



SIEMENS EDA

Calibre® Directed Self-Assembly User's and Reference Manual

Software Version 2021.2

Unpublished work. © 2021 Siemens

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software, Inc., its subsidiaries or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with Siemens. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This document is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made. Siemens disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' **End User License Agreement** may be viewed at: www.plm.automation.siemens.com/global/en/legal/online-terms/index.html.

No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

TRADEMARKS: The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux[®] is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: support.sw.siemens.com

Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Table of Contents

Chapter 1

Calibre Directed Self-Assembly Overview	9
Syntax Conventions	10
Calibre DSA Requirements	11
Calibre DSA Workflow	12
About Directed Self-Assembly	13

Chapter 2

DSA Model Creation and Verification	15
Creating an Average Contour	15
Creating a DSA Lithomodel	17
Loading DSA Data in nmModelflow	19
Adding a DSA Stage in nmModelflow	22
Creating a DSA Calibration Job	25

Chapter 3

Calibre DSA Reference	31
Model File Formats for DSA	32
CDSA Cylinder Model File Format	33
CDSA Lamella Model File Format	35
SVRF Commands for DSA	37
LITHO DSA DENSEOPC	38
LITHO DSA OPCVERIFY	40
TVF Commands for DSA	41
cdsa::DSA_GROUP	42
Litho Setup File Commands for DSA	49
cdsa_model_load	50
dsa_options	51
modelpath	55
setlayer cdsa_simulator	56
setlayer cdsa_synthesizer	57

Glossary

Index

Third-Party Information

List of Figures

Figure 1-1. DSA Components	10
Figure 1-2. Calibre Post-Tapeout Flow With DSA	12
Figure 1-3. Periodicity and Phase Transitions	13
Figure 2-1. Creating an Average Contour	15
Figure 2-2. Removing Outliers From Contour Sets	16
Figure 3-1. Lamella Model Measurement Region	36
Figure 3-2. Forbidden Geometries to Litho and Grouping Violations	44
Figure 3-3. Forbidden Geometries Due to Collinearity Violation	45
Figure 3-4. Forbidden and Correct Geometries	46
Figure 3-5. Optimizing the Guiding Pattern With the Perpendicular Option	54
Figure 3-6. Simulator Exception Cause	57

List of Tables

Table 1-1. Syntax Conventions	10
Table 3-1. SVRF Commands	37
Table 3-2. TVF Commands	41
Table 3-3. Litho Setup File Commands	49
Table 3-4. Valid Commands for dsa_options	51

Chapter 1

Calibre Directed Self-Assembly Overview

Directed self-assembly (DSA) is a semiconductor manufacturing technique in which chemicals added to a guiding pattern react in a way that creates contacts (vias) or lines. This creates structures smaller than can be manufactured through traditional deposition and etch processes. Calibre® DSA software aids in creating models for DSA processes and allows the DSA models to be used in other Calibre products.

Calibre DSA comprises three parts:

- **DSA Modeling** — The Compact Directed Self-Assembly (CDSA) model gives users the ability to simulate graphoepitaxy-guided cylindrical self-assembled structures for full chips. The simulations allow users to decompose, group, synthesize, and verify DSA processes for via and contact layers.

The model minimizes the potential energy of a copolymer system to determine the best placement of the DSA contact inside a template. It can also identify when templates would lead to defective structures.

Models are created and calibrated using Calibre® ContourCal through the Tcl mdm commands in the Calibre® WORKbench™ shell interface. You do not need a separate license to create calibrated DSA models.

- **DSA Synthesis** — The Calibre DSA Mask Synthesis product reads the DSA models and your proposed design and returns guiding pattern templates.

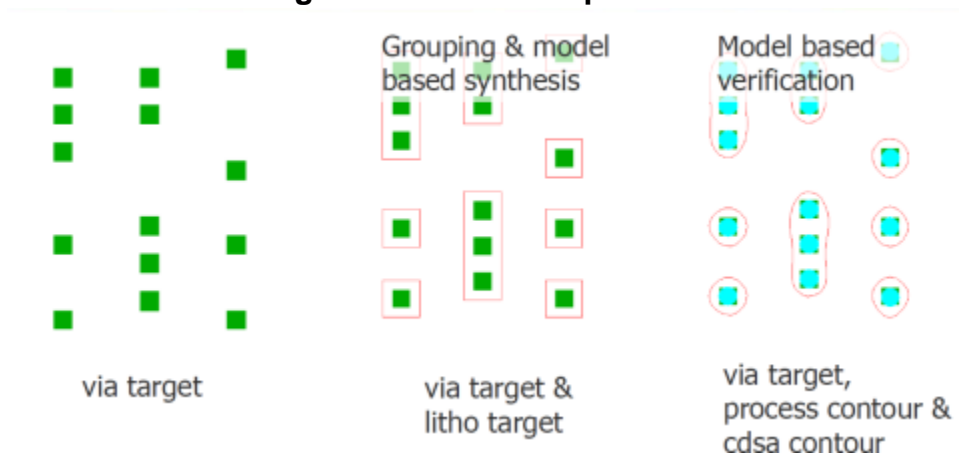
DSA synthesis runs as part of a Calibre® nmOPC™ flow. It is included in an SVRF file through the command LITHO DSA DENSEOPC.

- **DSA Verification** — The Calibre DSA Verification product runs after OPC and litho verification. It simulates where cylinders are likely to form based on the DSA models and the simulated contours of the post-OPC guiding patterns.

Because the manufacturing process always includes some variation, verification is an important step to allow you to identify potential hotspots before the masks are created.

DSA verification runs as part of a Calibre® OPCverify™ flow. It is included in an SVRF file through the command LITHO DSA OPCVERIFY.

Figure 1-1. DSA Components



Syntax Conventions	10
Calibre DSA Requirements	11
Calibre DSA Workflow	12
About Directed Self-Assembly	13

Syntax Conventions

The command descriptions use font properties and several metacharacters to document the command syntax.

Table 1-1. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	Underlining indicates either the default argument or the default value of an argument.
UPPerCase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.

Table 1-1. Syntax Conventions (cont.)

Convention	Description
' '	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
or	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
...	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.
Example: DEVICE { <i>element_name</i> ['(' <i>model_name</i> ')']} <i>device_layer</i> { <i>pin_layer</i> ['(' <i>pin_name</i> ')'] ...} ['<' <i>auxiliary_layer</i> '>' ...] ['(' <i>swap_list</i> ')'] ...] [BY NET] BY SHAPE]	

Calibre DSA Requirements

To run Calibre DSA you need a correctly configured environment.

Licensing

Calibre DSA requires separate licenses for creating models, synthesizing guiding patterns, and verifying output.

- Modeling requires Calibre® nmModelflow™.
- Synthesis requires Calibre nmOPC and a DSA Mask Synthesis license.
- Verification requires Calibre OPCverify and a DSA Verification license.

Refer to the [Calibre Administrator's Guide](#) for more information on licensing these products.

At this time, Calibre DSA cannot be used with the separately licensed Calibre® FullScale™ platform. Calibre DSA runs should be performed with the Calibre hierarchical engine.

Environment Variables

As with all Calibre products, you must have CALIBRE_HOME set to the path of the Calibre software tree.

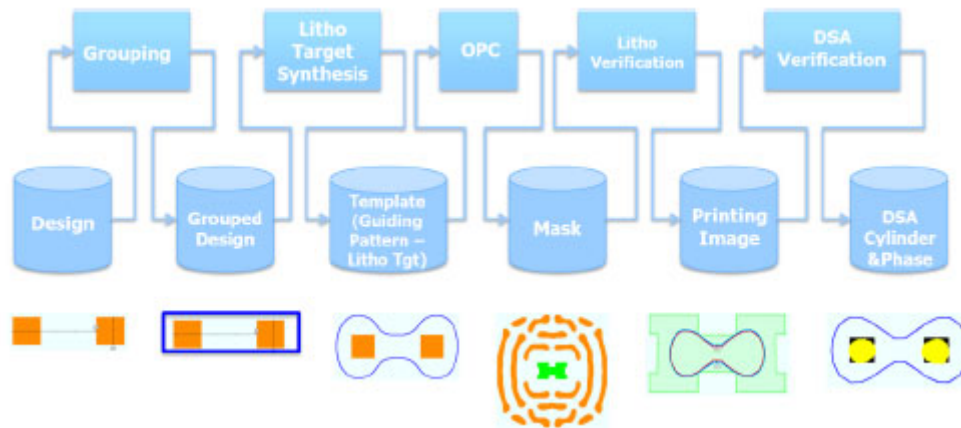
Process Technology

At this time, the Calibre DSA tools only support graphoepitaxy DSA and contact layers. All contacts must be square and of a fixed size.

Calibre DSA Workflow

The Calibre DSA products fit into a typical Calibre post-tapeout flow.

Figure 1-2. Calibre Post-Tapeout Flow With DSA



1. The foundry provides a Compact DSA (CDSA) model (not shown in the figure) as part of its process development kit (PDK).
2. The layout engineers include the CDSA model when developing blocks. They can use the DSA Synthesis product to perform grouping and avoid placing contacts in such a way that they are prone to manufacturing defects.
3. After the chip is laid out but before performing OPC, the contact layer is processed with the DSA grouping utility. The groups are then run through DSA synthesis to create guiding pattern shapes.
4. The layer containing the guiding pattern shapes undergoes OPC using Calibre nmOPC along with the rest of the chip design. To enable additional simulation for DSA, use the SVRF command “LITHO DSA DENSEOPC”. The resulting mask layer undergoes normal verification with Calibre OPCverify.
5. After verifying the mask layer will create the intended guiding patterns even with typical process variation, the guiding pattern contours are simulated to verify they will create the intended contacts. This stage is done with the SVRF command “LITHO DSA OPCVERIFY”.

About Directed Self-Assembly

Directed self-assembly (DSA) is a semiconductor manufacturing technique that integrates self-assembling materials with traditional manufacturing processes. It makes use of a copolymer. The copolymer's component polymers are typically referred to as “block A” and “block B”. The most common ones used in semiconductor manufacturing are polystyrene (PS) and polymethylmethacrylate (PMMA).

Depending on the ratio of the length of the blocks, the copolymer might assemble into lines or cylinders. With the appropriate guidance, the placement can be controlled and used to create fins or contacts, for example.

There are two different methods to guide (or direct) the assembly:

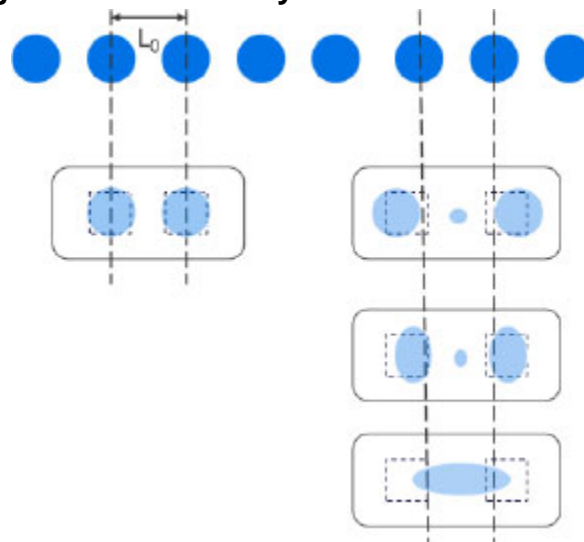
- **chemoepitaxy** — Chemicals that repulse one of the blocks align the copolymer.
- **graphoepitaxy** — Trenches of a particular size contain the copolymer. The shape is such that it encourages the formation of aligned structures.

The Calibre DSA tools are designed for graphoepitaxy processes.

DSA processing techniques are typically used for sub-14nm technology nodes. It complements multi-patterning, reducing the total number of masks. It is particularly useful for contacts and vias.

Block copolymers exhibit a natural periodicity, referred to in the literature as L_0 . When the guiding shape is not close to an integer multiple of L_0 , the blocks do not form a regular pattern. This is referred to as a phase transition. Phase transition conditions are metastable and may result in distorted structures, missing or extra cylinders, or clumping. [Figure 1-3](#) shows a good guiding pattern on the left and the possible results of a poorly sized guiding pattern on the right.

