

SIEMENS EDA

Calibre® nmMPC™ and Calibre® nmCLMPC User's and Reference Manual

Software Version 2021.2

SIEMENS

Unpublished work.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	
Introduction to Calibre nmMPC	
and Calibre nmCLMPC.....	13
Calibre nmMPC Overview	13
Calibre nmMPC Workflow	14
Calibre nmMPC Process Flow.....	14
Calibre nmMPC Prerequisites	15
Calibre nmMPC Operations.....	16
Calibre nmDRC-H Command Line Syntax.....	19
Considerations for Calibre FullScale Runs	20
Syntax Conventions	22
Chapter 2	
Generate a Test Layout	23
Test Layout Prerequisites.....	23
Test Layout Generation	23
Chapter 3	
Create and Calibrate Models.....	27
MPC Modeling Prerequisites.....	27
MPC Modeling Overview	29
Proximity Modeling Based on VEB	30
Model Creation and Calibration Process	31
Stage 1 - Long Range Density Convolve	32
Loading the Gauge File	32
Configuring Long Range Optimization	33
Configuring Optimization Parameters	34
Input Model or Layer File	35
Stage 2 - Build Uniformity Polynomial	36
Stage 3 - Remove LR Effects From Short Range Data	38
Stage 4 - Build Short Range Model	40
Loading Corrected Short Range Measurements.....	40
Configuring the VEB Model Form.....	41
Configuring VEB Optimization Parameters.....	42
Input Model or Layer File and Build Final Short Range Model	43
Stage 5 - Build the E-Beam Model.....	45
Configuring and Optimizing an E-Beam Model	45
Additional Modeling Operations in MPC Center.....	46
MPC Center Menu Quick Reference	48

Chapter 4

Correct and Verify the Design.....	51
Correction Prerequisites.....	51
The Calibre nmMPC Setup File.....	53
Import and Define Models	56
Define Layers	58
Define an Image	62
Set Iterations	62
Set Feedback for Convergence	62
Deactivate Additional Fragmentation.....	63
Set Mask Constraints	63
Create a No-Correction Region	63
Smooth Biased Target Layers.....	64
SEM Box Correction	66
Tag Scripting.....	70
Correction for Curvilinear Layouts	70
Optional E-Beam Model Definition	71
How Calibre nmMPC Measures EPE	72
Long-Range Correction Process	73
Short Range (Etch and E-Beam) Correction Process	74
Verify Corrections	81
Verifying the Corrections With Calibre OPCverify	81
Verifying EPE Layers	84

Chapter 5

Calibre nmMPC Reference	89
Test Layout Generation Commands	92
mpcCompile	93
ptpCompile	94
ptpCompile Input File Format	96
uniCompile	103
uniCompile Input File Format	104
mtpCompile	105
mtpCompile Input File Format	107
MPC Calibration Batch Commands	113
mpcggConvert/mpcssConvert.....	114
mpclr.....	115
mpclropt.....	119
mpceval.....	122
Long-Range Model File Format	124
mpcfloow_v2	126
Model-Layer File Format	131
MPC Correction Commands	133
LITHO MPC.....	135
LITHO CLMPC	136
denseopc_options (for MPC)	138
direct_input.....	139
ebeam_model_load.....	141

Table of Contents

etch_model_load	145
layer	146
modelpath	147
processing_mode	148
setlayer ebeam_simulate	149
setlayer curve	151
setlayer curve_target	161
setlayer mpc	167
setlayer clmpc	169
MPC denseopc_options Commands	170
algorithm	174
allow_jog_full_move	175
background (for denseopc_options block)	176
check_movement	177
convex_concave_adj_len	178
convex_concave_correction	179
convex_concave_count	180
convex_concave_limit	181
convex_concave_metric	182
corner_control	183
epe_spacing	184
feedback	185
fragment_coincident	186
fragment_corner	188
fragment_flaglayer	197
fragment_inter	198
fragment_interlayers	212
fragment_island	214
fragment_layer	218
fragment_layer_order	220
fragment_max	222
fragment_min	225
fragment_minjog	226
fragment_nearly_coincident	227
fragment_visible	230
fragment_small	231
freeze_skew_edges	232
grids_for_angle_45_snap	233
image	234
image_options	236
jog_freeze	237
jog_ignore_metric	238
layer (denseopc_options Command)	240
line_end_fragment	242
max_angle_tolerance (LITHO CLMPC Only)	243
max_iterations	244
max_smoothing_range (LITHO CLMPC Only)	245
max_iter_movement	246
max_opc_move	247

measure_metric_mode	248
mpc_contour_search_radius	249
mpc_fragment_type_projection	250
mpc_poi_measurement (LITHO CLMPC Only)	251
mpc_preserve_angles (LITHO CLMPC Only)	252
mpc_pseudo_nyquist	253
mpc_sites_control	254
mpc_sites_mode	255
mpc_skip_fragments	257
mpc_use_nearest_epe	258
mrc_rule	259
no_opc_region	266
opc_grid_multiplier	267
retarget_layer	268
scale	272
sparse_ebeam_veb	273
step_size	274
tile_fragmentation_only	275
version	276
MPC Custom Tcl Scripting Commands	277
DISPLACEMENT	280
FEEDBACK	285
fragalign	287
FRAGMENT delete	290
FRAGMENT modify	291
FRAGMENT split	292
FRAGMENT_EPE_MEASURE_METHOD	294
FRAGMENT_MOVE	295
MRC	297
MRC_RULE (Custom Scripting Command)	299
NEWTAG all	301
NEWTAG blocked	302
NEWTAG cd	303
NEWTAG complete	305
NEWTAG displacement	307
NEWTAG edge	309
NEWTAG epe	313
NEWTAG external internal enclose enclosing	315
NEWTAG fragment	319
NEWTAG line_end space_end	324
NEWTAG neighbor	325
NEWTAG sequence	326
NEWTAG topological	328
NEWTAG vertical horizontal all_angle 45_angle	330
OPC_ITERATION	331
OUTPUT_MEASUREMENT_LOCATIONS	333
OUTPUT_OP_C	334
OUTPUT_SHAPE fragment	336
TAG and or subtract	339

Table of Contents

Calibre nmMPC File Formats	340
Lithomodel (Litho Model Format)	341

Appendix A

Calibre nmMPC in Calibre nmModelflow **343**

Key Concepts for Calibre nmModelflow	343
Calibre nmMPC in Calibre nmModelflow Workflow	344
Creating a Litho Model in Calibre nmModelflow	345
Loading Gauge Data into Calibre nmModelflow	349
Creating a Stage for Calibre nmMPC	350
E-Beam Calibration Stage	352
Etch Calibration Stage	353
Creating a Calibration Job in Calibre nmModelflow (Calibre nmMPC)	354

Index

Third-Party Information

Table of Contents

List of Figures

Figure 1-1. Example Mask Process Correction Workflow	15
Figure 2-1. Example MPC Test Pattern.....	24
Figure 3-1. Model Calibration Flow	29
Figure 3-2. Gauge Object in MPC Center	33
Figure 3-3. LR Tab (Add Kernels)	34
Figure 3-4. Optimize Tab (LR Sub-Tab Displayed)	35
Figure 3-5. Misc Tab (Long Range)	36
Figure 3-6. Example Mask Layer File (.mlf) for Long Range Model.....	36
Figure 3-7. LR Tab (BTERMS).....	37
Figure 3-8. SVRF Tab	38
Figure 3-9. Gauges Tab (LR Try)	39
Figure 3-10. Gauge Object.....	41
Figure 3-11. VEB Tab	41
Figure 3-12. Optimize Tab (VEB Sub-Tab Displayed).....	42
Figure 3-13. Misc Tab (VEB)	43
Figure 3-14. Example Mask Layer File (.mlf) for VEB Model.....	44
Figure 3-15. Proximity Model File Example.....	44
Figure 3-16. E-Beam Model Configuration.	45
Figure 3-17. Contour Tab	47
Figure 3-18. Remote Tab.....	47
Figure 4-1. Minimum Calibre nmMPC Setup File With an E-Beam Model.....	53
Figure 4-2. Minimum nmMPC Setup File With an Etch Model Only	54
Figure 4-3. Standard Calibre Gauge Measurement	66
Figure 4-4. SEM Box Method	66
Figure 4-5. MPC Output Comparison With SEM Box Method	68
Figure 4-6. Standard EPE Measurement	68
Figure 4-7. EPE Evaluation Points Added.	70
Figure 4-8. Example SVRF Block Generated from MPC Center	74
Figure 5-1. Example ptpCompile Output Pattern	94
Figure 5-2. line_space Example.....	97
Figure 5-3. post Example.....	97
Figure 5-4. contact Example	98
Figure 5-5. line_end Example	98
Figure 5-6. space_end Example.....	99
Figure 5-7. single_line_end Example	99
Figure 5-8. single_space_end Example	100
Figure 5-9. fidelity Example	100
Figure 5-10. chess Example.....	101
Figure 5-11. Example uniCompile Output Pattern	103
Figure 5-12. Example Mask Model	105

Figure 5-13. Example mtpCompile Patterns	106
Figure 5-14. Setting an Origin Point in the Coordinate System	108
Figure 5-15. Example of Line/Space End Scaling.	157
Figure 5-16. Length and Width Representation for setlayer curve	159
Figure 5-17. Effects of retarget_layer emulate and setlayer curve	160
Figure 5-18. Effect of colinearAngleTolerance 1	162
Figure 5-19. Curvature Example	163
Figure 5-20. Comparisons of linearRatio True Versus False	164
Figure 5-21. Changing the Coincident Layer to Affect Fragmentation.	187
Figure 5-22. Intrafeature Fragmentation Example.	193
Figure 5-23. corner_next and length_next.	196
Figure 5-24. fragment_inter -minshield 2 -shield 4 -ripples 1	200
Figure 5-25. Example -ripplestyle	203
Figure 5-26. Interfeature Shielding Example	204
Figure 5-27. -split Example	205
Figure 5-28. Using Shield With Interfeature Fragmentation.	206
Figure 5-29. -skipCorners Option	209
Figure 5-30. Typical Fragmentation Caused by a Scattering Bar	209
Figure 5-31. Symmetric Fragmentation Caused by a Scattering Bar and -sym	210
Figure 5-32. Effect of -combined	211
Figure 5-33. Without -dontcross Option	215
Figure 5-34. With -dontcross Option	215
Figure 5-35. Island Layer Fragmentation Example	217
Figure 5-36. fragment_max	224
Figure 5-37. jog_ignore_metric Behavior	238
Figure 5-38. Site Placement Using mpc_sites_mode	255
Figure 5-39. The Four Metrics.	262
Figure 5-40. Effects of retarget_layer emulate	269
Figure 5-41. EPE Measurements at Corners (Example 1)	270
Figure 5-42. EPE Measurements at Corners (Example 2)	271
Figure 5-43. The Four Metrics for Tag-Based MRC.	300
Figure 5-44. Corner and Len Definitions.	310
Figure 5-45. Effect of -perpendicular_only and -corner_block	316
Figure 5-46. Example of mcorner and rdisp	322
Figure 5-47. Output Per Iteration Example	334
Figure A-1. Flow Stage Wizard: E-Beam Calibration.	352
Figure A-2. Flow Stage Wizard: Etch Calibration	353

List of Tables

Table 1-1. Mask Process Correction Stages	14
Table 1-2. Related Products and Their Manuals	16
Table 1-3. Syntax Conventions	22
Table 3-1. Model Creation and Calibration Process	31
Table 3-2. Long Range Density Convolve Process Summary	32
Table 3-3. Build VEB Short Range Model Process Summary	40
Table 3-4. MPC Center Menu Summary	48
Table 4-1. Calibre nmMPC Setup File Summary	55
Table 4-2. Layer Descriptions	60
Table 4-3. SEM Box Commands Defaults	67
Table 5-1. Test Layout Generation Commands	92
Table 5-2. MPC Calibration Batch Commands	113
Table 5-3. Mask Process Correction Commands	133
Table 5-4. General Recommended Settings for setlayer curve	156
Table 5-5. Line/Space End-Specific Recommended Settings for setlayer curve	158
Table 5-6. Recommended Jog-Specific Settings for setlayer curve	159
Table 5-7. MPC denseopc_options Summary	170
Table 5-8. Available Flag Names	197
Table 5-9. Supported Fragmentation Keywords in fragment_layer Block	218
Table 5-10. Allowed Fragmentation in a fragment_layer Command	221
Table 5-11. Examples of Fragment Splitting	223
Table 5-12. layer Command layer_types (denseopc_options)	240
Table 5-13. Calibre nmMPC Fragment Command Defaults	272
Table 5-14. nmMPC Custom Scripting Commands Summary	277
Table 5-15. Calibre nmOPC File Formats	340
Table A-1. Calibre nmMPC in Calibre nmModelflow Steps	344
Table A-2. Flow Stage Wizard Choices for Calibre nmMPC Mode	351

Chapter 1

Introduction to Calibre nmMPC and Calibre nmCLMPC

Calibre® nmMPC™ is a suite of functions for modeling, simulating, and correcting distortions of the mask manufacturing process. Models generated by this process are utilized by Calibre nmMPC and FRACTURE tools to accurately reflect the final output of the mask process.

Calibre® nmCLMPC utilizes functions similar to Calibre nmMPC, but is targeted specifically for curvilinear layouts (layouts with skew-edge shapes), known as curvilinear mask process correction or CLMPC. Calibre nmCLMPC follows the same workflow as Calibre nmMPC with exceptions that are highlighted in this document.

Calibre nmMPC Overview	13
Calibre nmMPC Workflow.....	14
Calibre nmMPC Process Flow	14
Calibre nmMPC Prerequisites	15
Calibre nmMPC Operations.....	16
Calibre nmDRC-H Command Line Syntax	19
Considerations for Calibre FullScale Runs.....	20
Syntax Conventions	22

Calibre nmMPC Overview

Calibre nmMPC accounts for errors that occur during the mask process, improving mask Critical Distance (CD) uniformity and linearity. Total mask CD error is typically a combination of proximity effects (referred to as short range effects) and long range effects:

- Proximity Errors (Etch and E-beam Effects)

In the range of 0 to 1 microns, proximity errors are caused by finite patterning resolution, resist, and etch effects. A special class of short-range effects, known as E-beam effects, are typically triggered by forward scattering and beam blur.

- Long-Range Effects

In the range above 50 microns, long-range CD errors are caused by process non-uniformity and pattern density-based effects (which includes electron fogging, and so on).

Calibre nmMPC Workflow

The Mask Process Correction (MPC) works along with the Optical Process Correction (OPC) workflow. Different models are generated to account for mask process effects (etch, e-beam, and long range effects) and the models are then used by Calibre to make corrections to the design layout, much as the OPC process uses models to account for optical, resist, and etch effects on the wafer.

The workflow can be summarized as follows:

1. The basic Calibre nmMPC flow starts by building a mask model from mask test data.
2. Calibre nmMPC corrects the MPC test pattern used to create the MPC model.
3. MPC is run using the MPC model for simulation.
4. Finally, the MPC output mask target is corrected using the MPC model to produce the mask writer input.

Calibre nmMPC Process Flow

The overall mask process correction flow for Calibre nmMPC can be summarized in three basic stages.

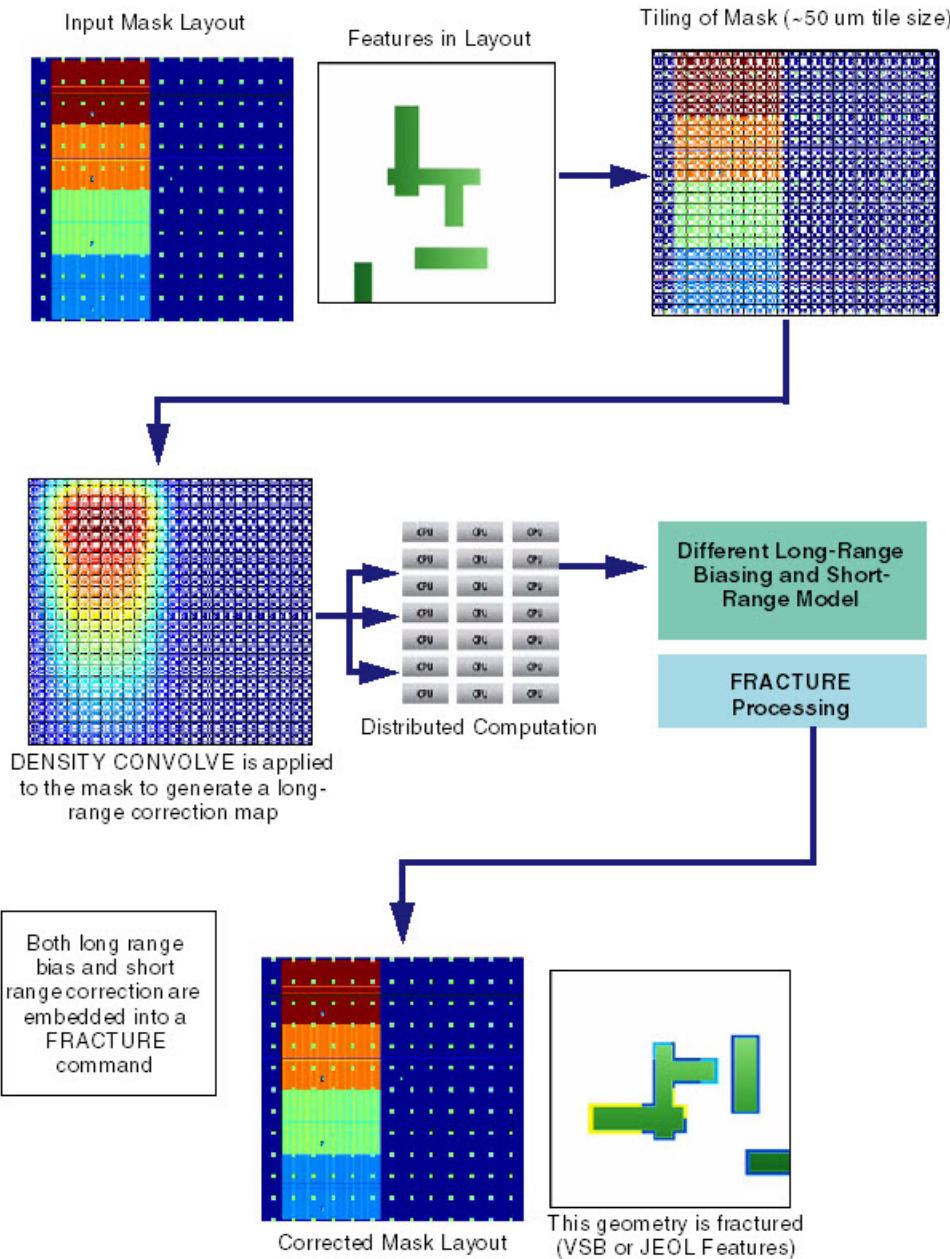
The basic stages are described in [Table 1-1](#).

Table 1-1. Mask Process Correction Stages

Stage	Process	Description
1	Generate a Test Layout	Create a test mask with initial data models (short range and long range) and test patterns.
2	Create and Calibrate Models	Extract calibrated models from the measured data.
3	Correct and Verify the Design	Use the modeling information to simulate the mask process and apply corrections (edge biasing) to the layout.

[Figure 1-1](#) illustrates a typical data flow for both short- and long-range mask process correction. Many variants are possible depending on the particular application context and mask process used.

Figure 1-1. Example Mask Process Correction Workflow



Calibre nmMPC Prerequisites

Before you begin the Calibre nmMPC process, there are several prerequisites.

- **Platform support** — Calibre MDP is available on all supported platforms found in the [Calibre Administrator's Guide](#). Refer to that document for instructions on how to install Calibre software.

- **Licensing** — You must have valid licenses for the following:
 - Calibre® FRACTURE formats and MDP utilities.
 - Calibre® WORKbench™ (Version 2008.4 or higher).
 - Calibre® nmMPC™ for Calibre nmMPC correction.
 - Calibre® nmCLMPC for Calibre nmCLMPC correction.
 - Calibre® MPCVerify for mask-based verification. Calibre® RVE™ is also recommended to view Calibre MPCVerify histograms.

For more information on licensing, refer to the [Calibre Administrator's Guide](#).

Calibre nmMPC Operations

The MPC flow touches on several Calibre tools.

These include the following:

- The Tcl batch script parser available in Calibre WORKbench to invoke utilities that generate test pattern layouts and gauge files, and calibrate the long-range model. Refer to the [Calibre WORKbench User's and Reference Manual](#) for complete information on how to run Calibre WORKbench.
- MPC Center in Calibre WORKbench to calibrate the long range and proximity models from data in the gauge files.
- Calibre nmDRC-H to process the final SVRF file, including an MDP FRACTURE block.

Table 1-2 lists the documents associated with the related Calibre tools.

Table 1-2. Related Products and Their Manuals

Related Products	Documentation
All post-tapeout products	Calibre Post-Tapeout Flow User's Manual

Table 1-2. Related Products and Their Manuals (cont.)

Related Products	Documentation
Calibre® FRACTUREc™ Calibre® FRACTUREh™ Calibre® FRACTUREj™ Calibre® FRACTUREm™ Calibre® FRACTUREt™ Calibre® FRACTUREv™ Calibre® MDPmerge™ Calibre® MDPstat™ Calibre® MDPverify™ Calibre® MPCpro™ Calibre® MASKOPT™ Calibre® MDP Embedded SVRF	<i>Calibre Mask Data Preparation User's and Reference Manual</i> <i>Calibre Release Notes</i>
Calibre® MPCverify	<i>Calibre MPCverify User's and Reference Manual</i>
Calibre® MDPview™	<i>Calibre MDPview User's and Reference Manual</i> <i>Calibre DESIGNrev Layout Viewer User's Manual</i> <i>Calibre Release Notes</i>
Calibre® Interactive™ Calibre® RVE™	<i>Calibre Interactive User's Manual</i> <i>Calibre RVE User's Manual</i>
Calibre® nmDRC™ Calibre® nmDRC-H™	<i>Calibre Release Notes</i> <i>Calibre Verification User's Manual</i> <i>Standard Verification Rule Format (SVRF) Manual</i>
Calibre® WORKbench™	<i>Calibre WORKbench User's and Reference Manual</i>
Tcl/Tk Batch Commands	<i>Calibre DESIGNrev Reference Manual</i>
Calibre® Metrology API (MAPI)	<i>Calibre Metrology API (MAPI) User's and Reference Manual</i>
Calibre® Metrology API (MAPI)	<i>Calibre Metrology Interface (CMi) User's Manual</i>
Calibre® Job Deck Editor	<i>Calibre Job Deck Editor User's Manual</i>
Calibre® MDPDefectAvoidance™	<i>Calibre MDPDefectAvoidance User's Manual</i>

Table 1-2. Related Products and Their Manuals (cont.)

Related Products	Documentation
Calibre® MPCVerify	<i>Calibre MPCVerify User's and Reference Manual</i>
Calibre® DefectReview™	<i>Calibre DefectReview User's Manual</i>
Calibre® MDPAutoClassify™	<i>Calibre MDPAutoClassify User's Manual</i>
Calibre®DefectClassify™	<i>Calibre DefectClassify User's Manual</i>

Calibre nmDRC-H Command Line Syntax

The correction stage of Calibre nmMPC runs in batch mode, which requires an SVRF file to be processed by the Calibre nmDRC-H hierarchical database engine. The processing is invoked from a shell tool command line.

Usage

```
calibre_path/calibre {-drc -hier | -pto} [-turbo [number_of_processors]] [-turbo_all]  
[-turbo_litho [number_of_processors]] [-remotefile filename] svrf_filename
```

Arguments

- *calibre_path*
Specifies the path to your Calibre executables.
- **-drc -hier**
Selects hierarchical nmDRC checking.
- **-pto**
Enables Calibre® FullScale™, a separately licensed Calibre engine that improves scalability and runtime for typical post-tapeout designs. See “[Considerations for Calibre FullScale Runs](#)” on page 20 for information.
- **-turbo [number_of_processors]**
Instructs Calibre nmDRC-H to use multithreaded parallel processing.

The optional *number_of_processors* argument is a positive integer that specifies the number of CPUs to use. If you do not specify a *number_of_processors*, Calibre nmDRC-H runs on the maximum number of CPUs available for which you have licenses. In general, you should avoid specifying *number_of_processors*.

Calibre nmDRC-H runs on the maximum number of CPUs available if you specify a number greater than the maximum available. For example:

```
calibre -drc -hier ... -turbo 3 ...
```

operates on two processors for a 2-CPU machine.

See “License Consumption for Distributed Calibre” in the *Calibre Administrator’s Guide* for additional information about license scaling with CPU count.

- **-turbo_all**
This option is used with the -turbo option. This option halts Calibre tool invocation if the tool cannot obtain the exact number of CPUs you specified.

For example:

```
calibre -drc -hier -turbo -turbo_all rule_file
```

executed on an 8-CPU machine for a hierarchical DRC MT run is the same as specifying this:

```
calibre -drc -hier -turbo 8 -turbo_all rule_file
```

Without -turbo_all, the Calibre tool normally uses fewer threads than requested if the requested number of licenses or CPUs is unavailable.

See “License Consumption for Distributed Calibre” in the *Calibre Administrator’s Guide* for additional information about license scaling with CPU count.

- **-turbo_litho [number_of_processors]**

Specifies the number of processors to use for RET and MDP applications. May only be specified with -turbo.

The optional *number_of_processors* argument is a positive integer that specifies the number of CPUs to use for RET and MDP processes. If you do not specify *number_of_processors*, Calibre runs on the maximum number of CPUs available for which you have licenses, regardless of the -turbo setting.

You can specify the -turbo and the -turbo_litho options concurrently in a single command line and the respective *number_of_processors* arguments can vary between the two options.

- **-remotefile filename**

This option is part of the MTflex multithreaded, parallel processing architecture, which enables multithreaded operation on remote hosts of a distributed network. The remote hosts do not have to be of the same platform type when using this option. It must be specified in conjunction with the -turbo and -turbo_litho option, which specifies the number of processors you are using, including those on the remote hosts. The filename specifies the pathname of a configuration file containing information for the local and remote hosts. You must have the required number of licenses for your job.

This option applies only to hierarchical applications. For more details, see the *Calibre Administrator’s Guide*.

- **svrf_filename**

Specifies the SVRF filename.

Considerations for Calibre FullScale Runs

Calibre nmMPC can be run on the Calibre® FullScale™ platform. However, there are some limitations.

Calibre FullScale Overview

Calibre FullScale is a separately licensed Calibre engine that improves scalability and runtime for typical post-tapeout flows. Invoke Calibre FullScale with “calibre -pto” instead of “calibre -drc -hier.”

Edge results may differ between -pto and -drc -hier runs due to skew edges crossing section boundaries. The skew edges can be from either the input or the intermediate layers. For example, operations such as External REGION can generate non-Manhattan edges.

Calibre FullScale does not support all SVRF operations¹. If a -pto run is started with a rule file that contains unsupported operations, the run exits with an error “Forbidden operations:” followed by the unsupported operations. Generally, Boolean and topological DRC operations are supported, and a subset of the DFM, Litho, RET, and MDP keywords. See “[Calibre FullScale SVRF Support](#)” in the *Calibre Post-Tapeout Flow User’s Manual* for a complete list.

Calibre FullScale cannot be used for LVS, PERC, or parasitic extraction. Calibre FullScale is incompatible with hyperscaling (-hyper).

Changes to Transcript

The Calibre FullScale platform produces a different transcript than the Calibre hierarchical engine. The transcripts also include all the information traditionally reported by the Calibre hierarchical engine. For more details on interpreting the transcript, see “[Interpreting the Calibre Transcript for PTO Applications](#)” in the *Calibre Post-Tapeout Flow User’s Manual*.

Hardware Requirements

The number of remote hosts can stress the primary host. The memory load can be distributed across the cluster by using remote data servers (RDS). When you use a cluster, you must invoke Calibre FullScale with -remotedata. 8 GB per core is recommended. See “[Calibre FullScale Platform](#)” in the *Calibre Post-Tapeout Flow User’s Manual* for more information on RDS.

RDS benefits from high bandwidth. To avoid saturating the network, the bandwidth should be greater than 1 Gbit/sec.

Calibre nmMPC in Calibre FullScale

Calibre nmMPC rule files may require some changes to run when run on the Calibre FullScale platform. Check the rule file for these operations and keywords:

- The following Calibre nmMPC setlayer commands are supported in Calibre Fullscale:
 - setlayer curve
 - setlayer ebeam_simulate
 - setlayer mpc

1. Most SVRF specifications are supported. Operations are functions which, when evaluated, create a derived layer.

Syntax Conventions

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

Table 1-3. 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.
‘ ’	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:	
<pre>DEvice {element_name ['(model_name)']} device_layer {pin_layer ['(pin_name)'] ...} ['<auxiliary_layer>' ...] ['(swap_list)' ...] [BY NET BY SHAPE]</pre>	

Chapter 2

Generate a Test Layout

The first stage of a Mask Process Correction flow is generating a test mask with initial data models to measure long range and proximity effects. The calibration mask contains the test structures needed to calibrate the models.

Test Layout Prerequisites	23
Test Layout Generation.....	23

Test Layout Prerequisites

Before you begin this stage, you must have the prerequisite information available as input.

- Specification files:
 - Proximity test pattern specification file (for example, proximity.in): This is used to generate proximity model (short-range) test layouts, consisting of calibration structures, validation structures, and corner rounding model calibration structures. Refer to “[ptpCompile Input File Format](#)” on page 96 for complete details.
 - Uniformity test pattern specification file (for example, uniformity.in): This is used to generate a uniformity test mask layout and corresponding gauge output. This file uses the same format as the proximity file. Refer to “[uniCompile Input File Format](#)” on page 104 for complete details.
 - Mask test pattern specification file (for example, example.in): Generates mask (short-range and long-range) test layouts, which consists of proximity, density loading and uniformity layouts. Refer to “[mtpCompile Input File Format](#)” on page 107 for further information.

Test Layout Generation

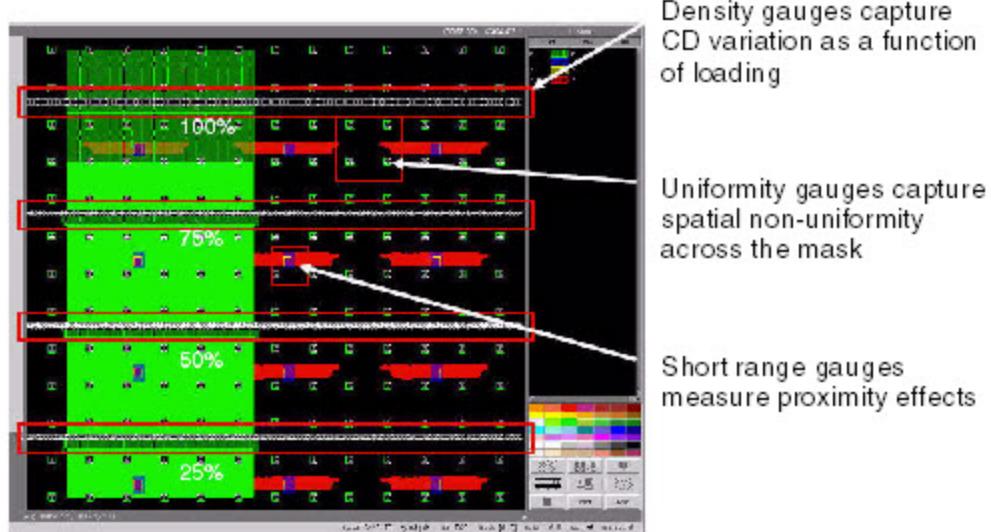
To generate a test layout, run the mpcCompile Tcl command.

```
cwb% mpcCompile example.in proximity.in uniformity.in
```

This is equivalent to running the [ptpCompile](#), [uniCompile](#), and [mtpCompile](#) commands in succession (you can alternatively run each one of these commands individually).

In these layouts, by default, layer 0 of the layout contains all features, layer 1 contains all text notations, layer 2 contains the border, and layer 3 contains information for measurements recorded in the gauge output. [Figure 2-1](#) shows an example test calibration mask layout.

Figure 2-1. Example MPC Test Pattern



Once the compilation is complete, the test pattern can be used to measure features on the mask and the resulting measurements output in gauge files. After that is accomplished, the data in the gauge files may be used for model calibration.

When the process is complete, the test layout generation tool generates the following:

- A test mask layout (OASISTM¹ file)
- Several output gauge files (with a .gg suffix) containing the corresponding measurement locations and parameters of the mask layout:
 - Proximity pattern test structures
 - Density loading pattern test structures
 - Uniformity pattern test structures

The output gauge files use the same format used by OPC (for detailed info about the gauge format, please refer to the [Calibre WORKbench User's and Reference Manual](#)). The basic syntax is:

```
# Flag Struct Row Col X1 Y1 X2 Y2 Loc Drawn Other Meas Simul MEtch SEtch
Prec Weight Stdev Count 95%ci
```

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.

where, briefly summarized:

- **Flag** specifies the activate or deactivate flag.
- **Struct** specifies the name of the measured feature.
- **Row** and **Col** specifies the location on mask of measured feature.
- **X1, Y1, X2, and Y2** specify the coordinates of the measurement locations.
- **Loc** is set to 0 or 1, where 0 means measuring the pattern width, and 1 means measuring the space.
- **Drawn** is the measured feature CD.
- **Other** means some other useful info.
- **Meas** represents the actual measured CD.
- **Simul** is the simulated CD.
- **MEtch** is measured CD for etch data.
- **SEtch** is simulated CD for etch data.
- **Weight** is the priority given to balance the measurement among locations.
- **Stdev** sets the standard deviation in nm for characterizing measurement statistics.
- **Count** represents number of samples used to obtain the **Stdev** value.
- **95%ci** contains the calculated confidence interval based on the following equation:

$$95\%ci = t\text{-value} * stdev / \sqrt{count}$$

where t-value is a two-tailed Student's probability of 0.05 with count-1 degrees of freedom, and **Stdev** and **Count** are the input values supplied. Refer to the [Calibre WORKbench User's and Reference Manual](#) for details.

Chapter 3

Create and Calibrate Models

The second stage of the Calibre nmMPC flow is model creation and calibration.

After the test mask is measured, several distinct models must be generated: proximity (short-range) models, and a long-range model. At the conclusion of this stage, you should have the following:

- Calibrated MPC models that include a short range (Variable Bias Etch), E-beam, and a long range model. This document will cover the calibration for the each of these models.
- A generated SVRF block to insert into the FRACTURE file.

MPC Modeling Prerequisites	27
MPC Modeling Overview	29
Proximity Modeling Based on VEB	30
Model Creation and Calibration Process	31
Stage 1 - Long Range Density Convolve	32
Stage 2 - Build Uniformity Polynomial	36
Stage 3 - Remove LR Effects From Short Range Data	38
Stage 4 - Build Short Range Model	40
Stage 5 - Build the E-Beam Model.....	45
Additional Modeling Operations in MPC Center.....	46
MPC Center Menu Quick Reference.....	48

MPC Modeling Prerequisites

Before starting the modeling stage, there are several prerequisites.

- Calibration mask layouts generated from the “[Generate a Test Layout](#)” stage.
- Gauge files generated from the “[Generate a Test Layout](#)” stage, filled in with actual measurement data.
- A Model-Layer File (.mlf), one for each of the models (Long Range and Short Range). A model-layer file is a type of setup file specifically for entering parameters to generate a model.
 - For Long Range (LR) models, it contains the specification of the model form to be calibrated, input layer information, and the window size for density calculation.
 - For Short Range models, it contains the input layer information.

The Model-Layer File format is documented in the *Calibre WORKbench User's and Reference Manual*.

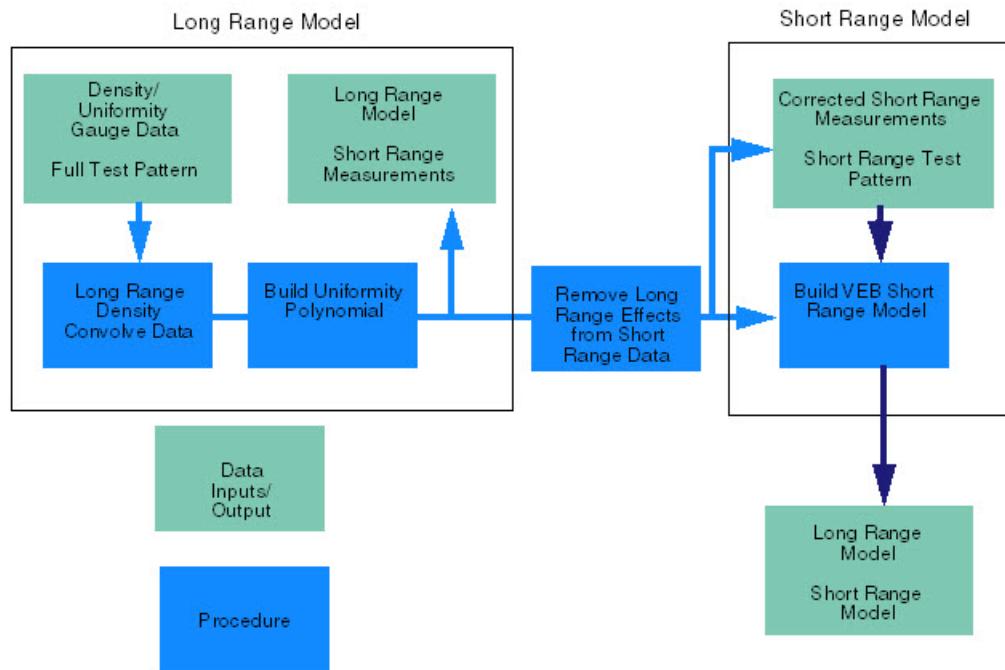
MPC Modeling Overview

Calibre nmMPC uses a modified version of the Calibre WORKbench test pattern engine to create its proximity (density) and uniformity (long-range) patterns, which are compiled into a single composite test pattern. The tools designed for Calibre nmMPC output gauge data files, each of which must be filled in according to the results of measuring a test chip on the structures.

After the test mask is measured, generate the models, which can be short range models (etch or E-beam), and a long-range model.

Figure 3-1 illustrates the overall model calibration flow for both long range and short range models.

Figure 3-1. Model Calibration Flow



You are going to use the MPC Center to separate the long range effects from the proximity data (both are present in the gauge files with measurement data), since the modeling of each of those effects is affected by the presence of the data from the other data type.

- The goal of modeling the long term effects is the production of a modeling polynomial and a Gaussian parameter vector that can be used with the SVRF command DENSITY CONVOLVE.
- The goal of modeling the proximity effects is a VEB model that can be used with Calibre nmMPC. Refer to the following section, “[Proximity Modeling Based on VEB](#)” for further details.

Proximity Modeling Based on VEB 30

Proximity Modeling Based on VEB

The largest sources of mask systematic error are treated with a Variable Etch Bias (VEB) model. Rigorous “first-principles” based modeling of etch effects are typically too complex for full-chip type applications.

