si (Silicon) - Palik"

  - material_type: "Dielectric"
    name: "SiO2: non-dispersive"
    material_name: "SiO2 (Glass) - Palik"

#Optimization Parameters
optimization_variables:

  #Wavelength scale
  - start: 1300.0e-9
    stop: 1800.0e-9
    points: 21

  #Optimizer
    max_iter: 10 #Maximum Iterations of Optimizations to Run
    method: "L-BFGS-B" #Quasi Newton Method
    pgtol: 1.0e-5 #Gradient tolerance parameter
    ftol: 1.0e-5 #Stops optimizations when changes in FOM are less than this

  #Mode source
    direction: "Forward"
    injection_axis: "x-axis"
    center_wavelength: 1.550e-6
    mode_selection: "fundamental TE mode"
```

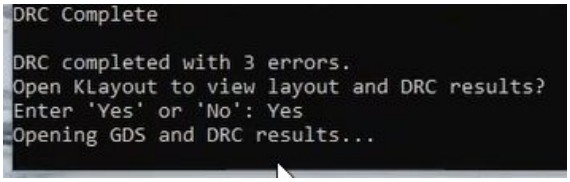
2. After configuring the settings in the YAML file, run main_y_branch_generation.py and the following command window should appear

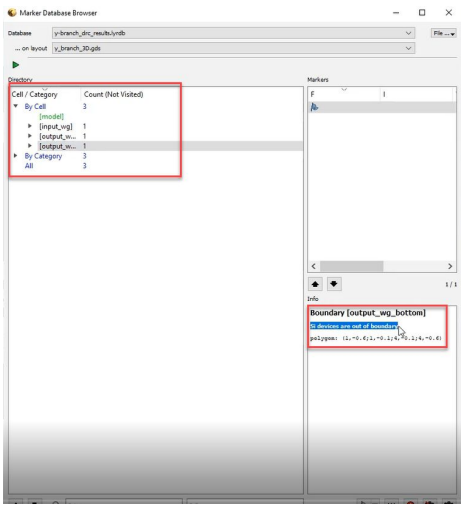
```
C:\WINDOWS\system32\cmd.exe - "C:\Users\victo\anaconda3\python.exe" "C:\Users\victo\Documents\SiEPIC_Work\Inverse Design Y Branch\main_y...
Lumerical lumapi.py path: C:\Program Files\Lumerical\v202\api\python
Simulation project path: C:\Users\victo\Documents\SiEPIC_Work\Inverse Design Y Branch
CONFIGURATION FILE {'root': 'C:\\Users\\victo\\Documents\\SiEPIC_Work\\Inverse Design Y Branch', 'lumapi': 'C:\\Program
Files\\Lumerical\\v202\\api\\python'}
```

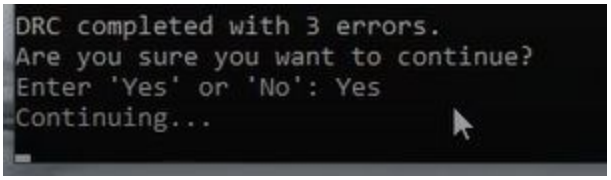
3. The script will now generate a GDS file based on the parameters specified in the YAML file and attempt to open KLayout to run a DRC check.



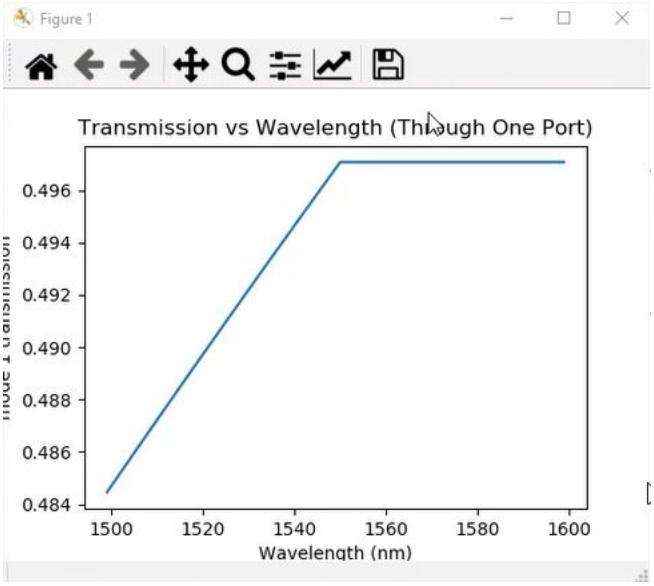
4. Once the first DRC check is completed, the script will prompt the user to open KLayout to view/fix/errors. After exiting KLayout, the user will be prompted to continue the process. Entering ‘Yes’ will continue to simulations/optimizations. Entering ‘No’ will exit out of the program.



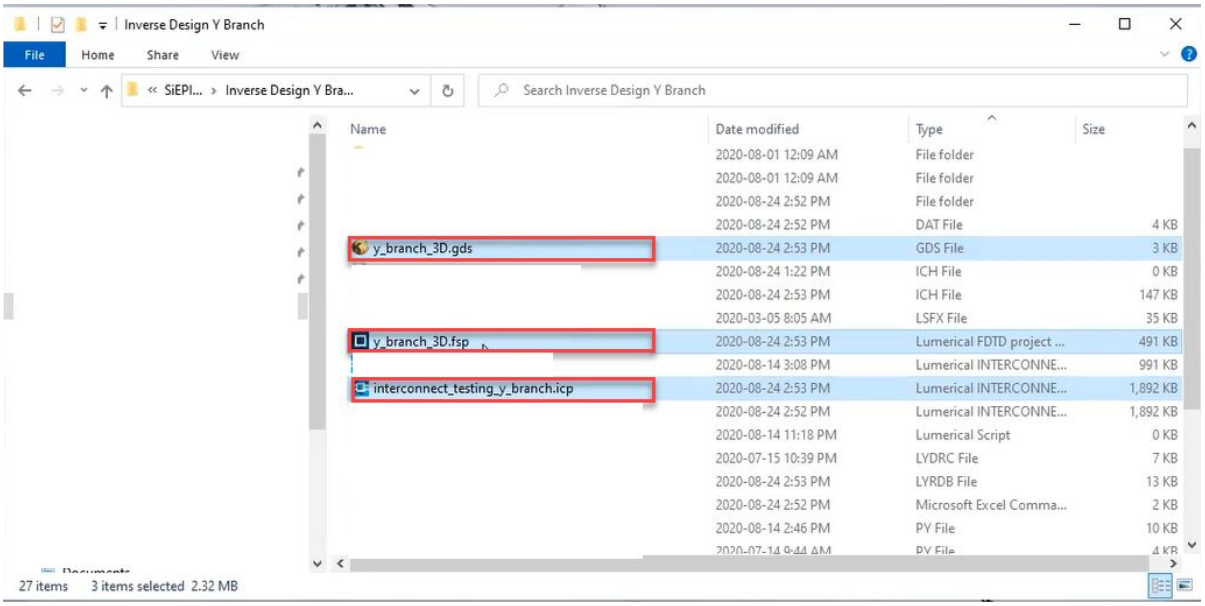