Figure 1-3. Periodicity and Phase Transitions



Chapter 2

DSA Model Creation and Verification

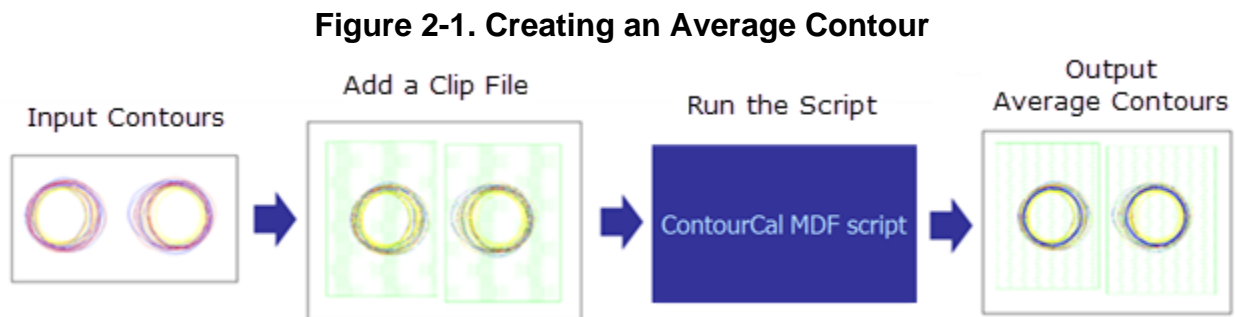
Accurate DSA models can be created through proper calibration and verification. Proper calibration requires contour sets. Averaged contour sets can be created from initial contour data sets. These contour sets provide better calibration results.

Creating an Average Contour	15
Creating a DSA Lithomodel	17
Loading DSA Data in nmModelflow	19
Adding a DSA Stage in nmModelflow	22
Creating a DSA Calibration Job	25

Creating an Average Contour

Model calibration uses sets of contours, which should have outliers removed. A ContourCal MDF script can output an average contour using input contours and a clip layer.

Figure 2-1 summarizes the procedure.



Prerequisites

- A layout with two or more contours for each structure that will be used in model calibration
- At least one layer containing polygons that each enclose a single contour (the clip layer)

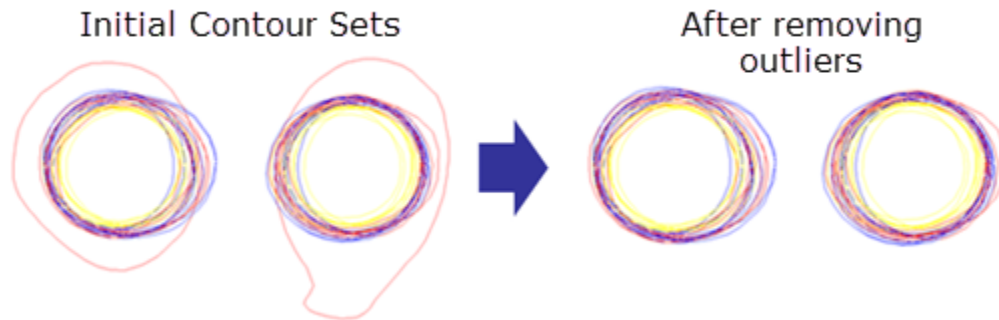
Procedure

1. Open the layout file in Calibre WORKbench. Remove outlier contours.

In Figure 2-2, the two contour sets on the left contain outliers. The two contour sets on the right more closely represent ideal contours and are thus more useful for creating an average contour. If the outlier contour forms a single line, right-click the contour and

select **Delete**. This may not be the case for imported SEM images, in which several segments must be selected and deleted.

Figure 2-2. Removing Outliers From Contour Sets




2. Create a Contour Layer Info (CLI) file.

A CLI file describes a clip layer contained in a design file, and can be used to identify the layer information and averaging options. An example CLI file is shown:

```
version 1
contourSetCount 1
contourLog 0.0 100000
contourCostMethod dense
contourAverage on 1 // Turns on averaging
contourSet 1 {
  contourlayer 101 // Identifies layer with contours to average.
  contourlayer 102 // Can be one layer with multiple 'paths' or
                  // multiple contourlayer statements with polygon
                  // layers.
  cliplayer 2000
  dose 1.00
  defocus 0.00
  weight 1
  outstartlayer 2001 3
}
```

For more information, see “[Contour Layer Info \(CLI\) File Format](#)” in the *Calibre WORKbench User’s and Reference Manual*.

Note

 If the contours are paths, contours need to be on the same layer, and thus the CLI file will only contain one layer reference.

3. Create a Calibre nmModelflow script that loads the layout and CLI file, and runs a simulation. The script should also save the output layout (for example, an OASIS[®]¹ file). An example script is shown:

```
#Load the layout file.
mdf contour layout gds/contour.gds

#Load the CLI file with contour input data.
mdf contour cli cli/contour.cli -average_only

#Run the simulation.
mdf simulate

#Save the layout to a different file.
mdf contour output layout averaged_contour.oas oasis
```

4. Run the Calibre nmModelflow script from step 3. In the shell prompt, enter the following, using the name of your script:

```
calibrewb mdf_script
```

Results

The script outputs a layout file containing the average contour.

Creating a DSA Lithomodel

Litho models are a collection of models that can be referenced as a single item in order to simplify the syntax of Calibre[®] nmOPC[™] and Calibre OPCverify commands by specifying a single litho model instead of multiple models in commands. Litho models can be created in Calibre nmModelflow in order to specify the type of DSA model being used as well as the parameter values.

Restrictions and Limitations

- Litho models only support a single resist model per litho model definition.
- Optical models must use absolute paths for the source and Jones pupil files.

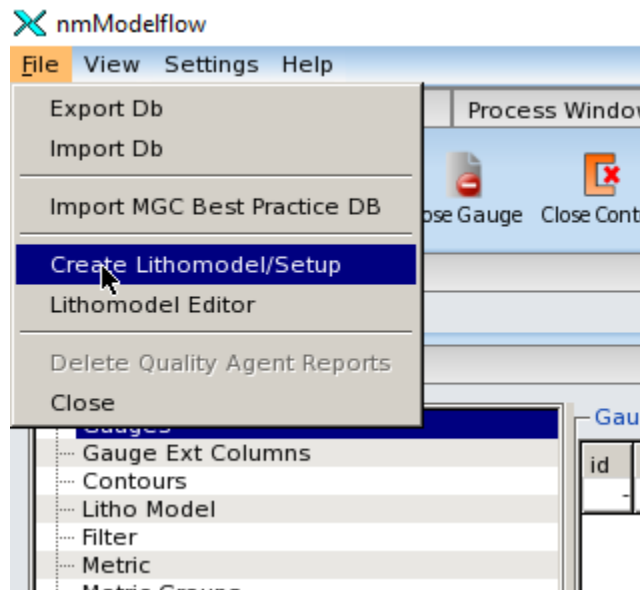
Prerequisites

- Any model components needed to describe the lithographic exposure.
- Calibre WORKbench invoked and Calibre nmModelflow activated.

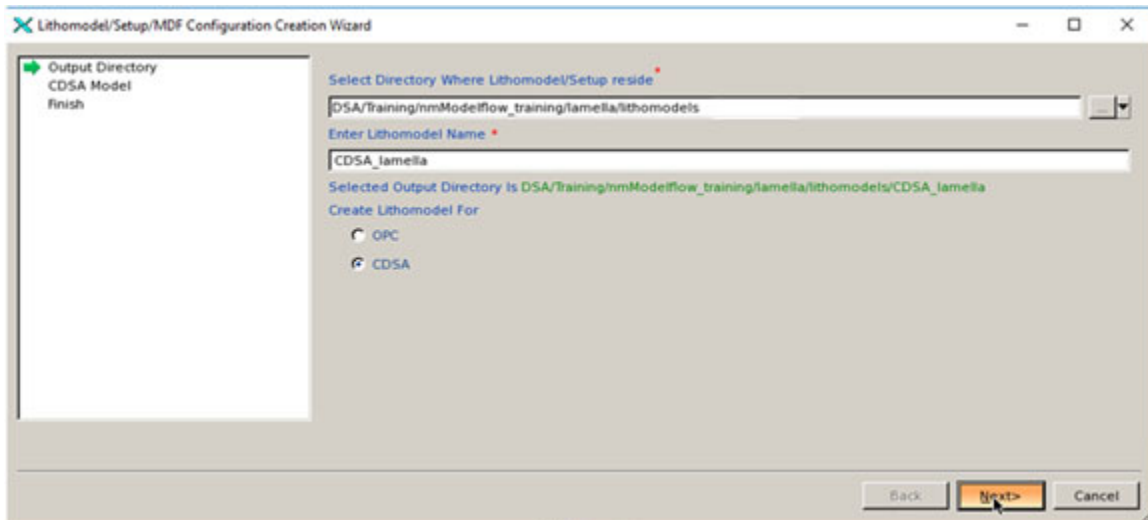
1. OASIS[®] is a registered trademark of Thomas Grebinski and licensed for use to SEMI[®], San Jose. SEMI[®] is a registered trademark of Semiconductor Equipment and Materials International.

Procedure

1. Click **File > Create Lithomodel/Setup** to open the Lithomodel/Setup/MDF Creation Wizard for Calibre nmModelflow. The wizard lists each step that requires input (Output Directory, Lithomodel Components, Exposure, Finish).

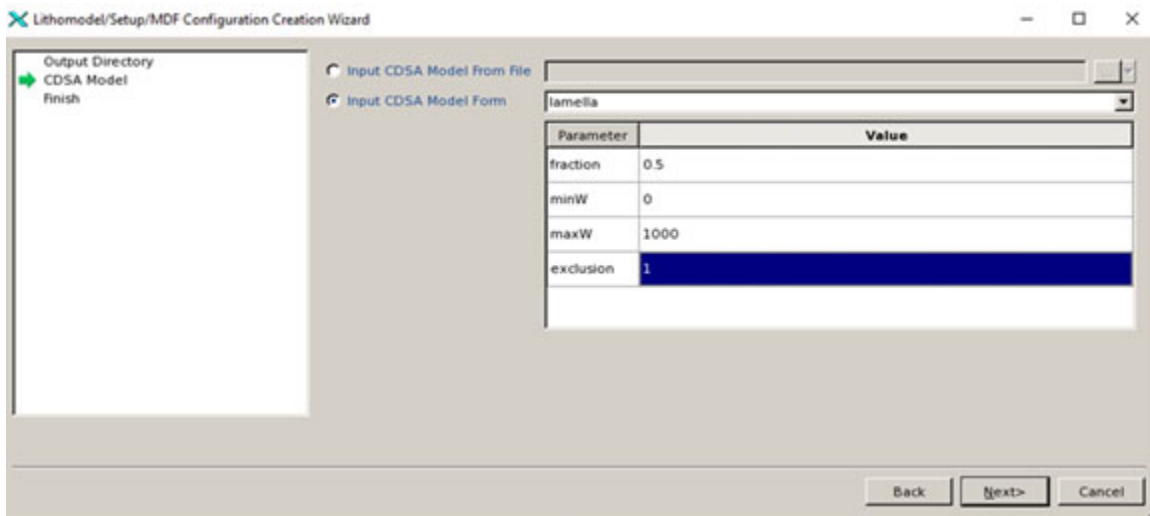


2. For the Output Directory screen, enter or navigate to an empty working directory to contain the completed litho model. (You can create a new directory from the file chooser dialog.) Select the CDSA radio button and click **Next**.

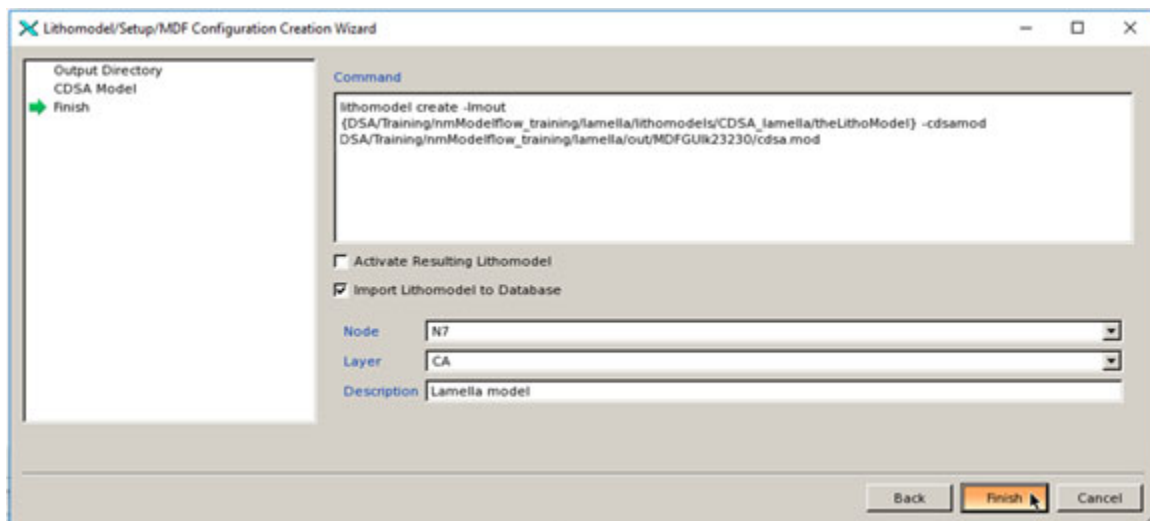


3. Select the “Input CDSA Model Form” radio button and select either ‘cylinder’ or ‘lamella’ for the CDSA model type. Enter the nominal values for the CDSA model.