There are two primary pattern-based influences on etch rates that VEB attempts to model:

- Density based effects - Gaussian kernels convolved with pattern density
- Aperture based effects - Visible kernels integration of etch feature opening

A Variable Etch Bias model describes the bias as a function of local density and visibility.

- Shifted Gaussian convolution - Single density determined by Gaussian convolution
- Visible kernels - Directly characterizes line width or space
- VEB models position-dependent bias relative to the target mask edge
- Model based on OPC etch modeling
 - Etch signature expected to be primary uncorrected mask CD error component
- Calibration supports both 1D and 2D measurements
 - CD measurements
 - 2D CD SEM contour calibration

Model Creation and Calibration Process

Long-range effect models are calibrated and built using a particular sequence.

Table 3-1. Model Creation and Calibration Process

Stage	Task	Description	Data	MPC Model
1	Stage 1 - Long Range Density Convolve	Optimize the Gaussian sigma (kernel size) for a Density Convolve model only.	Density/Uniformity sites with drawn CD = 1000 nm	Long Range Density Convolve
2	Stage 2 - Build Uniformity Polynomial	Build a uniformity polynomial in MPC Center.	Density/Uniformity sites with drawn CD = 1000 nm	Long Range Density Convolve + Spatial Uniformity
3	Stage 3 - Remove LR Effects From Short Range Data	Use the T (Try) option for Long Range models to extract Long Range effects from the Short Range data	Proximity sites	Long Range Density Convolve + Spatial Uniformity
4	Stage 4 - Build Short Range Model	Build the calibrated short range model.	Proximity sites	VEB
5	Stage 5 - Build the E-Beam Model	Build the E-beam model.	Proximity sites	VEB
Optional	Additional Modeling Operations in MPC Center	You can optionally: <ul style="list-style-type: none"> • read in a Contour Layer Information (CLI) file • create a remote host file and implement distributed processing 	Proximity sites corrected for Long Range effects	VEB

Note the following:

- The **DENSITY CONVOLVE** SVRF command calibrates influence of density over a mm scale. Areas influenced include:
 - Fogging (though most writers correct for this)
 - Microloading in etch processes
- “Spacial uniformity with polynomial fit” is a systematic non-uniformity in the mask process (plate uniformity, resist thickness, and so on).

Stage 1 - Long Range Density Convolve

Optimize the Gaussian sigma (kernel size) for a Density Convolve model only.

[Table 3-2](#) summarizes the main phases in the Long Range Density Convolve stage.

Table 3-2. Long Range Density Convolve Process Summary

Phase	Section	Description
1	Loading the Gauge File	Load the test pattern layout (known as a gauge file) into MPC Center in Calibre WORKbench.
2	Configuring Long Range Optimization	Add kernels, uniformity order, and BTERMS for the Long Range model.
3	Configuring Optimization Parameters	Set optimization options in MPC Center for the Long Range model.
4	Input Model or Layer File	Input a model or layer file in MPC Center.

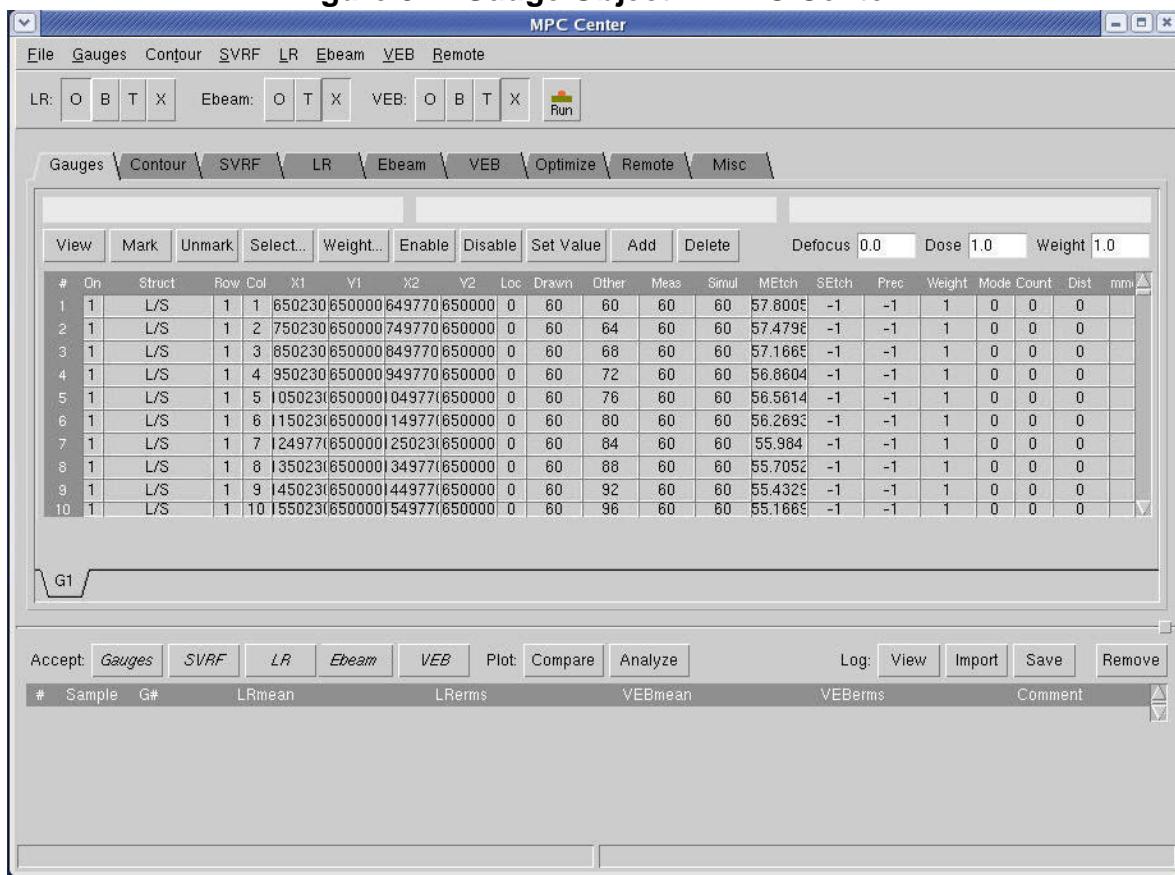
Loading the Gauge File

Load the test pattern layout (known as a gauge file) into MPC Center in Calibre WORKbench.

Procedure

1. Invoke Calibre WORKbench.
2. Load a test pattern layout into Calibre WORKbench. The layout can be hierarchical or flat.
3. In Calibre WORKbench, select **Tools > MPC Center** to open the MPC Center.
4. In MPC Center, select the **Gauges > Open** menu option. Import the proximity data gauge file. It should appear in the Gauges tab in MPC Center ([Figure 3-2](#)).

Figure 3-2. Gauge Object in MPC Center



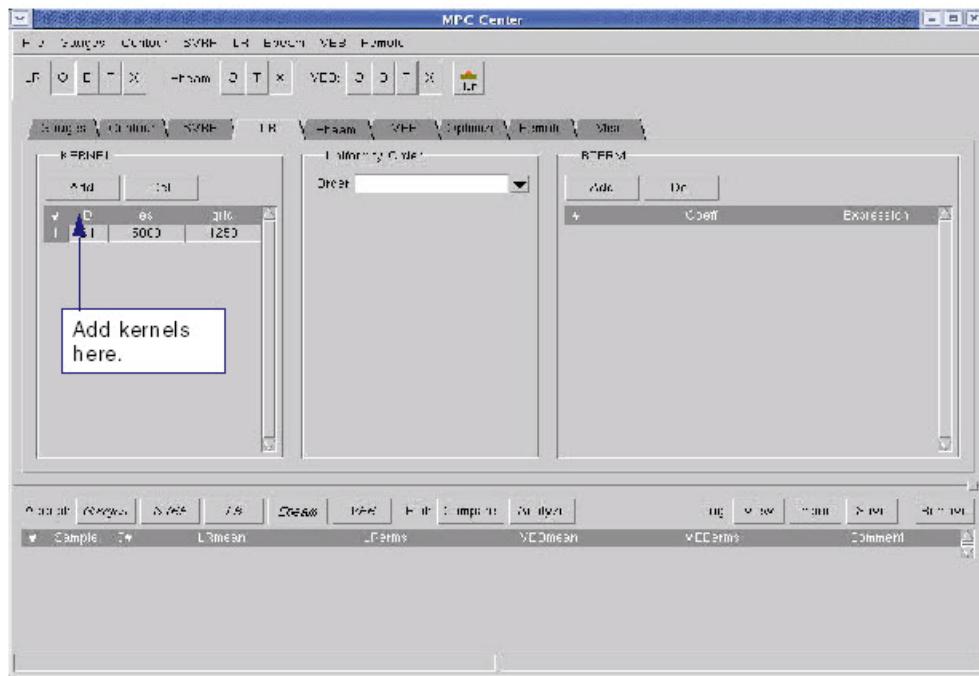
Configuring Long Range Optimization

Add kernels, uniformity order, and BTERMS for the Long Range model.

Procedure

- Click the **LR** tab (see [Figure 3-3](#)).

Figure 3-3. LR Tab (Add Kernels)



2. In the LR page, make sure that the Uniformity Order is set to 0. This will be reset at a later stage.
3. Add Kernels. Click the **Add** button to add Gaussian kernels. You can start with 1 kernel and perform an initial run to check if the model approximates your data. If you require more coefficients for a better fit, you can return and add the second kernel.

This is equivalent to filling in kernel data in the Long Term model file form (see “[Long-Range Model File Format](#)” on page 124).

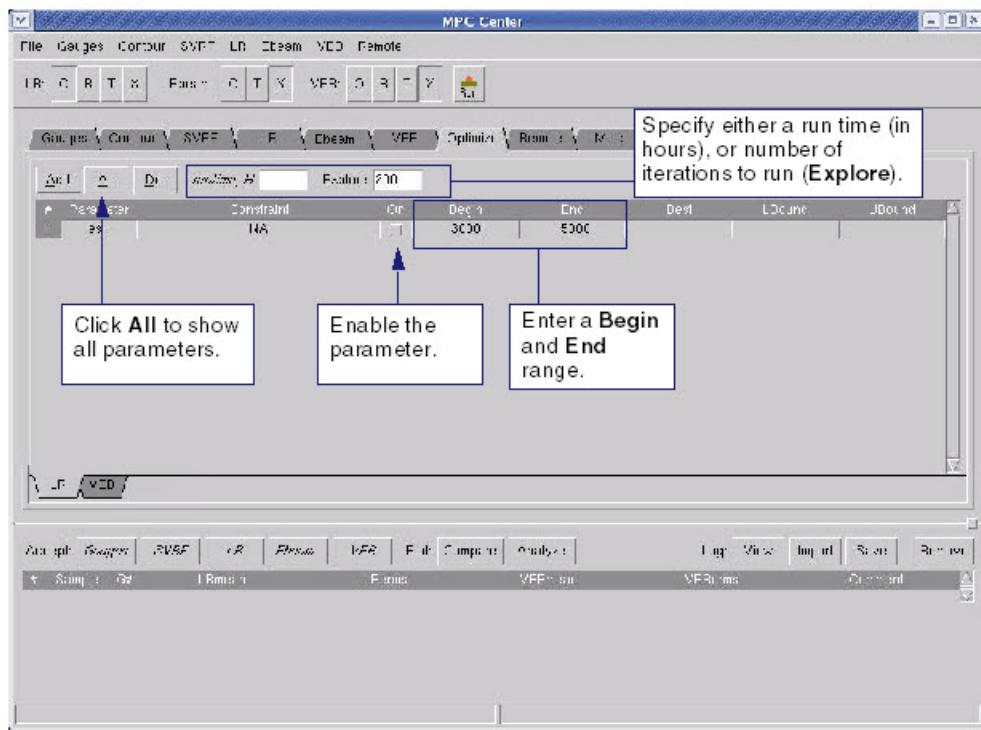
Configuring Optimization Parameters

Set optimization options in MPC Center for the Long Range model.

Procedure

1. Click the **Optimize** tab. In the Optimize page, click the **LR** sub-tab (see [Figure 3-4](#)).

Figure 3-4. Optimize Tab (LR Sub-Tab Displayed)



2. In the **LR** sub-tab in the Optimize page, click **All** to see all the parameters in the gauge file.
3. Enable the parameters you want to optimize clicking the radio button in the **On** column.
4. Set lower and upper bounds by entering **Begin** and **End** ranges for your enabled parameters.
5. Set either the number of iterations to run in the **Explore** field, or a run time value in the **Runtime** field (the **H** stands for number of Hours).

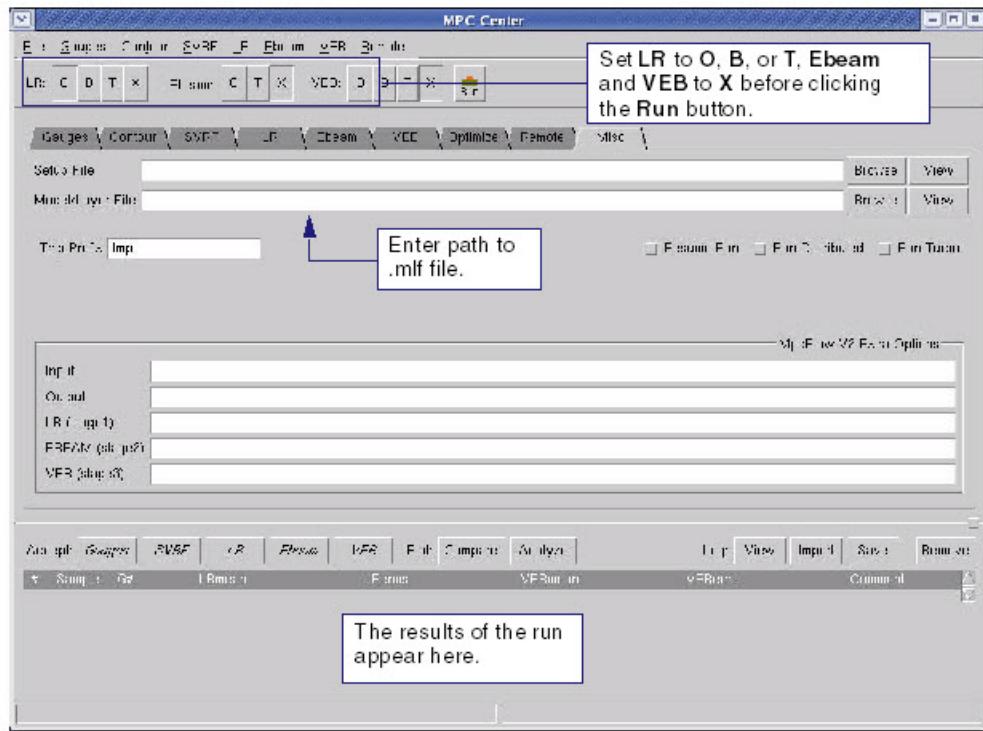
Input Model or Layer File

Input a model or layer file in MPC Center.

Procedure

1. Click the **Misc** tab (see [Figure 3-5](#)).
2. Set **LR** to **O**, **B**, or **T** and **Ebeam**, and **VEB** to **X**.

Figure 3-5. Misc Tab (Long Range)



3. In the **Misc** tab, enter the path to the **.mlf** file (see [Figure 3-6](#) for an example **.mlf** file) in the **Model/Layer File** field.

Figure 3-6. Example Mask Layer File (.mlf) for Long Range Model

```
modelform COMBO
input layer 0
//window 50
window 100
```

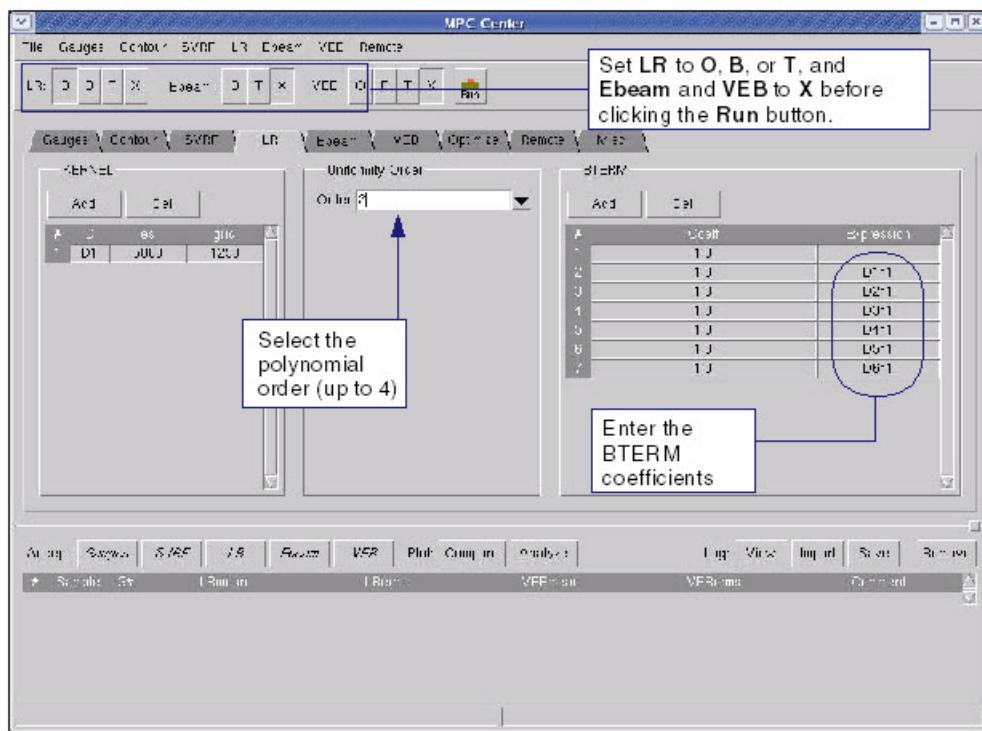
Stage 2 - Build Uniformity Polynomial

Build a uniformity polynomial in MPC Center.

Procedure

1. Click the **LR** tab again (see [Figure 3-7](#)).
2. Set **LR** to **O, B, or T**, and **Ebeam** and **VEB** to **X**.

Figure 3-7. LR Tab (BTERMS)

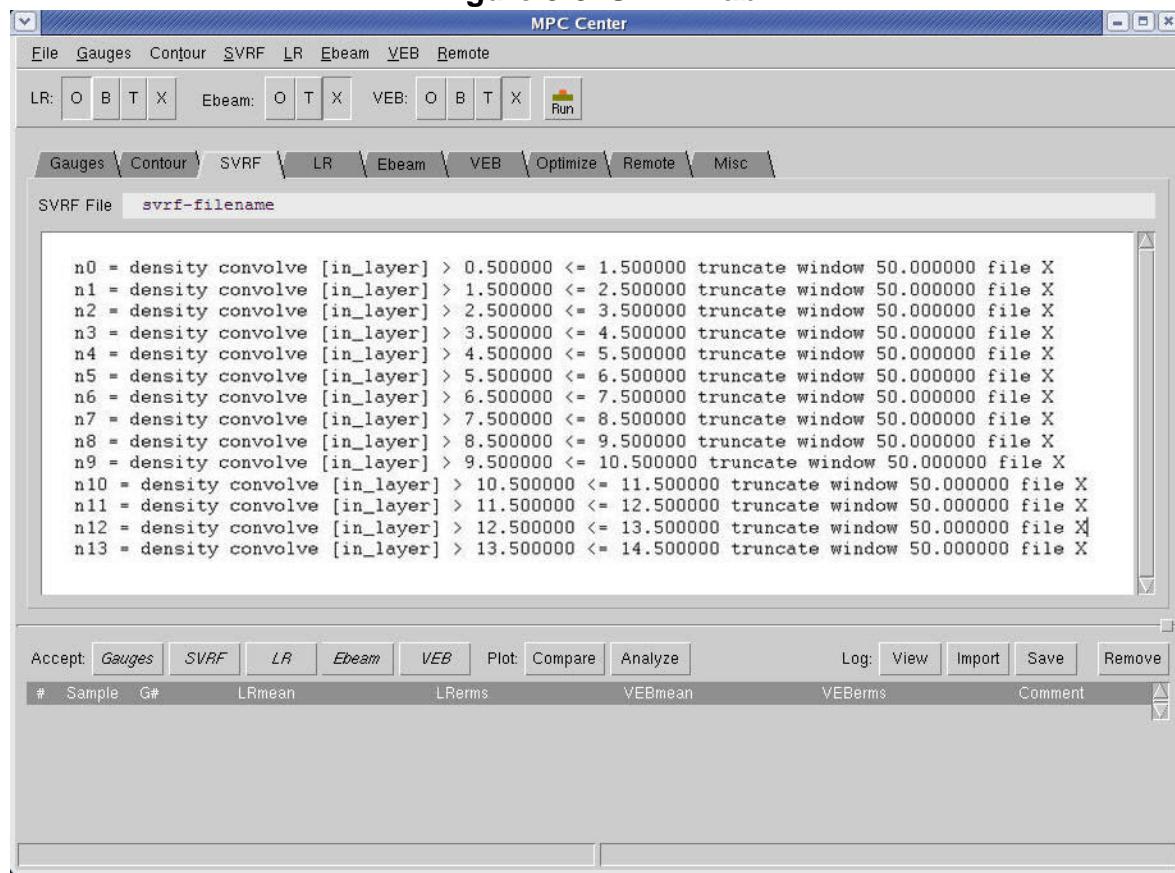


3. In the **LR** tab, reset the Uniformity Order to a nonzero integer. This represents the spatial uniformity, or an attempt to map out the non-uniformity of CDs as a function of position. This can be specified as a polynomial up to the 4th order.
 4. Set the BTERMS. These are constant BTERM coefficients (of pattern density and uniformity) for the kernels. You need the following coefficient terms:
 - 1 constant BTERM coefficient
 - 1 for the diffusion kernel
 - 5 for the first order terms (x, y, x^2, y^2, xy). You need to type in X, Y, X2, Y2 in the Coeff field, and XY in the fields next to their coefficients.

This is equivalent to filling in Uniformity Order and BTERM data in the Long Term model file form (see “[Long-Range Model File Format](#)” on page 124).

5. In MPC Center button bar, set the run control so that only the LR options are enabled. Set **LR** to **O** (Optimize), Build (B), or T (Try), and **Ebeam** and **VEB** to **X**.
 6. Click the **Run** button to generate the model. This produces an SVRF block that you can view in the **SVRF** tab, as well as a sample .mod file (this is only to view, not to be used in the rule file).

Figure 3-8. SVRF Tab



You can also see the model fit results at the bottom of the screen in the LR page. If you wish to add further complexity to your models for better fit results, you can add another kernel and re-run the process.

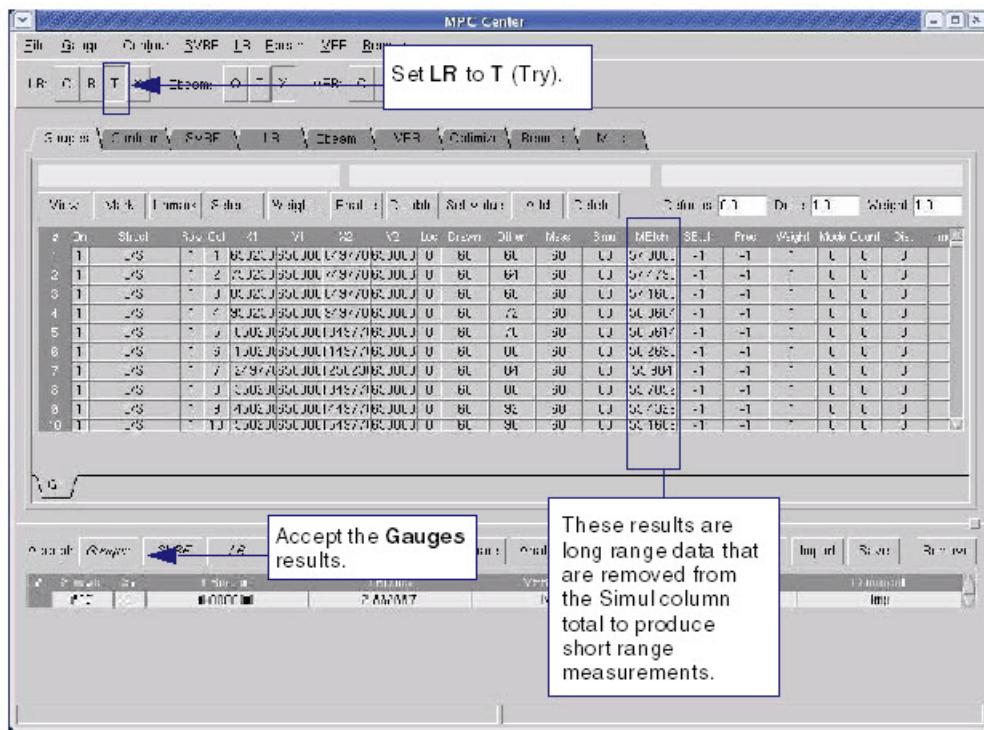
Stage 3 - Remove LR Effects From Short Range Data

Use the T (Try) option for Long Range models to extract Long Range effects from the Short Range data

Procedure

1. Click the **Gauges** tab.
2. In MPC Center button bar, set the run control so that only the LR options are enabled. Set **LR** to **T** (Try).

Figure 3-9. Gauges Tab (LR Try)



3. Click **Run**.
4. Examine the results in the Simul column and the MEtch column. The Simul column represents the results from the long range simulation. The difference between the numbers in the Simul column and the corresponding values in the MEtch column represent the short range data.
5. Click the **Gauges** button in the Accept button bar (at the bottom of the dialog box). Accepting the gauges result removes the long range data from the short range measurements.

Stage 4 - Build Short Range Model

Build the calibrated short range model.

Table 3-3 summarizes the main phases in the Short Range (VEB) model process stage.

Table 3-3. Build VEB Short Range Model Process Summary

Phase	Section	Description
1	Loading Corrected Short Range Measurements	Load the corrected short range measurements into MPC Center.
2	Configuring the VEB Model Form	Enter the kernels and term coefficients for the VEB model form.
3	Configuring VEB Optimization Parameters	Enter VEB model optimization parameters in MPC Center.
4	Input Model or Layer File and Build Final Short Range Model	Enter a model or layer file in MPC Center and build the corrected VEB Short Range Model.

E-Beam models are another type of short-range model, but these are created using the [ebeam_model_load](#) command and are currently calibrated using either the [mpcflow_v2](#) batch command or the Ebeam tab of MPC Center (see “[Stage 5 - Build the E-Beam Model](#)” on page 45 for information).

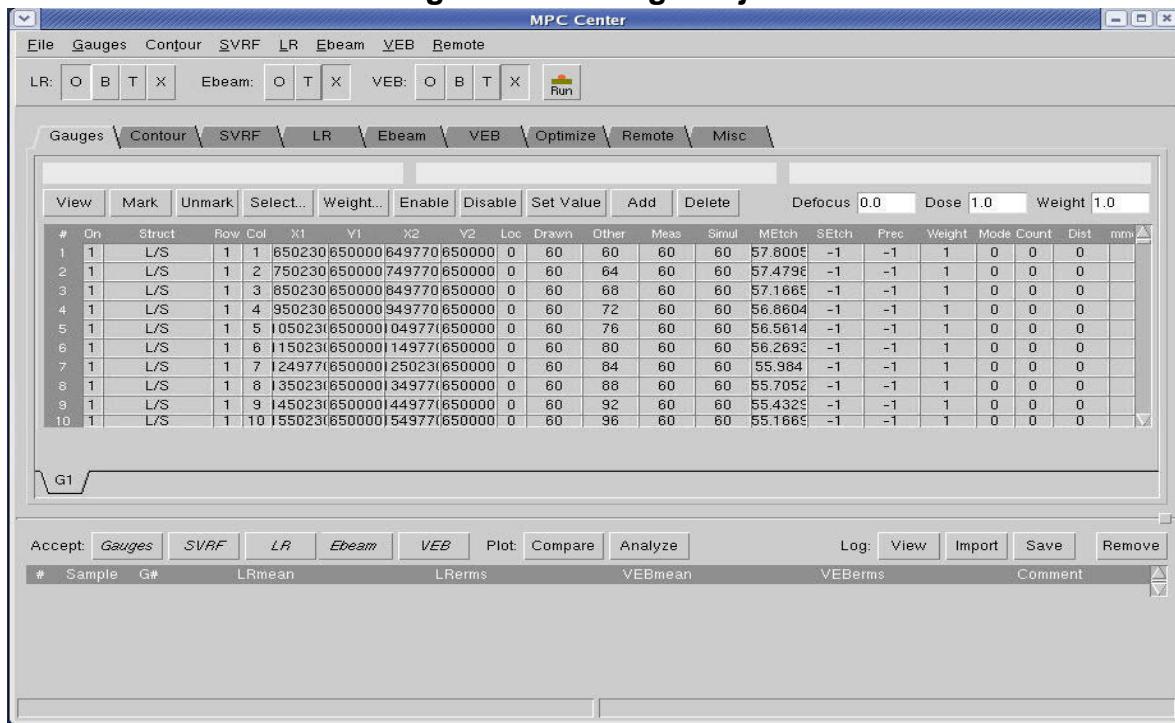
Loading Corrected Short Range Measurements

Load the corrected short range measurements into MPC Center.

Procedure

1. Invoke Calibre WORKbench.
2. Load a test pattern file into Calibre WORKbench. This file must be flat.
3. In Calibre WORKbench, select **Tools > MPC Center** to open the MPC Center.
4. In MPC Center, select the **Gauges > Open** menu option. Import the proximity data gauge file. It should appear in the Gauges tab in MPC Center ([Figure 3-10](#)).

Figure 3-10. Gauge Object



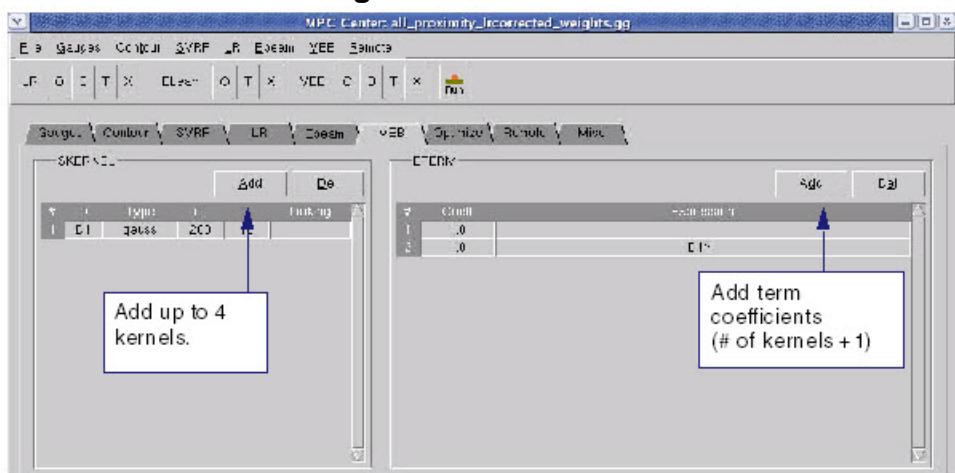
Configuring the VEB Model Form

Enter the kernels and term coefficients for the VEB model form.

Procedure

1. Click the **VEB** tab (see [Figure 3-11](#)).

Figure 3-11. VEB Tab



2. In the VEB page, enter the following information:

- SKERNEL: Click the **Add** button to add up to 4 kernels. You can start with 1 kernel and perform an initial run to check if the model approximates your data. If you require more coefficients for a better fit, you can return and add more kernels as part of an iterative process. The Kernels show the Gaussian sigmas and shifts from the gauge file data.
- BTERM: These are constant term coefficients for the kernels. You need coefficient terms for each kernel plus one extra (for instance, 2 kernels require 3 BTERM coefficients).

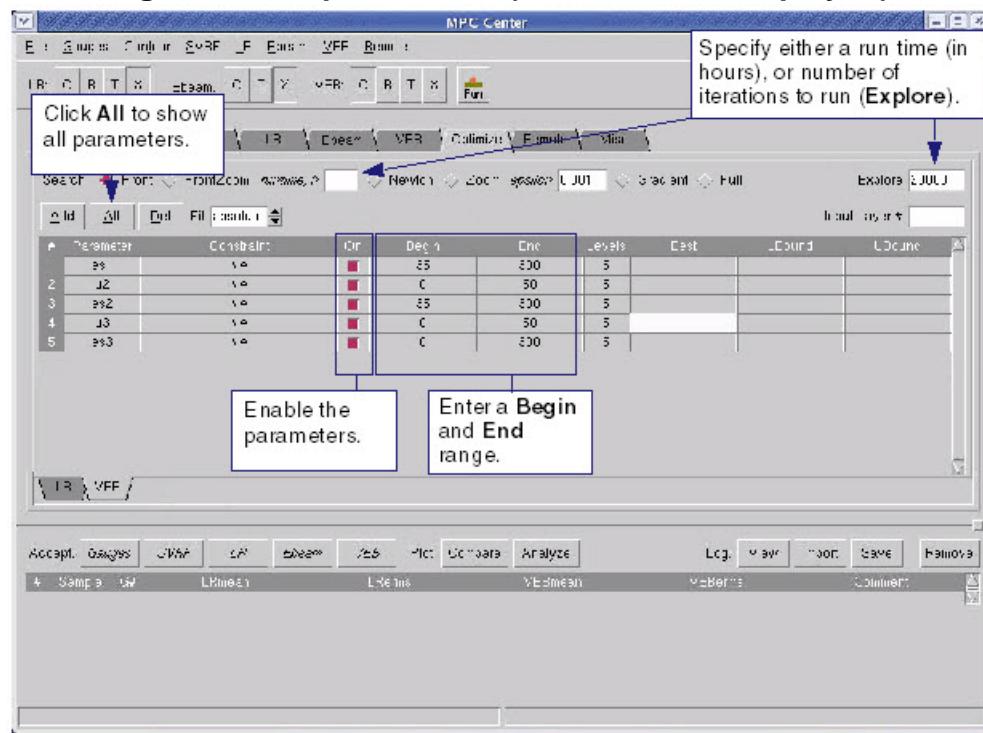
Configuring VEB Optimization Parameters

Enter VEB model optimization parameters in MPC Center.

Procedure

1. Click the **Optimize** tab. In the Optimize page, click the **VEB** sub-tab.

Figure 3-12. Optimize Tab (VEB Sub-Tab Displayed)



2. In the **VEB** tab of the Optimize page, click **All** to see all the parameters from the gauge file.
3. Enable the parameters you want to optimize by clicking the radio button in the **On** column.

4. Set lower and upper bounds by entering **Begin** and **End** ranges for your enabled parameters.
5. Set either the number of iterations to run in the **Explore** field, or a run time value in the **Runtime** field (the **H** stands for number of Hours).
6. Leave Fit at **Absolute** and **Front** search.

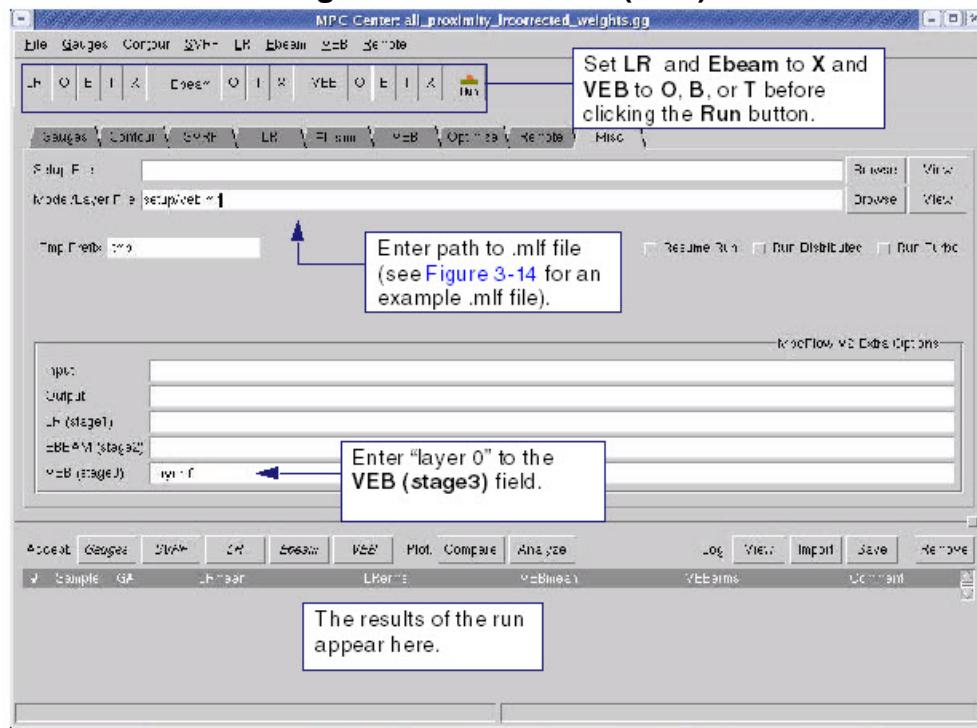
Input Model or Layer File and Build Final Short Range Model

Enter a model or layer file in MPC Center and build the corrected VEB Short Range Model.

Procedure

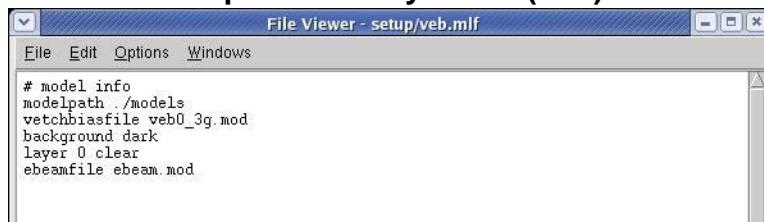
1. Click the **Misc** tab (see [Figure 3-13](#)).

Figure 3-13. Misc Tab (VEB)



2. In the Misc page, enter the path to the .mlf file in the **Model/Layer File** field,
3. Enter “inputlayer 0” in the **VEB (stage 3)** field, which indicates that the design file contours are incorporated as part of the calibration calculations. [Figure 3-14](#) shows an example .mlf file.

Figure 3-14. Example Mask Layer File (.mlf) for VEB Model



```
# model.info
modelpath ./models
wetchbiasfile veb0_3g.mod
background dark
layer 0 clear
ebeamfile ebeam.mod
```

4. In the MPC Center button bar, set the run control so that only the VEB options are enabled. Set **VEB** to **O** (Optimize), **B** (Build), or **T** (Try) and **LR** and **Ebeam** to **X**.
5. Click the **Run** button to generate the model. This produces a .mod file, which is the VEB model you reference in your SVRF file for proximity corrections (see Figure 3-15).