5. Optimizations will then be performed in FDTD. The final optimized Y branch design will be simulated in INTERCONNECT and there will be a design requirement check (for losses/transmission). If the check passes, then a graph will be displayed showing the optimized results.



6. A GDS check will be performed again once the graph is closed for the optimized Y branch design. Again, the script will prompt the user to open KLayout to view/fix errors
7. Once exiting KLayout and continuing the process by entering ‘Yes’ once more, the process will be completed and 3 files will be generated in the users local library:
    a. A Lumerical FSP file with simulated/optimized design
    b. A KLayout GDS file of the simulated and optimized design
    c. An INTERCONNECT file for testing



Parameterized Cell Simulation/Optimization: PSR Design Process

To be added later in the week

Definitions

Keyword	Definition	Notes
Process YAML	Configuration file for a foundry's process, including design rules, layer stack, etc.	
Design Parameter	A mostly physical parameter used for designing a photonic component (i.e. width = 60 nm, fill factor = 50%)	
Design Intent	A specific technical requirement that a photonic component must accomplish (ie: transmission, gain, FSR (free spectral range)	
Figure of Merit (FOM)	The ultimate design goal that distinguishes a design from other designs or from unoptimized designs (i.e. insertion loss)	
FDTD	Full fledged Maxwell's equations solver. Used to characterize a photonic component with **zero** approximations.	FDTD is not the same as MODE or EME
MODE	Eigenmode solver used to solve for modes of a waveguide.	
EME	Uses frequency domain solver to solve Maxwell's equations	
INTERCONNECT	Photonic circuit simulation software.	
GDS	File format for layout	
OAS	File format for layout	
CML Compiler	Lumerical software used to generate compact models from data (i.e. s-params)	