Alternatively, select the “Input CDSA Model From File” radio button, and enter or navigate to the desired file. Click **Next**.



4. Select the desired Node and Layer from the respective drop down menus. Enter a description for the model. Click **Finish**.



Results

A litho model directory is created in the specified directory. Selecting the **Import Lithomodel to Database** check box also adds the completed *Lithomodel* file to the current database.

Loading DSA Data in nmModelflow

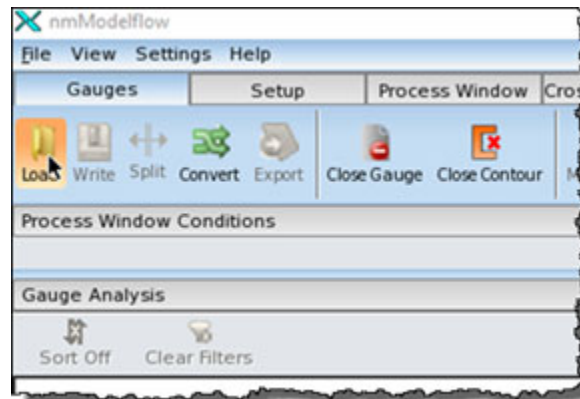
Contours can be loaded into nmModelflow for the corresponding litho model and layout. These components are needed in order to perform a calibration job for a DSA model.

Prerequisites

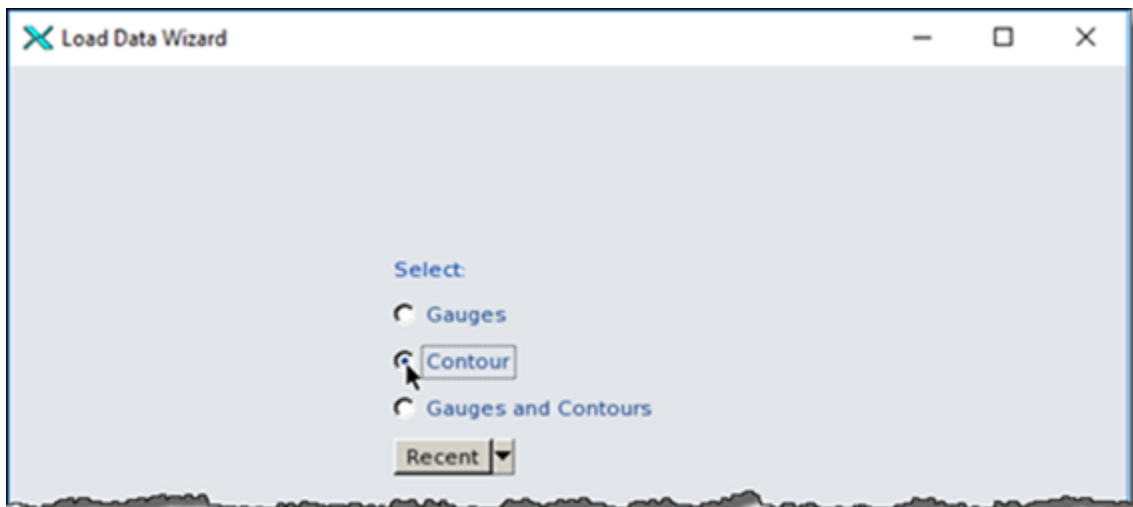
- A Contour (CLI) or CDSA setup file
- A Lithomodel file or a Calibre nmModelflow or Calibre OPCverify setup file
- The corresponding layout file

Procedure

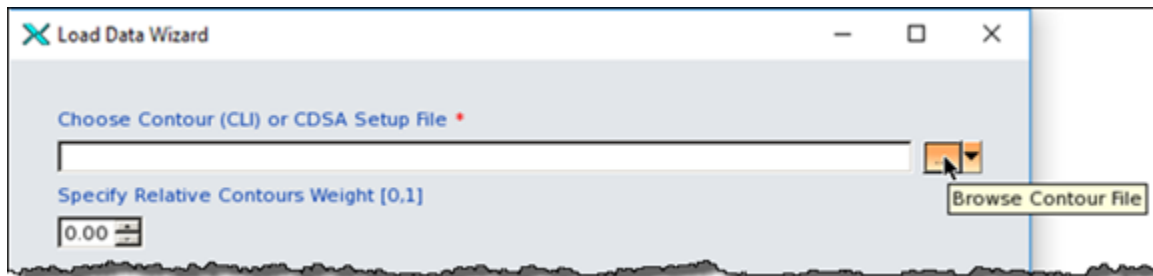
1. In the **Gauges** tab, click the **Load** button.



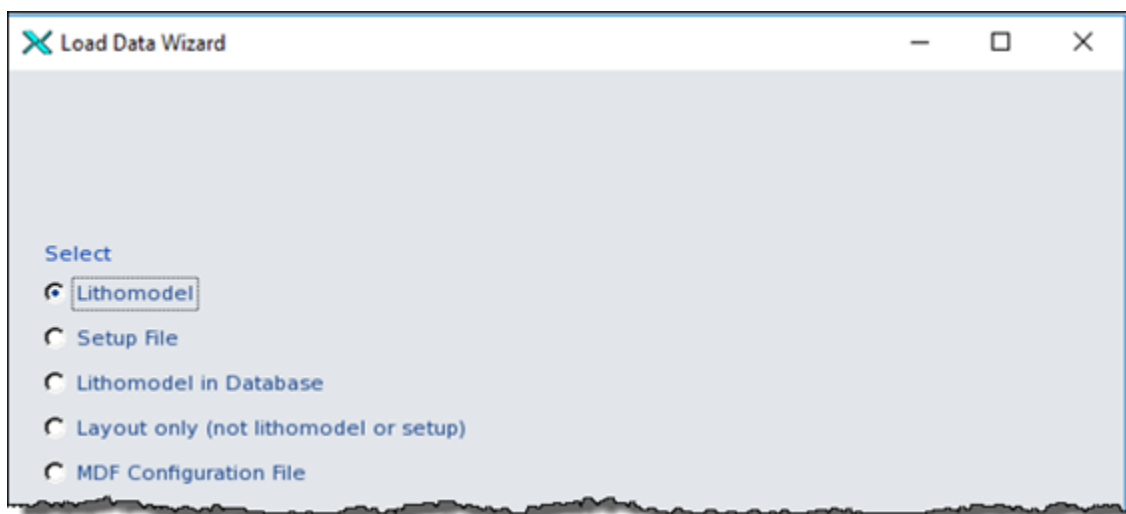
2. Select the Contour radio button (gauges are not supported for CDSA).



3. Enter the contour (CLI) or CDSA setup file.



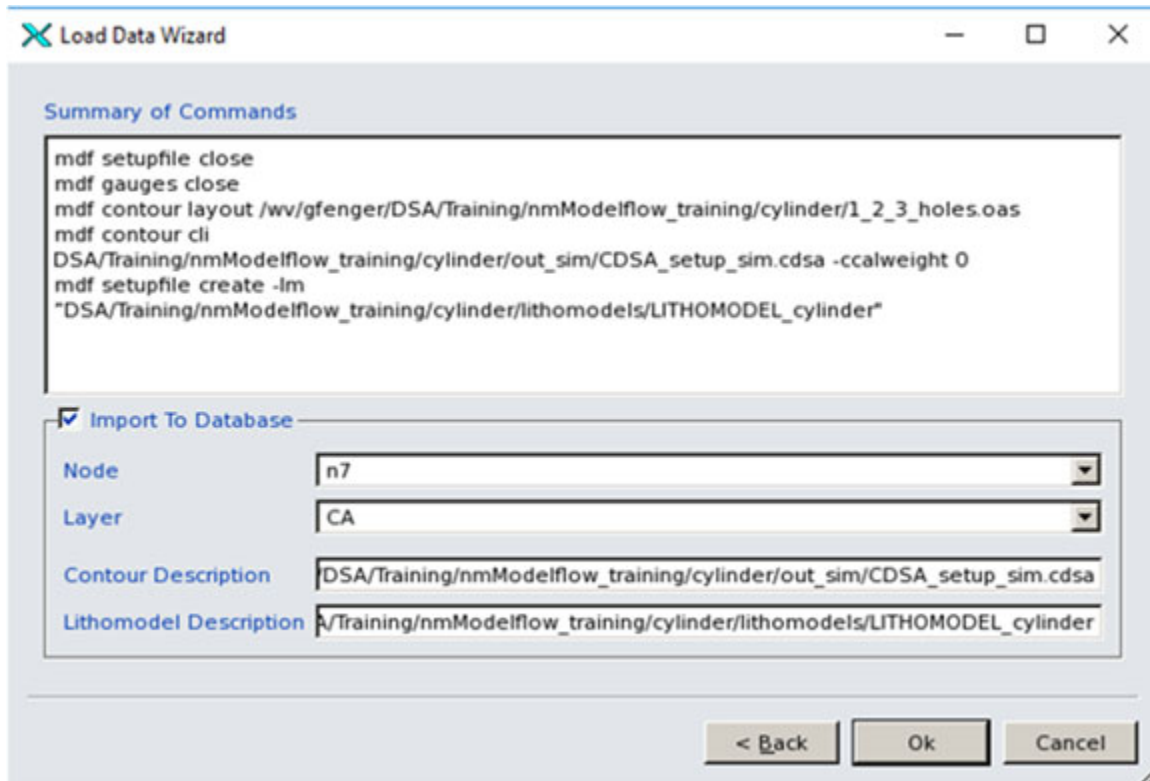
4. Select the Lithomodel or “Setup File” radio button and click **Next**.



5. Enter the corresponding layout file and click **Next**.



6. Select the “Import to Database” check box. Review the Summary of Commands for the setup and click **OK**.



Results

Calibre nmModelflow generates a setup file, runs the commands, and loads all data into the Calibre nmModelflow GUI.

Adding a DSA Stage in nmModelflow

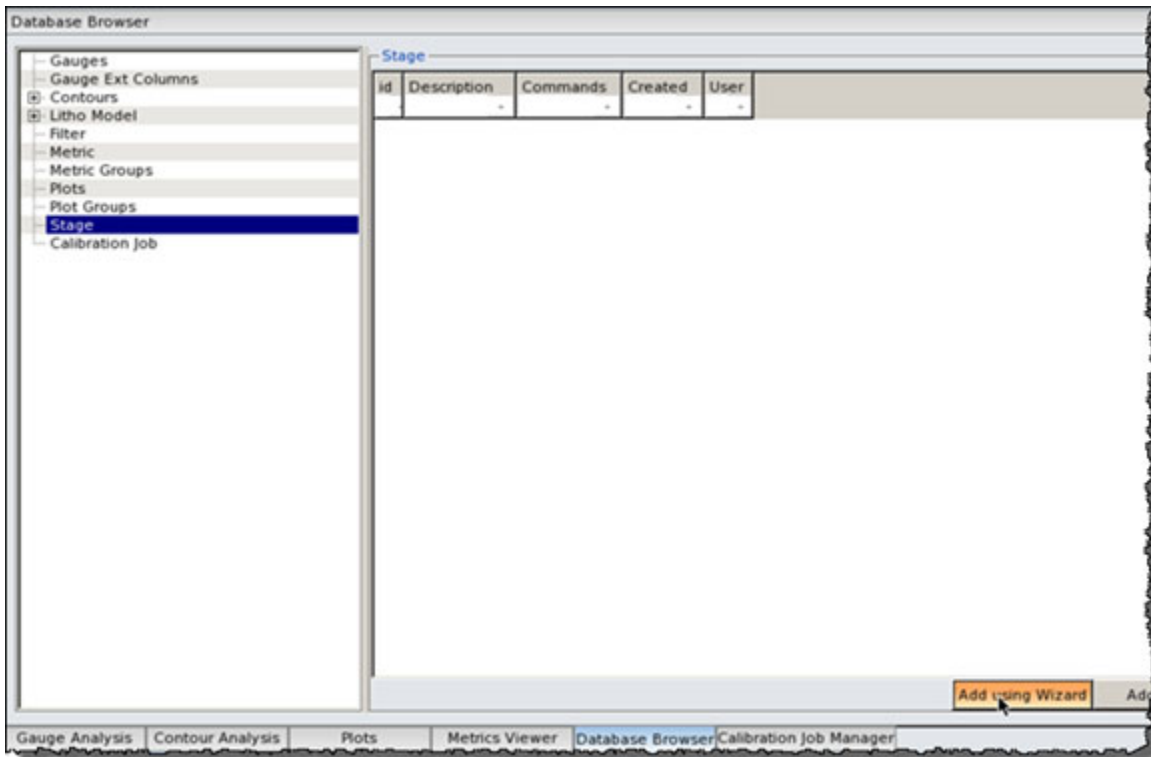
Stages represent simulation and calibration commands in a job. When you add a stage, you define simulation commands and set parameters and optimizer settings for an calibration run. Calibration jobs can be run independently, or can be linked with the results of another calibration job.