Figure 3-15. Proximity Model File Example

```
# VERSION v2009.1_0.4 (pre-production) Wed Oct 22 02:34:36 PDT 2008
# RUNNING ON Linux redbean 2.6.9-67.ELsmp #1 SMP Wed Nov 7 13:56:44 EST 2007 x86_64
# USER tlin
# TIME Fri Oct 24 15:14:43 2008

version 1
modelType VEB
SKERNEL D1 type=gauss es=167.5 u=5
SKERNEL D2 type=gauss es=61.5 u=25
SKERNEL D3 type=gauss es=167.5 u=25
btermCount 4
BTERM 22.076
BTERM -16.3266 D1^1
BTERM -11.7086 D2^1
BTERM 7.30751 D3^1
.tmp_mfv2_eo.mod (END)
```

You can also see the model fit results at the bottom of the screen in the VEB page. If you want to add further complexity to your models for better fit results, you can add another kernel and re-run the process.

Stage 5 - Build the E-Beam Model

The E-beam model captures very short range effects caused by electron beam forward scattering and beam blur.

For E-beam models, a specified kernel (narrow Gaussian) can be applied to capture short range effects resulting from forward scattering and beam blur, which can improve the modeling accuracy and correction fidelity for narrow lines, line ends, corners and 2D features in general.

You can create a fast E-beam model that models corner rounding and performs simulation and rule based modes.

Configuring and Optimizing an E-Beam Model..... 45

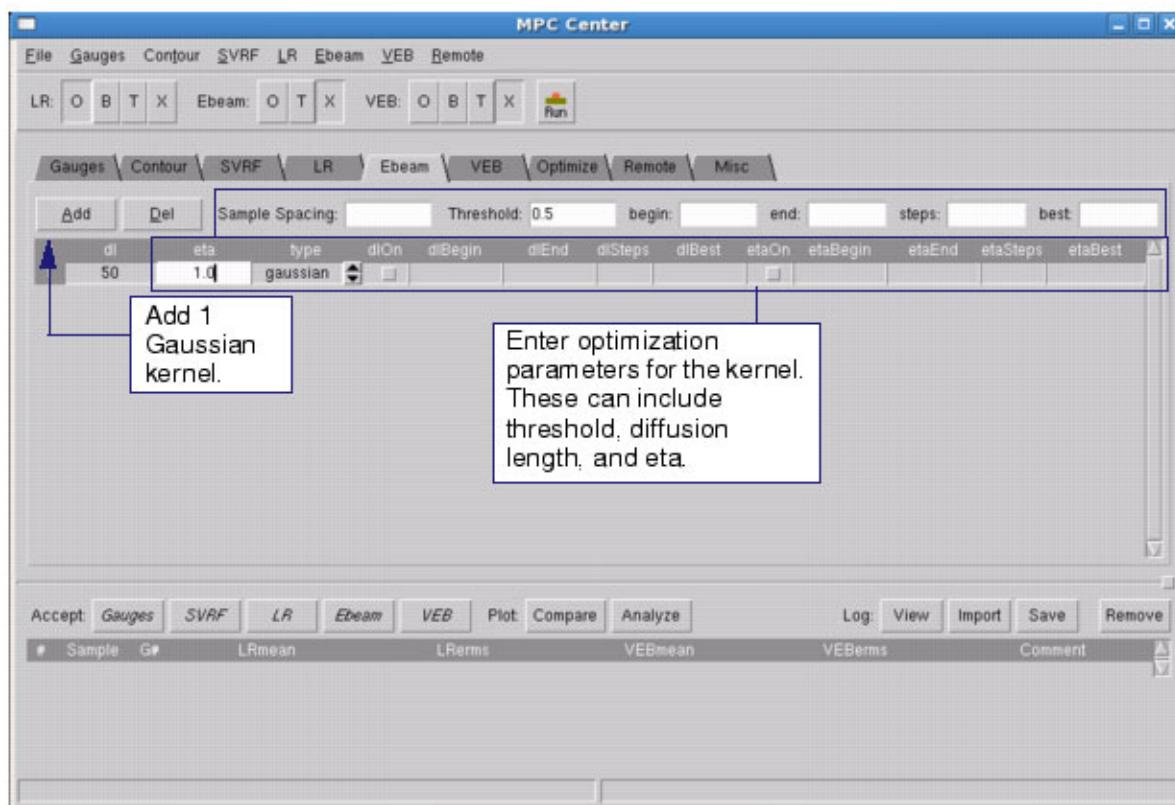
Configuring and Optimizing an E-Beam Model

Configure the E-beam model in the **Ebeam** tab of MPC Center.

Procedure

1. In MPC Center, click the **Ebeam** tab (see [Figure 3-16](#)).

Figure 3-16. E-Beam Model Configuration



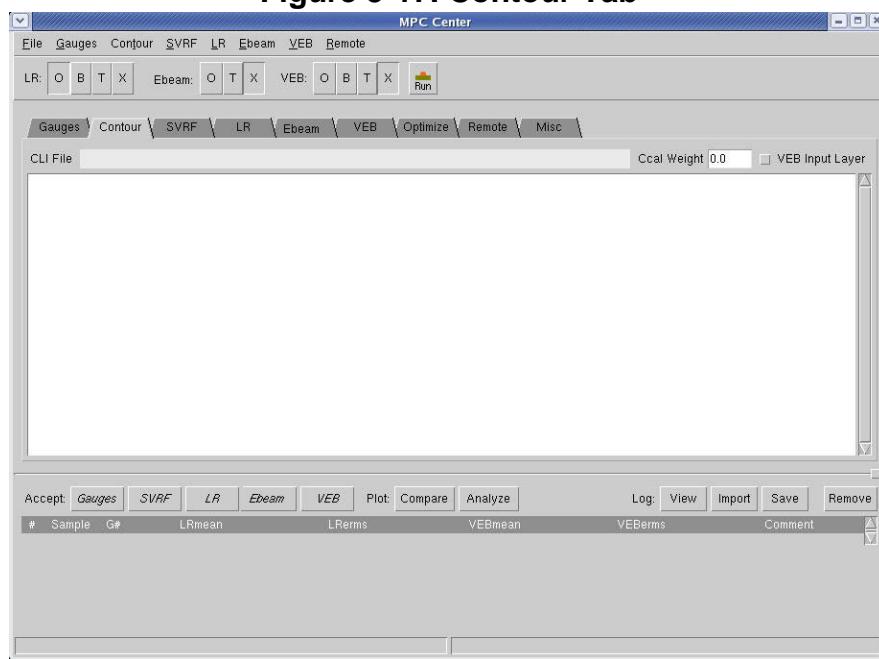
2. In the **Ebeam** tab, perform the following steps:
 - a. Click the **Add** button to add a Gaussian kernel. This is used to perform lithographic simulations using geometry on the input layer by convolving the input image with the Gaussian kernel of the diffusion length. Typical values are between 20-50 nm.
 - b. Enter parameters for the added kernel. This can include the threshold, diffusion length, and eta. Qualifiers for each parameter include the beginning and end range, best “fixed” value, and number of steps. These parameters correspond directly to the same used in the [ebeam_model_load](#) setup file command.
3. In the MPC Center button bar, set the run control so that only the E-beam and VEB options are enabled. Set **Ebeam** and **VEB** to **O** (Optimize) or **T** (Try) and **LR** to **X**.
4. Click the **Run** button to generate the E-beam model.
5. The E-beam model can also be created using the [setlayer ebeam_simulate](#) and [ebeam_model_load](#) commands in the MPC setup file. Refer to “[Optional E-Beam Model Definition](#)” on page 71 for information. E-beam calibration can also be performed using a batch command, [mpcflow_v2](#) (see “[mpcflow_v2](#)” on page 126 for a complete description).

Additional Modeling Operations in MPC Center

MPC Center also contains two additional tabs, **Contour** and **Remote**.

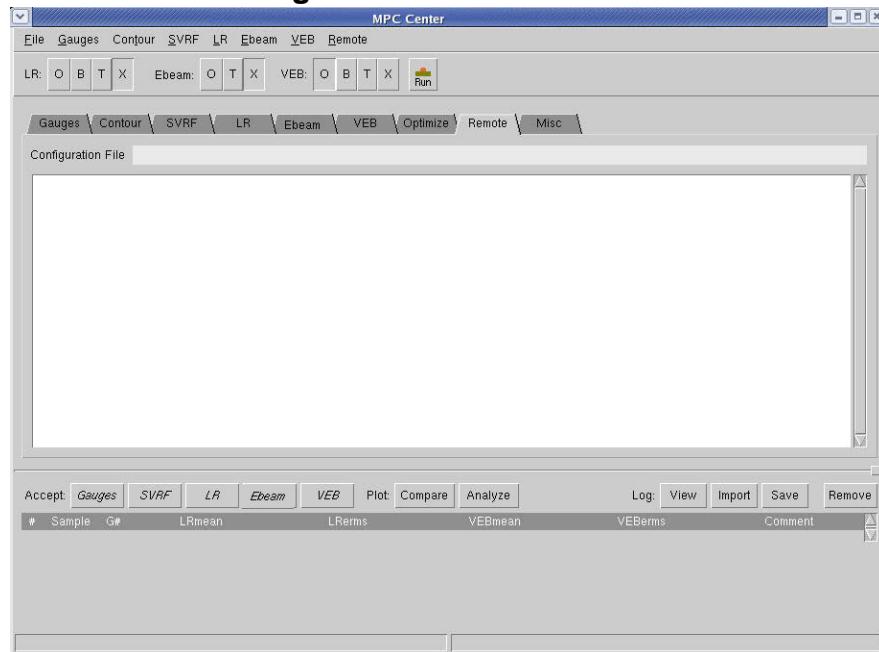
- The Contour tab in MPC Center is identical to the **Contour** tab (see [Figure 3-17](#)) described for the CM1 Center as documented in the [Calibre WORKbench User’s and Reference Manual](#). This tab is used to read a Contour Layer Information (CLI) file, which is supplies information about an additional contour calibration layout file.

Figure 3-17. Contour Tab



- If you have a distributed hardware system, the **Remote** tab (see [Figure 3-18](#)) is used to create a remote host file and implement distributed processing. See the [Calibre WORKbench User's and Reference Manual](#) for complete details.

Figure 3-18. Remote Tab



MPC Center Menu Quick Reference

The MPC Center contains a number of menu-based operations that are identical to those used in CM1 Center.

These operations are summarized in [Table 3-4](#) and full descriptions can be found in the *Calibre WORKbench User's and Reference Manual*.

Table 3-4. MPC Center Menu Summary

Menu	Description	
File		
	Open Session	Open MPC Center session file.
	View Session	View MPC Center session file.
	New Session	Create new MPC Center session file.
	Save Session	Save MPC Center session file.
	Export Run Script	Saves the session file as a run file. Allows you to run script as mpcflow_v2 instead of reconfiguring the MPC Center.
	Close Window	Close MPC Center.
Gauges		
	Open	Open gauge file object.
	Append	Appends to existing gauge file object.
	Save	Saves gauge file object.
	Delete	Deletes selected gauge file object.
	Open Super Gauge	Open super gauge data file.
	Save as Super Gauge	Saves object as super gauge data file.
	Import Super SS	Converts a super spreadsheet calibration file to a super gauge file. This operation is similar to Import SS.
	Import SS	Converts a sample spreadsheet calibration file to a gauge file. See " Converting Sample Spreadsheets to Gauge Files " in the <i>Calibre WORKbench User's and Reference</i> .
	Append SS	Appends to an existing spreadsheet.
	Export Markers	Draw gauge markers on an empty layer. Refer to " Cleaning the Sample Spreadsheet and Gauge Object " in the <i>Calibre WORKbench User's and Reference</i> .
	Import Markers	Import gauge objects (markers) from sample file spreadsheets. Refer to " Importing Drawn Gauges From a Layout " in the <i>Calibre WORKbench User's and Reference</i> .
	Measure Layout for CD	Measures the selected gauge object for CDs.

Table 3-4. MPC Center Menu Summary (cont.)

Menu		Description
	Clip Layout	Create a reduced version of the layout file. Refer to “ Creating a Clipped Layout File ” in the <i>Calibre WORKbench User’s and Reference</i> .
	Bossung Plot	Create a Bossung Plot graph for one or more individually-selected test pattern structure lines. Refer to “ Bossung Graphs for Super Gauge Objects ” in the <i>Calibre WORKbench User’s and Reference</i> .
	Bossung Compare	Compare Bossung curves for several multiply-selected test pattern structures in a single graph. Refer to “ Bossung Graphs for Super Gauge Objects ” in the <i>Calibre WORKbench User’s and Reference</i> .
	Adjust Gauge	Adjust the base measurement of the gauge object in nms.
Contour, SVRF, LR, and Ebeam		
	Open...	Open tab session file.
	Save Session...	Save tab session file.
VEB		
	Open...	Open tab session file.
	Save Session...	Save tab session file.
	Create Default...	Creates a default VEB model and optimizations using the settings from the VEB tab.
Remote		
	Open...	Open tab session file.
	Save Session...	Save tab session file.
	Termplate...	Creates remote host configuration file.

Chapter 4

Correct and Verify the Design

With the long-range and proximity models now calibrated, a rule deck can be constructed to simulate the mask and apply corrections to the layout (as well as perform a FRACTURE in the same run).

Correction Prerequisites	51
The Calibre nmMPC Setup File.....	53
Import and Define Models	56
Define Layers	58
Define an Image	62
Set Iterations	62
Set Feedback for Convergence	62
Deactivate Additional Fragmentation.....	63
Set Mask Constraints	63
Create a No-Correction Region	63
Smooth Biased Target Layers.....	64
SEM Box Correction	66
Tag Scripting.	70
Correction for Curvilinear Layouts.....	70
Optional E-Beam Model Definition	71
How Calibre nmMPC Measures EPE	72
Long-Range Correction Process.....	73
Short Range (Etch and E-Beam) Correction Process.....	74
Verify Corrections	81
Verifying the Corrections With Calibre OPCverify	81
Verifying EPE Layers.....	84

Correction Prerequisites

Before you begin this stage, you should already fulfilled a number of prerequisites.

- Your design layout
- For long-range correction:
 - Calibrated model information for proximity and long-range effects
 - An SVRF command block generated by MPC Center for long-range effects

- For short range correction:
 - A calibrated VEB model produced by MPC Center

When the process is complete, you should have the following:

- A corrected layout

The Calibre nmMPC Setup File

Once the models are created and calibrated, the mask can now be corrected by creating a Calibre nmMPC setup file, a file that contains commands that reference the long range and short range simulation models and uses them to adjust the mask polygons.

The setup file contains all of the information necessary for running correction and verification, including:

- Model definitions
- Input layer definitions
- The `processing_mode` command must be set to flat
- MPC controls

Figure 4-1 and Figure 4-2 shows Calibre nmMPC setup files for an e-beam and etch model. The like-colored text (such as the red `ebeamModel` text) indicates the same object reference.

Figure 4-1. Minimum Calibre nmMPC Setup File With an E-Beam Model

```

maskout = LITHO MPC polyfrac
FILE mpc_setup MAP maskout
    |-----| Derive SVRF layers using
    |-----| LITHO MPC.

LITHO FILE mpc_setup [
    |-----| Begin the setup file with an
    |-----| SVRF LITHO FILE statement.

    modelpath ./models
    etch_model_load etchModel etch.mod
    ebeam_model_load ebeamModel {
        |-----| Import and define layers and
        |-----| set processing_mode to flat.

        model_type simulation
        diff_length 0.024 }
    processing_mode flat
    |-----| Define input layers from
    |-----| LITHO MPC into the setup file

    layer maskin
    |-----| Begin the denseopc_options
    |-----| block.

    denseopc_options corr {
        |-----| Set Calibre version and
        |-----| algorithm to 1, enable
        |-----| tile_fragmentation_only, and
        |-----| define a background.

        version 1
        algorithm 1
        tile_fragmentation_only 1
        background dark
        |-----| Pass layers to MPC.

        layer maskin opc clear
        |-----| Define an image.

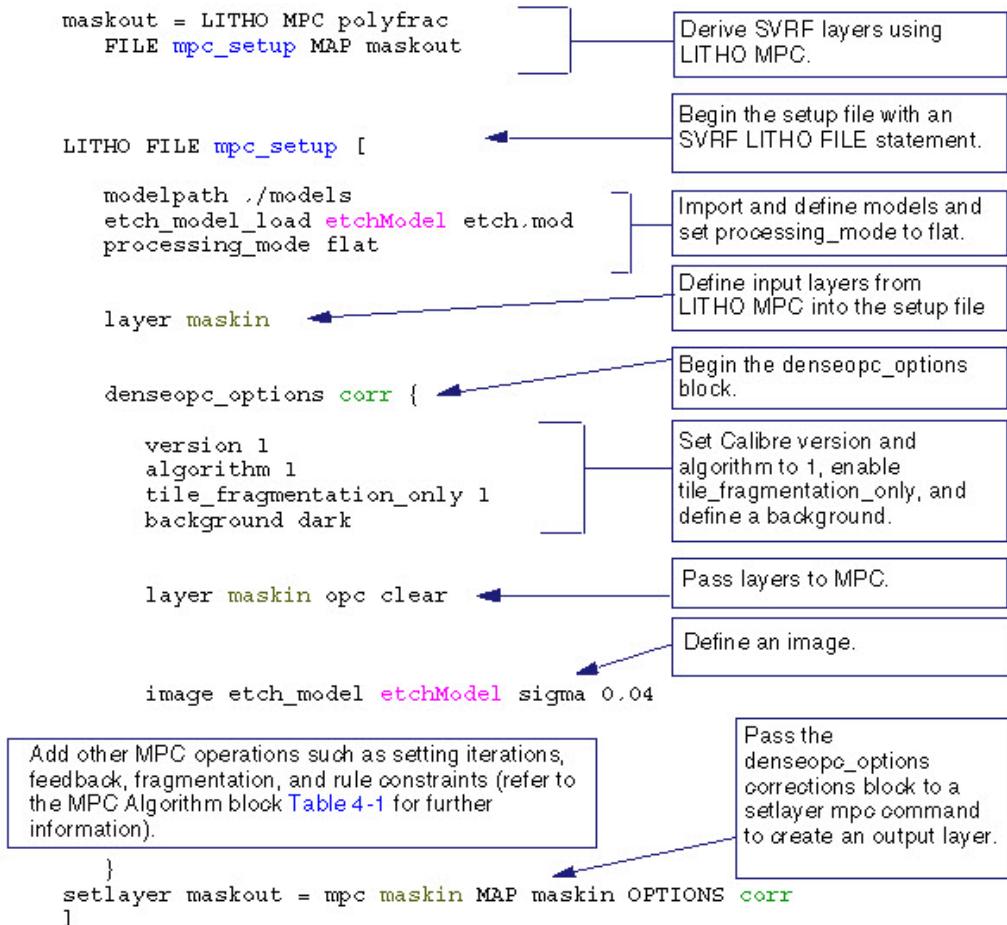
        image ebeam_model ebeamModel etch_model etchModel
        |-----| Add other MPC operations such as setting iterations,
        |-----| feedback, fragmentation, and rule constraints (refer to
        |-----| the MPC Algorithm block Table 4-1 for further
        |-----| information).

    }
    |-----| Pass the
    |-----| denseopc_options
    |-----| corrections block to a
    |-----| setlayer mpc command
    |-----| to create an output layer.

    setlayer maskout = mpc maskin MAP maskin OPTIONS corr
]

```

Figure 4-2. Minimum nmMPC Setup File With an Etch Model Only



The Calibre nmMPC setup file can be a separate file or can be defined in an SVRF script with LITHO FILE. A minimum setup file will have:

- Models
- Input layer definitions
 - Maps input layers from SVRF to simulation names
 - This mapping is strictly by order
 - At least one is required
- The `processing_mode` command must be set to flat
- A `setlayer` command to generate MPC output
- Correction options
 - Version (set to 1)
 - `algorithm` (set to 1)

- `tile_fragmentation_only`
- Background transmission (required to declare but not used)
- MPC layer definition
- The layer names match the names in the setlayer line

The MPC Setup File makes use of the following SVRF commands:

- **LITHO MPC**: This is a layer derivation that passes inputs to nmMPC and creates outputs.
- **LITHO FILE**: A specification command that defines a setup file. This is not required if a separate file is used.
- **VARIABLE**: Constants defined with the VARIABLE statement can be passed to Calibre nmMPC.

The Calibre nmMPC setup file has the following basic parts: the Simulation section, and the MPC Control block (defined by the `denseopc_options (for MPC)` block). The sections and their operations are summarized in [Table 4-1](#).

Table 4-1. Calibre nmMPC Setup File Summary

Task	Description	nmMPC Command Used
Simulation		
Import and Define Models	Specify the path to models, and define E-beam and etch models.	<code>modelpath</code> <code>etch_model_load</code> for etch models <code>ebeam_model_load</code> for E-beam models
Define Input Layers (see Define Layers)	Define layers being passed from LITHO MPC into the setup file.	<code>layer name</code>
MPC Control Block (<code>denseopc_options</code> Block)		
Define an Image	Generate a nominal image.	<code>image</code> <code>etch_model</code> if no E-beam is used <code>image</code> <code>ebeam_model</code> if E-beam is used
Create an MPC Algorithm		

Table 4-1. Calibre nmMPC Setup File Summary (cont.)

Task	Description	nmMPC Command Used
Set Version and Background	Set version to 1 and a dark background. The background statement is required to declare, but not used by Calibre nmMPC.	version background (for denseopc_options block)
Set algorithm and tile_fragmentation_only	Set algorithm to 1 and tile_fragmentation_only.	algorithm tile_fragmentation_only
Pass Layers to MPC (see Define Layers)	Define layers passed inside the setup file.	layer (denseopc_options Command) name type transmission
Set Iterations	Set maximum correction iterations (1-3 recommended).	max_iterations
Set Feedback for Convergence	Set feedback settings.	feedback
Deactivate Additional Fragmentation	Normally, turn off fragmentation	tile_fragmentation_only
Set Mask Constraints	Optionally add mask rule constraints.	mrc_rule
Create a No-Correction Region	Optionally create “no-correction” areas on the mask	no_opc_region
Generate Output Layers (see Define Layers)	Pass the denseopc_options (for MPC) block to a setlayer denseopc command to create an output layer.	setlayer mpc
Tag Scripting	Create tag scripts to apply minor corrections to fragments.	—

Import and Define Models

You should have your model information prepared prior to creating the setup file. Once created, import the models into Calibre nmMPC.

Use the [modelpath](#) command to define the path (more than one can be defined) to the directory containing your modeling information.

```
modelpath dir1:dir2:dirn
```

For example:

```
modelpath ./models
```

Use the VEB [etch_model_load](#) command to import the etch model file. Assign a unique name to this model (one or more of model can be defined).

```
etch_model_load name {filename | inline}
```

The [ebeam_model_load](#) command enables you to load an E-beam model file. For example:

```
ebeam_model_load ebeam_model_name {  
    diff_length 0.024  
    model_type simulation  
    sample_spacing 0.008  
}
```

Instead of loading models discretely, you may use a litho model. A litho model is a directory which must be found within the model path. The litho model contains a key file called Lithomodel. There is no command required to explicitly load the litho model or its sub-models. It is automatically loaded when the litho model name is detected in the denseopc_options block.

If you use a litho model:

- You do not use the etch_model_load and ebeam_model_load commands in the setup file. Their information is taken instead from the litho model. If specified, they will be ignored.

You do not specify the e-beam model name or dose in the [setlayer ebeam_simulate](#) command. You will only need to refer to an [image_options](#) block that references the litho model.

- For Calibre nmMPC, the denseopc_options block must either use the litho model method or the discrete model-loading method. You cannot use both model methods in the same file. However, you can specify several litho models in a single setup file.
- The e-beam dose specification must be included in the setup file.

The following is an example of the Lithomodel file:

```
% cat duv10/Lithomodel
version 1
lithotype EBEAM
etch etch.mod
ebeam ebeam.mod
```

The dose specification in the setup file (using an [image_options](#) block) may look similar to the following example:

```
modelpath ./myModels
layer M0
layer M2
layer M3
layer M4
layer M5
layer M6
layer M7
tilemicrons 100
image_options MDF {
    layer M0 visible ebeam_dose 1.0
    layer M2 visible ebeam_dose 0.9
    layer M3 visible ebeam_dose 1.1
    layer M4 visible ebeam_dose 1.2
    layer M5 visible ebeam_dose 1.3
    layer M6 visible ebeam_dose 1.4
    layer M7 visible ebeam_dose 1.5
    litho_model myLatestEuvModel
}
setlayer im = ebeam_simulate MDF
```

For further information on the litho model format, refer to “[Lithomodel \(Litho Model Format\)](#)” on page 341.

Define Layers

A layer contains polygon information that is used by the Calibre RET batch tools to simulate and correct your mask layout design.

You define how the LITHO operation uses these layers by writing definitions using the commands in the setup file. Each layer definition assigns a type and number, identifies the transmission type, and defines how the layer affects, or is affected by fragmentation.

Layers are defined in two places (with two different commands):

- Input layers passed from LITHO MPC SVRF command into the setup file
- Layers passed to MPC by the setlayer command

Two layer declarations are required due to the fact that layers used in MPC may not be the same as the input layers defined in SVRF.

Define Input Layers

Each layer passed from LITHO MPC as input must have a “layer” statement. Any minimum setup file will map input layers from SVRF to simulation names.

The layer statement uses the following basic syntax:

layer name

- *name*: A user-specified alphanumeric string that names the layer.

The input layers are matched to SVRF LAYER statements strictly by order, which means the layer names do not necessarily need to match. Inside the setup file, layers are referenced by name.

Pass Layers to MPC

MPC layers are defined inside denseopc_options block. All layers passed on the denseopc setlayer command line must have a layer statement. The number of layers passed must equal the number defined.

The basic syntax for a layer passed to MPC is as follows:

layer name type transmission

- *name*: A user-specified alphanumeric string that names the layer.
- *type*: A parameter that defines the layer type.
- *transmission*: An argument that defines the mask layer’s optical transmission type. Values accepted are dark and clear. The layer transmission should always be opposite of the background (for example, the background is clear and layer is dark).

Layer definitions should be directly after the version statement in the [denseopc_options \(for MPC\)](#) block. Any layer name used as a keyword argument must be already defined.

Layer Types

Layers are used for a number of different purposes in Calibre MPC, depending on the specific type.

There are several different layer types supported by Calibre nmMPC (see the following table). These layer types have meaning only with respect to the MPC toolset.

Table 4-2. Layer Descriptions

Layer Type	Function
opc	<p>The opc layer serves as the target layer (the layer that will have MPC performed on it) for LITHO operations.</p> <p>During processing, an opc layer is impacted as follows:</p> <ul style="list-style-type: none"> • Its edges are fragmented. • Its edges are moved during MPC • The output (the MPC modifications to this layer) represents the final mask. <p>At least one opc layer is required. You can pass multiple opc layers during a LITHO operation and define the opc layers in a setup file.</p> <p>When the opc layer is used for retargeting, the opc layer is fragmented and corrected in order to make a contour that matches the retarget layer.</p> <p>Multiple opc layers are allowed. This allows layer operations such as fragmentation and MRC enforcement to be tailored to specific needs. During simulation and correction, the OR of all the opc layers is used as input to the simulator and to determine which edges are movable.</p>
hidden	<p>By default, any layer not specified in the setup file is marked as a hidden layer. Hidden layers are used for a number of functions in Calibre nmMPC, including serving as retargeting layers and marker layers.</p>
visible	<p>Visible layers contain geometries that are not corrected yet are optically visible - that is, they appear on the mask and interact with other geometries to affect the final printed design.</p> <p>The data on visible layers can be non-printing data, such as scattering bars, or it can be printing data that does not need correction and needs to be factored into density calculations. By default, visible layers do not induce fragmentation. Visible layers are not required.</p> <p>Visible layers should not interact in any way with opc layers. Any overlap or abutment between an opc layer and a visible layer will result in false edges on the opc layer. False edges are layer edges that do not have any corresponding mask edges. In cases where correction is not wanted on some polygons, those polygons should be put on an opc layer. A copy of the layer can be passed and used with the no_opc_region keyword.</p>

Table 4-2. Layer Descriptions (cont.)

Layer Type	Function
correction	<p>Correction layers contain edges that are moved during MPC. Typically, correction layers are used to generate multiple mask images, such as phase shifting masks. They are used only with Calibre nmMPC batch tool or Calibre WORKbench.</p> <p>During processing, a correction layer:</p> <ul style="list-style-type: none"> • Has its edges fragmented. • Has its fragments mapped to fragments on an opc layer. • Has its edge fragments moved during OPC. <p>Correction layers are not required. If you do not pass a correction layer to the batch tool, it moves the edges on the opc layer instead.</p>
external	<p>External layers contain corrected mask data. If present:</p> <ul style="list-style-type: none"> • The simulated mask is derived only from external layers. • The opc layer is not used by the simulator. <p>This enables you to provide a separate mask layer for simulation only.</p> <p>The following example illustrates the implementation of an external layer in a denseopc_options block:</p> <pre>denseopc_options "EPE.LayerGen" { layer poly opc clear layer mpc_poly external clear image ebeam_model EBEAM \ etch_model ETCH feedback 1.0 ... } setlayer epe_layer = mpc_poly \ mpc_poly MAP_poly OPTIONS\ "EPE.LayerGen"</pre> <p>This code is used for EPE layer verification, where measurement sites are in the target layer and the external layer (the MPC-corrected layer) is used for simulation. Refer to “Verifying EPE Layers” on page 84 for further information.</p>

Generate Output Layers

To generate output layers, use the setlayer command in the setup file.

setlayer name = operation inputs

MPC is performed using the `denseopc setlayer` command, as shown in the following example:

```
setlayer maskout = mpc maskin MAP maskin OPTIONS corr
```

In this example, “maskout” is the name of the output target layer. One layer is defined, “maskin.” MAP is a required keyword followed by the associated argument indicating the layer or tag containing the output to be returned (in this case, only the poly layer is output), and OPTIONS is a required option that specifies the name of a [denseopc_options \(for MPC\)](#) sub-command block of LITHO MPC setup file commands.

Define an Image

A Calibre nmMPC image command is required in the `denseopc_options` block. Any non-empty combination of E-beam or etch models can be used; E-beam-only, etch-only, or E-beam and etch. This statement parameterizes the simulated image that is used to compute EPEs. For example:

```
image ebeam_model ebeamModel etch_model etchModel
```

If no E-beam model is defined, only use the `image etch_model` option instead. For example:

```
image etch_model etchModel sigma 0.04
```

Set Iterations

The number of iterations can be controlled with the command `max_iterations`. Depending on the model and correction accuracy requirements, only 1 to 3 iterations will be needed in most cases.

For example:

```
max_iterations 3
```

Set Feedback for Convergence

Convergence occurs when the results of MPC edge correction reflect the closest possible representation of the original mask feature (the simulation results “converge” on the original design). Although convergence is the intended result of MPC, achieving that convergence can often be difficult. Calibre nmMPC makes corrections based on an iteration of measurement, corrective edge movement, then further measurement.

Use the command `feedback` to adjust edge movement to achieve convergence.

Edge movement is set by EPE * feedback value. The default starting feedback is -0.4. For MPC, feedback should be set at or near -1.0 (-1.0 for one iteration, -0.98 for two or more iterations). For example:

```
feedback -1.0
```

Feedback of each fragment may be slightly increased or decreased on the second iteration. For instance, for three iterations, you can try the following:

```
feedback -0.95 -0.99 -1.0
```

Deactivate Additional Fragmentation

Post-OPC data already has fragmented edges. In most cases no additional fragmentation is needed or warranted.

Set the command [tile_fragmentation_only](#) to 1 to turn off all default fragmentation. The only fragmentation performed is for tile boundaries. Few, if any, new jogs should be introduced in the output.

Set Mask Constraints

A constraint is a behavioral limitation observed by MPC. A Mask Rule Constraint (MRC) is a type of spacing-based constraint where the rules are enforced on the photomask.

Mask constraints are defined as blocks using the Calibre nmMPC [mrc_rule](#) keyword. A mask rule can be one of the following:

- Width or spacing
- Within one layer or between two layers
- Applied in selected areas based on marker layers

Constraints are defined either based on edge lengths or on projection between the edges.

Create a No-Correction Region

A “No Correction” region is an area designated on the layout where MPC will not be performed. This prevents any unintended fragment edge movement that can impact the final mask (particularly in areas where no further correction is warranted).

A marker layer can be passed in to define areas that should not get corrections using the [no_opc_region](#) setup file command. For example:

```
no_opc_region markerlayer
```

Any fragment that is not outside of the marker layer will not be moved. This means fragments are completely or partly inside the marker. Shapes inside the layer are visible to the model. Marker layers are type hidden in the layer statement.

Smooth Biased Target Layers

Biasing operations can occasionally introduce many small irregularities (such as excess jogs or overcorrection) that degrade the overall shapes, preventing clean fragmentation and making the target layer unsuitable for correction.

To correct this, use the [retarget_layer](#) command to separate the target from the layer being corrected. The opc layer is copied to a retarget layer and “smoothed.” The original layer is now the opc layer, and the correction target is the “retarget layer.” In this case, EPE is measured from the image to the retarget layer, then the fragments on the opc layer are moved accordingly (fragments on the retarget layer cannot be moved since smoothed or rounded corners cannot be corrected).

There are three basic approaches to deal with biasing issues:

- Correct a layer with small jogs, then pass in original layer (a pre-biased layer) before biasing as an opc layer, using the layer with jogs as retarget layer.
- Use a smoothed target.
- Smooth the jogs and the retarget layer.

The smoothing process on the retarget layer represents a more physical target, reducing the dependency on corner fragment length. It also reduces overcorrection at corners that results in “ringing” in the image.

Smoothed layers can be generated using the [setlayer curve](#) command and are passed to MPC as retarget layers. The setlayer curve command can be used to produce a smooth target. The curve command implements a b-spline, and control points are derived from existing polygon vertices. Additional points are added before and after corners and at least one point is added in the center of short segments.

Additionally, the purpose of the [setlayer curve_target](#) command is to create a target that mimics what the final image is going to look like. Using this command, you look at the differences between the image and the curve target and adjust the parameters to more closely match what the final image looks like. By having a feasible target (one that can actually be printed on masks), the optimization can be simplified, resulting in better correction results.

A general process for smoothing layers can be described as:

1. Get reasonable good correction results without using smooth target.
2. Use post-correction simulation contour to estimate the amount of residual corner rounding.

3. Adjust curve parameters to match existing contour.
4. When done, compare correction results with and without smoothed input.
 - a. Results should show improved corrections.
 - b. Some fragment length adjustment may help.

A smoothed target can be used for EPE measurement against the contour(s), with the following notes:

- EPE measurements are made orthogonal to the smooth target.
- Convex and line end fragments still use the maximum EPE.
- Concave corners use the minimum EPE.

The following is an example Calibre nmMPC setup file performing target smoothing using setlayer curve_target:

```

LAYER poly 4
poly { copy poly } DRC CHECK MAP poly 4

LITHO FILE "EBeam.setup" [
    ebeam_model_load ebeamModel {
        model_type simulation
        diff_length 0.030
    }

    layer poly

    denseopc_options srcOptions {
        version 1
        image ebeam_model ebeamModel
        algorithm 1
        background clear
        layer poly opc dark
        layer tgt hidden dark
        retarget_layer tgt

        max_iterations 1
        feedback -0.98
        tile_fragmentation_only 1
    }

    setlayer tgt = curve_target poly criticalDistance 0.040 \
        cornerRadius 0.040 cornerRadiusConcave 0.020 \
        cornerRadiusConvex 0.040 lineEndRatio 1.0 maxJog 0.006 \
        roundCorner true
    setlayer mpc_poly = mpc poly tgt MAP poly OPTIONS srcOptions
]

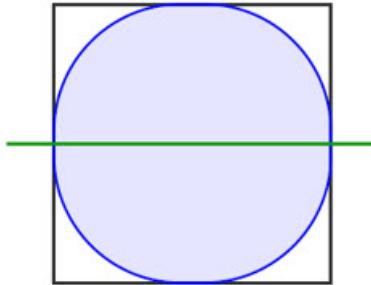
target {LITHO MPC poly MAP tgt FILE "EBeam.setup"} DRC CHECK MAP target 5
mpc_poly {LITHO MPC poly MAP mpc_poly FILE "EBeam.setup"} DRC CHECK \
    MAP mpc_poly 301

```

SEM Box Correction

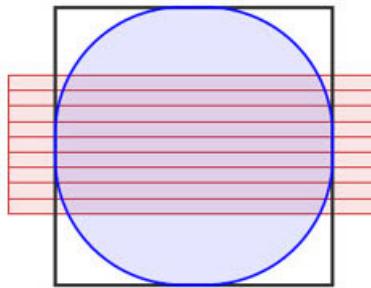
A special correction mode, called SEM Box correction, can be enabled for convex and concave corner fragments.

Figure 4-3. Standard Calibre Gauge Measurement



In the standard CD measurement method, Calibre measures a single gauge line. In the presence of corner rounding, Calibre typically measures the maximum CD for contacts or the minimum CD for tip-to-tip structures. In the SEM Box method, CD is calculated by creating a measurement box, as shown in the following figure.

Figure 4-4. SEM Box Method



The box contains a user-specified number of measurements in the box, and the box size is based on a percentage of the target CD. The box method outputs the average CD as the result.

In Calibre nmMPC, SEM Box mode is implemented through several commands to measure the same 2D CD on a post-MPC structure (such as small contacts or line-end tip-to-tip measurement types) as you would measure on a pre-MPC structure (the calibration mask). These commands include:

- **convex_concave_correction** — A command that enables SEM Box correction mode. This enables several commands used for SEM Box correction:
 - **convex_concave_limit** — Specifies a size limit for convex and concave fragments considered for correction. This command is required for SEM Box mode, as it specifies how line and space ends are considered.

- **convex_concave_adj_len** — Defines the adjacent length of a fragment being processed by the SEM Box method of correction.
- **convex_concave_metric** — Specifies the percentage of a fragment to be considered for placing measurement points for correction.
- **convex_concave_count** — Specifies the number of measurement points to be placed on convex and concave fragments for correction.

The defaults for each of these commands are listed in the following table.

Table 4-3. SEM Box Commands Defaults

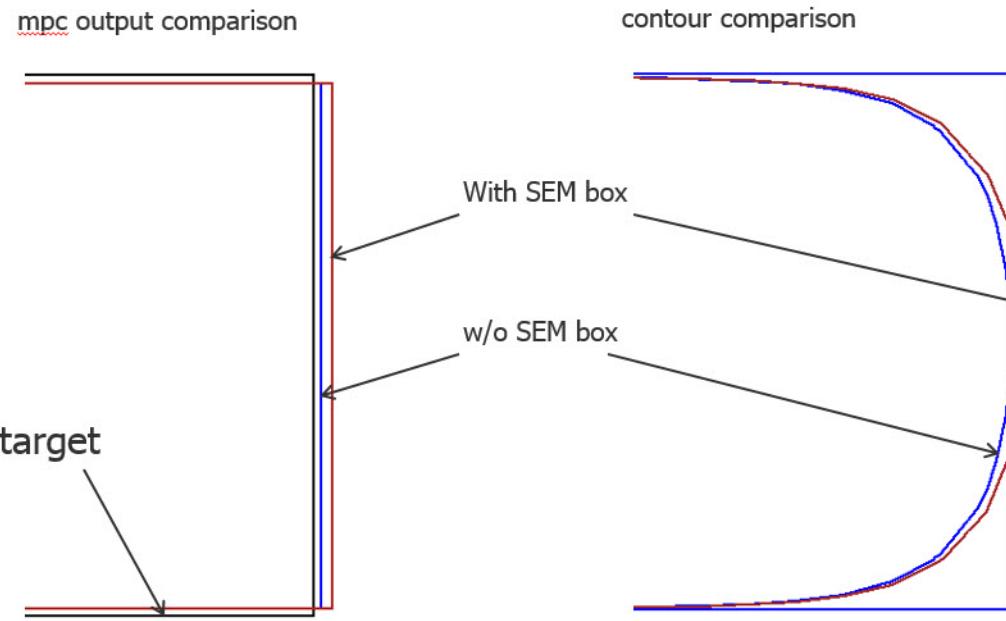
Command	Default
<code>convex_concave_correction</code>	off
<code>convex_concave_limit</code>	0 (no fragments are selected for SEM Box mode)
<code>convex_concave_adj_len</code>	3 * smallest e-beam kernel size if the e-beam model is present; 0 otherwise.
<code>convex_concave_metric</code>	0.5 (50% of the middle of the fragment is taken for placing the measurement points.)
<code>convex_concave_count</code>	3 measurement points

The following is an example of a SEM Box command block.

```
convex_concave_correction on // Turn on SEM Box correction
convex_concave_adj_len 0.01 // Filter jogs from SEM Box method.
convex_concave_metric 0.5 // Measure EPE within 50% of the fragment
convex_concave_limit 0.1 // Limit the fragment length to less than 100nm
convex_concave_count 3 // Select 3 points for EPE measurement
                           // and average them as effective epe for MPC
                           // correction
```

Note that with the SEM Box method, over-correction can occur that is not flagged in SEM measurement. The following figure shows the results of the SEM Box method on MPC output, showing the overcorrection with the MPC output and contours generated.

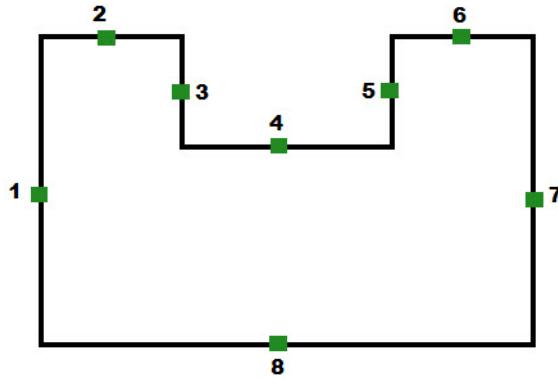
Figure 4-5. MPC Output Comparison With SEM Box Method



SEM Box Method and EPE Measurement

Figure 4-6 illustrates an example of the standard method of EPE measurement when not using the SEM Box method. An initial measurement point is centered on a fragment and additional measurement points ripple outward from the initial measurement point, just as in Calibre nmOPC. The measurement points are spaced `epe_spacing` apart. In this figure, a very large value for `epe_spacing` was chosen so that only one centered measurement point per fragment remains.

Figure 4-6. Standard EPE Measurement



How Calibre nmMPC measures EPE depends on the type of fragment being measured. For convex fragments, these are fragments with two convex corners, the maximum EPE of all evaluation points on this fragment is taken. For concave fragments, the minimum EPE is taken and for all other fragments the average EPE is used for correction.

The SEM Box method changes this metric and calculates the average EPE at all measurement points on concave or convex fragments if they are smaller than `convex_concave_limit`. Setting this command is required for the SEM Box mode, as it determines how line and space ends are considered.

If you have a layout that contains a large number of fragments with short adjacent fragment length (for example, a high volume of jogs on a metal layer), the SEM Box process could consume far more run time in unnecessarily processing each of those fragments. Using the `convex_concave_adj_len` command allows you to specify and filter these types of fragments out of SEM Box processing to save run time.

The SEM Box method also introduces additional measurement points on those fragments, determined by the value of `convex_concave_count`. Suggested values for `convex_concave_count` are between 3 and 7. The value can only be odd and the minimum meaningful value is 3 since using a value of 1 would give the same results as not using the SEM Box mode. Values above 7 are possible but very little accuracy improvement is expected beyond 7.

Note that the SEM Box method does not spread measurement points equally along the fragment, but limits the distribution of measurement points to the center fraction of the fragment given by `convex_concave_metric`. In the following code example, this means only the center 50% of the fragment is used for measuring EPE at evaluation points.

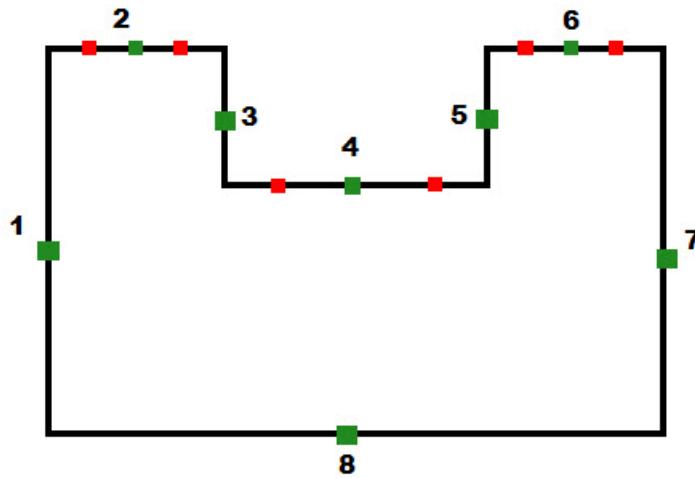
```
convex_concave_correction on    // Turn on SEM Box correction
convex_concave_metric 0.5      // Measure EPE within 50% of the fragment
convex_concave_limit 0.1       // Limit the fragment
                                // length to less than 100nm
convex_concave_count 3        // Select 3 points for EPE measurement
                                // and average them as effective epe for MPC
                                // correction
```

This code block instructs Calibre nmMPC to insert 3 EPE measurement points on fragments based on the following criteria:

- The fragment length is less than 100 nm
- The corners on each end must be of the same type (convex or concave, not both)

In [Figure 4-7](#), fragments 2, 4, and 6 are less than 100 nm. Fragments 2 and 6 both have convex corners at each end, and fragment 4 has concave corners. These fragments qualify for additional EPE evaluation points through the SEM Box method. Fragments 1, 3, 5, 7, and 8 continue to use the standard EPE measurement method.

Figure 4-7. EPE Evaluation Points Added



Additional EPE points are added to fragments 2, 4, and 6. Two additional points are symmetrically distributed on both sides of the existing center-placed EPE point within 50% of the fragment length, as specified by the `convex_concave_metric` command.

Tag Scripting

Once you fragment a polygon, you may only wish to target minor adjustments to certain edges, rather than all fragments. To target those specific fragment edges, you can place unique tags to separate them from the rest of the fragments.

Tagging is the process of defining named subsets of edge fragments that you can reference and manipulate separate from all other fragments.

The commands listed in “[MPC Custom Tcl Scripting Commands](#)” on page 277 are supported in a Calibre nmMPC tagging script.

Correction for Curvilinear Layouts

Calibre nmCLMPC uses an alternative setup file targeted specifically for curvilinear layouts (layouts with skew-edge shapes), known as curvilinear mask process correction or CLMPC.

The alternative setup file is similar to the LITHO MPC-based setup file, but it instead uses a few alternate commands.

- [LITHO CLMPC](#) — Used as an alternative to LITHO MPC to define a LITHO command block, optimized for handling curvilinear mask shapes.
- [setlayer clmpc](#) — The matching command to `setlayer mpc` to define output layers.

- [max_angle_tolerance \(LITHO CLMPC Only\)](#) — Unique to LITHO CLMPC, this command is used in conjunction with the max_smoothing_range to specify the angle tolerance to be considered, along with distance for smoothing edges.
- [max_smoothing_range \(LITHO CLMPC Only\)](#) — Unique to LITHO CLMPC, this command is used for specifying the range along edges used for smoothing. Larger values result in smoother output contours.
- [mpc_poi_measurement \(LITHO CLMPC Only\)](#) — Unique to LITHO CLMPC, this command optimizes EPE measurement time for CLMPC by reducing EPE measurements.
- [mpc_preserve_angles \(LITHO CLMPC Only\)](#) — Unique to LITHO CLMPC, this command preserves edge angles in CLMPC movement.

The following is an example LITHO CLMPC file.

```
LITHO FILE "clmpc.recipe" [
  denseopc_options clmpcOptions {
    algorithm 1
    max_smoothing_range 0.025
    max_angle_tolerance 40
    max_iterations 2
    feedback -0.8 -0.6
    image ebeam_model ebeam0 etch_model etch0
  }
  setlayer mpcOut = clmpc mpcIn MAP mpcIn OPTIONS clmpcOption
]
mpcOut = LITHO CLMPC poly MAP mpcOut FILE "clmpc.recipe"
```

Optional E-Beam Model Definition

The E-beam model simulates a special class of short-range effects, known as E-beam effects, are typically triggered by forward scattering and beam blur.

For E-beam models, a specified kernel (narrow Gaussian) can be applied to capture short range effects triggered by forward scattering and beam blur, which can improve the modeling accuracy and correction fidelity for narrow lines, line ends, corners and 2D features in general. The E-beam simulation component can also improve the fidelity of etch simulation.

You can create a fast E-beam model that models corner rounding and performs simulation. The E-beam model is generated using the [setlayer ebeam_simulate](#) and [ebeam_model_load](#) commands in the MPC setup file.

The ebeam_model_load command defines an E-beam simulation model. The setlayer ebeam_simulate command simulates the deposited energy pattern resist using a Gaussian model, which models forward electron scattering.

For example:

```
ebeam_model_load fastEB {  
    model_type simulation  
    diff_length 0.025  
}
```

You can also reference the E-beam model with the [image](#) command inside the [denseopc_options \(for MPC\)](#) block. For example:

```
image ebeam_model ebeamModel etch_model etchModel
```

An E-beam model can also be generated and calibrated using the [mpcflow_v2](#) command.

An E-beam model can also be generated and calibrated using MPC Center or the [mpcflow_v2](#) command. Refer to “[Stage 5 - Build the E-Beam Model](#)” on page 45 for complete information.

How Calibre nmMPC Measures EPE

Edge placement error (EPE) is the difference between the drawn edge of a polygon and either the simulated silicon edge or actual silicon edge when printed. How EPE is measured varies according to the feature (such as if a fragment is adjacent to a convex or concave corner, at a line end, or if it is a non-corner edge).

In Calibre nmMPC:

- In post-MPC data, every fragment has 2 corners.
- Corner rounding effects must be considered.
- The initial measurement point is centered in a normal fragment and ripple outward from the initial measurement point, just as in nmOPC. The measurement points are spaced `epe_spacing` apart, the default value for nmMPC being 40 nm
- In Calibre nmMPC, adjacent edges have no effect on the measurement of an edge.
- The distance for filtering out EPEs at corners depends only on the smallest sigma value among the e-beam kernels. No EPE measurement points are placed within three times the smallest e-beam sigma from a corner. For example, if you have a 200 nm edge (fragment) and the smallest e-beam kernel has a sigma of 20 nm, then Calibre nmMPC does not place any EPE measurement points within 60 nm from both end-points of the edge. Only the middle 80 nm portion has EPE evaluation points placed. If the fragment length less than 6 times of smallest ebeam sigma, then the EPE measurement will be in the center of the fragment.

Long-Range Correction Process

Long-range pattern density correction utilizes the DENSITY CONVOLVE SVRF statement to generate the biasing map over the mask, and MAPSIZE to resize elements of the layout.

- The DENSITY CONVOLVE SVRF statement allows you to create a representation of your input layer that creates a bias map after applying convolution algorithms. Convolution is a mathematical procedure that produces a weighted moving average of two functions. In this case, it evaluates the density value of a density window by taking into consideration the values of those windows surrounding it. Therefore the density values are spread out to nearby density windows.
- The MDP MAPSIZE SVRF statement allows you to variably resize elements of your layout based on their geographical locations, which are determined by map-size regions. The sizing values are selected as a response to the strength of a physical process effect that requires a data-based correction. Examples are density-based process-loading effects and variation of processing by radial location within a particular machine.

To prepare your data for map-sizing, you must partition your data into non-overlapping map-size regions by creating additional layers in your design data. These new layers are referred to as map layers. You can create these layers either manually, by drawing marker shapes in a layout editor like Calibre MDPview, or derive them inside Calibre using DENSITY, DENSITY CONVOLVE, or other SVRF operations.

To correct for long-range effects:

1. In the SVRF rule deck, insert the Long Range model SVRF block generated by MPC Center. This block contains the calibrated long-range model definition as well as correction layer generated by DENSITY CONVOLVE.
2. You can add a [Short Range \(Etch and E-Beam\) Correction Process](#) section to the same rule file, along with FRACTURE settings.
3. Process the SVRF rule deck using the Calibre engine. Your layout should then be corrected using the modeling data.

Figure 4-8. Example SVRF Block Generated from MPC Center

```
### This generated segment is to be inserted into the correction rule deck

### This is the long-range model definition for DENSITY CONVOLVE
litho file X [
SCALE 8.400366 GAUSS 50.000000
SCALE -6.343858 GAUSS 100.000000
SCALE 1.725309 GAUSS 1000.000000
spatialPolynomial 6.716368e+00 -5.230650e-05x -3.597974e-05y -2.193321e-10xy 1.597709e-10x2
6.286192e-11y2 -5.549886e-16x2y 2.903201e-15y2x 4.177375e-15x3 1.843460e-15y3
]

### Correction layers generated by DENSITY CONVOLVE

n0 = density convolve [in_layer] > 0.500000 <= 1.500000 truncate window 50.000000 file X
n1 = density convolve [in_layer] > 1.500000 <= 2.500000 truncate window 50.000000 file X
n2 = density convolve [in_layer] > 2.500000 <= 3.500000 truncate window 50.000000 file X
n3 = density convolve [in_layer] > 3.500000 <= 4.500000 truncate window 50.000000 file X
n4 = density convolve [in_layer] > 4.500000 <= 5.500000 truncate window 50.000000 file X
n5 = density convolve [in_layer] > 5.500000 <= 6.500000 truncate window 50.000000 file X
n6 = density convolve [in_layer] > 6.500000 <= 7.500000 truncate window 50.000000 file X
n7 = density convolve [in_layer] > 7.500000 <= 8.500000 truncate window 50.000000 file X
n8 = density convolve [in_layer] > 8.500000 <= 9.500000 truncate window 50.000000 file X
n9 = density convolve [in_layer] > 9.500000 <= 10.500000 truncate window 50.000000 file X
n10 = density convolve [in_layer] > 10.500000 <= 11.500000 truncate window 50.000000 file X
n11 = density convolve [in_layer] > 11.500000 <= 12.500000 truncate window 50.000000 file X
n12 = density convolve [in_layer] > 12.500000 <= 13.500000 truncate window 50.000000 file X
n13 = density convolve [in_layer] > 13.500000 <= 14.500000 truncate window 50.000000 file X

### Call to MDP MAPSIZE to perform long-range correction

my_mapsize [MDP MAPSIZE [in_layer] n0 n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12 n13 FILE [
map_size -0.000500 -0.001000 -0.001500 -0.002000 -0.002500 -0.003000 -0.003500 -0.004000
-0.004500 -0.005000 -0.005500 -0.006000 -0.006500 -0.007000
]
```

Short Range (Etch and E-Beam) Correction Process

Proximity or short-range correction can be handled through two different available models.

- Variable Etch Bias model (position-dependent bias relative to the mask edge). In this model, you create a Variable Etch Bias (VEB) setup file block to account for etch effects on the mask.
- E-beam model. In this case, a specified kernel (narrow Gaussian) can be applied to capture short range effects triggered by forward scattering and beam blur, which can improve the modeling accuracy and correction fidelity for narrow lines, line ends, corners and 2D features in general. The E-beam simulation component can also improve the fidelity of etch simulation.

To implement short range correction.

1. Create a setup file using the [LITHO MPC](#) statement.

2. In the setup file, insert the VEB model information generated from the previous stage described in ““Stage 4 - Build Short Range Model” on page 40”. For example:

```

## This is the beginning of the VEB setup file
litho file mpc.setup [
    ## This is the proximity model information (a VEB model named
    ## EM_Nominal to be referenced later in the setup file).
    etch_model_load EM_Nominal {
        version 1
        modelType VEB
        SKERNEL D1 type=gauss es=91 u=25
        SKERNEL D2 type=gauss es=36 u=25
        SKERNEL D3 type=gauss es=287 u=20
        btermCount 4
        BTERM 50
        BTERM -221 D1^1
        BTERM 99 D2^1
        BTERM 0.35 D3^1
    }
    layer masktarget
    ...
]

```

3. For E-beam simulation, create and reference the E-beam model. Refer to the [mpcflow_v2](#) command for further details on creating and calibrating the E-beam model.
4. Create a [denseopc_options \(for MPC\)](#) block. In the denseopc_options block:
 - a. Reference the models (VEB and E-beam) using the [image](#) command.
 - b. Make sure that no new fragmentation is introduced.
 - c. Refer to [Table 4-1](#) on page 55 for other possible operations you can add to the denseopc_options block.
5. Process the SVRF rule deck using the Calibre engine. Your layout should then be corrected using the modeling data.

Correct and Verify the Design Short Range (Etch and E-Beam) Correction Process

For example:

```
litho file "mpc.setup" [
    etch_model_load EM_Nominal {
        version 1
        modelType VEB
        SKERNEL D1 type=gauss es=91 u=25
        SKERNEL D2 type=gauss es=36 u=25
        SKERNEL D3 type=gauss es=287 u=20
        btermCount 4
        BTERM 50
        BTERM -221 D1^1
        BTERM 99 D2^1
        BTERM 0.35 D3^1
    }
    ebeam_model_load fastEB {
        model_type simulation
        diff_length 0.025
    }
    processing_mode flat
    background clear
    layer masktarget visible dark
    denseopc_options srcOptions {
        version 1
        algorithm 1
        tile_fragmentation_only 1
        background dark
        layer masktarget opc clear
        max_iterations 2
        feedback -0.98
        image ebeam_model fastEB etch_model EM_Nominal
    }
    ...
    setlayer srcModelResult = mpc masktarget MAP masktarget OPTIONS srcOptions
]
```

This diagram illustrates the structure of a Calibre mpc.setup script with annotations explaining specific sections:

- This is the VEB model.**: Points to the line `modelType VEB`.
- This is an E-Beam model.**: Points to the line `model_type simulation`.
- Start the denseopc_options block**: Points to the opening brace `{` of the `denseopc_options` block.
- Reference the models**: Points to the `srcOptions` block.
- Enable retargeting for VEB**: Points to the `image` command within the `srcOptions` block.

Once the setup file is complete, it can be run from the SVRF rule deck.

6. Run the SVRF rule deck using the Calibre engine. Your fractured output should then be corrected using the specified models.

```

litho file mpc.setup [
    etch_model_load EM_Nominal {
        version 1
        modelType VEB
        SKERNEL D1 type=gauss es=91 u=25
        SKERNEL D2 type=gauss es=36 u=25
        SKERNEL D3 type=gauss es=287 u=20
        btermCount 4
        BTERM 50
        BTERM -221 D1^1
        BTERM 99 D2^1
        BTERM 0.35 D3^1
    }
    ebeam_model_load fastEB {
        model_type simulation
        diff_length 0.025
    }
    processing_mode flat
    background clear
    layer masktarget visible dark
    denseopc_options srcOptions {
        version 1
        algorithm 1
        tile_fragmentation_only 1
        background dark
        layer masktarget opc clear
        max_iterations 2
        feedback -0.98
        image ebeam_model fastEB etch_model EM_Nominal
    }
    setlayer srcModelResult = mpc masktarget MAP masktarget \
        OPTIONS srcOptions
]

```

Combined Model Example with FRACTURE

For most cases, you can perform both long-range and proximity corrections in the same SVRF rule deck. The following example shows both long-range and proximity corrections along with a FRACTURE run in an SVRF rule deck.

Correct and Verify the Design

Short Range (Etch and E-Beam) Correction Process

```
layout system oasis
layout path "../input.oas"
layout primary "*"
precision 10000
layout error on input no

layer masktarget 2

drc maximum results all
drc results database "/dev/null" oasis

// Density convolve to generate the map-regions for long-range correction.
// coefficients
//    1.8823063e+01 (A0)
//    1.6239218e+01 (C1, sigma=8000 microns)
//    1.9131294e-05 (A1, x nm/microns)
//    1.9628250e-05 (A2, y nm/microns)
//    9.0055064e-11 (A3, xy nm/microns^2)
//    6.6496318e-10 (A4, x^2 nm/microns^2)
//    3.2541606e-10 (A5, y^2 nm/microns^2)

litho file X [
    scale 1.6239218e+01 gauss 8000.000000
    spatialPolynomial 1.8823063e+01 1.9131294e-05x \
        1.9628250e-05y 9.0055064e-11xy 6.6496318e-10x2 3.2541606e-10y2
]

n0 = density convolve masktarget > 13.500000 <= 14.500000 \
    TRUNCATE window 50.000000 file X
n1 = density convolve masktarget > 14.500000 <= 15.500000 \
    TRUNCATE window 50.000000 file X
n2 = density convolve masktarget > 15.500000 <= 16.500000 \
    TRUNCATE window 50.000000 file X
n3 = density convolve masktarget > 16.500000 <= 17.500000 \
    TRUNCATE window 50.000000 file X
n4 = density convolve masktarget > 17.500000 <= 18.500000 \
    TRUNCATE window 50.000000 file X
n5 = density convolve masktarget > 18.500000 <= 19.500000 \
    TRUNCATE window 50.000000 file X
n6 = density convolve masktarget > 19.500000 <= 20.500000 \
    TRUNCATE window 50.000000 file X
n7 = density convolve masktarget > 20.500000 <= 21.500000 \
    TRUNCATE window 50.000000 file X
n8 = density convolve masktarget > 21.500000 <= 22.500000 \
    TRUNCATE window 50.000000 file X
n9 = density convolve masktarget > 22.500000 <= 23.500000 \
    TRUNCATE window 50.000000 file X
n10 = density convolve masktarget > 23.500000 <= 24.500000 \
    TRUNCATE window 50.000000 file X
n11 = density convolve masktarget > 24.500000 <= 25.500000 \
    TRUNCATE window 50.000000 file X
n12 = density convolve masktarget > 25.500000 <= 26.500000 \
    TRUNCATE window 50.000000 file X
n13 = density convolve masktarget > 26.500000 <= 27.500000 \
    TRUNCATE window 50.000000 file X
n14 = density convolve masktarget > 27.500000 <= 28.500000 \
    TRUNCATE window 50.000000 file X
n15 = density convolve masktarget > 28.500000 <= 29.500000 \
```

```

        TRUNCATE window 50.000000 file X
n16 = density convolve masktarget > 29.500000 <= 30.500000 \
        TRUNCATE window 50.000000 file X
n17 = density convolve masktarget > 30.500000 <= 31.500000 \
        TRUNCATE window 50.000000 file X
n18 = density convolve masktarget > 31.500000 <= 32.500000 \
        TRUNCATE window 50.000000 file X
n19 = density convolve masktarget > 32.500000 <= 33.500000 \
        TRUNCATE window 50.000000 file X
n20 = density convolve masktarget > 33.500000 <= 34.500000 \
        TRUNCATE window 50.000000 file X
n21 = density convolve masktarget > 34.500000 <= 35.500000 \
        TRUNCATE window 50.000000 file X
n22 = density convolve masktarget > 35.500000 <= 36.500000 \
        TRUNCATE window 50.000000 file X
n23 = density convolve masktarget > 36.500000 <= 37.500000 \
        TRUNCATE window 50.000000 file X

// MPC

litho file mpc.setup [
    layer masktarget
    etch_model_load EM_Nominal {
        version 1
        modelType VEB
        SKERNEL D1 type=gauss es=61 u=4.8
        SKERNEL D2 type=gauss es=481 u=-340
        btermCount 3
        BTERM 214
        BTERM -381 D1^1
        BTERM -16 D2^1
    }
    ebeam_model_load fastEB {
        model_type simulation
        diff_length 0.025
    }
    processor_mode flat
    denseopc_options srcOptions {
        version 1
        algorithm 1
        background dark
        layer masktarget opc clear
        max_iterations 2
        feedback -0.98
        image ebeam_model fastEB etch_model EM_Nominal
    }
    setlayer srcResult = mpc masktarget MAP masktarget \
        OPTIONS srcOptions
]

// Short-range correction.
srcResult = litho mpc masktarget map srcResult file "mpc.setup"

lrcResult = mdp mapsizer srcResult n0 n1 n2 n3 n4 n5 n6 n7 n8 n9 \
            n10 n11 n12 n13 n14 n15 n16 n17 n18 n19 n20 n21 n22 \
            n23 file
[

```

Correct and Verify the Design

Short Range (Etch and E-Beam) Correction Process

```
soft_boundary 1
map_size -0.007000 -0.007500 -0.008000 -0.008500 \
           -0.009000 -0.009500 \ -0.01000 -0.010500 -0.011000 \
           -0.011500 -0.012000 -0.012500 -0.013000 \
           -0.013500 -0.014000 -0.014500 -0.01500 \
           -0.015500 -0.016000 -0.016500 \
           -0.017000 -0.017500 -0.018000 -0.018500
]

// Output result.
output = copy lrcResult

// FRACTURE SECTION: section-mode MPC vsb12 fracture.
my_mpc {
    fracture nuflare output file [
        // VSB12 fracture parameters.
        max_skew_approximation_error 0.01
        conversion_address_unit 0.001
        frame_width 256
        chip_directory "mpc.vsb12"
        log_file_name "mpc.vsb12.log"
        version 12
        small_value 0.1
        cd internal auto
        computation_mode section
    ]
}
```

Verify Corrections

Validate the output of your correction run using Calibre OPCverify and with Calibre nmMPC.

Verifying the Corrections With Calibre OPCverify.....	81
Verifying EPE Layers	84

Verifying the Corrections With Calibre OPCverify

You can validate the output of your correction run using the Calibre OPCverify command measure_ope. Specifically, you can find the MPC fragments where the EPE lies in a specified interval.

Procedure

1. Pass the target layer as an OPC layer and the Calibre nmOPC layer as a correction layer.
2. Simulate the contour of the corrected mask layer.
3. Use the Calibre OPCverify command measure_ope to measure EPE from the target layer to the contour layer edges.

Examples

The following is an example rule deck containing a verification through measurement of EPE intervals.

```

LAYOUT SYSTEM OASIS
LAYOUT PATH 'gds/mpc_out.oas'
LAYOUT PRIMARY '*'
DRC MAXIMUM RESULTS ALL
PRECISION 4000
DRC RESULTS DATABASE 'gds/mpcVerify.oas' OASIS

LAYER poly 0 // Mask Target.
LAYER polyMPC 1 // MPC output for the target.

LITHO FILE mpcin [
    etch_model_load etch {
        version 1
        modelType VEB
        SKERNEL D1 type=gauss es=131.5 u=-49.6
        SKERNEL D2 type=gauss es=131.5 u=-29.6
        SKERNEL D3 type=gauss es=538.19 u=10.016
        SKERNEL D4 type=visible es=284.64 u=10 looking=out
        btermCount 11
        BTERM 19.3383
        BTERM 19.6769 D1^1
        BTERM -60.7564 D2^1
        BTERM -15.7873 D3^1
        BTERM 23.2994 D4^1
        BTERM 7.89299 D1^1*D2^1
        BTERM 52.7954 D1^1*D3^1
        BTERM 7.96471 D1^1*D4^1
        BTERM -5.18151 D2^1*D3^1
        BTERM -42.7745 D2^1*D4^1
        BTERM -2.99799 D3^1*D4^1
    }
    ebeam_model_load eb {
        model_type simulation
        diff_length .021
    }
    procssing_mode flat
    background dark
    tilemicrons 100.0
    layer poly visible clear
    layer polyMPC visible clear

// Simulate the contour of the corrected mask.
setlayer ebeamContour = ebeam_simulate polyMPC ebeam_model eb
setlayer maskContour = veb_simulate ebeamContour etch_model etch
setlayer jog_filter = filter_generate poly jog 0.06 0.05 expand 0.01
setlayer
neg = measure_epe maskContour poly epe_spacing 0.040 outside \
    jog_filter only <= -0.001 function min min_featsize 0.010 \
    \
    max_edgelen 0.100 output_expanded_edges 0.001 trim_ends 0.002 \
    property {min}
setlayer
neg2 = measure_epe maskContour poly epe_spacing 0.040 \
    outside jog_filter only > -0.001 <= -0.0005 function min \
    \
    min_featsize 0.010 max_edgelen 0.100 output_expanded_edges \
    0.001 \
    trim_ends 0.002 property {min}

```

```

setlayer
neg1 = measure_epe maskContour poly epe_spacing 0.040 \
    outside
jog_filter only > -0.0005 <= -0.00025 function min \
    min_featsize
0.010 max_edgelen 0.100 output_expanded_edges 0.001 \
    trim_ends
0.002 property {min}
setlayer
zero = measure_epe maskContour poly epe_spacing 0.040 \
    outside jog_filter only > -0.00025 < 0.00025 function min \
    min_featsize 0.010 max_edgelen 0.100 output_expanded_edges 0.001 \
    trim_ends 0.002 property {min}
setlayer
pos1 = measure_epe maskContour poly epe_spacing 0.040 \
    outside jog_filter only >= 0.00025 < 0.0005 function max \
    min_featsize 0.010 max_edgelen 0.100 output_expanded_edges 0.001 \
    trim_ends 0.002 property {max}
setlayer
pos2 = measure_epe maskContour poly epe_spacing 0.040 \
    outside jog_filter only >= 0.0005 < 0.001 function max \
    min_featsize 0.010 max_edgelen 0.100 output_expanded_edges 0.001 \
    trim_ends 0.002 property {max}
setlayer
pos = measure_epe maskContour poly epe_spacing 0.040 \
    outside jog_filter only >= 0.001 function max min_featsize 0.010 \
    max_edgelen 0.100 output_expanded_edges 0.001 trim_ends 0.002 \
    property {max}
// Classify one of the outputs using 'copy'.
setlayer posReduced = copy pos classify {
    context poly
    halo 0.020
}
]
// Compute EPE bins.
neg = LITHO MPC poly polyMPC MAP neg FILE mpcin
neg2 = LITHO MPC poly polyMPC MAP neg2 FILE mpcin
neg1 = LITHO MPC poly polyMPC MAP neg1 FILE mpcin
zero = LITHO MPC poly polyMPC MAP zero FILE mpcin
pos1 = LITHO MPC poly polyMPC MAP pos1 FILE mpcin
pos2 = LITHO MPC poly polyMPC MAP pos2 FILE mpcin
pos = LITHO MPC poly polyMPC MAP pos FILE mpcin
posReduced = LITHO MPC poly polyMPC MAP posReduced FILE mpcin
neg { copy neg }
neg2 { copy neg2 }
neg1 { copy neg1 }
zero { copy zero }
pos1 { copy pos1 }
pos2 { copy pos2 }
pos { copy pos }
posReduced { copy posReduced }

DRC CHECK MAP neg 101
DRC CHECK MAP neg2 102
DRC CHECK MAP neg1 103
DRC CHECK MAP zero 104

```

```
DRC CHECK MAP pos1 105
DRC CHECK MAP pos2 106
DRC CHECK MAP pos 107
DRC CHECK MAP posReduced 108
dfm_rdb_101 { DFM RDB neg "reports/epeProps.dfmdb" }
dfm_rdb_102 { DFM RDB neg2 "reports/epeProps.dfmdb" }
dfm_rdb_103 { DFM RDB neg1 "reports/epeProps.dfmdb" }
dfm_rdb_104 { DFM RDB zero "reports/epeProps.dfmdb" }
dfm_rdb_105 { DFM RDB pos1 "reports/epeProps.dfmdb" }
dfm_rdb_106 { DFM RDB pos2 "reports/epeProps.dfmdb" }
dfm_rdb_107 { DFM RDB pos "reports/epeProps.dfmdb" }
```

Verifying EPE Layers

EPEs in an MPC recipe can be verified by deriving a specific EPE layer, running a Calibre nmMPC simulation, and then passing it to Calibre MPCVerify for verification.

When generating the EPE layer, the measurement locations should be consistent with the correction layer. This means specifying an opc layer as the target layer, and an external layer as the corrected layer for EPE layer verification. Measurement sites are placed in the opc target layer, and the external layer is used for simulation. This ensures that both the correction flow and verification flow get the same measurement points, provided the same recipe settings for fragmentation, epe_spacing, and so on, in both flows.

Procedure

1. In your Calibre nmMPC recipe, create an EPE layer using the following rules:
 - Specify the target layer as the opc layer.
 - Specify the corrected layer as an external layer. This means that only this layer is to be simulated. Note that, in this particular context, this is not the final simulated layer as typically generated in an MPC recipe. This is used specifically for EPE layer verification.
 - Set max_iterations to 1
 - Set feedback to 1
2. Simulate the corrected layer.
3. Use the Calibre MPCVerify command measure_epe to measure EPE from the target layer to the contour layer edges.

Results

In hier or hybrid mode, tiling is slightly shifted in EPE layer generation compared to the correction tiling. This is due to the additional correction layer as input to the EPE layer generation recipe, which slightly changes the cell extents. This issue exists only in hier or hybrid mode, where tiling depends on the cell extents. In flat mode, the tiling is consistent in both correction and EPE layer generation flows and the measurement points are consistent.

Due to this shift in the tiling in hier or hybrid mode, a few fragments at tile boundaries may get different measurement points and EPEs computed in correction may differ slightly on these tile boundary fragments.

Examples

The following example illustrates both a correction and verification rule file for EPE layer verification:

```
// --- MPC RECIPE FOR GENERATING EPE LAYER
LITHO FILE MPC.recipe [
...
    layer MpcTarget
    layer MpcOut
...
    tilemicrons 40
...
    denseopc_options mpcOptions {
...
        version          1
        background       dark
        algorithm        1
...
// --- Specify an opc target layer and an external layer specifically
// --- for edge movement
        layer MpcTarget      opc      clear
        layer MpcOut        external clear
...
// --- Set max_iterations to 1 and feedback to 1
        max_iterations 1
        feedback 1
...
        OPC_ITERATION 1
...
    }
setlayer EPE_layer = mpc MpcTarget MpcOut MAP MpcTarget OPTIONS mpcOptions
]

LITHO FILE MPCV.recipe [
...
    layer MpcTarget
    layer EPE_layer
...
    setlayer lineEnds = identify_edge MpcTarget length <= 0.300 > 0.120 \
                        length1 >= 0.150 length2 >= 0.150 \
                        corner1 convex corner2 convex extend -0.045 \
                        expand $filterWidth
...
    setlayer epeLE = measure_epe EPE_layer MpcTarget inside lineEnds \
                     epe_spacing 0.030 function max \
                     min_featsize 0.030 max_edgelen 0.35 \
                     fragment only not > -0.0006 < 0.0006 \
                     output_expanded_edges 0.001 trim_ends 0.002 \
                     property {magnitude_max
                     } classify {
                     context MpcTarget
                     halo 0.05
                     maximum_error_number 100000
                     anchor MpcTarget
```

```
    reflections_rotations_match yes
    score bin_size 0.001 largest
    worst 500 exact
}
...
]

// --- Generate the EPE layer using the MPC recipe
EPE_layer = LITHO MPC MpcTarget MpcOut MAP EPE_layer FILE MPC.recipe

// --- Generate verify output using MPCv recipe
epeLE = LITHO MPCVERIFY MpcTarget EPE_layer MAP epeLE FILE MPCV.recipe
```


Chapter 5

Calibre nmMPC Reference

Calibre nmMPC is largely a batch command-based tool. All major operations can be performed through batch commands in the setup file.

Test Layout Generation Commands	92
mpcCompile	93
ptpCompile	94
ptpCompile Input File Format	96
uniCompile	103
uniCompile Input File Format	104
mtpCompile	105
mtpCompile Input File Format	107
MPC Calibration Batch Commands	113
mpcggConvert/mpcssConvert	114
mpclr	115
mpclrop	119
mpceval	122
Long-Range Model File Format	124
mpcflow_v2	126
Model-Layer File Format	131
MPC Correction Commands	133
LITHO MPC	135
LITHO CLMPC	136
denseopc_options (for MPC)	138
direct_input	139
ebeam_model_load	141
etch_model_load	145
layer	146
modelpath	147
processing_mode	148
setlayer ebeam_simulate	149
setlayer curve	151
setlayer curve_target	161
setlayer mpc	167
setlayer clmpc	169
MPC denseopc_options Commands	170
algorithm	174
allow_jog_full_move	175
background (for denseopc_options block)	176
check_movement	177

convex_concave_adj_len	178
convex_concave_correction	179
convex_concave_count	180
convex_concave_limit	181
convex_concave_metric	182
corner_control	183
epe_spacing	184
feedback	185
fragment_coincident	186
fragment_corner	188
fragment_flaglayer	197
fragment_inter	198
fragment_interlayers	212
fragment_island	214
fragment_layer	218
fragment_layer_order	220
fragment_max	222
fragment_min	225
fragment_minjog	226
fragment_nearly_coincident	227
fragment_visible	230
fragment_small	231
freeze_skew_edges	232
grids_for_angle_45_snap	233
image	234
image_options	236
jog_freeze	237
jog_ignore_metric	238
layer (denseopc_options Command)	240
line_end_fragment	242
max_angle_tolerance (LITHO CLMPC Only)	243
max_iterations	244
max_smoothing_range (LITHO CLMPC Only)	245
max_iter_movement	246
max_opc_move	247
measure_metric_mode	248
mpc_contour_search_radius	249
mpc_fragment_type_projection	250
mpc_poi_measurement (LITHO CLMPC Only)	251
mpc_preserve_angles (LITHO CLMPC Only)	252
mpc_pseudo_nyquist	253
mpc_sites_control	254
mpc_sites_mode	255
mpc_skip_fragments	257
mpc_use_nearest_epe	258

mrc_rule	259
no_opc_region	266
opc_grid_multiplier	267
retarget_layer	268
scale	272
sparse_ebeam_veb	273
step_size	274
tile_fragmentation_only	275
version	276
MPC Custom Tcl Scripting Commands	277
DISPLACEMENT	280
FEEDBACK	285
fragalign	287
FRAGMENT delete	290
FRAGMENT modify	291
FRAGMENT split	292
FRAGMENT_EPE_MEASURE_METHOD	294
FRAGMENT_MOVE	295
MRC	297
MRC_RULE (Custom Scripting Command)	299
NEWTAG all	301
NEWTAG blocked	302
NEWTAG cd	303
NEWTAG complete	305
NEWTAG displacement	307
NEWTAG edge	309
NEWTAG epe	313
NEWTAG external internal enclose enclosing	315
NEWTAG fragment	319
NEWTAG line_end space_end	324
NEWTAG neighbor	325
NEWTAG sequence	326
NEWTAG topological	328
NEWTAG vertical horizontal all_angle 45_angle	330
OPC_ITERATION	331
OUTPUT_MEASUREMENT_LOCATIONS	333
OUTPUT_OPCT	334
OUTPUT_SHAPE fragment	336
TAG and or subtract	339
Calibre nmMPC File Formats	340
Lithomodel (Litho Model Format)	341

Test Layout Generation Commands

The following table summarizes the commands used for test layout generation.