Prerequisites

- An active gauge object in the **Gauge Analysis** tab
- An active *Lithomodel* file in the database

Procedure

1. Select the **Database Browser** tab in the list of primary display tabs.

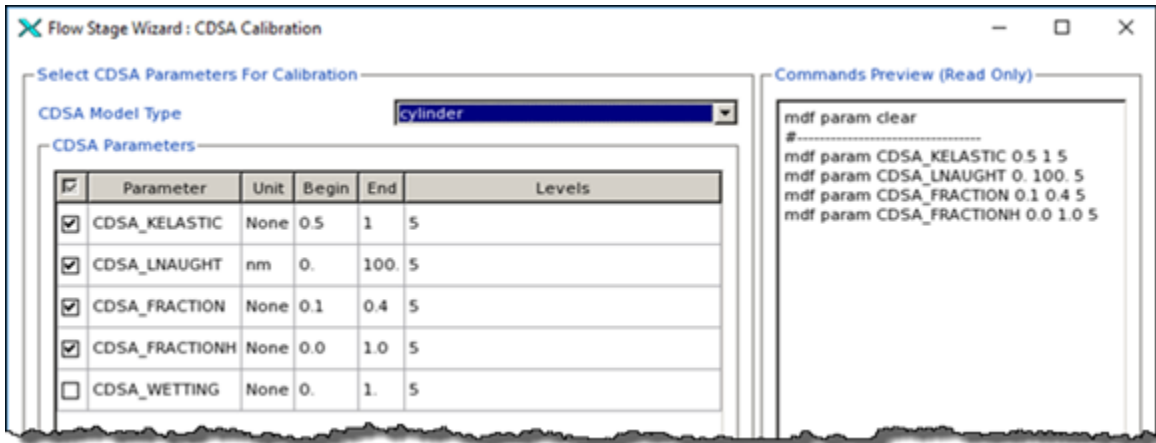


2. Select **Stage** in the left column.
3. Click the **Add Using Wizard** button.
4. In the Select Stage Type dialog, select the Calibrate option and the CDSA check box.



5. Click **Next**.

6. Select either cylinder or lamella from the CDSA Model Type dropdown list and the desired CDSA parameters using the corresponding check boxes.



7. Click **Next**.
8. Select the desired optimizer settings. The default settings can be used.



9. Click **Next**.
10. Click **Edit** and remove the “mdf optimize subsearch threshold optimize” command.



11. Click **Finish**.

Results

A new entry is added to the Stage table in the database.

Creating a DSA Calibration Job

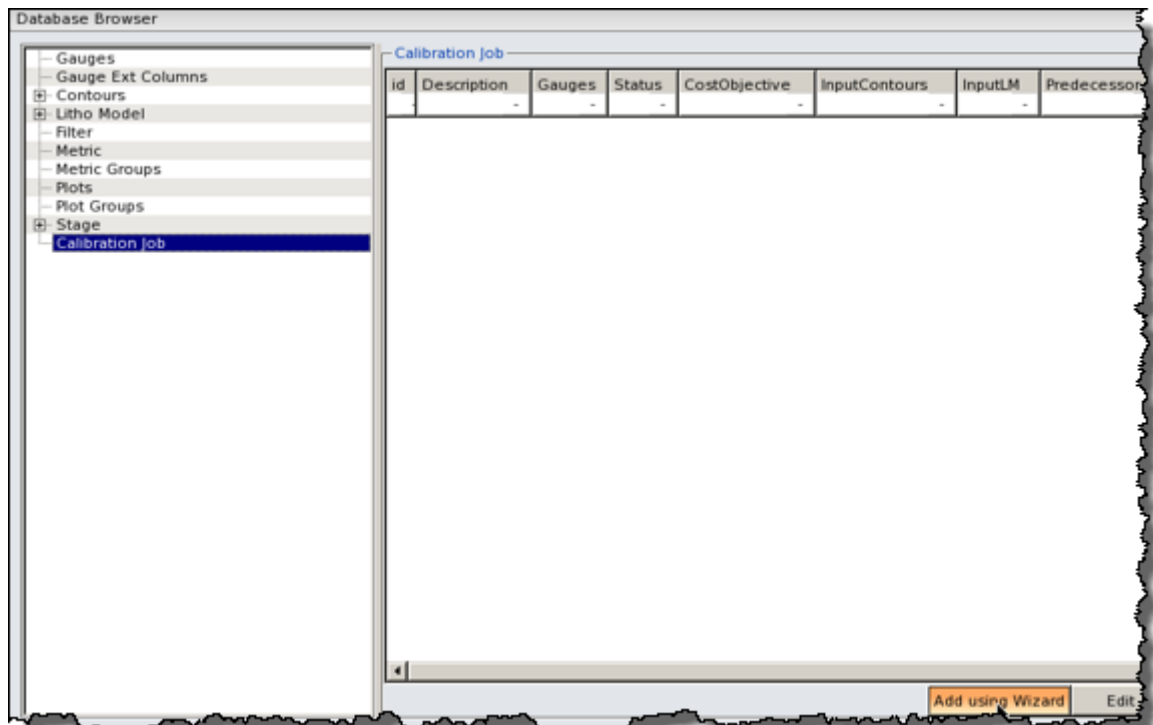
Calibration jobs are the core of optimization for Calibre nmModelflow. You must set up a calibration job in order to generate output.

Prerequisites

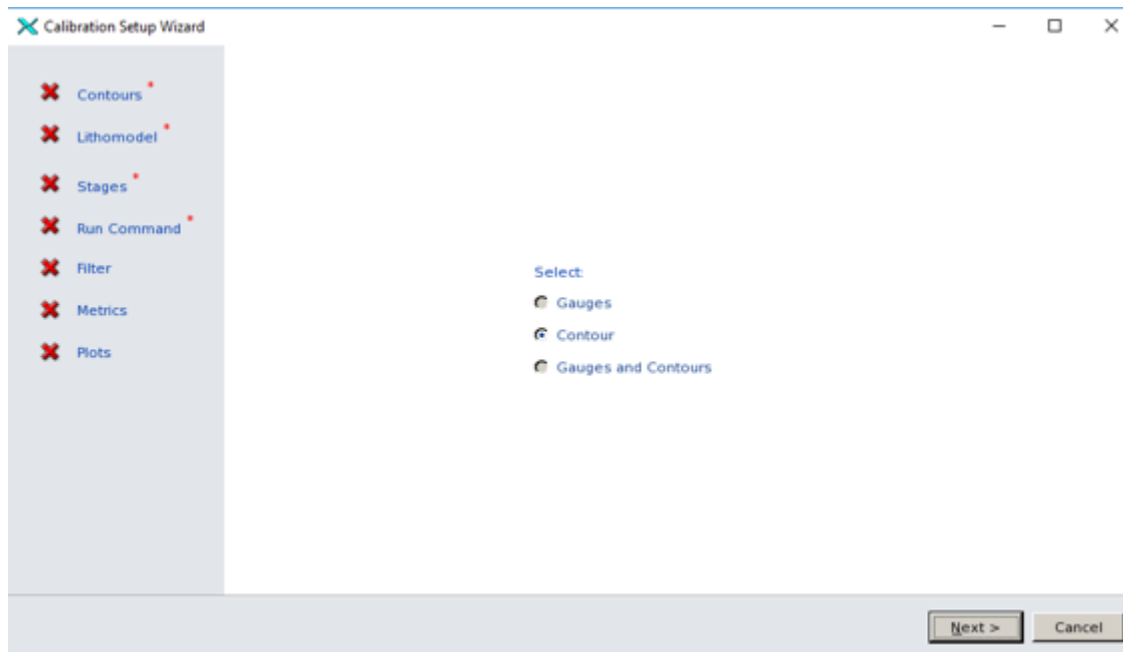
- A gauge object (contour)
- A litho model
- A stage

Procedure

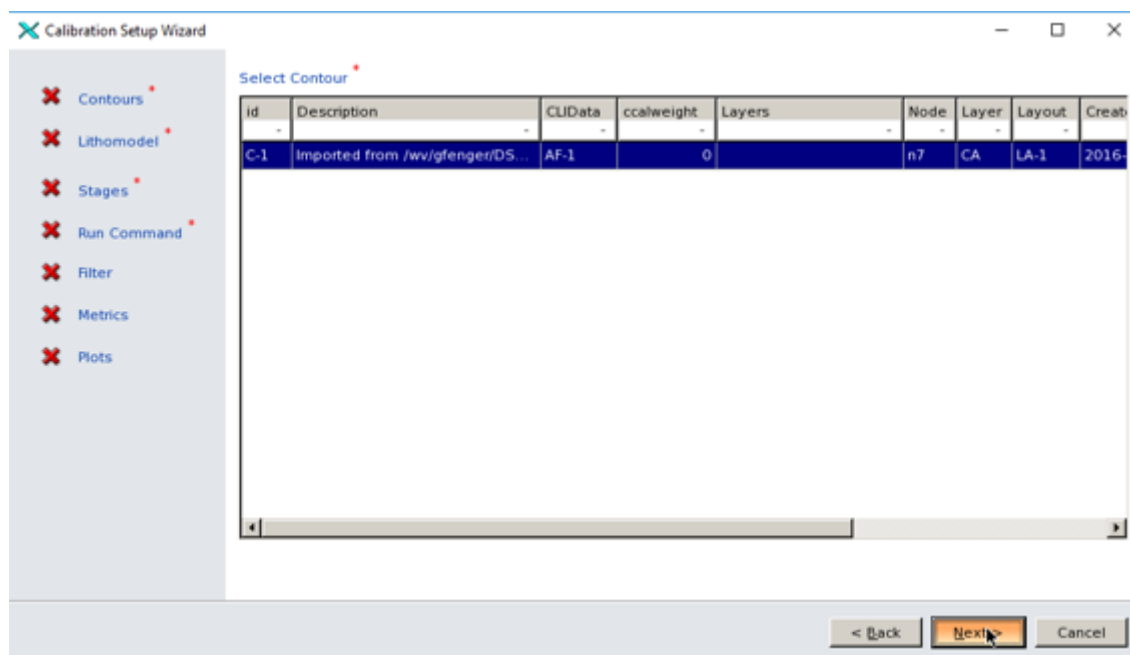
1. Click **Add using Wizard** in the Calibration Job database browser.