Table 5-1. Test Layout Generation Commands

Tcl Command	Description
mpcCompile	Generates a proximity, uniformity, and mask model test layout (combines the execution of ptpCompile , uniCompile , and mtpCompile).
ptpCompile	Generates a proximity model (short range) test layout and corresponding measurement spreadsheet.
uniCompile	Generates a uniformity test mask layout and corresponding spreadsheet.
mtpCompile	Generates a mask model (short-range and long range) test layouts, which consists of proximity, density loading, and uniformity models.

mpcCompile

Test Layout Generation Commands

Generates a proximity, uniformity, and mask model test layout (combines the execution of ptpCompile, uniCompile, and mtpCompile).

Usage

mpcCompile
mtpfilename

ptpfilename

unifilename

Arguments

- ***mtpfilename***

Specifies the input file. For information on the input file. Refer to “[mtpCompile Input File Format](#)” on page 107 for further information.

- ***ptpfilename***

Specifies the proximity test pattern specification file. Refer to “[ptpCompile Input File Format](#)” on page 96 for further information.

- ***unifilename***

Specifies the uniformity test pattern specification file and corresponding gauge file. The file format is the same as the ptpCompile input file. Refer to “[ptpCompile Input File Format](#)” on page 96 for further information.

Description

This command combines the [ptpCompile](#), [mtpCompile](#), and [uniCompile](#) commands into a single execution. The inputs are the same for each of the separate commands.

ptpCompile

Test Layout Generation Commands

Generates a proximity model (short range) test layout and corresponding measurement spreadsheet.

Usage

ptpCompile *ptpfilename*

Arguments

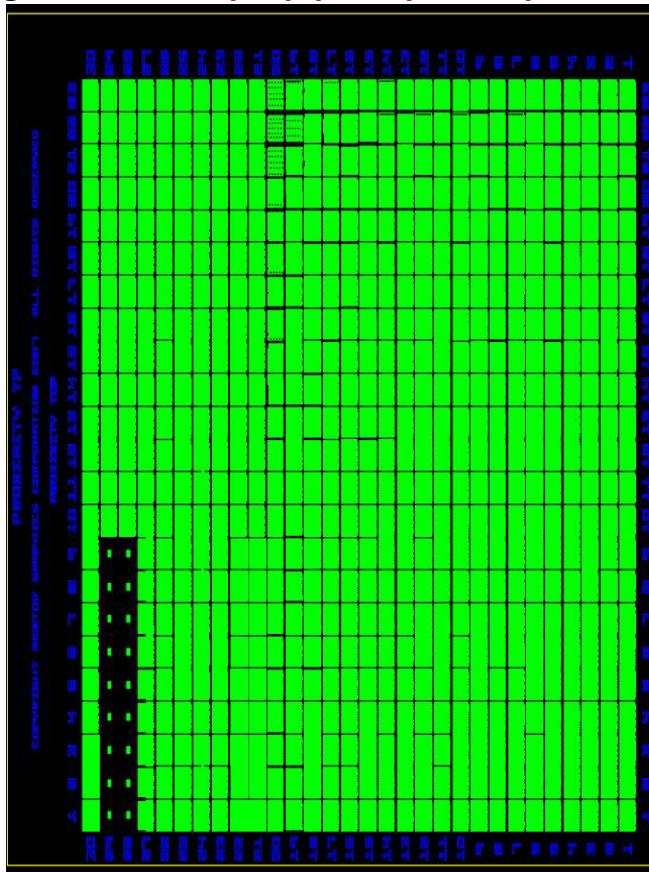
- *ptpfilename*

Specifies the proximity test pattern specification file, refer to “[ptpCompile Input File Format](#)” on page 96 for further information.

Description

Generates proximity model (short-range) test layouts and corresponding gauge output, which consists of calibration structures, validation structures, and corner rounding model calibration structures. [Figure 5-1](#) shows an example output pattern generated by ptpCompile.

Figure 5-1. Example ptpCompile Output Pattern



Examples

Create a proximity test layout from a specified input file.

```
% ptpCompile proximity.in
```

ptpCompile Input File Format

Input file for ptpCompile

The ptpCompile input file is used to define the parameters for the ptpCompile run.

Format

The ptpCompile input file consists of several lines of keywords. A keyword defines the scope for any subsequent text lines until another keyword is found. A line that begins with a # character is a comment.

Parameters

- **dbggrid *val***
Database grid, in microns, in which to specify the test layout features.
- **designgrid *val***
Design grid, in microns. By default, it is assumed to be the value of dbggrid.
- **topcellname *top_cell_name***
The name of the top cell to create for the current test layout.
- **gdsfile *gds_file_name***
The output GDS file to write.
- **ggfile *gg_file_name***
The output gauge file to write.
- **inset *val***
Sets the gap between two adjacent features, in microns. The default is 10 microns.
- **blocksize *val***
Sets the size of each feature block, in microns. The default is 50.0 microns.
- **FEATURE *feature_name***
Begins the definition of a feature in proximity model. The *feature_name* syntax is described in the following section, “”.

Supported Feature Names

The list of structure names along with the required parameter list are shown below. The *val* is in microns, and should never exceed (blocksize-2*inset).

- **line_space**

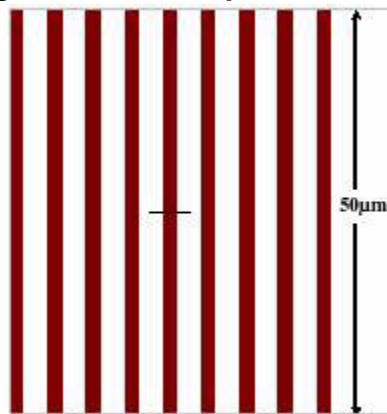
Specifies periodic line and spaces. Enter a set of line widths and a set of space values. Line/Space features are generated for each combination. There is one CD measurement location, as marked by the drawn line, which is the line width at the center of the structure.

Dimension values include:

Spaces {*val*}+

Widths {*val*}+

Figure 5-2. line_space Example



- post

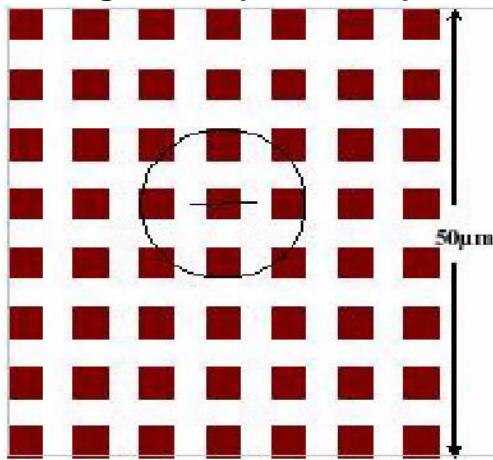
Specifies periodic posts. Enter the following information:

Spacing values: Spaces {*val*}+

Post widths: Widths {*val*}+

Post features are generated for each combination. There is one CD measurement location, as marked by the drawn line, which is the post width at the center of the structure.

Figure 5-3. post Example



- contact

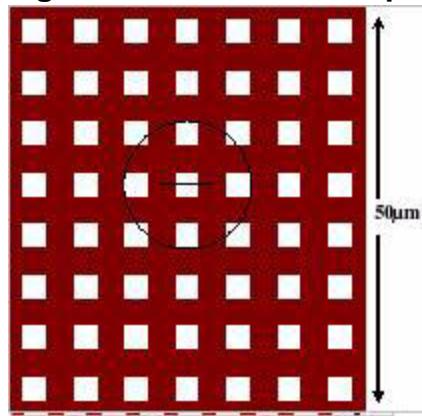
Specifies periodic contacts. Enter the following information:

Spacing values: Spaces {*val*}+

Contact widths: Widths {*val*}+

Contact features are generated for each combination. There is one CD measurement location, as marked by the drawn line, which is the contact space at the center of the structure.

Figure 5-4. contact Example



- **line_end**

Specifies periodic line/space features with line-end gaps. Enter the following information:

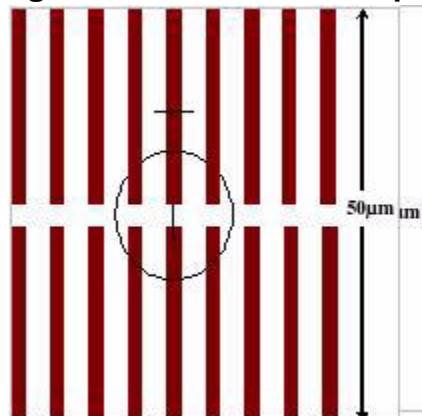
Spacing values: Spaces {*val*}+

Line widths: Widths {*val*}+

Line-end gaps: Gaps {*val*}+

Line/Space features are generated for each combination. There are two measurement locations, as marked by the lines shown. One is the gap along the vertical direction at the center of the structure, the other is line width along the horizontal direction at the upper center of the structure.

Figure 5-5. line_end Example



- **space_end**

Specifies periodic line/space features with space-end gaps. Enter the following information:

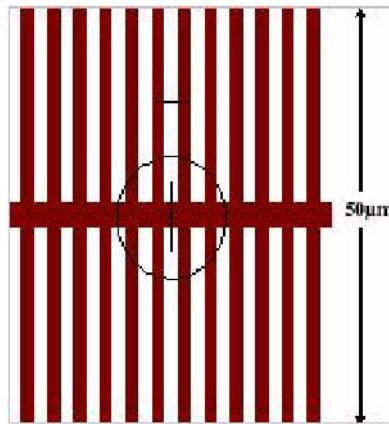
Spacing values: Spaces {*val*}+

Line widths: Widths {*val*}+

Space-end gaps: Gaps {*val*}+

Line/Space features are generated for each combination. There are two measurement locations, as shown by the marked lines. One is the gap along the vertical direction at the center of the structure, the other is space alone horizontal direction at the upper center of the structure.

Figure 5-6. space_end Example



- **single_line_end**

Specifies a periodic line/space features with single line-end gap.

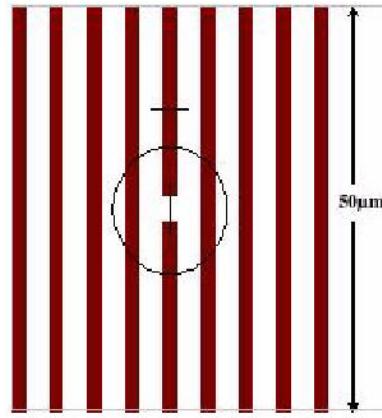
Spacing values: Spaces $\{val\}+$

Line widths: Widths $\{val\}+$

Line-end gaps: Gaps $\{val\}+$

Line/Space features are generated for each combination. There are two measurement locations, as marked by the lines shown. One is the gap along the vertical direction at the center of the structure, the other is line width along the horizontal direction at the upper center of the structure.

Figure 5-7. single_line_end Example



- **single_space_end**

Specifies periodic line/space features with single space-end gap.

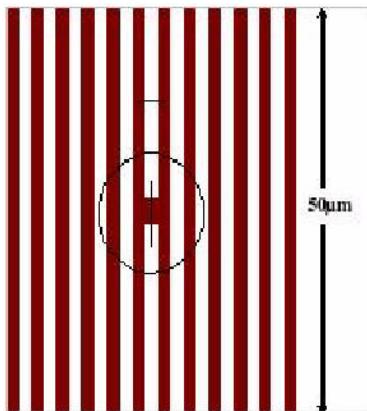
Spacing values: Spaces {val}+

Line widths: Widths {val}+

Space-end gaps: Gaps {val}+

Line/Space features are generated for each combination. There are two measurement locations, as shown by the marked lines. One is the gap along the vertical direction at the center of the structure, the other is space alone horizontal direction at the upper center of the structure.

Figure 5-8. single_space_end Example



- fidelity

Specifies fidelity patterns. Enter the following information:

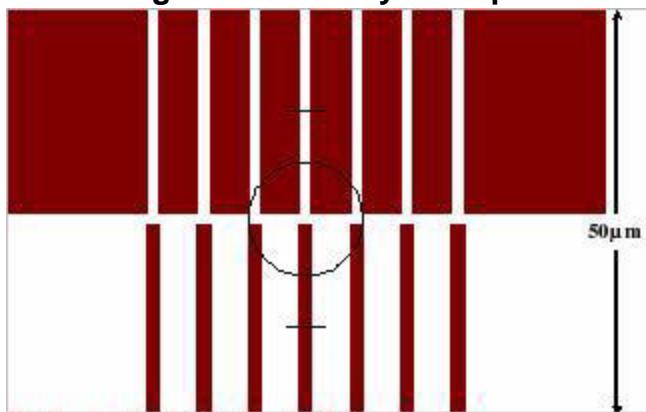
Spacing values: Spaces {val}+

Widths: Widths {val}+

Gap values: Gaps {val}+

Fidelity patterns are generated for each combination. There are two measurement locations, as marked by the drawn lines. One is line width at lower center of the structure, the other is space at the upper center of the structure.

Figure 5-9. fidelity Example



- post_corner

Specifies periodic posts. It has the same structure and measurement location as feature post, except its size is only 1/10th of the size of the feature post. Enter the following information:

Spacing values: Spaces {*val*}+

Post widths: Widths {*val*}+

Post features are generated for each combination. Post width values are usually very small, (for example, 40, 60, 80, 100, and 120 nm).

- contact_corner

Specifies periodic contacts. It has the same structure and measurement location as feature contact, except its size is only 1/10th of the size of the feature contact. Enter the following information:

Spacing values: Spaces {*val*}+

Contact widths: Widths {*val*}+

Contact features are generated for each combination. Contact width values are usually very small (for example, 40, 60, 80, 100, and 120 nm).

- chess

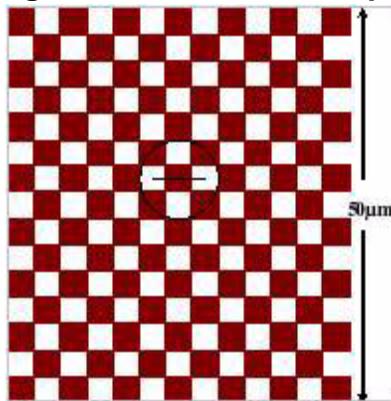
Specifies a chess-board pattern. Enter the following information:

Spacing values: Spaces {*val*}+

Contact widths: Widths {*val*}+

Contact features are generated for each combination. There is one measurement location, as marked by the drawn line, which is the line width at the center of the structure.

Figure 5-10. chess Example



- line_space_fill

Specifies a line_space_fill pattern. Enter a pair of space width values to generate line-and-space patterns to fill unused space on the mask:

Spacing values: Spaces {*val*}+

Widths: Widths {*val*}+

Examples

The following file, **proximity.in** is used for generating proximity structures:

```
PATTERN "proximity"
dbggrid 0.001
designgrid 0.001
dbunit 1
topcellname PROXIMITY_TOP
blocksize 50
gdsfile proximity.gds
ssfile proximity.gg
markers none
border 10
FEATURE line_space
Spaces 0.120 0.140 0.160 0.180 0.200 0.220 0.250 0.270 0.300 0.360 0.420 0
1.000 1.200 .460 0.520 0.600 0.700 0.800
0.5W2i0d t0h.s6 0 00 .F0E8A0T U0R.E0 9p0o s0t.100 0.110 0.120 0.140 0.160
0.180 0.200 0.220 0.250 0.270 0.300 0.360 0.420 0.460
Spaces 0.200 0.300 0.500
Widths 0.200 0.300 0.500
FEATURE contact
Spaces 0.200 0.300 0.500
Widths 0.200 0.300 0.500
FEATURE line_end
Spaces 0.200 0.250 0.300
Widths 0.200 0.250 0.300
Gaps 0.320 0.500
FEATURE space_end
Spaces 0.200 0.250 0.300
Widths 0.200 0.250 0.300
Gaps 0.320 0.500
FEATURE single_line_end
Spaces 0.200 0.250 0.300
Widths 0.200 0.250 0.300
Gaps 0.320 0.500
FEATURE single_space_end
Spaces 0.200 0.250 0.300
Widths 0.200 0.250 0.300
Gaps 0.320 0.500
FEATURE fidelity
Spaces 1.000 1.000 1.000
Widths 0.200 0.250 0.300
Gaps 0.320
FEATURE post_corner
Spaces 0.040 0.060 0.080
Widths 0.040 0.060 0.080
FEATURE contact_corner
Spaces 0.040 0.060 0.080
Widths 0.040 0.060 0.080
FEATURE chess
Widths 0.200 0.250 0.300
FEATURE line_space_fill
Spaces 0.250
Widths 0.250
```

uniCompile

Test Layout Generation Commands

Generates a uniformity test mask layout and corresponding spreadsheet.

Usage

uniCompile *unifilename*

Arguments

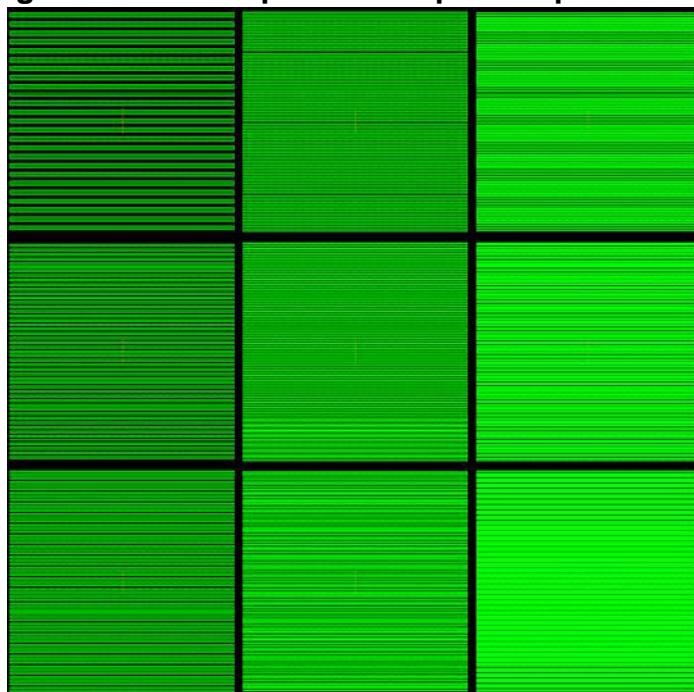
- ***unifilename***

Specifies the uniformity test pattern specification file and corresponding gauge file. The file format is the same as the ptpCompile input file (see “[ptpCompile Input File Format](#)” on page 96).

Description

Generates a uniformity test mask layout and corresponding spreadsheet.

Figure 5-11. Example uniCompile Output Pattern



Examples

Create a uniformity test pattern:

```
% uniCompile uniformity.in
```

uniCompile Input File Format

The input file for the uniCompile utility

The uniCompile input file defines the parameters for the uniCompile run.

Format

The uniCompile input file uses the same format as the ptptCompile input file. See “[ptptCompile Input File Format](#)” on page 96 for a complete description.

Parameters

The parameters are identical to the ptptCompile input file.

Examples

```
# Test Pattern File format
PATTERN "proximity"
dbgrid 0.001
designgrid 0.001
dbunit 1
topcellname UNIFORMITY_TOP
blocksize 50
gdsfile uniformity.gds
ssfile uniformity.gg
markers none
border 2
FEATURE line_space
Spaces 0.2 0.22 0.25 0.28 0.3 0.35 0.4 0.5 1.0
Width 0.2 0.22 0.25 0.28 0.3 0.35 0.4 0.5 1.0
#End of sample 1
```

mtpCompile

Test Layout Generation Commands

Generates a mask model (short-range and long range) test layouts, which consists of proximity, density loading, and uniformity models.

Usage

mtpCompile *mtpfilename*

Arguments

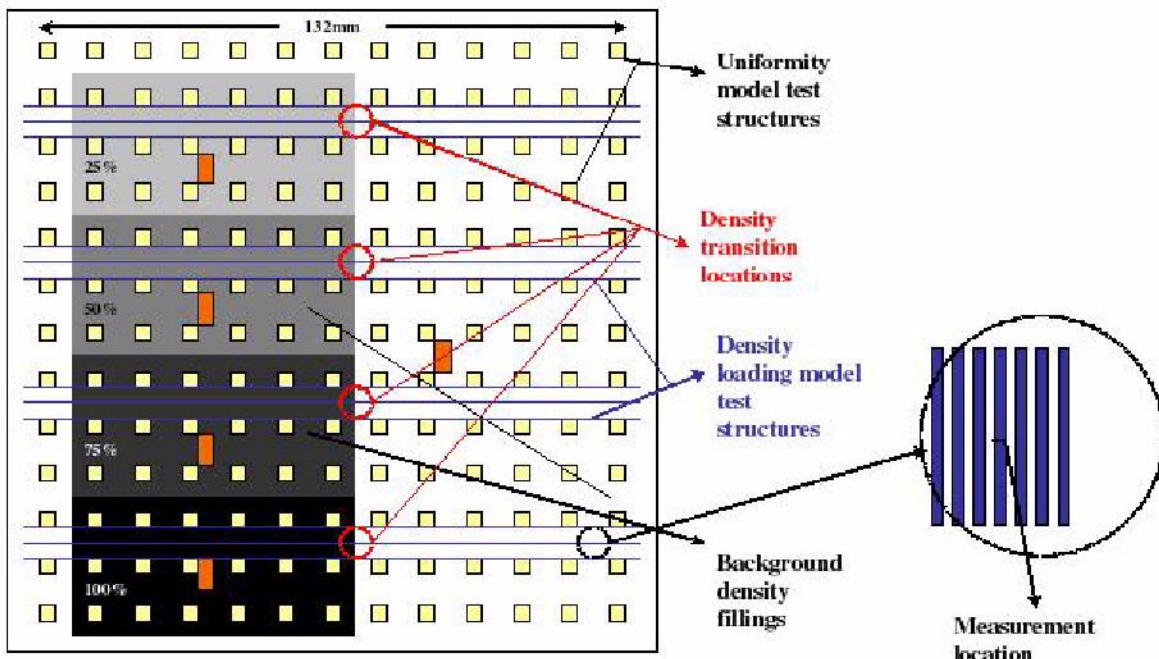
- ***mtpfilename***

Specifies the input file. For information on the input file, refer to “[mtpCompile Input File Format](#)” on page 107 for further information.

Description

Creates a whole mask test layout (short-range and long-range), which consists of proximity, density loading, and uniformity models (an example pattern is shown in [Figure 5-12](#)), as well as the gauge file from a specified input file.

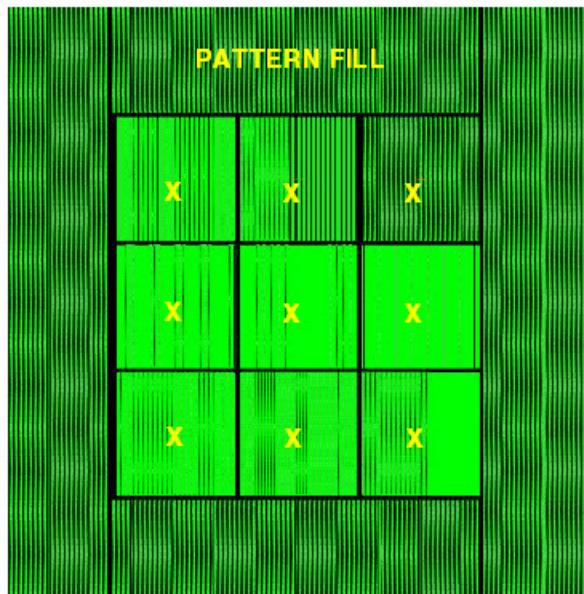
Figure 5-12. Example Mask Model



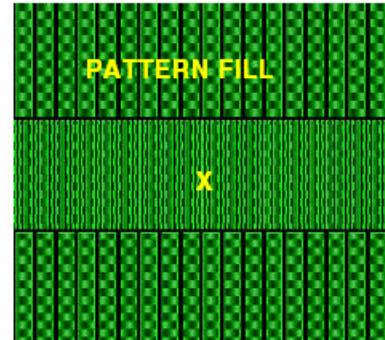
Density loading model consists of background density filling structures (simple line and space structures with varying duty), and density loading structures (simple line and spaces printed across mask).

Figure 5-13. Example mtpCompile Patterns

Uniformity Measurement Sites



Density Measurement Sites



Examples

Create a whole mask test layout and spreadsheet from a specified input file.

```
% mtpCompile mask.in
```

mtpCompile Input File Format

Input file for ptCompile

The mtpCompile input file is used to define the parameters for the mtpCompile run.

Format

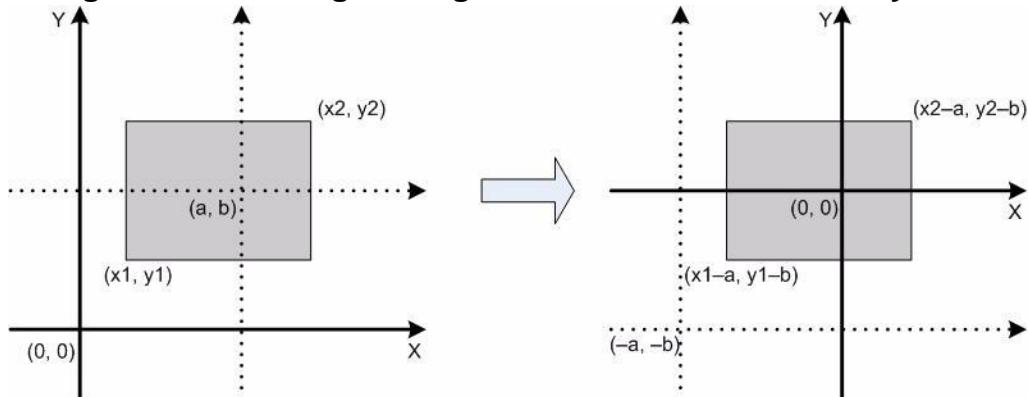
This input file consists of a series text lines of keywords. Keyword is the first word in the text file. A line starting with a # character is a comment. A keyword defines the scope for any subsequent text lines until another keyword is found. All *val* parameters are in microns.

Parameters

Global Keywords

- **dbgrid *db_grid***
Database grid, in user units (microns), in which to specify the test layout features.
- **designgrid *val***
Design grid, in microns. Currently, it is equal to dbgrid.
- **topcellname *top_cell_name***
Specifies the name of the top cell to create for this test layout.
- **inset *val***
Sets the gap between two adjacent features, in microns. The default is 10 microns.
- **masksize *val***
Sets the size of the whole mask, in microns. The default is 132000.0 microns.
- **outlayout *output_layout_file***
Specifies the output layout file name.
- **outorigin *val1 val2***
Sets the coordinates of the origin point in user's coordinate system. In the Calibre coordinate system, it is assumed the origin point is at the lower-left corner of the mask. You can set up your own coordinate system by specifying the location of their origin point in the Calibre coordinate system (see [Figure 5-14](#)).

Figure 5-14. Setting an Origin Point in the Coordinate System



Uniformity Model Keywords

The following keywords are specifically for the uniformity model:

The uniformity block is filled on the mask from center out, symmetrically. When given a uniformity block size and the distance between two adjacent blocks, the maximum size of the block array that can be placed within the boundary of the mask is automatically calculated. The pattern inside a uniformity block is generated from existing test pattern .gds and .gg files. You can specify background density filling for uniformity test structures.

Uniformity test structures are measured at every placement of the uniformity structures (for example, periodic line/space) at the center of the center line. Typically, there are more than one line/space features in the uniformity test structures.

- **uniformity_size *val***
Sets the size of a uniformity block.
- **uniformity_interval *val***
Sets the distance between any two adjacent uniformity blocks.
- **uniformity_bg_space *val***
Sets space of the background density in a uniformity block.
- **uniformity_bg_width *val***
Sets width of the background density in a uniformity block.
- **uniformity_gds *uni_gds_file***
Specifies the input uniformity GDS file name.
- **uniformity_gg *uni_gg_file***
Specifies the input uniformity gauge file name.
- **uniformity_gg_out *uni_gg_out***
Specifies the output gauge file name for the uniformity model.

Background Density Model Keywords

The following keywords are for the background density model only. You can specify line width, density and number of locations for different background densities.

- **bg_density [0, 1]**
Specifies an array of densities of the background density blocks.
- **bg_location_x val**
Sets the x coordinate of the left-bottom background density block.
- **bg_location_y val**
Sets the y coordinate of the left-bottom background density block.
- **bg_height val**
Sets the height of each background density block.
- **bg_smaller_width val**
Sets the width of each background density block.
- **bg_linewidth val**
Sets the line width of the line and space structures in a background density block.

Density Loading Model Keywords

The following keywords are for density loading model. Density loading model test structures are simply lines and spaces printed across the mask. You can specify the number of different line/space combinations and their values. By default, density loading pattern starts at x=0 and ends at x=mask_size.

On each density loading model structure (line/space) printed across the mask, starting from x=0, at every dl_interval place, the closest line width is measured at the center of the line.

- **dl_location_y {val}+**
Specifies an array of Y-coordinates of the left-bottom points of the density loading blocks. These blocks are arranged one by one from bottom to top in the Y-direction.
- **dl_height val**
Sets height of each the density loading block.
- **dl_gap val**
Sets the distance between two adjacent density loading blocks in Y direction.
- **dl_width {val}+**
Specifies the width of the line and space features in density loading blocks.
- **dl_space {val}+**
Specifies the space of the line and space features in density loading blocks.

- **dl_measure_interval *val***
Specifies the distance between two adjacent measurement locations.
- **dl_gg_out *dl_gg_out***
Specifies the output gauge file name for density loading model.

Proximity Model Keywords

The following keywords are for the proximity model.

- **proximity_gds *pro_gds_file***
Specifies the input GDS file name of a proximity model.
- **proximity_gg *pro_gg_file***
Specifies the input gauge file name of a proximity model.
- **proximity_x {*val*}+**
Specifies an array of x coordinates of the left-bottom points of the proximity blocks.
- **proximity_y {*val*}+**
Specifies an array of y coordinates of the left-bottom points of the proximity blocks.
- **proximity_gg_out *pro_gg_out***
Specifies the output gauge file name for a proximity model.

VEB Model Keywords

The following keywords are for a VEB model:

- **veb_gds *veb_gds_file***
Specifies the input GDS file name of a VEB model.
- **veb_gg *veb_gg_file***
Specifies the input gauge file name of a VEB model.
- **veb_x {*val*}+**
Specifies an array of x coordinates of the left-bottom points of the VEB blocks.
- **veb_y {*val*}+**
Specifies an array of y coordinates of the left-bottom points of the VEB blocks.
- **veb_gg_out *veb_gg_out***
Specifies the output gauge file name for a VEB model.

User-Integrated Model Keywords

You can integrate your own test layout to the whole-mask layout by specifying their layout GDS input, the corresponding spreadsheet file, and the locations on mask. You can specify as many integrated models needed by specifying multiple user-integrated models.

The following keywords are for a User-Integrated model:

- **integrated_gds *integrated_gds_file***
Specifies the input GDS file name of an integrated model.
- **integrated_gg *integrated_gg_file***
Specifies the input gauge file name of an integrated model.
- **integrated_x {*val*}+**
Specifies an array of x coordinates of the left-bottom points of the integrated blocks.
- **integrated_y {*val*}+**
Specifies an array of y coordinates of the left-bottom points of the integrated blocks.
- **integrated_gg_out *integrated_gg_out***
Specifies the output gauge file name for an integrated model.

Examples

Create a whole mask test layout from a specified input file.

```
% mtpCompile mask.in

bg_density 0.25 0.5 0.75 1
bg_location_x 10000
bg_location_y 6000
bg_height 30000
bg_width 50000
bg_smaller_width 2
##### Density loading model parameters. #####
##### y: the y location of set of density loding model #####
##### height: the heighth of one density loading
##### gap: the distance inside the set of density loading models #####
##### space, width: the parameters for filling line-and-space #####
dl_location_y 19000 49000 79000 109000
dl_height 50
dl_gap 1000
dl_width 0.2 0.25 0.3 0.5 1.0
dl_space 0.2 0.25 0.3 0.5 1.0
dl_measure_interval 1000
dl_measure_addition 50 100 200 400 800
dl_gg_out example_density_out.gg
##### Proximity model parameters. #####
##### gds,ss: input pattern #####
##### x,y: the left-bottom position of proximity models #####
proximity_gds proximity.gds
proximity_gg proximity.gg
proximity_x 28000 28000 28000 28000 68000 68000 68000 68000 108000 108000
proximity_y 8000 38000 68000 98000 8000 38000 68000 98000 8000 38000
proximity_hh_out example_proximity_out.gg
#####
output
outlayout example.gds
outorigin 66000 66000
##### VEB model parameters. There has to be one instance of this.
#####
veb_gds user_supplied_patterns/standard_80_120.gds
veb_gg user_supplied_patterns/standard_80_120.gg
veb_x 27000 27000 27000 67000 67000 67000 107000 107000 107000 107000
veb_y 10000 40000 70000 100000 10000 40000 70000 100000 10000 40000 70000
veb_gg_out example_standard_out.gg
#####
additional user defined proximity calibration patterns. There can be
##### as many instances of this as desired.
# integrated_gds user_supplied_patterns/etch_80_120.gds
# integrated_gg user_supplied_patterns/etch_80_120.gg
# integrated_x 30000 30000 30000 30000 70000 70000 70000 70000 110000
# 110000 110000 110000
# integrated_y 10000 40000 70000 100000 10000 40000 70000 100000 10000
# 40000 70000 100000
# integrated_gg_out example-etch_out.gg
```

MPC Calibration Batch Commands

The following table lists commands used for MPC Calibration batch mode.

Table 5-2. MPC Calibration Batch Commands

Tcl Command	Description
mpcgConvert/mpcssConvert	Removes the proximity bias from long-range spreadsheets and gauge files respectively.
mpclr	Computes the coefficients of pattern density and uniformity long-range models, and outputs a partial SVRF deck for DENSITY CONVOLVE and MAPSIZE.
mpclropf	Determines the optimized Gaussian sigmas in specified ranges that give the best fit to the density data. Then the function computes the coefficients of pattern density and uniformity long-range models, and output an SVRF rule deck for DENSITY CONVOLVE and MDP MAPSIZE.
mpceval	Evaluates the long-range model at each location in the input spreadsheet, subtracts the implied bias from the spreadsheet entry, and outputs the new spreadsheet with prediction CDs and fit errors.
mpcflow_v2	Calibrates an E-beam model, working with the VEB model to search for the optimized Gaussian sigma value in the E-beam model.

mpcggConvert/mpcssConvert

MPC Calibration Batch Commands

Removes the proximity bias from long-range spreadsheets and gauge files respectively.

Usage

mpcggConvert -layout *input_gg* *output_gg* *base_CD*

mpcssConvert -layout *input_ss* *output_ss* *base_CD*

Description

These functions remove the proximity bias from long-range gauge files and spreadsheets respectively.

Arguments

- ***input_gg***
A required argument that specifies the input gauge file.
- ***output_gg***
A required argument that specifies the output gauge file.
- ***input_ss***
A required argument that specifies the input spreadsheet.
- ***output_ss***
A required argument that specifies the output spreadsheet.
- ***base_CD***
A required argument that specifies the base CD that other CDs will be adjusted by the proximity bias of the base CD. Note that base CD has to be one of the CD values in the specified input spreadsheet or input gauge file.

Examples

```
mpcggConvert input.gg output.gg 500
mpcssConvert input.ss output.ss 500
```

mpclr

MPC Calibration Batch Commands

Computes the coefficients of pattern density and uniformity long-range models, and outputs a partial SVRF deck for DENSITY CONVOLVE and MAPSIZE.

Usage

```
mpclr [-sectionSize sect_size] -layout layout_handler
-input_layer input_layer_list
[-input_spreadsheet input_ss | -input_gauge input_gg]
-model input_model
[-modelform {COMBO | DENSITY | UNIFORMITY}]
[-window val]
[-CD_bin_width cd_val]
-svrf_out svrf_name
[-ss_out spreadsheet_name | -gg_out gauge_file_name]
-model_out output_model
```

Description

The MPC Calibration function, mpclr, computes the coefficients of pattern density and uniformity long-range models, and output an SVRF rule deck for DENSITY CONVOLVE and MDP MAPSIZE. The mpclr command calibrates the long range model using the following process:

1. The mpclr command first computes the density of the geometry on specified layer using the specified window value.
2. The resulting density grid is then convolved according to specified Gaussian kernels.
3. The mpclr command then computes the coefficients for these Gaussian kernels. For uniformity long-range model, it simply computes the coefficient for up to 4th order spatial polynomial terms. In specific, the model has the form:

$$\begin{aligned} \text{deltaCD} = & A0 + \text{Sum over } i (C_i D(x,y) * \exp(-(x^2 - y^2)/(2*Si^2))) + A1 x + A2 y + A3 xy \\ & + A4 x^2 + A5 y^2 + A6 x^2y + A7 xy^2 + A8 x^3 + A9 y^3 + A10 x^3y + A11 x^2y^2 + A12 \\ & xy^3 + A13 y^4 + A14 y^4 \end{aligned}$$

Arguments

- **-sectionSize sect_size**

When this option is specified, the layout is divided into sections of specified size, and is loaded in and processed section by section. The section size should be positive and in ums.

- **-layout layout_handler**

A required argument that specifies the input test pattern layout.

- **-input_layer *input_layer_list***

Required polygon input layers used to generate long-range and uniformity models. If input layer is more than 1, these layers will be drc_or internally, then calibrated.

- **-input_spreadsheet *input_ss* | -input_gauge *input_gg***

Specifies the input file name of a spreadsheet or gauge with long-range pattern density calibration data, uniformity model calibration data, or combined data, depending on what was specified with the -modelform option.

- **-model *input_model***

Specifies an input model. The input model file is similar to an LR model file, but without the BTERM items, which will be calibrated and output later in an LR model file.

- **-modelform {COMBO | DENSITY | UNIFORMITY}**

Specifies the model type: DENSITY, UNIFORMITY, or COMBO.

- If the model type is DENSITY, then a long-range model that is based on CD bias and Gaussian convolution with the pattern density is computed using measured data in a spreadsheet specified by the -input_spreadsheet option.
- If the model type is UNIFORMITY, then a long-range model that is based on CD uniformity is computed using measured data in a spreadsheet specified by the -input_spreadsheet option.
- If the model type is COMBO, then a combined long-range model based on bias, pattern density, and CD uniformity is computed using measured data in a spreadsheet specified by the -input_spreadsheet option. This is the default option.