2. Select the Contour radio button.

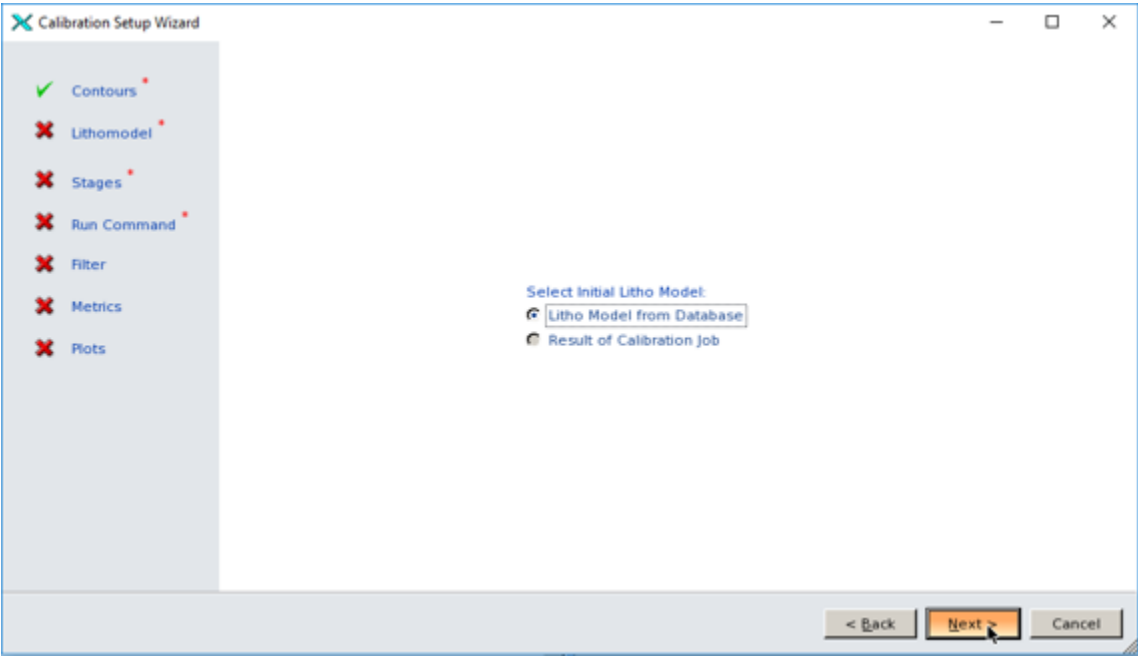


3. Click **Next**.
4. Select the previously loaded contour.



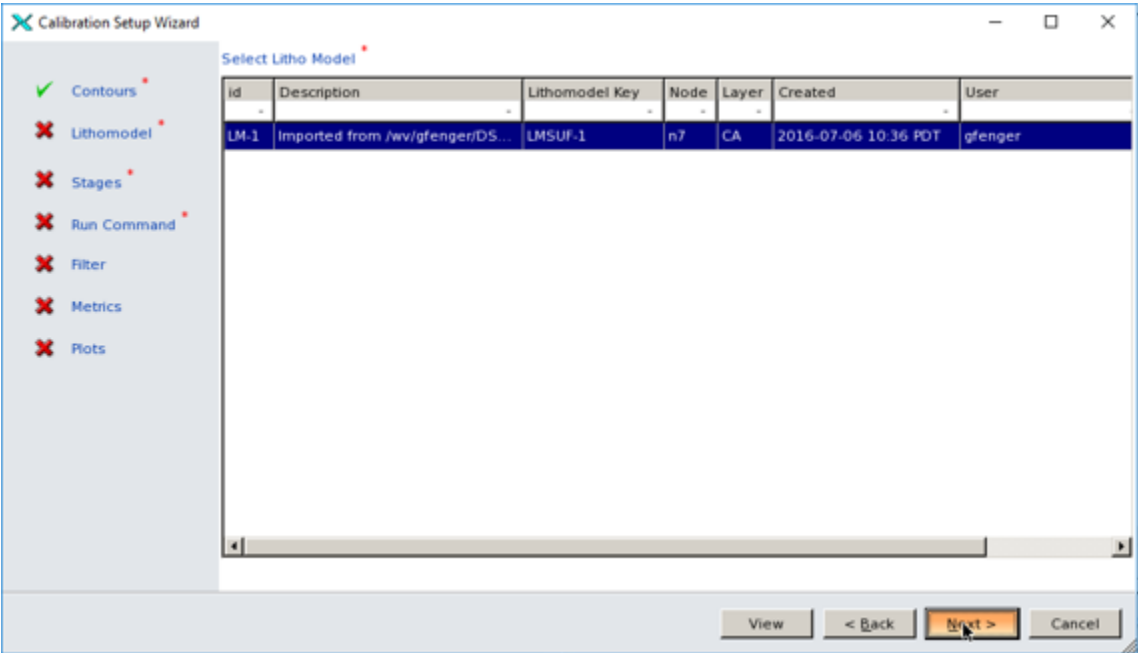
5. Click **Next**.

6. Select the “Lithomodel from Database” radio button.



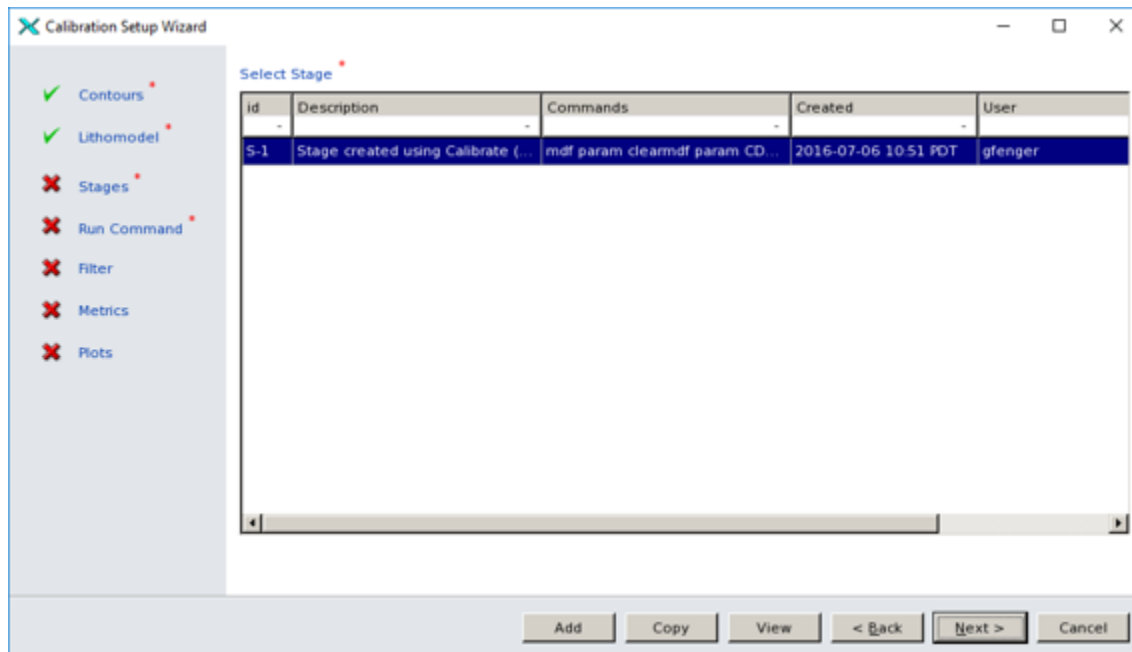
7. Click **Next**.

8. Select the previously loaded Lithomodel.

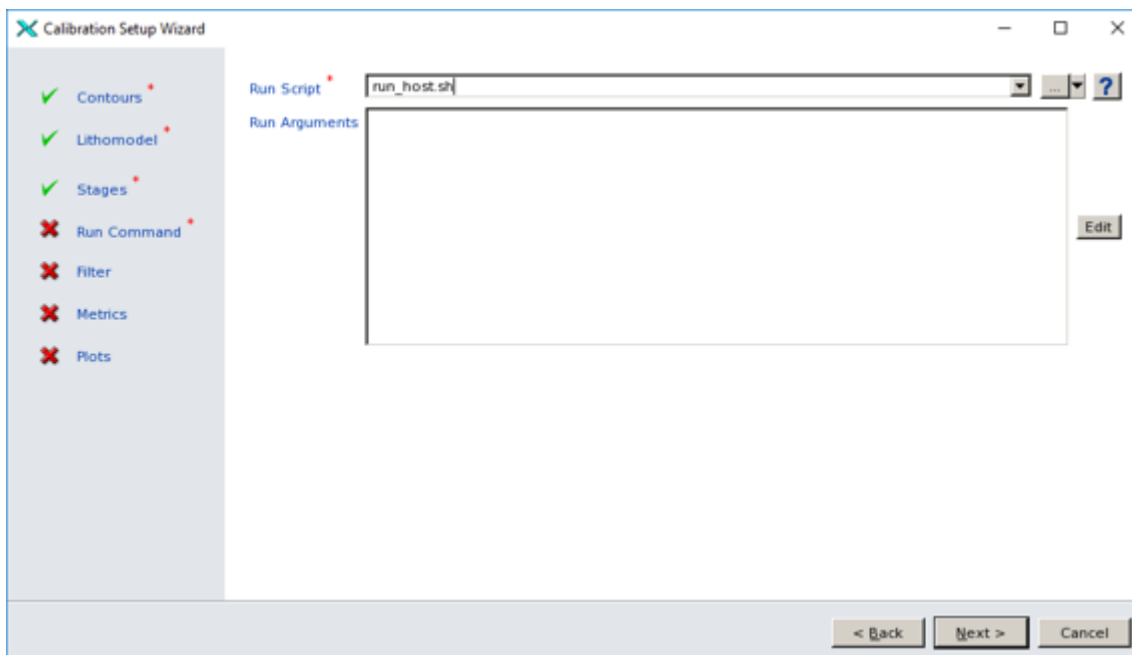


9. Click **Next**.

10. Select the previously loaded created stage.

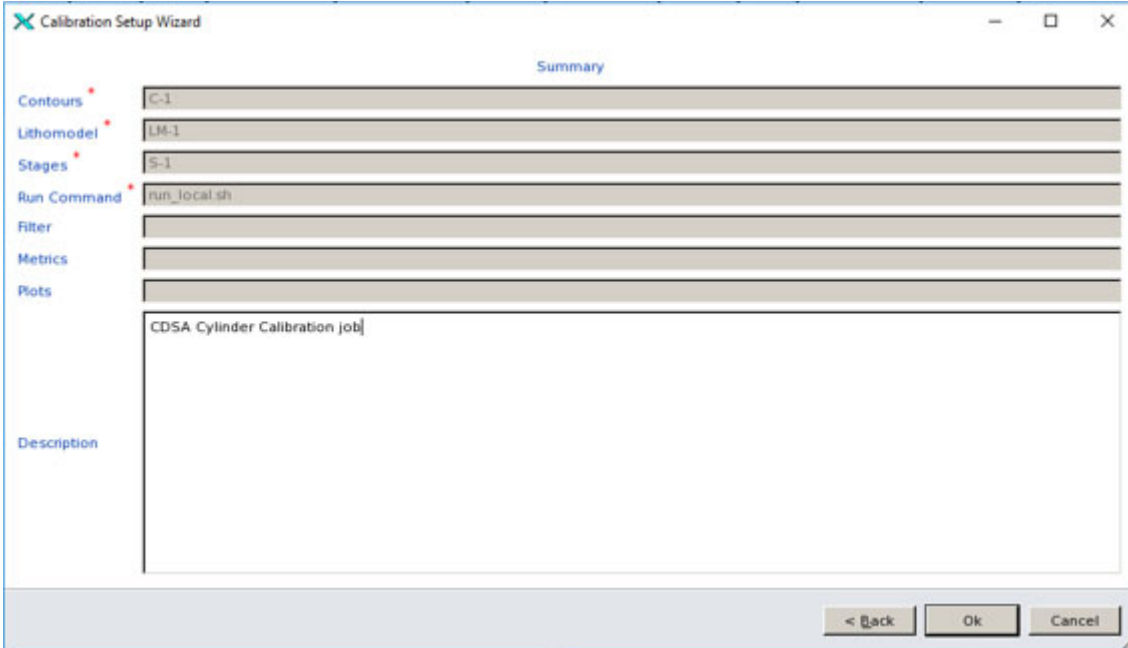


11. Click **Next**.
12. Select the appropriate run script.



13. Click **Next**.
14. Click **Finish**.

15. Add a description.



The screenshot shows the 'Calibration Setup Wizard' window, specifically the 'Summary' tab. On the left, there is a vertical list of configuration categories: Contours, Lithomodel, Stages, Run Command, Filter, Metrics, Plots, and Description. Each category has a corresponding text field to its right. The 'Contours' field contains 'C-1', 'Lithomodel' contains 'LM-1', 'Stages' contains 'S-1', and 'Run Command' contains 'run_local.sh'. The 'Description' field is a larger text area at the bottom, containing the text 'CDSA Cylinder Calibration job'. At the bottom right of the window, there are three buttons: '< Back', 'Ok', and 'Cancel'.

16. Click **OK**.

Results

The new job appears in the Calibration Job Manager list in the main window.

Chapter 3

Calibre DSA Reference

The Calibre DSA additions include a new type of process model, SVRF commands, and setup file keywords.

Commands used with Calibre nmModelflow are described in the Calibre nmModelflow command reference, invoked from its **Help** menu.

Model File Formats for DSA	32
CDSA Cylinder Model File Format	33
CDSA Lamella Model File Format	35
SVRF Commands for DSA	37
LITHO DSA DENSEOPC	38
LITHO DSA OPCVERIFY	40
TVF Commands for DSA	41
cdsa::DSA_GROUP	42
Litho Setup File Commands for DSA	49
cdsa_model_load	50
dsa_options	51
modelpath	55
setlayer cdsa_simulator	56
setlayer cdsa_synthesizer	57

Model File Formats for DSA

Calibre DSA models are produced by the Calibre nmModelflow calibration platform. Once calibrated, they are used in synthesis and simulation through the setup file.

The following rules apply to CDSA model files:

- Keywords are case sensitive.
- Each keyword must appear on its own line.
- Comments can appear anywhere, and run from the comment character (#) to the end of the line.
- Keywords can appear in any order.

CDSA Cylinder Model File Format	33
CDSA Lamella Model File Format	35

CDSA Cylinder Model File Format

Generated by: Calibre nmModelflow

Input for: DSA setup files

The CDSA cylinder model is a compact model used for creating guiding patterns and predicting where shapes will be created in a graphoepitaxy DSA manufacturing process. Use the cylinder model for round contacts.

Format

The following rules apply to CDSA model files:

- Keywords are case sensitive.
- Each keyword must appear on its own line.
- Comments can appear anywhere, and run from the comment character (#) to the end of the line.
- Keywords can appear in any order.

Parameters

- #
An optional comment. The rest of the line is not parsed by the Calibre software.
- **fraction *f***
A required keyword and value specifying the relative amount by copolymer volume of block A, the molecule which becomes the vias. For example, if A is 10% by volume of the total copolymer, set *f* to 0.1. $0 < f < 1$.
- **fractionH *fh***
A required keyword and value specifying the relative amount by volume of wetting homopolymer to the total volume of homopolymer and copolymer. $0 < fh < 1$.
- **kelastic *k***
A required keyword and value specifying the copolymer potential coefficient of cell-to-cell interactions. The coefficient is unitless. $0 < k < 1$.
- **lnaught *q***
A required keyword and value specifying the natural distance in nanometers between the A structures. This is sometimes referred to as the radius of gyration. $q > 0$.
- **type cylinder**
An optional keyword and value that defines the model type.
- **wetting {0 | 1}**
A required keyword and value (either 0 or 1) identifying which portion of the solution wets the walls of the guiding pattern.

0 — The majority block wets the sidewalls.

1 — The minority block wets the sidewalls.



- *number_contacts normalized_energy*

An optional pair of values specifying the phase transition points. The *number_contacts* is the number of contacts per group for the given threshold *normalized_energy*. If the calculated energy for a given group is greater than *normalized_energy*, the guiding pattern is assumed to be in phase transition. *normalized_energy* is unitless and empirically determined during model calibration.

Examples

This is a basic example that might be used to begin calibration against measured data:

```
kelastic 0.9  
fraction 0.1  
fractionH 0.15  
lnaught 39  
wetting 0  
type cylinder
```

CDSA Lamella Model File Format

Generated by: Calibre nmModelflow

Input for: DSA setup files

The CDSA lamella model captures properties that affect the shape of rectangular structures formed within the guiding pattern.

This model provides the shape formed within the guiding pattern and the conditions under which punch-through fails. Because of local density limits, some structures do not propagate homogeneously.

Use the lamella model for bar vias or to minimize line-edge roughness (LER) in the guiding pattern. It cannot be used for cylindrical systems.

Format

The following rules apply to CDSA model files:

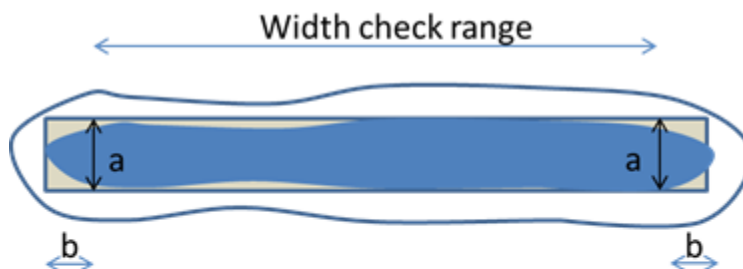
- Keywords are case sensitive.
- Each keyword must appear on its own line.
- Comments can appear anywhere, and run from the comment character (#) to the end of the line.
- Keywords can appear in any order.

Parameters

- **type lamella**
An optional keyword and value identifying the CDSA model type.
- **F *fraction***
A required keyword and value specifying the relative amount by copolymer volume of block A, the molecule which becomes the rectangle. For example, if A is 50% by volume of the total copolymer, set *fraction* to 0.5. $0 < \textit{fraction} < 1$
- **exclusion nm**
A required keyword and value that sets an exclusion zone from the target's line ends. To determine phase transition, the model needs to simulate the assembled block copolymer's width. Width measurements will not be taken in the exclusion zone.

In [Figure 3-1](#), the exclusion parameter would be set to b.

Figure 3-1. Lamella Model Measurement Region



Tips are excluded from the check range

b = distance from tip in nm
 $a \in [a_{min}, a_{max}]$

- **maxW nm**

A required keyword and value that specify the maximum width within the measurement region of an acceptable assembled lamella. If any portion of the assembled block copolymer is greater than maxW it is tagged as being in phase transition.

- **minW nm**

A required keyword and value that specify the minimum width within the measurement region of an acceptable assembled lamella. If any portion of the assembled block copolymer is less than minW it is tagged as being in phase transition.

Note that the lamella ends are rounded; set exclusion to keep the rounded tips from being measured.

SVRF Commands for DSA

Calibre DSA extends Calibre nmOPC and Calibre OPCverify commands by adding the keyword “DSA”. The DSA extension can be used jointly with the EUV extension.

The SVRF commands are added to an existing SVRF rule file. The litho setup file called by the SVRF commands can be included in the SVRF rule file using the [Litho File](#) SVRF command.

Table 3-1. SVRF Commands

Command	Description
LITHO DSA DENSEOPC	Creates a layer with guiding patterns from the grouped contacts. Can also be used to output debug layers.
LITHO DSA OPCVERIFY	Runs DSA verification as part of a Calibre OPCverify run.

LITHO DSA DENSEOPC

Creates a layer with guiding patterns from the grouped contacts. Can also be used to output debug layers.

Usage

LITHO DSA [EUV] **DENSEOPC** *input_layer...* **FILE** *setup_file* **MAP** *layer_name*

Arguments

- **EUV**
An optional argument that invokes the EUV libraries. If this keyword is not included, Calibre nmOPC uses libraries designed for 193i lithography.
The EUV argument must appear between DSA and DENSEOPC if used.
- ***input_layer***
A required argument identifying the input layers in the design. You must specify at least one layer.
For DSA synthesis, Calibre nmOPC needs at least the grouping layer generated from [cdsa::DSA_GROUP](#) and the layer containing the drawn contacts to be used as a target layer.
- **FILE *setup_file***
A required argument identifying the litho setup file with the DSA synthesis settings. This setup file calls [setlayer cdsa_synthesizer](#) to produce the output layer (*layer_name*).
- **MAP *layer_name***
A required argument specifying the name of the derived layer *out_layer* from [setlayer cdsa_synthesizer](#). The two layer names must match.

Description

The SVRF operation LITHO DSA DENSEOPC passes layers between the SVRF rule file and the Calibre OPC engine. The DSA keyword triggers additional functionality for directed self-assembly.

The DSA algorithms are controlled by the settings in the *setup_file*. This file must have the settings needed for DSA synthesis. Depending on the [setlayer cdsa_synthesizer](#) specification, the returned layer may contain individual contacts, the polygons that marked groups of contacts, or the guiding patterns. Unlike non-DSA LITHO DENSEOPC output, the layer has *not* received OPC correction.

Examples

The following code uses concurrency to run the DSA synthesizer once to generate both guiding patterns and debug layers:

```
LITHO FILE dsa_synth {  
  
...  
  
setlayer good_vias = cdsa_synthesizer options dsa_opt1  
setlayer bad_vias1 = cdsa_synthesizer options dsa_opt1 map error_layer  
setlayer bad_vias2 = cdsa_synthesizer options dsa_opt1 map rejected  
}  
  
guiding_patterns = LITHO DSA DENSEOPC grp ct FILE dsa_synth MAP good_vias  
bad_placement    = LITHO DSA DENSEOPC grp ct FILE dsa_synth MAP bad_vias1  
rejected_vias     = LITHO DSA DENSEOPC grp ct FILE dsa_synth MAP bad_vias2
```

LITHO DSA OPCVERIFY

Runs DSA verification as part of a Calibre OPCverify run.

Usage

LITHO DSA [EUV] **OPCVERIFY** *input_layer...* **FILE** *setup_file* **MAP** *layer_name*

Arguments

- **EUV**
An optional argument that invokes the EUV libraries. If this keyword is not included, Calibre OPCverify uses libraries designed for 193i lithography.
The EUV argument must appear between DSA and OPCVERIFY if used.
- ***input_layer***
A required argument identifying the input layers in the design. You must specify at least one layer.
For DSA verification, Calibre OPCverify needs at least the contour layer generated from the guiding pattern mask and the layer containing the drawn contacts to be used as a target layer.
- **FILE *setup_file***
A required argument identifying the litho setup file with the DSA verification settings. This setup file should call [setlayer cdsa_simulator](#).
- **MAP *layer_name***
A required argument specifying the name of the derived layer *out_layer* from [setlayer cdsa_simulator](#). The two layer names must match.

TVF Commands for DSA

There is a Tcl Verification Format (TVF) cdsa package.

To use the TVF commands in a program, the file must include the following lines:

```
#! tvf
package require cdsa
```

Table 3-2. TVF Commands

Command	Description
cdsa::DSA_GROUP	Creates groups of vias (contacts) that are clustered into a single guiding pattern, and also identifies arrangements that cannot be manufactured with the specified settings. This command name is case sensitive.

cdsa::DSA_GROUP

Creates groups of vias (contacts) that are clustered into a single guiding pattern, and also identifies arrangements that cannot be manufactured with the specified settings. This command name is case sensitive.

Usage

Standard Form

```
cdsa::DSA_GROUP -layer input_layer -lnaught pitch  
-dsaLength max_extent -lithoLength spacing  
-group group_layer  
[-compliant output_layer]  
[-forbidden output_layer]  
[-type via -viaWidth width]  
[-groupStyle {manhattan | string}]  
[-viaTolerance tolerance]  
[-maxGroupSize integer]  
[-forbiddenLitho output_layer]  
[-forbiddenGroup output_layer]  
[-forbiddenCollinear output_layer]  
[-forbiddenGeometry output_layer]  
[-pregrouped input_layer]  
[-autogrouped output_layer]  
[-checkSet {all | autoonly}]
```

Double Patterning Form

```
cdsa::DSA_GROUP -layer input_layer -lnaught pitch  
-dsaLength max_extent -lithoLength spacing  
-group group_layer -group2 group_layer  
[-compliant output_layer] [-compliant2 output_layer]  
[-forbidden output_layer] [-forbidden2 output_layer]  
[-type via -viaWidth width]  
[-groupStyle {manhattan | string}]  
[-viaTolerance tolerance]  
[-maxGroupSize integer]  
[-forbiddenLitho output_layer] [-forbiddenLitho2 output_layer]  
[-forbiddenGroup output_layer] [-forbiddenGroup2 output_layer]  
[-forbiddenCollinear output_layer] [-forbiddenCollinear2 output_layer]  
[-forbiddenGeometry output_layer]  
[-pregrouped input_layer]  
[-precolored input_layer] [-precolored2 input_layer]  
[-autogrouped output_layer] [-autogrouped2 output_layer]  
[-autocolored output_layer] [-autocolored2 output_layer]  
[-checkSet {all | autoonly}]
```

Arguments

- **-layer *input_layer***
A required argument specifying the input layer name. The input layer should contain vias to group. Only via layers are supported (as opposed to lamellar structures).
- **-lnaught *pitch***
A required argument specifying the minimum pitch between vias possible with the DSA process. The pitch is given in microns. Anything smaller than this pitch is considered forbidden. (Pitches smaller than -lnaught can be manufactured using double patterning but must be on different layers.)
- **-dsaLength *max_extent***
A required argument specifying the maximum extent of a group in microns. Vias farther apart than -dsaLength are never placed in the same group.
- **-lithoLength *spacing***
A required argument specifying the spacing in microns between groups. (A single via can be its own group.) Typically -lithoLength is set to the minimum resolvable pitch for mask shapes.
- **-group *group_layer* [-group2 *group_layer*]**
A required argument specifying the output layer. Polygons outlining and connecting grouped vias are output to this layer. This layer is the group layer read by [setlayer cdsa_synthesizer](#).

The optional keyword -group2 specifies the name of a second output layer for use with double patterning.
- **-compliant *output_layer* -compliant2 *output_layer***
An optional argument specifying that DSA_GROUP should output compliant geometries to the named layer.

The optional keyword -compliant2 specifies the name of a second output layer for use with double patterning.
- **-forbidden *output_layer* -forbidden2 *output_layer***
An optional argument specifying that DSA_GROUP should output to the named layer the geometries that cannot be grouped for any reason.

The optional keyword -forbidden2 specifies the name of a second output layer for use with double patterning.
- **-type via -viaWidth *width***
An optional pair of arguments specifying additional constraints on the input geometry in microns. To specify -viaWidth (an upper limit on acceptable via sizes) you must also specify -type via.