- **-window *val***

Specifies the window size used to compute density. The default value is 50 um. If window is larger than the boundary in x or y direction, it is truncated to the boundary size. Refer to the TRUNCATE option in the Calibre DENSITY CONVOLVE command for more detail. The unit is in ums.

- **-CD_bin_width *cd_val***

Specifies a model range to create density convolve regions. The default value is 1.0 nm. Regardless of the model, the bins are centered in such a way that one of the bins would be centered on 0.

- **-svrf_out *svrf_name***

Specifies the output SVRF file name. Default value is the input spreadsheet file name plus extension .svrf. The output file will be a SVRF command scripts as input to next correction stage. The default name is modelform.svrf.

- **-ss_outspreadsheet_name | -gg_out gauge_file_name**

Specifies the output spreadsheet or gauge file name. Default value for -ss_out is the input spreadsheet file name, plus “_out”, plus the extension .ss. The default value for -gg_out is the input spreadsheet file name, plus “_out”, plus the extension .gg. The file has the same

columns as the input spreadsheet plus two new columns, one contains mask CDs computed by long-range model, the other contains the errors that between measured CDs and predicted CDs (errors = measured CD - predicted CD).

- **-model_out svrf_name**

Specifies the output LR model file name. The default value is the input spreadsheet file name plus extension .mod.

Examples

In the Calibre WORKbench running environment, the following Tcl commands are issued:

```
layout create 65_full_reticle.gds
```

This will return layout handler “layout0.”

```
mpclr -sectionSize 50000 -layout layout0 -input_layer "0 5 8"
      -input_gauge in.gg -modelform COMBO -model model.in -window 50
      -CD_bin_width 1.0 -svrf_out out.svrf -gg_out out.gg -model_out out.mod
```

The mpclr function generates SVRF rule deck in following format:

```
litho file X[
SCALE C1 GAUSS S1
...
SCALE Cm GAUSS Sm
spatialPolynomial [<a0>] [<a1>x] [<a2>y] [<a3>xy] [<a4>x2] [<a5>y2]
]
n0 = density convolve [in_layer] <constraint> window window_size file X
...
n(k) = density convolve [in_layer] <constraint> window window_size file X
my_mapsize {MDP MAPSIZE [in_layer] n0 ... n(k) FILE [
map_size d0 ... d(k)
]
}
```

In this SVRF rule deck:

- S1 ... Sm: Specifies input gauss values.
- C1 ... Cm: Specifies coefficients for long-range model.
- a0 ... a5: Specifies coefficients for uniformity model.
- n0 ... n(k): Specifies density convolve output layers according to measurement bins of CD errors.
- constraint: Specifies CD errors.
- d0 ... d(k): Specifies the map size corresponding to each density convolve output layer.

The following is an example of an output SVRF rule deck:

```
litho file X [
SCALE 8.400366 GAUSS 50.000000
SCALE -6.343858 GAUSS 100.000000
SCALE 1.725309 GAUSS 1000.000000
spatialPolynomial 6.716368e+00 -5.230650e-05x -3.597974e-05y -2.193321e-
10xy 1.597709e-10x2 6.286192e-11y2 -5.549886e-16x2y 2.903201e-15y2x
4.177375e-15x3 1.843460e-15y3]
n0 = density convolve [in_layer] > 0.500000 <= 1.500000 truncate window
50.000000 file X
n1 = density convolve [in_layer] > 1.500000 <= 2.500000 truncate window
50.000000 file X
n2 = density convolve [in_layer] > 2.500000 <= 3.500000 truncate window
50.000000 file X
n3 = density convolve [in_layer] > 3.500000 <= 4.500000 truncate window
50.000000 file X
n4 = density convolve [in_layer] > 4.500000 <= 5.500000 truncate window
50.000000 file X
n5 = density convolve [in_layer] > 5.500000 <= 6.500000 truncate window
50.000000 file X
n6 = density convolve [in_layer] > 6.500000 <= 7.500000 truncate window
50.000000 file X
n7 = density convolve [in_layer] > 7.500000 <= 8.500000 truncate window
50.000000 file X
n8 = density convolve [in_layer] > 8.500000 <= 9.500000 truncate window
50.000000 file X
n9 = density convolve [in_layer] > 9.500000 <= 10.500000 truncate window
50.000000 file X
n10 = density convolve [in_layer] > 10.500000 <= 11.500000 truncate window
50.000000 file X
n11 = density convolve [in_layer] > 11.500000 <= 12.500000 truncate window
50.000000 file X
n12 = density convolve [in_layer] > 12.500000 <= 13.500000 truncate window
50.000000 file X
n13 = density convolve [in_layer] > 13.500000 <= 14.500000 truncate window
50.000000 file X
my_mapsize {MDP MAPSIZE [in_layer] n0 n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11
n12 n13 FILE [
map_size -0.000500 -0.001000 -0.001500 -0.002000 -0.002500 -0.003000 -
0.003500 -0.004000 -0.004500 -0.005000 -0.005500 -0.006000 -0.006500 -
0.007000
]
}
```

mpclropt

MPC Calibration Batch Commands

Determines the optimized Gaussian sigmas in specified ranges that give the best fit to the density data.

Usage

```
mpclropt -sectionSize sect_size -layout layout_handler
    -input_layer input_layer_list
        [-input_spreadsheet input_ss | -input_gauge input_gg]
        [-runtime time]
        [-explore opt_iters]
    -model input_model
        [-modelform {COMBO | DENSITY | UNIFORMITY}]
        [-gauss “esN minN maxN...”]
        [-opt optimization_parameter_list]
        [-window val]
        [-CD_bin_width cd_val]
        [-svrf_out svrf_name]
        [-ss_out spreadsheet_name | -gg_out gauge_file_name]
        [-model_out output_model]
```

Arguments

- **-sectionSize *sect_size***

When this option is specified, the layout is divided into sections of specified size, and is loaded in and processed section by section. The section size should be positive and in ums.

- **-layout *layout_handler***

A required argument that specifies the input test pattern layout.

- **-input_layer *input_layer_list***

Required polygon input layers used to generate long-range and uniformity models. If input layer is more than 1, these layers will be drc_or internally, then calibrated.

- **-input_spreadsheet *input_ss* | -input_gauge *input_gg***

An argument that specifies the input file name of a spreadsheet or gauge file with long-range pattern density calibration data, uniformity model calibration data, or combined data, depending on what was specified with the -modelform option.

- **-runtime *time***

Runtime specifies the upper bound of running time for FRONT engine to do optimization search. The unit is in hours.

- **-explore *opt_iters***

Explore specifies the upper bound of number of FRONT optimization iterations. The default number of optimization iterations is 20.

- **-model *input_model***

Specifies an input model. The input model file is similar to an LR model file, but without the BTERM items, which will be calibrated, and output later in an LR model file.

- **-modelform {COMBO | DENSITY | UNIFORMITY}**

Specifies the model type: COMBO, DENSITY, or UNIFORMITY.

- If the model type is COMBO, then a combined long-range model based on bias, pattern density, and CD uniformity is computed using measured data in a spreadsheet specified by the -input_spreadsheet option. This is the default option.
- If the model type is DENSITY, then a long-range model that is based on CD bias and Gaussian convolution with the pattern density is computed using measured data in a spreadsheet specified by the -input_spreadsheet option.
- If the model type is UNIFORMITY, then a long-range model that is based on CD uniformity is computed using measured data in a spreadsheet specified by the -input_spreadsheet option.

- **-gauss “esN minN maxN...”**

When -modelform is set to COMBO or DENSITY, this option is required to give the Gaussian sigmas range (for example, -gauss “es1 50 100 es2 100 1000”). The unit for gauss sigma values is in microns.

- **-opt *optimization_parameter_list***

The optimization parameter list is in the following format: the first number specifies the sections; the second number specifies the points; the third number specifies “nt” value. By default, the -opt is 5 3 2.

- **-window *val***

Specifies the window size used to compute density. The default value is 50 um. If window is larger than the boundary in x or y direction, it is truncated to the boundary size. Refer to the TRUNCATE option in the Calibre DENSITY CONVOLVE command for more detail. The unit is in ums.

- **-CD_bin_width *cd_val***

Specifies a model range to create density convolve regions. The default value is 1.0 nm. Regardless of the model, the bins are centered in such a way that one of the bins would be centered on 0.

- **-svrf_out *svrf_name***

Specifies the output SVRF file name. The default value is the input spreadsheet file name with the .svrf extension. The output file will be a SVRF command scripts as input to next correction stage. The default name is in_ss.svrf.

- `-ss_out spreadsheet_name | -gg_out gauge_file_name`

Specifies the output spreadsheet or gauge file name. The default value for `-ss_out` is the input spreadsheet file name, plus “_out”, plus the extension .ss. The default value for `-gg_out` is the input spreadsheet file name, plus “_out”, plus the extension .gg. The file has the same columns as the input spreadsheet plus two new columns, one contains mask CDs computed by long-range model, the other contains the errors that between measured CDs and predicted CDs (errors = measured CD - predicted CD).

- `-model_out svrf_name`

Specifies the output LR model file name. The default value is the input spreadsheet file name plus the extension .mod.

Description

The optimized MPC Calibration function mpclropt improves the original MPC calibration function `mpclr` by automatically optimizing the Gaussian sigma parameters for long-range correction through a front optimization engine. Instead of specifying exact Gaussian sigmas, you only need to specify a number of Gaussian sigmas and their ranges correspondingly.

The function is able to figure out the optimized Gaussian sigmas in specified ranges that give the best fit to the density data. The function then computes the coefficients of pattern density and uniform long-range models, and outputs an SVRF rule deck for DENSITY CONVOLVE and MAPSIZE. The output SVRF rule deck format and the output spreadsheet format are the same as those of the `mpclr` command.

Examples

In the Calibre WORKbench running environment:

```
% layout create full_reticule.gds
```

This will return the layout handler “layout0.”

```
% mpclropt -sectionSize 50000 -layout layout0 -input_layer "0 5 8"  
-input_spreadsheet in.ss -modelform COMBO -model model.in  
-gauss "es1 50 100 es2 100 1000" -opt "5 3 2" -window 50 -CD_bin_width 1.0  
-svrf_out out.svrf -ss_out out.ss -model_out out.mod
```

mpceval

MPC Calibration Batch Commands

Evaluates the long-range model at each location in the input spreadsheet, subtracts the implied bias from the spreadsheet entry, and outputs the new spreadsheet with prediction CDs and fit errors.

Usage

```
mpceval -sectionSize sect_size -layout layout_handler -input_layer input_layer_list
  [-input_spreadsheet input_ss | -input_gauge input_gg]
  -model input_model
  [-window val]
  [-ss_out spreadsheet_name | -gg_out gauge_file_name]
```

Arguments

- **-sectionSize *sect_size***

When this option is specified, the layout is divided into sections of specified size, and is loaded in and processed section by section. The section size should be positive and in ums.

- **-layout *layout_handler***

A required argument that specifies the input test pattern layout.

- **-input_layer *input_layer_list***

Required polygon input layers used to generate long-range and uniformity models. If input layer is more than 1, these layers will be drc_or internally, then calibrated.

- **-input_spreadsheet *input_ss* | -input_gauge file *input_gg***

A required argument that specifies the input file name of a spreadsheet or gauge with long-range pattern density calibration data, uniformity model calibration data, or combined data, depending on what was specified with the -modelform option.

- **-model *input_model***

Specifies an input model. The input model file is similar to LR model file, but without the BTERM items, which will be calibrated and output later in LR model file.

- **-window *val***

Specifies the window size used to compute density. The default value is 50 um. If window is larger than the boundary in x or y direction, it is truncated to the boundary size. Refer to the TRUNCATE option in the Calibre DENSITY CONVOLVE command for more detail. The unit is in ums.

- **-ss_out *spreadsheet_name* | -gg_out *gauge_file_name***

Specifies the output spreadsheet or gauge file name. The default value for -ss_out is the input spreadsheet file name, plus “_out”, plus the extension .ss. The default value for - gg_out is the input spreadsheet file name, plus “_out”, plus the extension .gg. The file has the same columns as the input spreadsheet plus two new columns, one contains mask CDs

computed by long-range model, the other contains the errors that between measured CDs and predicted CDs (errors = measured CD - predicted CD).

Description

The MPC Evaluation function mpceval takes as its input a long-range model file (see “[Long-Range Model File Format](#)” on page 124 for the format of a long-range model file) and a spreadsheet of measurement locations. It then evaluates the long-range model at each location in the input spreadsheet, subtracts the implied bias from the spreadsheet entry, and outputs the new spreadsheet with prediction CDs and the fit errors.

Examples

In the Calibre WORKbench running environment:

```
layout create 65_full_reticle.gds
```

This will return layout handler “layout0.”

```
mpceval -sectionSize 50000 -layout layout0 -input_layer "0 5 8"  
-input_spreadsheet in.gg -model model.in -window 50 -ss_out out.gg
```

Long-Range Model File Format

Input file to define the Long-Range Model

The Long-Range Model File format defines long-range model parameters that are input into the MPC Evaluation function.

Format

The Long-Range (LR) file format is defined as follows:

```
#Comments are here
version version
modelType LR
SKERNEL D1 es = s
SKERNEL D2 es = s
...
btermCount n
uniformityOrder n
BTERM c1 D1
BTERM c2 D2
BTERM c3 X
BTERM c4 Y
BTERM c5 XY
...
```

Parameters

- **version** *version_number*

Specifies the version number. Currently, only a value of 1 is used.

- **modelType**

Specifies the model type, which must be LR.

- **SKERNEL**

Each SKERNEL entry represents a gauss kernel in the Long-Range model and contains the following field:

- **es**: The diffusion length of the kernel in nanometers.

- **btermCount** *n*

Specifies the total number of BTERMs.

- **uniformityOrder** *n*

Specifies the order of the uniformity items.

- **BTERM** *ci indicator*

The LR model term. The total number of Long-Range terms in the list must be reported using the **btermCount** *n* field.

- **c0**: The constant portion of the Long-Range model, in nanometers.

- **c1**: The model coefficients, in nanometers.

- o ***indicator***: Specifies the corresponding density kernels or uniformity items for the *ci* field (D1, D2, X, Y, and XY for instance).

Examples

```
#Comments are here
version 1
modelType LR
SKERNEL D1 es = 50.000
SKERNEL D2 es = 100.000btermCount 5
btermCount 8
uniformityOrder 2
BTERM 3.163141
BTERM -0.111031 D1
BTERM 2.758279 D2
BTERM 1.953366e-05 X
BTERM 1.414608e-05 Y
BTERM 1.333322e-05 XY
BTERM 1.222333e-05 X^2
BTERM 1.222222e-05 Y^2
```

mpcflow_v2

MPC Calibration Batch Commands

Calibrates an E-beam model, working with the VEB model to search for the optimized Gaussian sigma value in the E-beam model.

Usage

```
mpcflow_v2 {-m gds | -l layout}
{-gf gaugefile | -sgf supergaugefile}
[-centergauges true | false]
[-mlfile file | -mlstring string]

[output
 [log log]
 [gfout file]
 [etch file]
 [ebeam file]
]

[stage0
 [try]
 es value
]

[stage3
 [try]
 [fit {relative | absolute}]
 [inputlayer [integer] [cli]]
 [search {full | gradient | newton | front}
  [explore integer] [epsilon value]
  [un {begin end
 num_levels |= tcl_exp begin end}]
  [sm {begin end
 num_levels |= tcl_exp begin end}]
 ]
]
```

Arguments

Input Arguments

- **-m gds**

Specifies the input test pattern (either -m or -l is required). This argument cannot be used if -l is set.

- ***-l layout***

Specifies that input layout handle with the test pattern (*-l* is required). This argument cannot be used if *-m* is set. To avoid merging layers with data type, the layout must always be created using the *-dt_expand* option. For example

```
layout create "tp.gds" -dt_expand
```

- ***-gf gaugefile***

Specifies the input gauge file (*-gf* or *-sgf* is required if the stage3 inputlayer argument is not set). This argument cannot be used if *-sgf* is specified.

- ***-sgf supergaugefile***

Specifies the input supergauge file (*-gf* or *-sgf* is required if the stage3 inputlayer argument is not set). This argument cannot be used if *-gf* is specified.

- ***-centergauges true | false***

An optional keyword that specifies gauge centering. This option should be set to true if gauges in the gauge file need to centered along the polygon, and false otherwise. The default setting is false.

- ***-mlfile file***

Optionally specifies the input model-layer file. This option cannot be used if *-mlstring* is set. See the section “[Model-Layer File Format](#)” on page 131 for details on model-layer files.

- ***-mlstring {string}***

Optionally specifies the input model-layer string. This option cannot be used if *-mlfile* is set.

Output Arguments

- ***log log***

Outputs a log file. If *gsdout* is specified, the log info will be written to the *stdout* rather than to a file. The default file name is *modelflow.log*.

- ***gfout file***

Outputs gauge data with simulated CD/spaces measurements to disk.

- ***etch file***

Outputs the final etch model to disk. This option is meaningful only if stage3 etch bias optimization is specified.

- ***ebeam file***

Outputs the final E-beam model to disk. This option is meaningful only if stage0 E-beam optimization is specified.

Stage0 Optimization Arguments

An optional flag that uses the input model without fitting it to empirical data.

- `es value`

Specifies the diffusion length in microns of E-beam Gaussian sigma, and is a value between 0 and 10 um. The default is to use the nominal model setting. Note that sigma values above 2 um are used for calibrating the backscattering range of an E-beam writer and cannot be used in an E-beam model used for MPC correction. Instead, it is assumed that the optimized sigma value for backscattering is used for on-tool Proximity Effect Correction (PEC). Proximity Effect Correction is a process flow to be introduced in a future release.

Stage3 Optimization Arguments

- `try`

An optional flag that uses the input VEB model without fitting it to empirical data.

- `fit {relative | absolute}`

An optional flag that specifies the objective function used to optimize the etch model.

relative — Specifies that the etch model parameters are to be determined by minimizing the difference between simulated and measured etch bias. The latter is determined as the difference between etch and resist CD: CDetch - CDresist. This option cannot be used if the `inputlayer` argument is also specified.

absolute — Specifies that the etch model is to be fit by minimizing the difference between simulated and measured etch CD data; in other words, the model is fit directly to CD data rather than to bias data. This is the default setting.

- `inputlayer [layernumber] [cli]`

An optional flag that includes supplied resist contours in its VEB calibration calculations. They can be found by:

- the specified `layernumber`
- using the specified contour layer information (CLI) file specified in the `-clifile` or `-clistring` input arguments
- both the specified layer number and CLI file if `layernumber cli` is specified

Neither a resist nor an optical model are required in this case, but dummy files must be supplied in order to prevent parsing errors.

Resist contours must be polygons. The resist contour layer you specify has its information added to the gauge information for calculation purposes.

Resist contours should be available for all features crossed by gauges if you use this option.

Note

 For each of the following arguments, the allowed index entries are $j = 1, 2, 3$, or 4 .

- `uj`

Specifies a shift (in nanometers) of j -th etch density (which is the Dj term in the etch model). The default is to use the nominal model setting.

- es_j
Specifies the diffusion length (in nanometers) of j -th etch density, 0 es_j . The default is to use the nominal setting.
- c_j
Specifies the secondary density multiplier (in nanometers) of j -th etch transport kernel. The default is to use the nominal model setting.
- t_j
Specifies secondary diffusion length (in nanometers) of j -th etch transport kernel, 0 t_j . The default is to use the nominal model setting.

There are several limitations on the etch model and its parameters:

- Parameters of no more than 4 density kernels can be optimized simultaneously
- The u for the kernel of visible type must be non-negative
- The es for kernels of Gaussian type must be greater than or equal to 35 nm

Normal values of all etch optimization parameters are set to 100 nm.

Description

E-beam calibration runs together with VEB optimization to search for the optimized Gaussian sigma value in the E-beam model. This batch command interface is used to perform E-beam model calibration.

Backscattering Calibration

The mpcflow_v2 command can be used to calibrate back-scattering models with medium range kernels. Those medium range kernels can have a diff_length value between 2.0 um and 20 um. Up to four medium range kernels can be calibrated in one model and they can be combined with short range kernels below 2.0 um as shown in the following examples:

This ebeam model is calibrated with medium range kernels:

```
ebeamfile inline
inline_ebeam {
    model_type simulation
    sample_spacing 0.008
    transmission 1.00
    diff_length 0.0236 0.131 0.56 10.0
    eta 0.51 0.08 0.06 0.35
    kernel_type gaussian gaussian exponential gaussian
    calibrate diff_length 3 8.0 12.0 10
    calibrate eta 3 0.3 0.5 10
}
```

Medium range kernels can be calibrated with other kernels as shown in the following example:

```
ebeamfile inline
inline_ebeam {
    model_type simulation
    sample_spacing 0.008
    transmission 1.00
    diff_length 0.0236 0.131 0.56 10.0
    eta 0.51 0.08 0.06 0.35
    kernel_type gaussian gaussian exponential gaussian
    calibrate diff_length 2 0.46 0.66 10 diff_length 3 8.0 12.0 10
    calibrate eta 2 0.05 0.07 10 eta 3 0.3 0.5 10
}
```

Model-Layer File Format

Model and layer input for the mpcflow_v2 command

If the -ml option is specified in mpcflow_v2, a model-layer file is to be used as its input.

In general, the model-layer file/string consists of two parts:

- Model input specifies model paths and or inline models (optional).
- Layer input specifies layer and background parameters (required).

Each line of the model-layer file/string, provided that it is not a comment, must begin with a keyword. Only one keyword per line is allowed except keyword inline which does not count.

Format

```
layer layerNum[.datatype] {[clear | dark] [ebeam_dose d] }  
[modelpath path]  
[vetchbiasfile {filename | inline}]  
[inline_etch {' etch_model '}]  
[ebeamfile {filename | inline}]  
[inline_ebeam {' ebeam_model '}]
```

Parameters

- **layer *layerNum*[.*datatype*] {[clear | dark] [ebeam_dose *d*] }**

A required keyword that identifies the EGDS layer index to be used in simulation. The EGDS layer index can be a positive integer represented by layer number (*layerNum*) or it can also include a decimal portion to indicate a particular datatype (.datatype). The default datatype value is 0.

By default, the polygons are exposed by the mask writer, corresponding to the value “clear” to the optional second argument. Exposure of an area outside the polygons is indicated by setting this argument to “dark”. The exposure dosage for the polygons can be optionally specified using the ebeam_dose keyword. The dose value *d* is a floating point type.

- **modelpath *path***

An optional keyword that defines the path under which models reside. This is required if at least one of the models is not inlined.

- **vetchbiasfile {*filename* | inline}**

An optional keyword that specifies the name of the etch model file. If inline is specified, the resist model is specified inline.

- **inline_etch {' etch_model '}'**

Use this keyword with braces to enter the etch model when vetchbiasfile inline is set.

- **ebeamfile {*filename* | inline}**

An optional keyword that specifies the name of the ebeam model file. When used along with the keyword inline, instructs the tool that the resist model is specified inline.

- `inline_ebeam '{' ebeam_model '}'`

Use this keyword with curly brackets to enter the ebeam model (see “[ebeam_model_load](#)” on page 141 for the ebeam model parameters) when “ebeamfile inline” is set.

Examples

```
layer 0 clear
modelpath models
vetchbiasfile etch.mod
ebeamfile ebeam.mod
```

MPC Correction Commands

The following table lists all the commands used for MPC Correction.

Table 5-3. Mask Process Correction Commands

Command	Description
LITHO MPC Setup File Commands	
LITHO MPC	This command is used in conjunction with the setlayer mpc command to derive output target layers for correction.
LITHO CLMPC	This command is used in conjunction with the setlayer clmpc command to derive output target layers for correction in curvilinear or skew-edge layouts.
denseopc_options (for MPC)	Creates a uniquely-named sub-command block inside a LITHO MPC statement to set nmMPC options.
direct_input	Enables Direct Date Entry (DDE).
ebeam_model_load	Defines and loads the E-beam simulation model.
etch_model_load	Loads an etch model.
layer	Defines layer characteristics. Note that there is a denseopc_options -specific command. See “ layer (denseopc_options Command) ” on page 240.
modelpath	A colon (“.”) separated list of directories to search for E-beam and etch models.
processing_mode	Sets the processing mode for Calibre nmMPC.
setlayer ebeam_simulate	The ebeam_simulate command performs pseudo-lithographic simulations using geometry on the input layer.
setlayer curve	Generates a new layer with excess jogs and other small 2D effects removed from the output (a “smoothing effect”).
setlayer curve_target	Creates a target that mimics the final image.
setlayer mpc	Creates and derives layers for correction, depending on the options used. You create output layers using the design layers you defined with the layer command, and you subsequently use one or more of them as output in your LITHO MPC calls in your SVRF file.
setlayer clmpc	Creates and derives layers for correction on curvilinear and skew-edge layouts, depending on the options used. You create output layers using the design layers you defined with the layer command, and you subsequently use one or more of them as output in your LITHO CLMPC calls in your SVRF file.

Table 5-3. Mask Process Correction Commands (cont.)

Command	Description
	For commands under the denseopc_options block, refer to “ MPC denseopc_options Commands ” on page 170
	For the Tcl-based custom scripting commands, refer to “ MPC Custom Tcl Scripting Commands ” on page 277

LITHO MPC

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Defines the LITHO batch operation.

Usage

```
output_layer_name = LITHO MPC input_layer_name FILE parameter_block_name MAP  
          output_layer
```

Arguments

- ***output_layer_name***
A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer.
- ***input_layer_name***
Specifies the input layer. The layer specified must be a valid Calibre layer containing polygon data.
- **FILE *parameter_block_name***
A required keyword that specifies the name of a parameter block defined within the SVRF rule file, using the LITHO FILE command block. A separate setup file can also be specified.
- **MAP *output_layer***
A required keyword followed by the associated argument indicating the layer containing the output to be returned.

Description

The MPC command is a tool argument (Calibre nmMPC) to the Calibre LITHO batch operation. This command is used in conjunction with the [setlayer mpc](#) command to derive output target layers for SVRF.

Examples

```
LITHO MPC poly FILE mpc.setup MAP mpc_contour
```

LITHO CLMPC

LITHO CLMPC setup file command (see [Table 5-3](#) on page 133)

Defines the LITHO batch operation for Calibre nmCLMPC targeted for curvilinear and skew-edge layouts.

Usage

```
output_layer_name = LITHO CLMPC input_layer_name FILE parameter_block_name  
MAP output_layer
```

Arguments

- ***output_layer_name***

A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer.

- ***input_layer_name***

Specifies the input layer. The layer specified must be a valid Calibre layer containing polygon data.

- **FILE *parameter_block_name***

A required keyword that specifies the name of a parameter block defined within the SVRF rule file, using the LITHO FILE command block. A separate setup file can also be specified.

- **MAP *output_layer***

A required keyword followed by the associated argument indicating the layer containing the output to be returned.

Description

The LITHO CLMPC SVRF command is a tool argument used to run mask correction on curvilinear layouts (containing polygons with skew edges) and similar to the LITHO MPC command. This command is used in conjunction with the [setlayer clmpc](#) command to derive output target layers for SVRF.

There are differences of behavior in certain commands in a LITHO CLMPC block versus a LITHO MPC block.

- When [jog_freeze](#) is set, Calibre nmCLMPC only freezes fragments for polygons that do not contain skew edges (rectilinear polygons). For polygons that contain skew edges (curvilinear), the jogs are not frozen, even if jog_freeze is set.
- If a tag set contains jogs smaller than the [jog_freeze](#) limit, Calibre nmMPC does not move those jogs with the -hintOffset keyword in the [DISPLACEMENT](#) command. However, Calibre nmCLMPC does move those jogs, meaning all fragments in a tag set are moved, including those smaller than jog_freeze.

This difference in behavior of Calibre nmMPC can be avoided by excluding all jogs smaller than jog_freeze in the tag set and applying the -hintOffset only to the reduced tag set.

- For polygons that contain skew edges, Calibre nmCLMPC automatically sets `mpc_sites_mode` to 0 (the default site placement algorithm), even if you specifically set `mpc_sites_mode` to 1 (an alternate algorithm for advanced nodes to produce improved accuracy at corner fragments).
- When the SEM Box method (see “[SEM Box Correction](#)” on page 66) is specified in the Calibre nmCLMPC recipe, Calibre nmCLMPC only applies the SEM Box method to shapes that do not contain skew edges.

Examples

```
mpcOut = LITHO CLMPC poly MAP mpcOut FILE "clmpc.recipe"
```

denseopc_options (for MPC)

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Creates a uniquely-named sub-command block inside a LITHO MPC statement to set nmMPC options.

Usage

```
denseopc_options opt_name '{  
    options_list  
}'
```

Arguments

- *optname*

A required option that specifies the unique name of a sub-command block of LITHO MPC setup file commands (see [Table 5-3](#)) defined inside a LITHO FILE parameter block. This unique name will be passed to a [setlayer mpc](#) command for processing. Currently, there is a 15 character limit on the name.

- *options_list*

The sub-command block of LITHO MPC setup file commands (see [Table 5-3](#)) defined inside the denseopc_options parameter block.

Description

Creates a uniquely-named sub-command block inside a LITHO MPC statement to set nmMPC options. This uniquely-named block is then passed to a [setlayer mpc](#) denseopc command to be processed. The denseopc_options command is required in order to call a [setlayer mpc](#) denseopc command. The options block also must use the [image](#) command. Refer to [Table 5-3](#) for the complete list of supported commands.

It is required that you use [algorithm 1](#) in conjunction with simulation type models. For example:

```
denseopc_options srcOptions {  
    algorithm 1  
}  
output_layer_name = LITHO MPC input_layer_name FILE parameter_block_name  
MAP output_layer
```

direct_input

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Enables Direct Data Entry (DDE) for Calibre nmMPC.

Usage

```
direct_input '{  
    vboasis_path vboasis_filename [vboasis_injection [size bin_size]]  
    {[input input_number] layer number [datatype] [layer number [datatype]]...}...  
}'
```

Arguments

- **vboasis_path vboasis_filename**

A required argument supplying the absolute path to an OASIS file name to be used as the design source. A runtime error is returned if the input file is invalid. The filename must not be in single or double quotes.

- **vboasis_injection [size bin_size]**

The VB:OASIS file specified using vboasis_path may contain cells that have large amounts of data and have a large extent. These large cells degrade performance in section mode processing. To overcome this problem, cells can be partitioned into bins. The size of the cells that should be binned and the size of each bin (*bin_size*) is determined automatically by default when the vboasis_injection parameter is specified.

Alternately, the *bin_size* can be explicitly specified using the size *bin_size* in microns. Any cell whose extent has a width or height more than the *bin_size* is considered for binning. The vboasis_injection creates temporary data in a directory named *<file_path>.lcb*. The binning procedure is computation-intensive and can be performed in Calibre MT or MTflex modes.

The temporary data is read or written in a directory named *<file_path>.lcb* where *file_path* is argument specified to vboasis_path. The location of this directory is same as that of *file_path*.

- **[input input_number] layer number [datatype]**

A required keyword set that specifies one or more layers. At least one layer declaration is required.

input *input_number* — An optional keyword set that specifies which input layer from the setup file to map. The index starts at 1 for the first layer listed in the setup file. The specified layer is mapped to the VB:OASIS layer **number**, allowing you to use the input layers out of order. The value you specify for *input_number* cannot exceed the number of setup layers. If this keyword set is not specified, layers are assumed to be in the same order as the setup file. Either all layers must use *input_number*, or none of them.

layer number — A required keyword set defining a layer that is present in filename.

datatype — An optional argument that limits the datatypes that map onto the posttapeout operation's layers. If *datatype* is absent, all datatypes for that layer are used.

Description

The direct_input command enables you to use a method of direct input of OASIS files, also referred to as OASIS DDE (Direct Data Entry). OASIS DDE can accept any OASIS file. The braces ({{}}) are required.

This command works in conjunction with litho-flat mode and automatically activates it when a direct_input block is found.

This command causes the layers initially input to Calibre to be left unused. They are replaced with layers read directly from the OASIS database specified with the vboasis_path argument. Only the direct data covered by the HDB extent (as defined using a POLYGON statement in the SVRF file) is directly read from the given OASIS database. Therefore, it is not enough to pass in dummy layers through the LITHO MPC statement. At least one HDB layer must contain a square large enough to cover the given OASIS file in order to ensure that everything in it is processed.

Examples

```
LITHO FILE "mpc.setup" [
    ...
    direct_input {
        vboasis_path test.oas
        input 1 layer 99
        input 2 layer 0
    }

    layer opc_input1
    layer opc_input2
    layer opc_input3
    setlayer vbo_in1 = copy opc_input1
    setlayer vbo_in2 = copy opc_input2
]

vboais_in1 { LITHO MPC dummy1 dummy3 dummy4 MAP vbo_in1 FILE "mpc.setup" }
vbo_in2 { LITHO MPC dummy1 dummy3 dummy4 MAP vbo_in2 FILE "mpc.setup" }
```

ebeam_model_load

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Defines and loads the E-beam model.

Usage

```
ebeam_model_load ebeam_model_name {'  
    model_type simulation  
    diff_length diffusion_length...  
    [eta (eta_i)...]  
    [kernel_type {gaussian | exponential}...]  
    [sample_spacing sample_space_val]  
    [log_level 0-100]  
    [rsm {on | off}]  
    [ebeam_kernel_group_limit group_limit]  
    [calibrate {{{{diff_length | eta} kernel_index}} | threshold | transmission}  
        lower_bound upper_bound steps...]  
    [sequence LUMPED]  
'}
```

Arguments

- ***ebeam_model_name***
Specifies the E-beam model name.
- ****model_type simulation****
Performs lithographic simulations using geometry on the input layer by convolving the input image with a Gaussian kernel of the specified diffusion length.
- ****diff_length diffusion_length...****
Specifies the gaussian diffusion length sigma in user units (usually um). This value is $\sqrt{2}$ larger than in the “standard deviation” form of the gaussian function. There must be at least one “very short” range term, with a diffusion length less than 0.1 user unit. The maximum allowed diffusion length is 1.0 user unit.
- ***eta (eta_i)...***
Specifies the weight of the terms in the multigaussian kernel. The terms must be positive and add to 1.0. The number of arguments must match the number of arguments to *diff_length* and be in the same order. This keyword may be omitted if only one kernel is used.

If you are using multiple kernels, the eta values for the different kernels must add up to 1.0. The sequence of sigma values, eta values, and kernel types must match. For example, when using three kernels, you specify:

```
model_type simulation
diff_length 0.025    0.065    1.60
eta          0.70     0.24     0.06
kernel_type gaussian gaussian exponential
```

In the previous example, the third kernel is an exponential kernel with a 1.6um sigma and contributes 6% to the model signal.

- **kernel_type {gaussian | exponential}...**

Specifies the kernel type. The number of arguments must match the number of arguments to `diff_length` and have the same order. This keyword may be omitted if only one kernel is used.

- **sample_spacingsample_space_val**

Controls sampling spacing, which defaults to `diffusion_length/3`. Larger values tend to improve performance, and smaller values improve accuracy. In addition, `sample_spacing` influences the effect of `max_iter_movement`: the effective value of `max_iter_movement` cannot be less than that of `sample_spacing`. The option `sample_spacing` applies to simulation models only.

Note

 The `mpcflow_v2` command requires that `sample_spacing` be less than 0.5 times the lower bound of diffusion length tested.

- **log_level 0-100**

Controls the amount of information printed to stdout: 0 = nothing, 10 = start/stop, 20 = errors, 30 = warnings, 40 = info, 50 = debug, 60 = too much. The default setting is 0.

- **rsm {on | off}**

An optional argument that enables activation of rasterization with spectral matching (RSM) mode, which is an alternative method for mask rasterization. It is more accurate than conventional rasterization. This argument is on by default.

- **ebeam_kernel_group_limit group_limit**

An optional keyword that is used to group the EBEAM Gaussian kernels in the Calibre nmMPC simulation flow. Any kernel less than this value is considered a very short kernel and anything above is considered a short kernel. The default value is 0.100 microns and the valid range is 0.050 microns to 2.0 microns.

Note

 E-beam models are supported with the smallest kernel larger than 100 nm. However, the smallest e-beam kernel must always be smaller than the `ebeam_kernel_group_limit` so that it is part of the first group of kernels.

- calibrate {{diff_length | eta} kernel_index} | threshold | transmission lower_bound upper_bound steps...

Specifies which ebeam_model_load parameters to treat as optimization variables during model calibration (such as diff_length, eta, threshold, and transmission). This keyword is ignored except when the model is used for Calibre nmMPC model calibration. You can specify any combination of parameters to calibrate; those not specified will use the fixed values given by the corresponding model parameter keyword. For example, you can either specify a fixed value for diff_length, or use a range with a lower bound and upper bound.

```
diff_length 0.025 // single fixed value
calibrate diff_length 0 0.021 0.036 10
                    // ranged value plus number of steps
```

kernel_index — Specifies the index into the list of sorted kernels and starts at zero. This index only applies to “per-kernel” parameters such as diffusion length. The other arguments are used by the optimizer in the calibration function. Not all parameters are necessarily used by every type of optimizer.

lower_bound upper_bound steps — Specifies a range for optimization, including a lower and upper bound, as well as the number of steps.

This keyword may be specified multiple times.

- sequence LUMPED

An optional argument that enables a “lumped” model mode. In this mode, instead of applying the e-beam and VEB models in sequence (requiring saving the intermediate results of both models at the expense of run time), both models are treated as a single model and the signal of both is calculated in a single step.

Note

 When switching to “lumped” model mode, the model must be re-calibrated since the simulated signal using identical parameters changes slightly.

Description

The ebeam_model_load command defines an E-beam model. This is required if the **ebeam_model** form of the **setlayer ebeam_simulate** command is used. The setlayer ebeam_simulate command performs pseudo-lithographic simulations using geometry on the input layer. The E-beam model can also be referenced by the **image** command in the **denseopc_options (for MPC)** options block.

The E-beam model simulates a special class of short-range effects, known as E-beam effects, are typically triggered by forward scattering and beam blur.

The command **algorithm** must always be set to 1 and be specified in the corresponding **denseopc_options (for MPC)** options block.

Examples

```
ebeam_model_load ebeam_25nm {  
    diff_length 0.024  
    model_type simulation  
    sample_spacing 0.008  
}
```

etch_model_load

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Loads an etch model.

Usage

etch_model_load *etch_model_name* [*filepath* | '{' *inline* '}']

Arguments

- ***etch_model_name***

A required argument specifying a name that you will later use to refer to the etch model in the [image](#) commands.

One of either *filepath* or {*inline*} must be specified.

- ***filepath***

Specifies a path to the etch model.

- '{' *inline* '}'

Specifies the etch model specifications inline; the curly braces ({}) are required and must surround the inline model definition.

Description

Loads the specified etch model for visible etch bias (VEB) calculations.

layer

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Defines layer characteristics such as name, type, transmission, and mask.