By default, the input to DSA_GROUP must be the same size and square.

- `-groupStyle {manhattan | string}`

An optional argument specifying the allowed geometric arrangement for the via group used to generate the guiding pattern. Typically, manhattan corresponds to the 193i process and string corresponds to the EUV process.

manhattan — Vias are in a straight vertical or straight horizontal line; no bends. This is recommended, and is the default groupStyle if the argument is not specified.

string — Vias form a path with segments joined at non-acute (right or obtuse) angles; otherwise, a synthesizer error message will occur.

- `-viaTolerance tolerance`

An optional argument specifying a tolerance for via width in microns. Vias drawn within \pm viaTolerance are accepted as valid. The default is 1 nm.

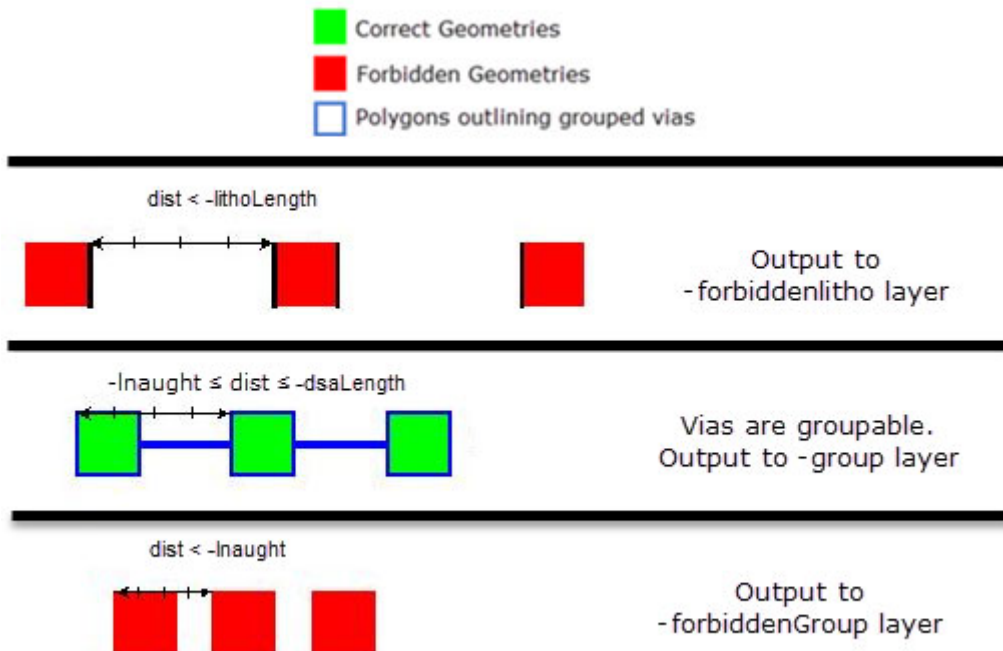
- `-maxGroupSize integer`

An optional argument specifying the maximum number of vias per group. The default is 5.

- `-forbiddenLitho output_layer -forbiddenLitho2 output_layer`

An optional argument that outputs markers indicating vias whose center-to-center distance is greater than -dsaLength (the vias cannot be grouped together), and whose edge-to-edge distance is less than -lithoLength (the vias cannot be placed on the same mask). The following figure shows an example of an output for -forbiddenLitho and forbiddenGroup, as well as a groupable configuration.

Figure 3-2. Forbidden Geometries to Litho and Grouping Violations



The optional keyword `-forbiddenLitho2` specifies the name of a second output layer for use with double patterning.

- `-forbiddenGroup output_layer -forbiddenGroup2 output_layer`

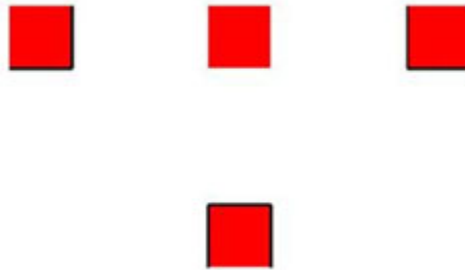
An optional argument specifying that DSA_GROUP should output to a named layer the vias that are closer together than `-lnaught`. When the pitch is less than `-lnaught`, the structures are forbidden since the assembly cannot happen at very tight pitches. An example of a configuration output to `-forbiddenGroup` is shown in [Figure 3-2](#).

The optional keyword `-forbiddenGroup2` specifies the name of a second output layer for use with double patterning.

- `-forbiddenCollinear output_layer -forbiddenCollinear2 output_layer`

An optional argument specifying that DSA_GROUP should output to the named layer the vias that could not be grouped in rectilinear strings. Independent of which process is used, configurations which create branches (vias grouped to more than two neighbors) are considered forbidden. This grouping violation is produced when the vias' centerlines do not share either the same vertical or horizontal coordinate (that is, the vias lie on a diagonal). The following figure shows an example of a via configuration that would result in a forbidden output for both manhattan and string group styles.

Figure 3-3. Forbidden Geometries Due to Collinearity Violation



The optional keyword `-forbiddenCollinear2` specifies the name of a second output layer for use with double patterning.

- `-forbiddenGeometry output_layer`

An optional argument specifying that DSA_GROUP should output contacts that are the wrong shape or size to the named layer. Contacts must be rectangles whose x and y dimensions are specified by `-viaWidth` with a tolerance specified by `-viaTolerance`. An example of an output for `-forbiddenGeometry` is shown in the following figure.

Figure 3-4. Forbidden and Correct Geometries



- `-pregrouped input_layer`

An optional argument specifying the name of an input layer indicating pregrouped vias on the layer specified by the `-layer` argument. Shapes that interact between these two specified layers are considered to be grouped.

For an example of how two layers interact, see “[Interact](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

- `-precolored input_layer -precolored2 input_layer`

An optional argument specifying the name of an input layer for use with double patterning. The layer should contain vias that are precolored with an initial color. The optional keyword `-precolored2` specifies the name of a second input layer for use with double patterning. This layer should contain vias that are precolored with a second color.

- `-autogrouped output_layer -autogrouped2 output_layer`

An optional argument specifying that DSA_GROUP should output automatically generated groups to the named layer. The optional keyword `-autogrouped2` specifies the name of a second output layer for use with double patterning.

- `-autocolored output_layer -autocolored2 output_layer`

An optional argument specifying that DSA_GROUP should output automatically colored groups to the named layer. The optional keyword `-autocolored2` specifies the name of a second output layer for use with double patterning.

- `-checkSet {all | autoonly}`

An optional argument specifying that no grouping or coloring will be done. Instead, the correctness of the groups is checked.

autoonly — Groups defined on the `-pregrouped` layer are assumed to be correct and are not checked.

all — All groups are checked.

Description

DSA_GROUP is a Tcl verification format (TVF) command that is part of the cdsa package. To use this command, your script must include the following lines:

```
#! tvf
package require cdsa
```

Only via layers are supported (as opposed to lamellar structures).

The command creates groups of vias that are clustered into a single guiding pattern. The command can also be used in double patterning flows to produce a pair of DSA masks, each containing DSA groups. Precolored layers can be specified to assign vias to a mask for double patterning using the -precolored argument. DSA_GROUP automatically assign colors to groups by default, which can be output to the -autocolored layer.

A group in DSA serves as a compliance check in that if vias cannot be properly grouped or be independent single vias, then they are considered “forbidden”. Two styles of grouping are supported: string and manhattan. String style grouping forms a path with segments joined at non-acute (right or obtuse) angles. Manhattan style grouping forms a straight vertical or horizontal line.

The following requirements must be met in order for vias to be groupable:

- Vias must be the same size and square. Bar vias are forbidden, and can be output to the -forbiddenGeometry layer.
- Grouping configurations cannot form branches (groups where vias have more than two neighbors); grouped vias must form a single chain. Branched configurations can be output to the -forbiddenCollinear layer.
- Overlapping groups, in which a via could belong to multiple groups, are not allowed. Uniformly spaced strings of vias where a via in the middle might be placed in any of several groups cannot be grouped for this reason. Increasing spacing either horizontally or vertically between rows of vias can resolve this issue. Overlapping groups can be output to the -forbiddenGroup layer.
- Groups are only formed if the vias are within -lnaught (L_0), the natural pitch of the polymer. This is because the assembly of the phases has only a limited tolerance. This distance may be increased by manipulating the shape of the guiding pattern, up to a distance of -dsaLength, within which the DSA process can reliably assemble vias. Distances exceeding -dsaLength are far beyond the natural pitch, and are thus not feasible for DSA. Grouping allows vias that are closer together than -lithoLength to be on the same mask if they are within -dsaLength; otherwise, the vias must be placed on different masks. Both -lnaught and -dsaLength are measured center-to-center, while -lithoLength is measured edge-to-edge. For the string group style, -lithoLength is much

smaller than in the manhattan group style, and may be less than -dsaLength or -lnaught. Vias that violate -lithoLength or -dsaLength can be output to the -forbiddenLitho layer.

The required -group layer outputs polygons outlining and connecting the grouped targets. Grouped contacts can be output to the -compliant layer. Vias can also be manually grouped by specifying a -pregrouped input layer. The pregrouped output is in the same form as the -group layer polygons regardless of the shape of the pregrouping input marker for the group, which allows more permissive shapes. Vias belonging to a pregroup may be precolored, as long as vias with opposite precolors are not included in the same pregroup. Pregrouped vias do not need to comply to the criteria for automatic grouping (which can be output to the -autogrouped layer). If non-compliant groups are given as predefined groups, the groups are included in the grouped output. They will also be flagged as forbidden if checking of them is specified with -checkSet.

Litho Setup File Commands for DSA

The litho setup file commands can be used in a litho setup file. Like the litho setup files used by Calibre nmOPC, the litho setup files used by Calibre DSA must load a model, collect product-specific settings in an options block, and invoke the product with a setlayer statement.

Table 3-3. Litho Setup File Commands

Command	Description
cdsa_model_load	Loads a CDSA model. Cannot be used with litho models.
dsa_options	In-lined subcommand block for setlayer cdsa_synthesizer and setlayer cdsa_simulator.
modelpath	Specifies the directories to search for CDSA or litho models. This command is optional. The command accepts both absolute and relative paths.
setlayer cdsa_simulator	Invokes the DSA simulator.
setlayer cdsa_synthesizer	Invokes the DSA synthesizer.

cdsa_model_load

Loads a CDSA model. Cannot be used with litho models.

Usage

cdsa_model_load *model_name model_path*

Arguments

- ***model_name***
A required argument that specifies the name by which other commands refer to the CDSA model.
- ***model_path***
A required argument specifying the name of the CDSA model as it exists in the file system. The directory of ***model_path*** must be in the [modelpath](#) list.

Description

A CDSA model must be specified to run the DSA simulator or synthesizer. If you are using conventional models, this is done with the combination of `modelpath`, `cdsa_model_load`, and the `dsa_options` keywords. If you are using litho models, use `modelpath` and the `dsa_options litho_model` argument instead.

Examples

To load a conventional (non-litho) model:

```
modelpath ../models:/lib/models/DSA
cdsa_model_load my_model lpp10_cdsa.mod
...
dsa_options opts {
    ...
    model my_model
    ...
}
...
```

If neither the file `../models/lpp10_cdsa.mod` or `/lib/models/DSA/lpp10_cdsa.mod` exists, the run halts with an error.

dsa_options

In-lined subcommand block for setlayer cdsa_synthesizer and setlayer cdsa_simulator.

Usage

```
dsa_options opt_block '{'
    arguments
  '}'
```

Arguments

- *opt_block*
A required argument naming the dsa_options block so that it may be referred to by other commands.
- *arguments*
A required list of commands for the DSA run surrounded by required braces ({ }). The commands must appear each on their own line. The pound character (#) comments out the rest of the line.

Table 3-4. Valid Commands for dsa_options

Argument	Description
dsa_group <i>layer_name</i>	<i>Required for setlayer cdsa_synthesizer.</i> The <i>layer_name</i> specifies a layer created by the cdsa::DSA_GROUP command, created manually, or generated in SVRF.
guiding_pattern <i>layer_name</i>	<i>Required for setlayer cdsa_simulator.</i> The <i>layer_name</i> specifies a layer previously created by the DSA synthesizer.
dsa_target <i>layer_name</i>	<i>Required.</i> Specifies a layer containing the individual vias.
litho_model <i>model_name</i>	Specifies a directory containing a litho model. The directory must be somewhere in the path defined by the modelpath command, and it must contain a CDSA model. All contents of the litho model except for the CDSA model are ignored. Either litho_model or model must be specified.
model <i>model_name</i>	Specifies the name of a DSA model previously loaded with cdsa_model_load . The <i>model_name</i> must be the name assigned to the model by cdsa_model_load. Either litho_model or model must be specified.
iterations <i>number</i>	An optional argument for setlayer cdsa_synthesizer. Specifies the maximum number of iterations during layout synthesis. Any contacts without a satisfactory solution after <i>number</i> iterations are written to the error layer. The default value is 15.

Table 3-4. Valid Commands for dsa_options (cont.)

Argument	Description
min_pitch <i>nm</i>	An optional argument for setlayer cdsa_synthesizer. Specifies the minimum feasible pitch. Any structures closer than <i>nm</i> are skipped during the synthesis process as no solution exists.
connector_width <i>nm</i>	An optional setting for the DSA synthesizer. Specifies the width of the guiding pattern in nanometers between adjacent contacts in the same group. The default value is the width of the contact.
corner <i>nm</i>	An optional setting for the DSA synthesizer. Specifies the length in nanometers of a 45-degree edge used to chop corners. The default value is 4 nm.
group_context_bound <i>nm</i>	An optional setting for the DSA synthesizer. Specifies an upper limit to the length of a guiding pattern. Shorter lengths generally produce better results but the length must support the maximum group size set in cdsa::DSA_GROUP . The default is 200 nm.
gp_aspect_ratio_bound <i>bound</i> [gp_aspect_ratio_start <i>start</i>]	An optional optimization setting for the DSA synthesizer. The parameters bound the search space for the width and height of the symmetrical octagonal sections that are derived from square contacts. The values bound and start are specified as fractions of the original dimensions, and must be between 0 and 1. <ul style="list-style-type: none"> gp_aspect_ratio_bound <i>bound</i> – The lower bound of the search space. gp_aspect_ratio_start <i>start</i> – (optional) The upper bound of the search space; the default start value is 1. If specified, start should be greater than bound. See the “Description” section for more detail.
target_width_tolerance <i>fraction</i>	An optional setting for the DSA synthesizer. Specifies the tolerance of how much smaller the final contact can be as a fraction of the original contact width. Specifying a tolerance allows the tool to try to further optimize the target location.
perpendicular	An optional optimization setting for the DSA synthesizer. When perpendicular is specified, the synthesizer explores both in-line and perpendicular orientations of the targets relative to adjacent structures, and selects the orientation that minimizes the internal energy. See the “Description” section for more detail.

Description

Creates a uniquely named block of product commands inside a litho setup file to set DSA options. This block is called by the [setlayer cdsa_simulator](#) and [setlayer cdsa_synthesizer](#) commands using the options argument.

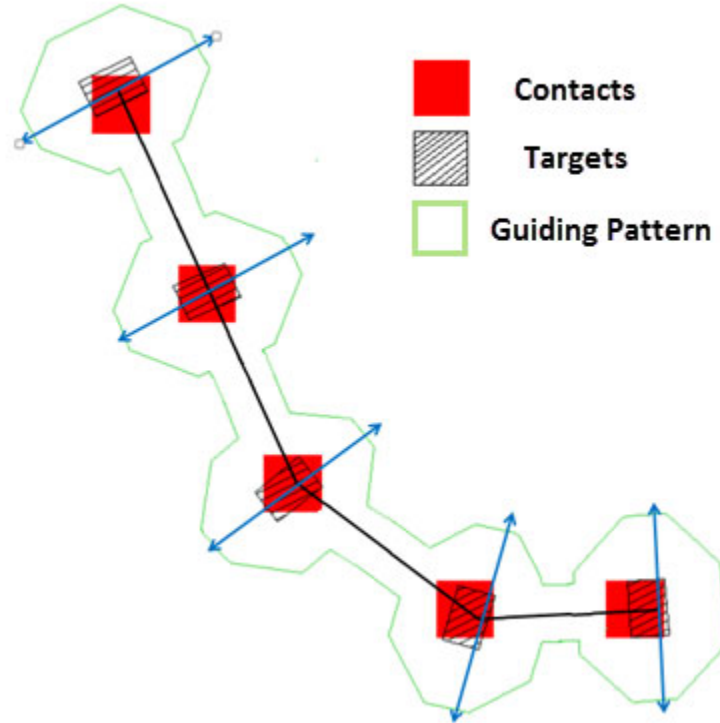
Many arguments are specific to either the DSA simulator or the DSA synthesizer.

Since contacts are grouped before synthesis, the guiding patterns and targets are oriented and sized to minimize the internal energy and optimize placement of the targets relative to the desired location of the contacts. Octagons are the default shape used for the guiding patterns. Using optional settings, the width of the octagonal guiding patterns changes dynamically in order to optimize target placement for each contact. Thus, the octagons do not have to be the same size. Additional optional settings are used to vary the width of the target, and place it either perpendicular or in-line with connecting targets, depending on what is determined to be optimal by the synthesizer.

The parameters `gp_aspect_ratio_bound` and `gp_aspect_ratio_start` bound the search space the tool explores for the width and height of the octagonal shapes in order to optimize the guiding pattern. Possible guiding patterns dimensions are searched in the range specified by parameters `gp_aspect_ratio_bound` and `gp_aspect_ratio_start`, and the guiding pattern with minimum energy is chosen.

When perpendicular is specified, the synthesizer explores both inline and perpendicular orientations of the targets relative to adjacent structures, and selects the orientation that minimizes the internal energy. If the target is in-between two adjacent targets, the target is oriented along the angle bisector, as shown in the following figure. The figure also illustrates the variable width of the octagonal guiding patterns.

Figure 3-5. Optimizing the Guiding Pattern With the Perpendicular Option



Examples

This example shows invoking the DSA simulator:

```
modelpath /lib/models/10nm:.\n\ndsa_options sim_block {\n    guiding_pattern gp1\n    dsa_target ct_poly\n    litho_model lpp10\n}\n\nsetlayer ct_out_energy = cdsa_simulator options sim_block property energy
```

modelpath

Specifies the directories to search for CDSA or litho models. This command is optional. The command accepts both absolute and relative paths.

Usage

modelpath *dir*

Arguments

- *dir*

Specifies a colon (:) separated list of directories to search. The default is the directory Calibre was invoked in.

Examples

Absolute path:

```
modelpath /export/home/calibre_wrk/models::/home/calibre/my_work/models
```

Relative path:

```
modelpath ../models::../home/calibre/my_work/models
```

setlayer cdsa_simulator

Invokes the DSA simulator.

Usage

setlayer *out_layer* = **cdsa_simulator** **options** *opt_block* [property energy]

Arguments

- *out_layer*
A required layer name to identify the layer containing the simulated contacts. The new layer is created and the results of cdsa_simulator are written to the layer. The name must be a valid Calibre layer name.
- **options** *opt_block*
A required keyword and argument indicating the name of the [dsa_options](#) block to use.
- property energy
An optional argument that adds a property to the contacts in the guiding pattern. The property value is the energy associated with that placement. This can be useful to determine the phase transition region and the validity of the simulated placement and contact shape.

Description

The setlayer cdsa_simulator command uses the DSA guiding pattern and target layer to generate simulated contacts. The guiding pattern and target layer are specified in the dsa_options block, along with other parameters. The target layer is necessary to determine how many contacts are desired within the guiding pattern. The simulator also calculates the energy required to most closely match that goal.

Examples

The following command uses the layers specified in “dsa_options dsa2” to create the output layer simulated_contacts. The contacts do not have the energy property attached.

```
setlayer simulated_contacts = cdsa_simulator options dsa2
```


setlayer cdsa_synthesizer

Invokes the DSA synthesizer.

Usage

```
setlayer out_layer = cdsa_synthesizer options opt_block [map {error_layer | final_target |  
rejected}]
```

Arguments

- **out_layer**

A required layer name to identify the layer containing the simulated contacts. The new layer is created and the results of cdsa_synthesizer are written to the layer. The name must be a valid Calibre layer name.

- **options opt_block**

A required keyword and argument indicating the name of the [dsa_options](#) block to use.

- **map {error_layer | final_target | rejected}**

An optional pair of keywords to change the content of the *out_layer*.

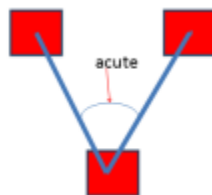
default (argument not specified) — The guiding patterns. The layer can be used in Calibre nmOPC as the opc layer.

error_layer — Group polygons for contacts for which no solution exists. In some instances the code determines it cannot converge on a solution. When “map error_layer” is used, the group polygons for the problematic contacts are written to the output layer. Simulating the output layer will identify groups of contacts for which no solution exists using the current settings.

final_target — The contacts without the guiding pattern. The layer can be used in Calibre nmOPC as a seed layer for the correction layer, or used to insert assist features. The contacts are shown in their final position prior to generating the guiding pattern.

rejected — Debug output. The layer contains the structures which were rejected during synthesis because of a simulator exception. The simulator exception only occurs when consecutive contacts within the group form an acute angle. Such structures should not have been grouped together because they are unlikely to self-assemble.

Figure 3-6. Simulator Exception Cause



Description

The `setlayer cdsa_synthesizer` command creates guiding patterns from the grouped contacts. The default output is a layer containing guiding patterns that create the grouped contacts given the process settings. The command can also be used to debug layouts with `map error_layer` or `map rejected`.

Note that the guiding patterns make no attempt to comply with MRC rules. It is assumed that the OPC or RET process is applied later in order to create the actual mask patterns. The output from this command is used only as a lithographic target; the RET correction must also check for MRC compliance.

Examples

The following code uses the layers specified in `dsa_opt1` to create guiding patterns and identify non-manufacturable contacts:

```
setlayer good_vias = cdsa_synthesizer options dsa_opt1
setlayer bad_vias1 = cdsa_synthesizer options dsa_opt1 map error_layer
setlayer bad_vias2 = cdsa_synthesizer options dsa_opt1 map rejected
```

Generally `bad_vias2` should be empty. The `bad_vias1` layer contains guiding patterns for, for example, contacts spaced at a forbidden pitch.

2D model

A model that supports objects that are not on a strict line.

A

The minority block copolymer, which becomes the cylinder.

guiding pattern

A shape of a particular size that can be etched on the wafer. It is proportioned such that it guides the block copolymers to form contacts (vias) in the desired arrangement.

phase transition

A change between stable assembly states that results in a distorted contact or line, or an additional partial structure.

— Symbols —

`[]`, 10

`{}`, 11

`|`, 11

— B —

Bar structures, 35

Block A, 59

Block B, 13

BlockA, 13

Bold words, 10

— C —

CALIBRE_HOME, 11

Case sensitivity, 32

CDSA model, 32, 50, 51

CDSA workflow, 12

`cdsa_model_load` command, 50

`cdsa_simulator`, 56

`cdsa_synthesizer`, 57

Clustering, 43

Command syntax, 10

Configuration, 51

Contours, 56

Copolymer description, 33

Courier font, 10

Cylinder model, 33

— D —

Directed self-assembly, 13

Directories, 55

Double pipes, 11

DSA group, 43, 51

DSA layer, 38

DSA options, 51

— E —

EUV, 38, 40

— F —

Fin structures, 35

Font conventions, 10

— G —

Group command, 43

Guiding patterns, 51, 57, 59

— H —

Heavy font, 10

— I —

Integrate DSA, 12

Italic font, 10

— L —

L0 *see* Lnaught, 13

Lamella model, 35

Licenses, 11

Litho DSA Denseopc, 38

Litho DSA OPCverify, 40

Litho models, 55

Lnaught, 13, 43

— M —

Mask synthesis, 9, 38

`mdf` commands, 31

MGC_HOME *see* CALIBRE_HOME, 11

Minimum keyword, 10

Modeling, 9, 32

`modelpath` setup command, 55

Models, 55

— O —

OPC, 38

Optical models, 55

— P —

Parentheses, 11

Phase transition, 13, 34, 59

Pipes, 11

Prerequisites, 11

Process technology, 12

— Q —

Quotation marks, [11](#)

— R —

Radius of gyration, [33](#)

Resist models, [55](#)

Round structures, [33](#)

— S —

Setlayer commands

 cdsa_simulator, [56](#)

 cdsa_synthesizer, [57](#)

Slanted words, [10](#)

Square parentheses, [10](#)

Synthesis, [57](#)

Synthesis product, [9](#)

— T —

Tcl package, [41](#)

TVF, [41](#)

— U —

Underlined words, [10](#)

Usage syntax, [10](#)

— V —

Verification, [9](#), [40](#), [56](#)

Vias, [33](#)

— W —

Workflow, [12](#)

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the *<your_software_installation_location>/legal* directory.