Usage

layer *layer_name*

Arguments

- ***layer_name***

A required argument specifying the name of the layer. Calibre nmMPC uses this name in the rest of the setup file to refer to this layer.

Layer names must be alphabetic and/or numeric strings less than 32 characters in length.

Description

A required command that specifies the input layer. Every layer to be used in setlayer denseopc commands must be declared in the setup parameters using this command.

Examples

```
layer SHAPES
```

modelpath

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Specifies default search location for model files.

Usage

modelpath *dir*

Arguments

- *dir*

Specifies a list of directories to search for optical or resist models. Each directory is separated with a colon (:).

Description

Specifies the directories to search for optical/resist models. This command is optional. The command accepts both absolute and relative paths.

Examples

Absolute path:

```
modelpath /export/home/calibre_wrk/models
```

Relative path:

```
modelpath ../models
```

processing_mode

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Sets the processing mode for Calibre nmMPC.

Usage

processing_mode [hierarchical | flat | hybrid]

Description

The hierarchical mode is the default setlayer processing mode. In this mode, the layout is split into cells (this is decided at HDB construction time, and depends on various SVRF settings such as LAYOUT ULTRA FLEX). Each cell is split into tiles that are processed separately on remote machines. This is a very efficient mode of processing when the layout is extremely hierarchical and there is a lot of redundancy encoded in the cell hierarchy of the input OASIS file.

The litho-flat mode (selected with the “flat” option) is an alternative way of processing hierarchy during the LITHO command. In this mode, the full chip is split into tiles, which are then processed separately on remote machines. This is an efficient mode of processing when the layout is almost flat and is required for Calibre nmMPC operations.

The hybrid mode selectively analyzes the cells and chooses either hierarchical or litho-flat mode depending on the characteristics of the cell. Any children of a cell that is processed in hierarchical mode is also processed in hierarchical mode. Therefore, if the top cell is considered a good hierarchical candidate, the entire design will be processed in hierarchical mode.

The hierarchical and flat settings turn on the specified processing modes regardless of the input hierarchy. This may cause the least efficient processing mode to be active in some cases.

A limitation of the litho-flat processing mode is that the amount of data generated may be larger (because the tool pushes flat output into a hierarchical database), which may cause a slowdown of downstream SVRF operations.

Arguments

- hierarchical | flat | hybrid

Selects the type of processing (hierarchical, litho-flat, or hybrid). hierarchical processing is the default value for LITHO. For Calibre nmMPC, this must be explicitly set to flat.

setlayer ebeam_simulate

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Simulates E-beam simulation on input layer.

Usage

```
setlayer output_layer_name = ebeam_simulate {  
    {input_layer_name ebeam_model ebeam_model_name [dose dose_value]...  
     [etch_model etch_model_name] [sample_spacing sample_spacing_val]  
     [etch_imagegrid etch_imagegrid_val] [vbe_kernel_group_limit val] | image_options name}
```

Arguments

- ***output_layer_name***

A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer. This layer must be a valid Calibre layer containing polygon data.

- ***input_layer_name***

Specifies the name of the input layer. The layer must be a valid Calibre layer containing polygon data.

- **{*input_layer_name ebeam_model ebeam_model_name*[dose *dose_value*]...}**

Specifies an E-beam model as created by [ebeam_model_load](#). The **ebeam_model** syntax requires **ebeam_model_name** be defined before **ebeam_simulate**. Each input layer block includes the layer name, the applied E-beam model, and the dose value (where the dose is optional) and the default value is 1. Currently, all input layers must share the same E-beam model. Multiple input layers can be specified.

If you are using a litho model, these keywords are not required. See “[Import and Define Models](#)” on page 56 and “[Lithomodel \(Litho Model Format\)](#)” on page 341 for further information.

- **etch_model *etch_model_name***

Specifies an optional etch model. This is not required if you use a litho model instead.

- **sample_spacing *sample_space_val***

Specifies an optional pixel size (in microns) to be used for E-beam simulation and controls sample spacing to improve accuracy. This parameter is functionally equivalent to the **sample_spacing** parameter utilized by [ebeam_model_load](#).

- **etch_imagegrid *etch_imagegrid_val***

Specifies an optional pixel size (in microns) to be used for etch simulation. This parameter is functionally equivalent to the **etch_imagegrid** command utilized by Calibre OPCVerify.

- **image_options *name***

Specifies the name of an [image_options](#) block. This is required if you use a litho model.

Description

The `ebeam_simulate` command performs pseudo-lithographic simulations using geometry on the input layer. It differs from the `image` command in that instead of performing a convolution for the entire contour, it computes explicitly a number of key points on the contour and connects them with edges that approximate the desired contour.

Examples

Multiple Input Layers

The following example shows a simulation model with multiple input layers.

```
ebeam_model_load "ebeam_25nm" {  
    model_type simulation  
    diff_length 0.025  
}  
setlayer eb_contour = ebeam_simulate poly0 ebeam_model \  
    "ebeam_25nm" dose 0.8 poly1 ebeam_model "ebeam_25nm" dose 1.2 \  
    etch_model etch
```

Default Sample Spacing

The following example shows a simulation model with default sample spacing.

```
ebeam_model_load "ebeam_25nm" {  
    model_type simulation  
    diff_length 0.025  
}  
setlayer eb_contour = ebeam_simulate ebeam_model "ebeam_25nm"
```

setlayer curve

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Generates a new layer with excess jogs and other small 2D effects removed from the output (a “smoothing effect”).

Usage

```
setlayer output_layer_name = curve input_layer_name
[order {linear | quadratic | cubic | num}] [cpdist cpdist_value] [pts_per_cp value]
[maxdist maxdist_value] [midpt midpt_value] [scale1 value] [scale2 value]
[midpt_offset value] [endpt_offset value] [active_layer active_layer]
[jog_tol tol_val jog_adj adj_val] [jog_cleanup_dist value] [jog_extra value]
[output_cpts val] [output_clean_target]
[concave [set_of_options]] [convex [set_of_options]]
[line_end [set_of_options] length value]
[space_end [set_of_options] length value]
[jog [set_of_options] length value]
[new_cp corner1 corner2 length constraint [length1 constraint] [length2 constraint]
offset value [perp value] [delete_corner] [delete_midpt]]]
```

where *set_of_options* is the following set of optional arguments:

```
[cpdist cpdist_value] [maxdist maxdist_value] [midpt midpt_value][midpt_offset value]
[endpt_offset value]
```

Arguments

- ***output_layer_name***

A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer. This layer must be a valid Calibre layer containing polygon data.

- ***input_layer_name***

A required keyword specifying the name of the input layer.

- **order {linear | quadratic | cubic | num}**

An optional parameter that specifies the B-spline order number. You can either specify a value *num* between 2 and 6, or specify linear (equivalent to a *num* of 2), quadratic (3), or cubic (4). The default value is quadratic.

- **cpdist *cpdist_value***

Specifies the distance between additional control points inserted to the left and right of an original polygon point. Longer distances increase rounding. The default value is 50 nm.

- **pts_per_cp *value***

Specifies the number of vertices for each control point in the output. The default is 10 vertices.

- maxdist *maxdist_value*

Specifies that the last control point should be at exactly *maxdist_value* away from the original polygon vertex. Though the spline curve will not trace out the same shape as the polygon, at *maxdist_value*, the curve will touch the polygon.

- active_layer *active_layer*

If the active_layer argument is specified, the output produces control points that overlap between the specified input layer and *active_layer*. This is implemented by making sure that no additional control points are placed within the *active_layer*.

The effect of this option is to force the output edge to conform to the drawn gate edge while still over the active layer before flaring out at the 90 degree polygon bend. This squares out any 90 degree turns in poly or metal layers that lie near an active region, minimizing any gate flare effect.

- midpt *midpt_value*

Specifies how many additional control points to add in the middle of a notch or nub that would have had no additional control points otherwise.

- scale1 *value*
scale2 *value*

If an adjacent edge is a notch or nub and is smaller than *cpdist_value* * scale1, control points on adjacent edges are put at a *cpdist_value* * scale2 distance instead of the *cpdist_value*. This is done for better curvature control for various line widths. Both scale1 and scale2 default to 1 if undeclared.

- midpt_offset *value*

All control points created by the midpt parameter (in the middle of line ends and space ends) are moved to increase the curvature. Line-end control points move towards the outside of the polygon. Space-end control points move towards the inside of the polygon.

- endpt_offset *value*

All control points situated at the polygon's corners are moved to increase the curvature (towards the outside of the polygon for convex corners and towards the inside of the polygon for concave corners).

- jog_tol *tol_val* jog_adj *adj_val*

When specified, this option will ignore all jogs less than or equal to *tol_val* that have neighbor edges smaller or equal to *adj_val*. This is to prevent small jogs from affecting the results of target smoothing prior to MPC.

- jog_cleanup_dist *value*

In the first stage of generating a spline, target jogs are removed according to the jog removal parameters jog_tol and jog_adj.

If a value for jog_cleanup_dist is set:

- Edges with their length \leq jog_tol and with at least one neighbor with length \leq jog_adj (if jog_adj is present), and with both neighbors' lengths $>$ jog_tol, and parallel to each other, are classified as jogs.
- Edges classified as jogs are removed.
- Unconnected edges (which are parallel by definition) are shortened by up to jog_cleanup_dist then connected by a line.

Note

This algorithm has a finite interaction distance, thus will cause no hierarchical differences.

If jog_cleanup_dist is not specified, setlayer curve defaults to 0.5 microns (the maximum value possible).

- **jog_extra value**

An optional argument that specifies to generate additional control points when edges are longer than jog_adj *adj_val*. The control point is inserted *value* from the jog along the edge.

- **output_cpts val**

If this parameter is present, squares with a half-width equal to the *val* are output instead of the actual curve. The squares represent the curve's control points.

- **output_clean_target**

If this parameter is present, the “clean” (after jog removal) target is output instead of the spline.

- **concave**

If this parameter is present, the global values set by the previous arguments are overridden in the case of concave corners.

- **convex**

If this parameter is present, the control points are set for convex corners.

- **line_end**

If this parameter is present, the control points are set to line ends.

- **space_end**

If this parameter is present, the control points are arranged for space ends.

- **jog**

If this parameter is present, the control points are set for jogs.

- **length value**

An optional argument (required if line_end, space_end, or jog are used) that specifies the fragment length of the feature.

Note

 If concave, convex, line_end, space_end, and jog are not used, or if they are not fully defined, then the values used will be the values set by the original keywords. If those values are not set, then the default values are used.

- `new_cp corner1 corner2 length constraint [length1 constraint] [length2 constraint]`
`offset value [perp value] [delete_corner] [delete_midpt]`

An optional argument set that specifies when to manually add new control points. When a point could be added by either new_cp or jog_extra, the one from new_cp is inserted.

corner1 corner2 length constraint — Specifies the edge to receive the control point. The corners may be “convex,” “concave,” “noncorner,” or “any.” Noncorner refers to the shallow corners inserted near jogs when jog_tol is specified. If jog_tol is not set, no corners match noncorner.

length1 constraint — An optional constraint that applies to the preceding edge.

length2 constraint — an optional constraint that applies to the edge after *corner2*.

The length constraints are in microns.

If the edge specification is symmetric (for example, new_cp convex convex length...) two control points are inserted, because it matches both corners.

offset value [perp value] — Specifies where to place the new control point. The offset value must be between -1.0 and 1.0, inclusive, measured inward along the edge from *corner1*.

perp value — An optional perpendicular offset. Positive values move the control point outward from the polygon. Specifying “offset 0.05 perp -0.05” will offset the control point 50 nm away from the first corner and 50 nm into the polygon’s region.

delete_corner — An optional argument to new_cp that removes an old corner control point to prevent interference with the new control point.

delete_midpt — An optional argument to new_cp that removes an old midpoint control point to prevent interference with the new control point.

Description

The setlayer curve command allows you to create a “smoothed” target based on the original target. The smoothed target allows you to prevent overcorrection on 2D effects and jogs.

The setlayer curve command implements the curve using a series of B-splines. B-splines are also known as basis splines, a spline function with minimal support with respect to a given degree, smoothness, and domain partition. Spline functions of a given degree, smoothness, and domain partition can be represented as a linear combination of B-splines of that same degree and smoothness over that same partition.

This command generates a spline or spline-derived output in the following sequence:

1. Target jogs are removed according to the jog removal parameters `jog_tol` and `jog_adj`.
The results are affected by whether or not you set a value for `jog_cleanup_dist`.
2. Spline control points are generated according to the following parameters: `order`, `cpdist`, `pts_per_cp`, `maxdist`, `midpt`, `scale1`, `scale2`, `midpt_offset`, `endpt_offset`, `active_layer`.
Control points are derived from existing polygon vertices; additional points are added before and after corners. At least one point is added in the center of short segments.
3. The output is generated according to the control points and the following parameters:
`pts_per_cp`, `output_clean_target`, `output_cpts`.

The general process of target smoothing involves the following steps:

1. Create a smoothed target using the `setlayer curve` command.
2. Run MPC, passing the smoothed target as a re-target layer.
3. Use the `retarget_layer emulate` option.

More specifically, you can do the following:

1. Try to obtain the MPC results without using the smoothed target.
2. Use post-MPC simulation contour to estimate the amount of residual corner rounding.
3. Adjust curve parameters to match the existing contour.
4. When done, compare MPC results with and without smoothed input. The results should show improved MPC, though some fragment length adjustment may help.

Warnings

If `jog_tol` is used without `jog_cleanup_dist`, the following warning appears in the transcript:

```
"setlayer curve": Using jog_tol without jog_cleanup_dist can cause  
hierarchical inconsistencies
```

Using `jog_tol` without `jog_cleanup_dist` is not recommended and will be deprecated in the future.

If `pts_per_cp` is set to a value higher than 5, the following warning appears in the transcript:

```
"setlayer curve": Using a pts_per_cp value larger than 5 can cause  
unnecessary slowdown in curve and subsequent ops
```

Optimize Results From setlayer curve

The following use cases apply to setlayer curve:

- Use setlayer curve when the target design is relatively simple and free of jogs of various sizes. This is often the case for active and poly layer.
- Do not use curve if the design is complicated and contains jog of various sizes. This is often the case for metal layers.
- There is no need to use curve if the design is extremely simple, such as contact or via layers.

There are a number of different parameters in setlayer curve that can be utilized to get the best results, depending on the kind of curvature you wish to produce.

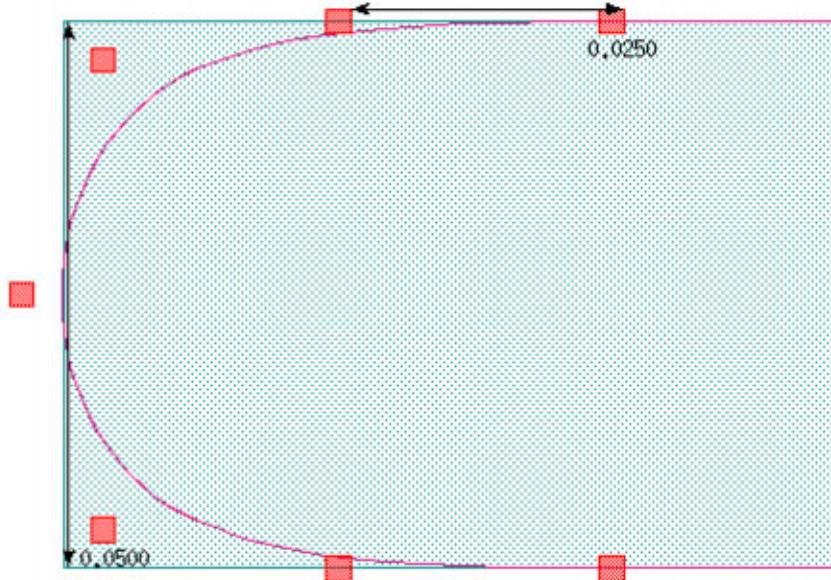
[Table 5-4](#) lists the recommended general setlayer curve parameter settings to get the best overall curvature fit.

Table 5-4. General Recommended Settings for setlayer curve

Parameter	Recommended Setting	Notes
order	6	An optional parameter that specifies the B-spline order. You can select linear, quadratic, or cubic.
cpdist	Set to 60% of your design's min_cd	Specifies the distance between additional control points inserted to the left and right of an original polygon point. Longer distances increase rounding.
maxcp	6	Specifies the maximum number of additional points inserted to the left and right of an original polygon point.
maxdist	Set to your design's minimum cd	Controls where the last control point should be at exactly the specified distance away from the original polygon vertex.
midpt	1	Specifies how many additional control points to add in the middle of a notch/nub that would have had no additional control points otherwise.
pts_per_cp	10	This parameter only controls the size of the output control points, it does not influence the curve parameter.

Table 5-4. General Recommended Settings for setlayer curve (cont.)

Parameter	Recommended Setting	Notes
scale1	2.0	If a line end or space end is smaller than $cpdist_value * scale1$, control points on adjacent edges are put at a $dist_value * scale2$ distance, instead of the $cpdist_value$. See Figure 5-15 for an example of line_end and space_end scaling.
scale2	0.5	Any line/space end that has width $< scale1 * cpdist$ will have a control point inserted on an adjacent edge with distance of $scale2 * width$. See Figure 5-15 for an example of line_end and space_end scaling.
endpt_offset	$> 0.25 * min_cd$ $< 0.4 * min_cd$	All control points situated at the polygon's corners are moved to increase the curvature.
midpt_offset	0.000	All control points created by the midpt parameter are moved to increase the curvature.

Figure 5-15. Example of Line/Space End Scaling

[Table 5-5](#) lists the recommended setlayer curve parameter settings for better handling of line ends and space ends.

Table 5-5. Line/Space End-Specific Recommended Settings for setlayer curve

Parameter	Recommended Setting	Notes
[line_end space_end]	Select either line_end or space_end, depending on the feature	Specifies feature as a line end or space end.
cpdist	0.6*min_cd	Specifies the distance between additional control points inserted to the left and right of an original polygon point. Longer distances increase rounding.
maxdist	[0.8 – 1.0]*min_cd	Controls where the last control point should be at exactly the specified distance away from the original polygon vertex.
midpt	1	Specifies how many additional control points to add in the middle of a notch/nub that would have had no additional control points otherwise.
midpt_offset	0.1*min_cd	All control points created by the midpt parameter are moved to increase the curvature.
endpt_offset	0.1*min_cd	All control points situated at the polygon's corners are moved to increase the curvature.
width	[1.0-1.5]*min_cd	Specifies the maximum width of the line end or space-end edge. See Figure 5-16 .
length	[1.0-1.5]*min_cd	Specifies the fragment length of the line end or space-end edge. See Figure 5-16 .

Figure 5-16. Length and Width Representation for setlayer curve

Table 5-6 lists the recommended parameter settings to reduce jogs in the rendered curve.

Table 5-6. Recommended Jog-Specific Settings for setlayer curve

Parameter	Recommended Setting	Notes
jog	Specify “jog” as the feature	Specifies feature as a jog.
cpdist	$0.6 * \text{min_cd}$	Specifies the distance between additional control points inserted to the left and right of an original polygon point. Longer distances increase rounding.
maxdist	$[0.8 - 1.0] * \text{min_cd}$	Controls where the last control point should be at exactly the specified distance away from the original polygon vertex.
midpt	1	Specifies how many additional control points to add in the middle of a notch/nub that would have had no additional control points otherwise.
midpt_offset	0.000	All control points created by the midpt parameter are moved to increase the curvature.
endpt_offset	$0.1 * \text{min_cd}$	All control points situated at the polygon's corners are moved to increase the curvature.
length	roughly $0.6 * \text{min_cd}$	Specifies the fragment length of the feature.

Note

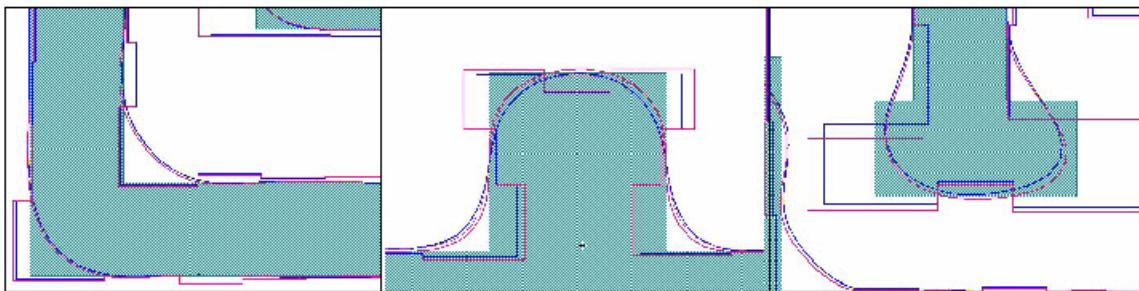
 A concave/convex-specific setting is generally not needed unless a different curvature is desired.

Retarget Layers and setlayer curve

Use the `retarget_layer` “emulate” option with the generated curve layer. This helps reduce ripples near corners especially at concave corners. The `setlayer curve` produces a more realistic target, which often results in less aggressive corner correction. However, you must still carefully tune your fragmentation and MPC recipes.

Figure 5-40 shows the effects of using `retarget_layer emulate`.

Figure 5-17. Effects of retarget_layer emulate and setlayer curve



The blue lines show the `retarget_layer emulate` effects, while the red lines show the effects without `retarget_layer emulate`.

You can also turn off retargeting through a tagging script. For example:

```
NEWTAG topological inside_edge M1 no_rettarget_region \
    -out frag_ignore_retarget
TARGET_FUNCTION 1 constant 0.0
TARGET_CONTROL frag_ignore_retarget 1
```

Examples

The following example creates a smoothed target layer:

```
setlayer CURVE1 = curve pctarget order 5 cpdist 0.07 maxdist 0.12
```

The following example filters out any jogs < 2nm with neighboring fragments < 50nm:

```
setlayer smooth = curve target order 6 cpdist 0.070 maxcp 6 pts_per_cp 4 \
    maxdist 0.050 midpt 3 scale1 1.25 scale2 0.7 endpt_offset -0.00 \
    jog_tol 0.002 jog_adj 0.050
```

setlayer curve_target

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Creates a target that mimics the final image.

Usage

```
setlayer output_layer_name = curve_target input_layer_name [criticalDistance val]
    [cornerRadius val] [cornerRadiusConcave val] [cornerRadiusConvex val]
    [colinearAngleTolerance degrees]
    [lineEndRatio val] [linearRatio {true | false}] [spaceEndRatio val]
    [lineEndWidth value] [spaceEndWidth value] [maxJog val]
    [roundCorner {true | false}] [roundedCornerRadius value]]
    [preferMid {true | false}] [midSpanExt val] [maxMidSpan val]] [enclose layer by val]
    [taglayer layer overrides]
```

Arguments

- ***output_layer_name***

A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer. This layer must be a valid Calibre layer containing polygon data.

- ***input_layer_name***

A required keyword specifying the name of the input layer.

- ***criticalDistance value***

The smallest feature that must be resolved. The default value is (Ro/3), where Ro is the resolution value obtained from the optical model parameters (lambda/NA).

This parameter is typically specified, rather than using the default value. You can optimize the value around the default value by plus or minus 50% to obtain the best results for your applications.

- ***cornerRadius val***

An optional argument that is used at corners to estimate the achievable target. You should optimize this parameter around the default value by plus or minus 50% to obtain the best results for their applications.

The value must be greater than 0. The default is criticalDistance.

- ***cornerRadiusConcave val***

The radius of curvature used to estimate the achievable target for concave corners only. This value is also used for space ends. The value must be greater than 0. The default value is cornerRadius, which in turn defaults to criticalDistance.

- `cornerRadiusConvex val`

The radius of curvature used to estimate the achievable target for convex corners only. This value is also used for line ends. The value must be greater than 0. The default value is `cornerRadius`, which in turn defaults to `criticalDistance`.

Note

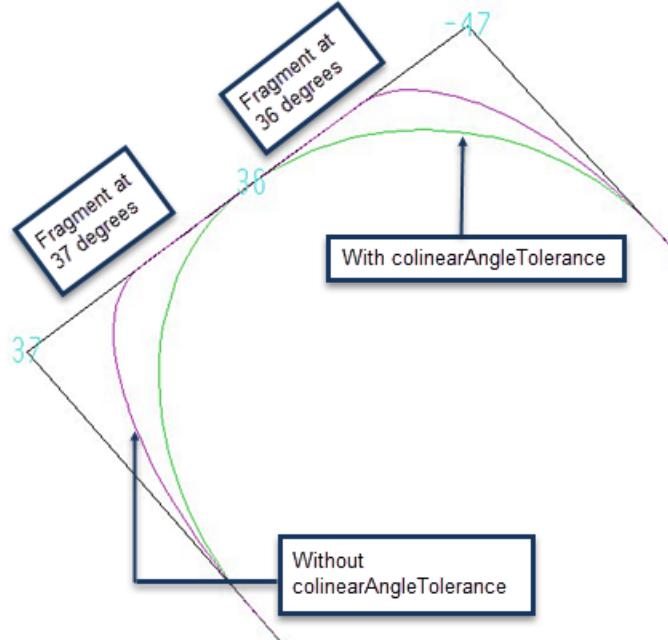
 In jogs and s-shapes that have overlapping convex and concave corners, the larger of `cornerRadiusConcave` or `cornerRadiusConvex` is used.

- `colinearAngleTolerance degrees`

An optional parameter that specifies in degrees how different two adjacent fragments may be and still be considered as collinear. The value must be an integer. The default value is 0 degrees.

This option is useful when a design contains non-45° skew angles as it can allow the algorithm to better recognize a straight edge, as shown in the figure below. The sharper outer contour (maroon) occurred because the two fragments of the edge were treated independently. Adding “`colinearAngleTolerance 1`” produced the smoother green contour.

Figure 5-18. Effect of colinearAngleTolerance 1



- `lineEndRatio val`

Allows for the specification of an elliptical curve at line ends. The curve is tangent to the side at a distance `val` from the corner (where `val` is computed by multiplying the width of the line by the ratio). The default ratio is 0.5 (resulting in a circular curve). This applies to narrow line ends whose width is less than two times the corner radius.

- linearRatio { true | false }

An optional keyword that, if set to false, Calibre nmMPC makes the following adjustments to the curvature of the smooth target:

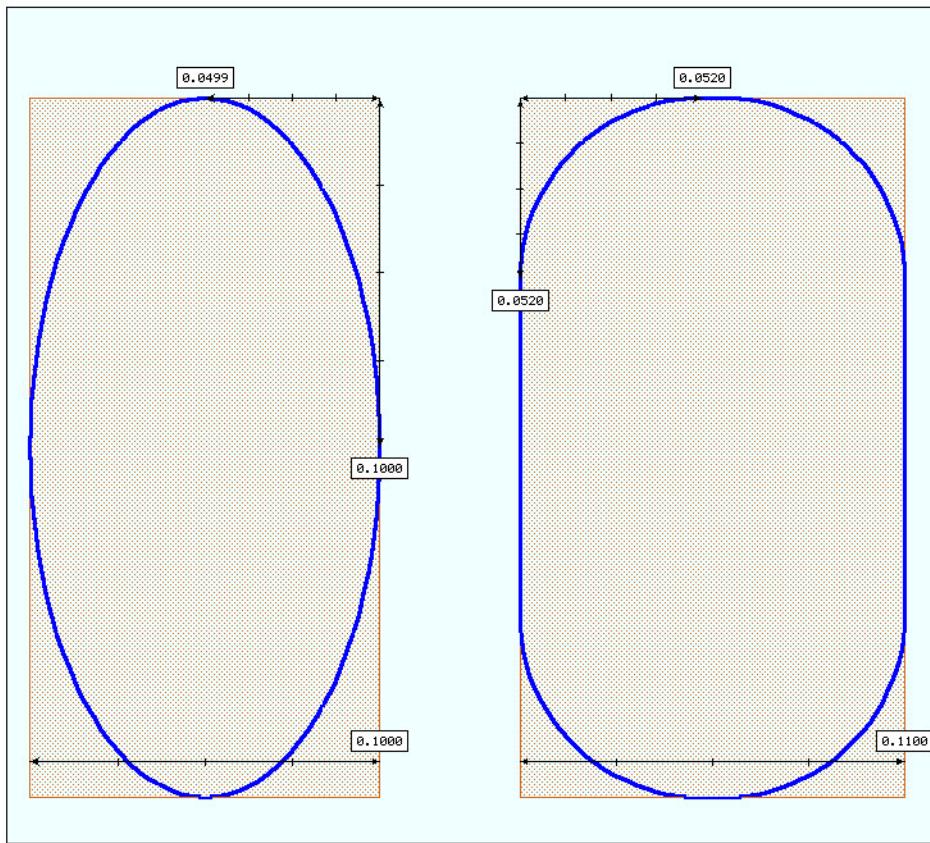
- For the line ends smaller than lineEndWidth (or if not defined, smaller than $2 * \text{cornerRadius}$):
 - On the line end (LE) fragment, the tangent point of the curve is set at the middle of the LE fragment.
 - On the LE adjacent fragment, the tangent point of the curve is set at a distance specified by that particular (LE CD) * lineEndRatio away from the corner.
- For line ends larger than lineEndWidth (or if not defined, smaller than $2 * \text{cornerRadius}$) the tangent point of the curve is set at a cornerRadius distance away from the corner for both LE and LE adjacent fragments.

[Figure 5-19](#) illustrates an example using the following settings:

```
criticalDistance 0.050 cornerRadius 0.052 lineEndRatio 1.0
```

Note that sharp transitions in curvature radius can occur when the lineEndWidth is reached.

Figure 5-19. Curvature Example



If linearRatio is set to true, Calibre nmMPC makes the following adjustments to the tangent point of the curvature at the LE adjacent fragment:

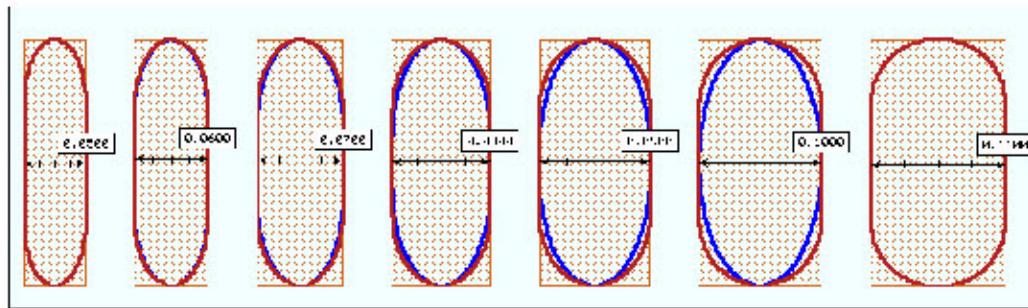
- No adjustment occurs for line ends smaller than cornerRadius.
- For the line ends larger than cornerRadius but smaller than $2 * \text{cornerRadius}$, the tangent point of the curve is set at a distance specified by that particular (LE CD) * lineEndRatio * multiplier.
- The multiplier is an additional scaling parameter that changes linearly between cornerRadius and $2 * \text{cornerRadius}$.
- No adjustment occurs for the line ends larger than $2 * \text{cornerRadius}$.

Figure 5-20 shows a comparison of smooth curves with linearRatio set to false (in blue) and true (in red) with the rest of the parameters as follows:

```
criticalDistance 0.050 cornerRadius 0.052 lineEndRatio 1.0
```

Setting linearRatio allows for smoother transitions.

Figure 5-20. Comparisons of linearRatio True Versus False



Red contours: linearRatio set to true
Blue contours: linearRatio set to false

In cases where the lineEndWidth is defined and is $\leq 2 * \text{cornerRadius}$:

- No adjustment occurs for line ends smaller than lineEndWidth.
- For the line ends larger than lineEndWidth but smaller than $2 * \text{cornerRadius}$, the tangent point of the curve is set at a distance specified by that particular (LE CD) * lineEndRatio * multiplier.
 - The multiplier is an additional scaling parameter that changes linearly between lineEndWidth and $2 * \text{cornerRadius}$.
- No adjustment occurs for line ends larger than $2 * \text{cornerRadius}$.

In cases where the lineEndWidth is defined and is $> 2 * \text{cornerRadius}$:

- If the curve's tangent point on the line end is farther from the corner than the cornerRadius value, the point is moved closer to the corner (by changing the lineEndRatio value).

This means that the tangent point on the line end with a width equal to lineEndWidth (maximal) is set at a distance (the cornerRadius value) from the corner. This provides a smoother transition.

- No adjustment occurs for line ends larger than lineEndWidth.

The default value is false.

- **lineEndWidth *value***

The minimum width at which an edge with two convex corners is not classified as a line end. Edges with length less than *value* are treated as line ends. The units for *value* are user units. The default value is $2 * \text{cornerRadiusConvex}$.

- **spaceEndWidth *value***

The minimum width at which an edge with two concave corners is not classified as a space end. Edges with length less than *value* are treated as space ends. The units for *value* are user units. The default value is $2 * \text{cornerRadiusConcave}$.

- **spaceEndRatio *val***

An optional flag that allows for the specification of an elliptical curve at space-ends. This option otherwise performs similarly to the lineEndRatio option. The default value is the lineEndRatio setting.

- **maxJog *val***

The maximum length at which an edge is considered a jog and not a feature. Smaller values cause fewer edges to be considered jogs. The units for *val* are user units. The default value is approximately $0.292893 * \text{radius}$, where radius is the minimum of cornerRadiusConcave, cornerRadiusConvex, and cornerRadius.

- **enclose *layer* by *val***

An optional flag that allows the target to be drawn around contacts and vias, where *val* specifies the amount of enclosure or internal distance requested for MPC. This can be useful if the target is being used outside of Calibre nmMPC.

- **roundCorner {true | false} [*roundedCornerRadius value*]**

An optional parameter that causes Calibre nmMPC to round concave corners that are shorter than cornerRadiusConcave. When set to true, the target is moved out towards the printed image by roundedCornerRadius. You can use the optional roundedCornerRadius keyword to increase the distance from the original corner that the target is moved. The default value for this parameter is false.

- preferMid {true | false}

If set to true, this parameter specifies that the midPoint handler should be used where possible, instead of S-steps or quarter circles. This means that instead of a curve being drawn, a straight line will be used. For some curve targets, this may be a more appropriate fit than curves that are typically used. The default value for this parameter is false.

- midSpanExt *val*

Specifies a floating point value between 0 and 10 that controls the slope of the line drawn when preferMid is enabled. The default value is 1, which draws a 45-degree angle line through the mid point of a span. Increasing the value of midSpanExt increases the length of the line drawn through the span and results in a shallower slope. By default, the extent of the line drawn is equal to the length of the span (*val* is set to 1). Increasing the value to 2 means that the extent of the line drawn will now be twice as long as the span, and the slope will be much shallower.

- maxMidSpan *val*

Specifies a length in user units (0 to 1 micron) that determines the maximum size of a span to be handled by the midpoint handler when preferMid is enabled. Spans longer than this value will be represented by curves. The default value is 2/3 of the critical distance (CD).

Description

The purpose of the setlayer curve_target command is to create a target that mimics what the final image is going to look like. Using this command, you look at the differences between the image and the curve target and adjust the parameters to more closely match what the final image looks like. By having a feasible target (one that can actually be printed on silicon), the optimization can be simplified, resulting in better MPC results.

Examples

```
setlayer ctarget = curve_target target_fill \
    criticalDistance 0.040 \
    cornerRadiusConcave 0.020 \
    cornerRadiusConvex 0.040 \
    lineEndRatio 1.0
```

setlayer mpc

LITHO MPC setup file command (see [Table 5-3](#) on page 133)

Creates and derives layers for MPC.

Usage

```
setlayer output_layer_name = mpc input_layer_name MAP output_layer OPTIONS
    options_block_name [property '{'property_block'}']
```

Arguments

- ***output_layer_name***

A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer. This layer must be a valid Calibre layer containing polygon data.

- ***input_layer_name***

Specifies the name of the input layer. The layer must be a valid Calibre layer containing polygon data.

- **MAP*output_layer***

A required keyword followed by the associated argument indicating the layer or tag containing the output to be returned. This keyword is required even for non-concurrent runs.

- ****OPTIONS** *options_block_name***

A required option that specifies the name of a denseopc_options block, a sub-command block of LITHO MPC setup file commands (see [Table 5-7](#)) defined inside a LITHO FILE parameter block.

- [property '{'property_block'}']

Specifies a property block. A property block is used to assign properties to layers, used primarily for error checking during the verification phase. The MPC version of the property block is identical to the blocks used by Calibre OPCverify.

For complete information on property blocks, refer to the [Calibre OPCverify User's and Reference Manual](#).

Description

This command creates and derives layers, depending on the options used. You create output layers using the design layers you defined with the layer command, and you will eventually use one or more of them as output in your LITHO MPC calls in your SVRF file.

The setlayer command also uses concurrency in order to map different outputs to setlayer commands. The MAP keyword is required even for non-concurrent runs. The OPTIONS keyword must be declared last in the command line. The order of layers passed should match the definitions of layers declared using LITHO MPC.

The [denseopc_options \(for MPC\)](#) command is required in order to call a setlayer denseopc command. The options block used with setlayer mpc must use the [image](#) command.

Examples

```
setlayer poly_mpc = mpc poly MAP poly OPTIONS ebeam_options
```

setlayer clmpc

LITHO CLMPC setup file command (see [Table 5-3](#) on page 133)

Creates and derives layers for CLMPC.

Usage

```
setlayer output_layer_name = clmpc input_layer_name MAP output_layer OPTIONS  
    options_block_name [property '{'property_block'}']
```

Arguments

- ***output_layer_name***

A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer. This layer must be a valid Calibre layer containing polygon data.

- ***input_layer_name***

Specifies the name of the input layer. The layer must be a valid Calibre layer containing polygon data.

- **MAP*output_layer***

A required keyword followed by the associated argument indicating the layer or tag containing the output to be returned. This keyword is required even for non-concurrent runs.

- **OPTIONS *options_block_name***

A required option that specifies the name of a denseopc_options block, a sub-command block of LITHO CLMPC setup file commands (see [Table 5-7](#)) defined inside a LITHO FILE parameter block.

- [property '{'property_block'}']

Specifies a property block. A property block is used to assign properties to layers, used primarily for error checking during the verification phase. The LITHO CLMPC version of the property block is identical to the blocks used by Calibre OPCVerify.

For complete information on property blocks, refer to the [Calibre OPCVerify User's and Reference Manual](#).

Description

This command generates or derives layers, depending on the options used, used in conjunction with [LITHO CLMPC](#), both created specifically for curvilinear and skew-edge layouts. You create output layers using the design layers you defined with the layer command, and you subsequently use one or more of them as output in your LITHO CLMPC calls in your SVRF file.

The setlayer command also uses concurrency in order to map different outputs to setlayer commands. The MAP keyword is required even for non-concurrent runs. The OPTIONS

keyword must be declared last in the command line. The order of layers passed must match the definitions of layers declared using LITHO CLMPC.

The [denseopc_options \(for MPC\)](#) command is required in order to call a setlayer denseopc command. The options block used with setlayer clmpc must use the [image](#) command.

Examples

```
setlayer mpcOut = clmpc mpcIn MAP mpcIn OPTIONS clmpcOption
```

MPC denseopc_options Commands

The OPTION *opt_name* argument specifies an inlined denseopc_options sub-parameter block (within the LITHO FILE block) containing a *lowercase only* command initialization block consisting of the following keywords, specified one per line:

Table 5-7. MPC denseopc_options Summary

Keyword	Description
algorithm	This must always be set to 1.
allow_jog_full_move	Enables the full movement of jog edges in Calibre nmMPC correction.
background (for denseopc_options block)	A required keyword that describing the imaging background.
check_movement	Specifies whether or not to check mask constraints.
corner_control	Controls the length of the first fragment next to corners.
convex_concave_adj_len	Defines the adjacent length of a fragment being processed by the SEM Box method of correction.
convex_concave_correction	Enables special MPC correction for convex and concave corner fragments.
convex_concave_count	Specifies the number of measurement points to be placed on convex and concave fragments for correction.
convex_concave_limit	Specifies a size limit for convex and concave fragments considered for correction.
convex_concave_metric	Specifies the percentage of a fragment to be considered for placing measurement points for correction.

Table 5-7. MPC denseopc_options Summary (cont.)

Keyword	Description
epe_spacing	Sets spacing between EPE or image samples along fragments.
feedback	This defines the default feedback values to be used per iteration.
fragalign	Aligns fragment break points on a specified layer.
fragment_coincident	Copies fragmentation from one layer to another where edges are coincident.
fragment_corner	Fragmentation parameter providing precise control over the number and length of intrafeature fragmented edges at convex and concave corners.
fragment_flaglayer	Controls the fragmentation operation for a specified layer.
fragment_inter	Fragmentation parameter breaking the design into enough edges to allow the MPC algorithm to improve the simulated EPE.
fragment_interlayers	Specifies layers for interfeature fragmentation.
fragment_island	Controls island layer fragmentation.
fragment_layer	Command block that allows for changes to global parameter settings for a specified layer.
fragment_layer_order	Used to change the default processing order of fragmentation schemes in a fragment_layer command block.
fragment_nearly_coincident	Copies fragments from one layer to another where the layer edges are within a specified distance.
fragment_max	Fragmentation parameter controlling the minimum edge produced by fragmentation.
fragment_min	Fragmentation parameter causing very long edges to be broken up.
fragment_minjog	Controls jog definition.
fragment_small	Specifies the upper size limit for fragments using a high-accuracy EPE measurement algorithm.
fragment_visible	Defines layers for visible layer fragmentation.

Table 5-7. MPC denseopc_options Summary (cont.)

Keyword	Description
freeze_skew_edges	Freezes skew edges and neighboring fragments.
grids_for_angle_45_snap	Specifies how 45-degree angle lines are moved during MPC.
image	Specifies the E-beam or etch model.
image_options	Creates a named image option block.
jog_freeze	Prevents a fragment from moving during MPC if it is a jog edge.
jog_ignore_metric	Changes a fragment's corner classification when it is shared with a short jog.
layer (denseopc_options Command)	Defines an input layer.
line_end_fragment	Activates or deactivates line-end fragmentation.
max_angle_tolerance (LITHO CLMPC Only)	Specifies the angle tolerance between two edge segments considered when smoothing edges of curvilinear and skew-edge layouts.
mpc_contour_search_radius	Overrides default contour search distance during EPE calculations.
max_iterations	Specifies the maximum iterations of MPC to perform.
max_iter_movement	Specifies the maximum distance to move an edge during an MPC iteration.
max_opc_move	Specifies maximum total distance all fragments are allowed to be moved by the entire opc run.
max_smoothing_range (LITHO CLMPC Only)	Specifies the local range along edges used for smoothing. Larger values result in smoother output contours.
measure_metric_mode	Specifies that the average metric is to be used for all fragment operations.
mpc_fragment_type_projection	Projects fragment types from opc layers to correction layers for multilayer MPC.
mpc_sites_control	Specifies distance measurement sites are to be placed away from corners.
mpc_sites_mode	Specifies the site placement algorithm.
mpc_poi_measurement (LITHO CLMPC Only)	Optimizes EPE measurement time for CLMPC by reducing EPE measurements.
mpc_preserve_angles (LITHO CLMPC Only)	Preserves edge angles in CLMPC movement.

Table 5-7. MPC denseopc_options Summary (cont.)

Keyword	Description
mpc_pseudo_nyquist	Sets the Nyquist value for MPC flows.
mpc_skip_fragments	Skips specified fragments from EPE measurement computations.
mpc_use_nearest_epe	Uses the nearest valid EPE for fragments that do not have a well-defined EPE.
mrc_rule	Specifies an internal/external constraint check between layers based on the distance between fragments or the fragment length.
no_opc_region	Specifies a “do not correct” layer where fragments that touch the specified layer do not have MPC performed on them.
opc_grid_multiplier	Refines the correction grid by a specified factor.
retarget_layer	Defines a new target layer (which should be a defined as a hidden layer with the “layer” command), but leaves the fragmentation on the opc layers.
scale	Used to resize fragmentation lengths according to a scale multiplier.
sparse_ebeam_veb	Specifies a sparse Gaussian simulation for e-beam and Variable Etch Bias (VEB) simulators.
step_size	Sets the minimum edge movement distance for MPC.
tile_fragmentation_only	Deactivates all default fragmentation except at tile boundaries. This must always be set to 1.
version	A required argument that specifies the version number of the MPC algorithm to use.

algorithm

[MPC denseopc_options Commands](#)

Bypasses contour creation to improve output consistency.

Usage

algorithm 1

Arguments

- **1**

Required setting to improve consistency.

Description

The algorithm command is used to bypass contour creation to improve consistency. In Calibre nmMPC, this command must always be set to 1.

allow_jog_full_move

[MPC denseopc_options Commands](#)

Enables the full movement of jog edges in Calibre nmMPC correction.

Usage

allow_jog_full_move {off | on}

Arguments

- **off | on**

Setting this parameter to **on** enables the full movement of jog edges in Calibre nmMPC correction. If set to **off**, the edge movement is bounded by its length. The default value is **off**.

Description

This command is used to manage jog edges during mask process correction, enabling full movement for small edges during correction to improve convergence.

background (for denseopc_options block)

[MPC denseopc_options Commands](#)

Defines background imaging characteristics.

Usage

background {dark | clear}...

Arguments

- **dark**

A literal argument designating dark field imaging. Transmission added to mask will be $\{\text{layertrans} + 0\}$ for this layer.

- **clear**

A literal argument designating clear field imaging. Transmission added to mask will be $\{\text{layertrans} + 1\}$ for this layer.

Description

A **required** configuration command describing the imaging background.

You must specify one background type per mask exposure. You must always define background opposite to the layer (for instance, layer set to clear and background set to dark), otherwise, an error will result.

Note

 This command is required for Calibre nmMPC and Calibre MPCVerify for syntax conformance only. You can specify any background, clear or dark. The drawn shape is exposed in either case. However, background and layer must always be opposites (for example, background set to dark and layer set to clear), otherwise the tool will issue a syntax error. Because drawn shapes are always exposed, always specify background as dark.

Examples

```
background dark
layer VIA opc clear
```

check_movement

[MPC denseopc_options Commands](#)

Activates or deactivates MPC edge movement constraints.

Usage

check_movement {on | off}

Arguments

- **on|off**

If check_movement is set to on, MPC mask constraints are checked. If set to off, the constraints are not checked. The default is on.

Description

An optional command that specifies whether or not to check MPC mask constraints. Turning this argument off will result in mask constraints not being checked. This command is particularly useful for debugging constraints.

convex_concave_adj_len

MPC denseopc_options Commands

Defines the adjacent length of a fragment being processed by the SEM Box method of correction.

Usage

convex_concave_adj_len *val*

Arguments

- ***val***

A required argument that specifies for the adjacent length of a fragment with both concave or convex corners, that the SEM Box method correction is applied to the fragment only if the adjacent fragment length are less than or equal to ***val***; otherwise the standard MPC correction is applied. The default value is equal to (3 * smallest e-beam kernel size) if the e-beam model is present; otherwise it is set to 0.

Description

The SEM Box method of correction, enabled by the [convex_concave_correction](#) command, typically allows **any** fragment with both corners (either concave or convex) to perform SEM Box correction. If you have a layout that contains a large number fragments with short adjacent fragment length (for example, a high volume of jogs on a metal layer), the SEM Box process could consume far more run time in unnecessarily processing each of those fragments. Using this command allows you to specify and filter these types of fragments out of SEM Box processing to save run time.

This command is valid only if the [convex_concave_correction](#) command is enabled. For complete information on SEM Box correction mode, refer to “[SEM Box Correction](#)” on page 66.

Examples

```
convex_concave_correction on // Turn on SEM Box correction
convex_concave_adj_len 0.01 // Filter jogs from SEM Box method.
convex_concave_metric 0.5 // Measure EPE within 50% of the fragment
convex_concave_limit 0.1 // Limit the fragment
                           // length to less than 100nm
convex_concave_count 3 // Select 3 points for EPE measurement
                        // and average them as effective epe for MPC
                        // correction
```

convex_concave_correction

Enables special MPC correction for convex and concave corner fragments.

Usage

convex_concave_correction {off | on}

Arguments

- **off | on**

A required argument pair that, if set to on, enables the SEM type averaging of EPE measurements. The default setting is off.

Description

This command enables a special MPC correction mode for convex and concave corner fragments. This mode enables several other corner-based commands such as:

- **convex_concave_limit** — Specifies a size limit for convex and concave fragments considered for correction.
- **convex_concave_adj_len** — Defines the adjacent length of a fragment being processed by the SEM Box method of correction.
- **convex_concave_metric** — Specifies the percentage of a fragment to be considered for placing measurement points for correction.
- **convex_concave_count** — Specifies the number of measurement points to be placed on convex and concave fragments for correction.

The convex_concave_correction block commands enable you to measure the same 2D CD on a post-MPC structure (such as small contacts or line-end tip-to-tip measurement types) as you would measure on a pre-MPC structure (the calibration mask).

The special correction mode is called SEM Box correction. For complete information, refer to “[SEM Box Correction](#)” on page 66.

Examples

```

convex_concave_correction on    // Turn on SEM Box correction
convex_concave_adj_len 0.01   // Filter jogs from SEM Box method.
convex_concave_metric 0.5     // Measure EPE within 50% of the fragment
convex_concave_limit 0.1      // Limit the fragment
                                // length to less than 100nm
convex_concave_count 3        // Select 3 points for EPE measurement
                                // and average them as effective epe for MPC
                                // correction

```

convex_concave_count

[MPC denseopc_options Commands](#)

Specifies the number of measurement points to be placed on convex and concave fragments for correction.

Usage

convex_concave_count *val*

Arguments

- ***val***

A required argument that specifies the number of measurement points to be placed on the convex and concave fragments. Only odd integers are allowed with this option. The default value is 3. The upper limit for this option is 21.

Description

This command specifies the number of measurement points to be placed on convex and concave fragments during the special correction mode enabled by [convex_concave_correction](#). A single measurement point is placed at the center of the fragment and any additional measurement points are placed symmetrically around the center of the fragment. This command is only valid if [convex_concave_correction](#) is enabled (set to on).

For complete information on SEM Box correction mode, refer to “[SEM Box Correction](#)” on page 66.

Examples

```
convex_concave_correction on // Turn on SEM Box correction
convex_concave_metric 0.5 // Measure EPE within 50% of the fragment
convex_concave_limit 0.1 // Limit the fragment length to less than 100nm
convex_concave_count 3 // Select 3 points for EPE measurement
                           // and average them as effective epe for MPC
                           // correction
```

convex_concave_limit

MPC denseopc_options Commands

Specifies a size limit for convex and concave fragments considered for correction.

Usage

convex_concave_limit *val*

Arguments

- *val*

A required argument that specifies that convex and concave fragments shorter than or equal to this value are considered for special treatment in correction. The default value is 0.

Description

This command specifies a value that identifies fragments for special treatment through the [convex_concave_correction](#) mode, if the convex or concave fragment is shorter than or equal to the specified value. This command is not valid unless [convex_concave_correction](#) is enabled, and it required when enabling SEM box correction mode. An error is generated if this command is not set explicitly.

For complete information on SEM Box correction mode, refer to “[SEM Box Correction](#)” on page 66.

Examples

```
convex_concave_correction on // Turn on SEM Box correction
convex_concave_metric 0.5 // Measure EPE within 50% of the fragment
convex_concave_limit 0.1 // Limit the fragment length to less than 100nm
convex_concave_count 3 // Select 3 points for EPE measurement
// and average them as effective epe for MPC
// correction
```

convex_concave_metric

[MPC denseopc_options Commands](#)

Specifies the percentage of a fragment to be considered for placing measurement points for correction.

Usage

convex_concave_metric *val*

Arguments

- ***val***

A required argument that specifies the percentage of the fragment considered for placing measurement points for correction. For example, a value of 0.5 means 50% of the middle of the fragment is taken for placing the measurement points. The valid range for this option is (0, 1.0).

Description

This command specifies the percentage of a fragment to be considered for placing measurement points for correction. This command is valid only if [convex_concave_correction](#) is enabled (set to on). The first measurement point is placed at the middle of a fragment, and the percentage of the fragment is based from that point.

For complete information on SEM Box correction mode, refer to “[SEM Box Correction](#)” on page 66.

Examples

```
convex_concave_correction on // Turn on SEM Box correction
convex_concave_metric 0.5 // Measure EPE within 50% of the fragment
convex_concave_limit 0.1 // Limit the fragment to less than 100nm
convex_concave_count 3 // Select 3 points for EPE measurement
                           // and average them as effective epe for MPC
                           // correction
```

corner_control

MPC denseopc_options Commands

Sets the length of the first fragment next to corners.

Usage

```
corner_control {corner_control_value | convex_value concave_value}
```

Arguments

- **corner_control_value**

Specifies the length of the first fragment next to any corner (concave and convex). The default is 0. If only one value is specified, then it is assumed to be for both.

- **convex_value**

Specifies the length of the first fragment next to a convex corner. This is declared as a pair with *concave_value*.

- **concave_value**

Specifies the length of the first fragment next to a concave corner. This is declared as a pair with *convex_value*.

Description

This optional command controls the length of the first fragment next to corners. The corner_control value can be set in a range from 0 to 1. Setting corner_control to 1 causes the longest first fragment and setting it to 0 causes the shortest first fragment.

Fragment lengths are based on the Nyquist value. The optional command **scale** is used to determine the final unit of fragmentation (the value is referred to as “frgu”). The value of frgu is (scale * Nyquist). The default value for scale is 1.0.

Examples

```
corner_control 0.8
```

epe_spacing

[MPC denseopc_options Commands](#)

Sets spacing between EPE or image samples along fragments.

Usage

epe_spacing *value*

Arguments

- *value*

Specifies a value for spacing between EPE or image samples along fragments. The default setting is 0.04.

Description

This command sets a value for the spacing between EPE or image samples along fragments.

Examples

```
epe_spacing 0.03
```

feedback

MPC denseopc_options Commands

Automatic convergence controller command.

Usage

feedback *feedback_val*

Arguments

- ***feedback_val***

Specifies the desired default feedback value. The default is -0.4. It is recommended that you set this close or at -1.0. The allowed range for feedback is -2 to 0. For Calibre nmMPC and Calibre MPCVerify, a feedback value of 1 is also allowed for EPE based verification layer generation.

Description

An optional command that defines the default feedback values to be used per iteration.

Edge movement is set by (EPE * feedback).

You can explicitly increase or decrease the feedback of each fragment to adjust edge movement. Supplying *n* values will set feedback for the first *n* iterations.

Examples

```
feedback -0.95 -0.99 -1.0
```

fragment_coincident

[MPC denseopc_options Commands](#)

Copies fragmentation from one layer onto another where edges are coincident.

Usage

fragment_coincident *layer1* -overlay 0 | 1

Arguments

- *layer1*

The layer name with which edges must be coincident in order to be fragmented. Layer1 must be of type **opc**, **correction**, **island**, or **visible**.

There is no default value for this keyword.

- **-overlay 0 | 1**

When this option is set to 1, the fragmentation points from the specified *layer1* are overlaid onto the current layer, keeping those points from *layer1*. The coincident edge copies the fragmentation, and the non-coincident edges are still fragmented.

When this option is set to 0, the coincident edges are active for fragmentation (not a copy), and the non-coincident edges do not get fragmented.

This option is used to align fragmentation between 2 layers. The default is 0.

Description

An optional user control for interfeature and intrafeature fragmentation. This keyword can only be used within a [fragment_layer](#) block. Edges on the layer to which the [fragment_layer](#) block applies will be fragmented if and only if they are coincident with the layer specified by this keyword.

This command is typically used to make sure that correction layer fragments align with opc layer fragments.

The fragment_coincident enforces [fragment_min](#), which means that each break on a coincident layer is checked to see if the break will violate [fragment_min](#) on the fragment layer. If inserting that break would violate [fragment_min](#), it is not inserted.

Additionally, fragment_coincident supports directional fragmentation. This means that if horizontal is specified as a direction for fragmentation, only fragments in the horizontal direction will be broken.

The overall result is that a fragment break cannot violate [fragment_min](#) and must be in the correct direction or it will not be added.

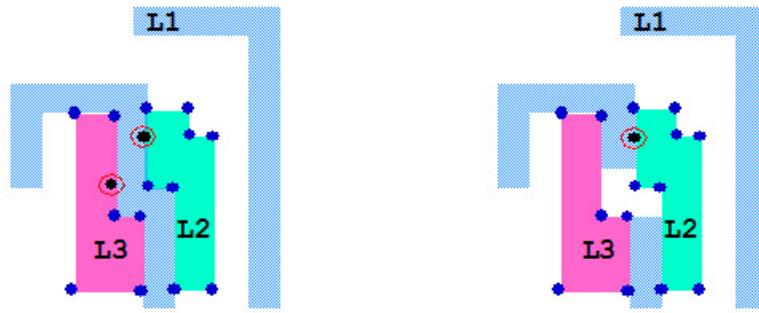
Examples

The following example shows fragmentation specified for edges coincident with layer L1. (both examples use the same setup file).

```
fragmentLayer L2 {  
    fragment_coincident L1  
}  
fragmentLayer L3 {  
    fragment_coincident L1  
}
```

The following figure shows the results (the highlighted dots represent interfeature vertices created on the coincident edges).

Figure 5-21. Changing the Coincident Layer to Affect Fragmentation



fragment_corner

[MPC denseopc_options Commands](#)

Fragmentation control: Used for fragmentation at corners.

Usage

```
fragment_corner edge_spec [fragment] rule [corner2 [fragment] rule]  
    [{breakinhalf | alwaysbreakinhalf} [enforce]]
```

where:

```
edge_spec = {corner_type [corner_subtype] [corner_type_next [corner_subtype]]}  
    [corner_type [corner_subtype] [corner_type_next [corner_subtype]]] [length constraint]  
    [length1 constraint] [length1_next constraint]  
    [length2 constraint] [length2_next constraint]  
    [jog_tol distance]
```

```
corner_type = {concave | convex}
```

```
corner_subtype = {end_adjacent | non_end_adjacent | angle_90 | angle_non90}
```

```
rule = [split n [force]] |  
    [len1...lenN priority rem r] [applyratio] [allowtrim amount]  
    [repeat length max_length force]
```

Arguments

Top Level Arguments

- fragment

An optional keyword that can be used to separate the fragmentation rule from the edge specification. You can also use this option with the length1/length2 keywords so that the length value is separated from the fragmentation values.

- corner2

An optional keyword that can only be used if there are two corners defined, and the definitions are unambiguously different. (For example, concave convex or concave angle_90 concave angle_non90.)

If corner2 is specified, it must be followed by a fragmentation rule for the second endpoint.

In cases where a single-corner rule and a two-corner rule both apply to a corner, the two-corner rule is used.

- breakinhalf [enforce]

An optional keyword that splits the middle fragment of edges in half when ripples are specified (*len1...lenN* or repeat *length*) but cannot be placed due to constraints. The halves of the split fragment must be greater than fragment_min.

Notice that by default, *breakinhalf* does not enforce the rem value of the rule. Instead, it ignores the rem value and enforces the fragment_min value for the layer. If you need to have the rem value enforced over the fragment_min value, use the enforce option.

If enforce is not specified, splitting only occurs if it does not cause the remaining fragments to be smaller than the fragment_layer's fragment_min value. See “[Example 2: breakinhalf](#)” and [Example 3: breakinhalf enforce](#).

The enforce option may only be specified immediately following breakinhalf.

***edge_spec* Arguments**

- ***corner_type***

*corner_type*_next

A required argument indicating whether a corner is concave or convex. At least one corner must be defined in the *edge_spec*. You can specify up to four corners using the _next suffix, for example:

```
fragment_corner convex concave_next convex concave_next ...
```

(The rule would apply to the edge between two convex corners that both have concave corners as their other neighbors. See “[Example 6: Using corner_next](#)” on page 195.)

The concave_next and convex_next keywords can only be specified after a ***corner_type*** [*corner_subtype*].

Note



If concave_next or convex_next is used, the corresponding length1/length2 values must be specified to prevent differences between flat and hierarchical runs. (The length1 and length2 values should provide an upper bound.)

For example,

```
fragment_corner convex convex_next fragment 0.1
```

produces the warning message

```
fragment_corner length1 must be specified and < value when a
corner1_next corner or length1_next is specified in order to limit
fragmentation inconsistencies.
```

- ***corner_subtype***

An optional argument further differentiating the corner. You can specify one per ***corner_type***. The same subtypes are used for both convex and concave. Valid values are the following:

end_adjacent — the corner satisfies the line-end or space-end adjacent criteria. (See line end fragment and space-end fragment in the glossary for line-end and space-end criteria.)

non_end_adjacent — the corner is not end-adjacent.

angle_90 — the corner is a 90 or 270 degree angle.

angle_non90 — the corner is not a right angle.

- length *constraint*

An optional argument specifying the length of the edge. The constraint takes the form of a DRC constraint and the value is specified in microns. For example:

```
length > 0.1 <= 0.2
length > 0.2
length == 0.3
```

- [length1 *constraint* [length1_next *constraint*]]
[length2 *constraint* [length2_next *constraint*]]

Optional keywords that specify the length of the edges at corner 1 and corner 2, and the lengths of the edges on either side of length1 and length2, respectively as illustrated in [Figure 5-23](#) on page 196. The constraint takes the form of a DRC constraint and the value is specified in microns.

Both length1 and length2 may be specified without the other. For example:

```
length1 <= 0.15
length1 > 0.2 length2 > 0.2
length2 < 0.24
```

When the rule includes a “corner corner_next”, the corresponding length1 or length2 must be specified, but length1_next and length2_next are completely optional. The length1_next and length2_next can only be specified after a length1 or length2.

- jog_tol *distance*

An optional argument that causes jogs less than or equal to *distance* to be ignored for length1, length1_next, length2, and length2_next. The *distance* value is in microns.

Note

 Jogs on the primary edge, measured by “length *constraint*”, are *not* ignored. The jog_tol option only ignores jogs on neighboring edges.

If jog_tol is not specified, jogs are handled like a corner.

rule Arguments

- split *n*

An optional argument specifying to break the fragment into *n* equal parts. For example, if the following command is applied to a 0.4 micron edge, the edge is split into four 0.1 micron fragments:

```
fragment_corner convex length <= 0.685 split 4
```

The *n* value must be an integer between 2 and 40, inclusive. If the resulting fragments violate fragment_min, the command generates an error. The preceding example would

likely generate an error because lengths of 0 would be split into four fragments. You can prevent this by providing a lower bound:

```
fragment_min 0.050
fragment_corner convex length >= 0.200 <= 0.685 split 4
```

The split *n* argument cannot be specified with any of the other **rule** arguments except force.

- force

An optional keyword for use with split. It overrides the fragment_min error. Use only when you are certain that violating fragment_min will not cause an error.

- *len1*

The distance in microns from a corner to the first fragmentation vertex that will be produced. The distance must be greater than fragment_min or the run halts with the following message:

```
intrafeature fragment is less than MinEdgeLength (value = len)
```

- *lenN*

The distance in microns from the previous fragmentation vertex to the next fragmentation vertex, measured sequentially from the corner inward.

- priority

An optional flag for two-corner rules indicating which corner has priority over the other. Typically, there is no fragmentation if the remainder (rem *r*) is violated after the first fragment at each corner is created. With the priority option, the fragment at the priority corner is created if the remainder (rem *r*) is not violated.

Intrafeature fragmentation selects an edge and attempts to create ripples on it, working inward from the corners. After the first ripple is created at each corner, the priority flag will no longer have any effect. Afterward, as ripples are created, intrafeature fragmentation will continue working inward toward the center of the edge, creating ripples. It continues until no ripples can be inserted without violating rem or fragment_min. It does this symmetrically from both corners, attempting to insert a ripple from each corner simultaneously in order to retain symmetry.

In summary, the priority flag is a “special case” handler for extremely short edges where it is not possible to insert even one ripple from each corner. If only one ripple from one corner can be inserted, the priority flag will allow you to choose which one. It is ignored outside of that case.

There are cases where using the priority argument in corner fragmentation can result in a “remainder violation” at non-priority corners. When priority is used, an edge is fragmented based on priority of corner A if the remainder length after *A_fragment_1st* is larger than rem_A or has satisfied the following condition:

$$A_fragment_2nd + \dots + A_fragment_nth + rem_A \leq remainder_length < B_fragment_1st + rem_B$$

This might result in a length violation of the remainder at non-priority corners if the length setting of *rem_B* is larger than for *rem_A*, given the priority keyword.

For example, you have an MPC recipe with the following corner fragment setting that applies to an edge length of 0.180 with convex and concave corners:

```
fragment_corner convex 0.065 0.060 rem 0.050 priority
fragment_corner concave 0.080 0.070 0.060 rem 0.060
```

The edge fragment result is based on the convex corner only with “0.065 0.060 0.055”. The remainder of the concave corner is only 0.055, which already violates the rem setting of 0.060 for concave corners.

To avoid remainder violations, set the same remainder (*rem r*) for both convex and concave corners for edge length less than (*A_fragment_1st + B_fragment_1st + rem*) before further fragmentation of convex and concave corners.

- **rem r**

An optional argument specifying the minimum allowed remainder. If fragmentation produces a fragment with length smaller than *r*, then it will not fragment the edge. The default value is *fragment_min*.

- **applyratio**

An optional argument specifying that an attempt should be made to split the edge in two and that the ratio of lengths for each corner should be maintained if possible. The fragments are lengthened and shortened as necessary but not allowed to violate *fragment_min*. If the edge is 2**fragment_min*, both fragments are kept at the same (minimal) length, despite the ratio.

- **allowtrim amount**

An optional argument specifying an amount of length reduction that can be done to the corner fragment in order to make room for a third fragment of minimum length.

- **repeat length [max_length] [force]**

An optional argument specifying a target size for repeated fragments in the remainder. The remainder is divided into fragments of at least this length (possibly longer). This is typically not allowed unless the rule has a length constraint with an upper bound.

max_length — An optional argument in microns specifying the maximum length to which the fragments can be adjusted. The remainder is split into equal-length fragments of a size between *length* and *max_length*.

force — An optional keyword that causes fragments to be exactly *length*.

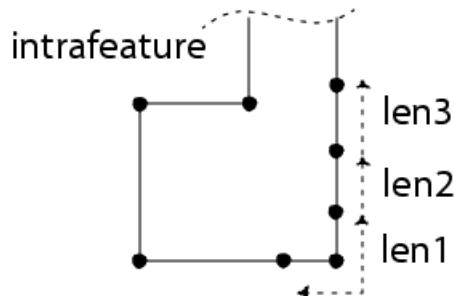
If force is specified and allowtrim is present, an attempt is made to trim the ends in order to allow the repeated fragments to have exactly the specified length. If possible, the end fragments are trimmed in an attempt to avoid creating fragments longer than this length.

Description

This keyword is an optional fragmentation parameter. This keyword gives you precise control over the number and length of intrafeature fragmented edges at concave or convex corners. The fragmentation process occurs automatically before MPC begins.

In intrafeature fragmentation, the edges connected to corners are broken up according to fragment_corner parameters. The following figure shows how the intrafeature fragmentation parameters are applied to produce fragmentation vertices.

Figure 5-22. Intrafeature Fragmentation Example



The end_adjacent subtype applies to both line-end adjacent and space-end adjacent corners.

Note About Promotion Radius in Hierarchical Mode

When an edge is longer than the promotion radius, it may cross a hierarchy boundary. When this occurs the type of corner for both corners cannot be reliably determined.

The workaround is when fragmenting long edges is to specify fragment_corner twice, one with an upper bound of the promotion radius and the other with it as a lower bound. See “[Example 5: Handling Long Edges](#)” on page 195.

Examples

Example 1: Declaration Order

The declaration order has an impact on the results. Each intrafeature command affects all the edges which meet its constraints. Therefore, more specific commands should be placed ahead of general commands. The one exception is when “fragment_layer_order full” is set. (See “Examples” in fragment_layer_order.)

For example, to specify different fragmentation parameters for short edges with 1 convex and 1 concave corner:

Incorrect Usage:

```
fragment_min 0.03
fragment_max 0.4
fragment_corner convex 0.035 0.05
fragment_corner concave 0.055
fragment_corner convex not_end_adjacent concave not_end_adjacent \
    length > 0.13 <= 0.15 0.040 corner2 0.05
```

The last command would not work since any corner it would affect would have already been fragmented by one of the first 2 fragment_corner commands.

Correct Usage:

```
fragment_min 0.03
fragment_max 0.4
fragment_corner convex not_end_adjacent concave not_end_adjacent \
    length > 0.13 <= 0.15 0.040 corner2 0.05
fragment_corner convex 0.035 0.05
fragment_corner concave 0.055
```

Example 2: breakinhalf

By default, breakinhalf respects the fragment_min setting. Consider a 0.12 micron long fragment with a corner at one end.

```
fragment_min 0.04
fragment_corner concave 0.05 0.05
```

If a rule that specifies two ripples of 0.05 with a fragment_min of 0.04 is applied, the fragment would not be changed because the ripples would leave a fragment of 0.02. If “breakinhalf” is added to the rule, the fragment is broken into two fragments of 0.06 each.

Example 3: breakinhalf enforce

Consider a long edge with

- a 50-nm center fragment
- a 25-nm fragment_min
- a 35-nm remainder (rem 0.035)

```
fragment_corner ... rem 0.035 breakinhalf
```

The 50-nm center fragment would still be broken to two 25-nm parts. This is because fragment_min is used to determine the minimum fragment length rather than the rem value. To enforce the rem value, use enforce:

```
fragment_corner ... rem 0.035 breakinhalf enforce
```

If “breakinhalf enforce” is used, the center fragment is not broken, due to the two 25 nm fragments created being less than the rem value, which would not be allowed.

Example 4: Controlling the Size of Fragments in a Central Remainder

You can combine the repeat and allowtrim arguments to create a fixed size for the middle fragment of an edge whose length is suitable for three fragments. It requires setting the corner fragments (*len1*) to the maximum desirable length, with the allowtrim *amount* set to the difference between the maximum and minimum fragment length.

For example, consider an edge of 0.2 microns, a grid size of 1 nm, and the following code:

```
fragment_corner concave not_end_adjacent concave not_end_adjacent \
    length >= 0.100 < 0.500 \
    fragment 0.048 rem 0.033 allowtrim .016 repeat 0.034 0.038
```

This sets the maximum length of the first fragment to 48 nm, but because of “allowtrim 0.016” the first fragments could be adjusted to as small as 32 nm if desirable. The initial pass leaves a center fragment of 104 nm. The “repeat 0.034 0.038” indicates the central fragment should be split into equal-length fragments from 34 to 38 nm long.

Three 34-nm fragments would consume 102 nm of the 104 nm. To keep the repeated fragments equal, the central fragment needs to be increased in multiples of 3 nm. With allowtrim, the final edge fragmentation is initial fragments reduced to 45 nm, and the central fragment further split into three 36 nm pieces.

Example 5: Handling Long Edges

Fragment_corner rules that operate on edges longer than the promotion radius can cause errors when the corners cannot “see” each other. For example, consider a promotion radius of 0.9 and this rule:

```
#This is a bad rule:
fragment_corner concave concave length > 0.8 fragment 0.15 0.15
```

Edges longer than 0.9 microns cause an error because the rule requires checking both corners. The safe way to handle these edges is to use a two-corner rule for edges less than the promotion radius, and a one-corner rule for edges longer than that, like so:

```
#This works better:
fragment_corner concave concave length >0.8 <=0.9 fragment 0.15 0.15
fragment_corner concave           length >0.9      fragment 0.15 0.15
```

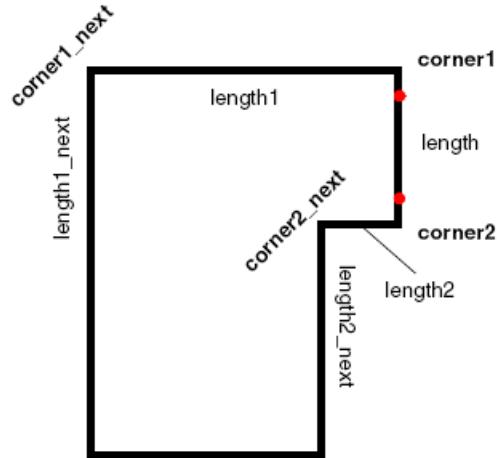
This division permits correct fragmentation despite hierarchy boundaries, and will generate the same results in hierarchical and flat runs.

Example 6: Using corner_next

The length options and corner options can be combined to allow highly specific edge selection without accidentally selecting other edges or using the tagging commands. For the shape shown,

the following code produces fragmentation where indicated by the red dots on the rightmost edge:

Figure 5-23. corner_next and length_next



```
fragment_corner convex convex_next convex concave_next length == 0.2 \
    length1 == 0.4 length1_next == 0.5 length2 == 0.1 length2_next == 0.3 \
    fragment 0.025
```

fragment_flaglayer

MPC denseopc_options Commands

Fragmentation control: Controls the fragmentation operation for a specified layer.

Usage

```
fragment_flaglayer [layerName] [flagName] {enable | disable}
```

Arguments

- *layerName*

Specifies the name of the layer.

- *flagName*

Specifies the name of the flag to change. Available flag names are described in the following table:

Table 5-8. Available Flag Names

Flag Name	Description
intrafeature_smalljogs	Do intrafeature fragmentation at small jogs
intrafeature_convex	Do intrafeature fragmentation at convex corners
intrafeature_concave	Do intrafeature fragmentation at concave corners
intrafeature_lineend	Do intrafeature fragmentation at line ends
interfeature_smalljogs	Do interfeature fragmentation from small jogs
intrafeature_false	Do intrafeature fragmentation on false edges
intersection_visible	Break layers crossed by this visible layer

- {**enable** | **disable**}

Indicates whether feature should be activated or deactivated.

Description

This command controls the fragmentation operation for layer *layerName*. This interface is designed to create a more descriptive control for the fragmentation operations. For example:

```
fragment_flaglayer TARGET intrafeature_smalljogs enable
```

This causes intrafeature fragmentation at small jogs to be enabled for the layer named TARGET.

fragment_inter

[MPC denseopc_options Commands](#)

Fragmentation control: Used to define parameters for interfeature fragmentation.

Usage

```
fragment_inter [-adjust] [-away] [-combined] [-conflictPriority type...] [-cornerDist w]
[-distancePriority {0 | 1}] [-externalOnly] [-internalOnly] [-interdistance y]
[-minshield distance] [-rem r] [-remext r]
[-ripples x | -num x] [-ripplelen z]... [-ripplestyle {0 | 1 | 2}]
[-shield n] [-shift z] [-skipCorners] [-split {0 | 1}] [-sym] [-symbisect]
```

Arguments

- **-adjust**

An optional keyword that adjusts the insertion point to avoid a minedgelength violation. Sometimes an insertion point is too close to a corner (or fragment break) to be inserted and prevents fragmentation. This results in no fragmentation occurring at all. Specifying -adjust allows the initial insertion point to be shifted to avoid a minedgelength violation and enables fragmentation. If shifting the initial insertion point up to minedgelength is still not enough to insert the point, no fragmentation occurs.

- **-away**

An optional argument that prevents corners from generating fragments across the interior of the polygon. This may be useful when a highly specific value of shielding is in use. The default is to allow interfeature fragmentation to generate fragmentation in all directions.

- **-combined**

An optional argument that causes fragment_inter to run concurrently on all layers listed in [fragment_interlayers](#). Normally, layers are processed one at a time, which gives points inserted by earlier layers priority over points inserted by later layers. If the layers are processed concurrently, then -conflictPriority considers the points from all layers during resolution. See “[Example 3 - Managing Conflict Between Multiple Layers](#)” on page 210.

- **-conflictPriority *type***

An optional argument that specifies how conflicts should be resolved. Conflicts can occur when two interfeature breakpoints are so close together that they violate fragment_min. Conflicts are resolved by comparing the two points and allowing the higher priority one to be inserted on the edge.

Specify any of the following keywords to build a priority list. Each *type* should be separated by a space.

cornerdist — The distance from either corner on the edge receiving fragmentation.

Points toward the middle of the edge are used in preference to points towards the ends.

distance — The distance between the corner and the edge receiving fragmentation.

Corners closer to the edge are used in preference to corners farther away.

projection — The projection of the corner’s edge to the edge receiving fragmentation.

Larger projection (that is, a longer common run) is used in preference to smaller projection.

pt1 — The distance from point 1 on the edge being broken. (Polygon edges are traversed such that the inside of the polygon is to the right. Point 1 is the first endpoint of the edge.) Corners closer to point 1 are used in preference to corners farther away.

shielding — The number of edges between the corner and the edge receiving fragmentation. Smaller shield counts are preferred over larger shield counts.

size — The width of the corner’s edge. Corners with longer edges are used in preference to narrower shapes.

The default priority order is shielding, distance, size, projection, cornerdist, pt1, and geometric. Types not specified are added to the end of the list of specified types in the order shown. For example, “-conflictPriority size projection” is treated as “-conflictPriority size projection shielding distance cornerdist pt1 geometric”.

- -cornerDist *w*

An optional argument that indicates the minimum distance at which an interfeature fragmentation point can be inserted from a corner. If the point is less than the distance *w* from a corner, it will not be inserted. If it is equal to or greater than distance *w*, it will be inserted. Priority is established by rules described in the section “[Fragmentation Point Priority](#)” on page 207.

In general, use -cornerDist if performing interfeature fragmentation before intrafeature, or when there might be multiple fragments within an area you do not want further fragmented. Use -skipCorners if intrafeature fragmentation runs first, as it ignores any fragments touching either convex or concave corners.

- -distancePriority {0 | 1}

When this value is set to 1, then the priority insertion of interfeature fragmentation points is decided by the distance from the projection point. When -distancePriority 1 is set, the closer of the two projection points “wins.” When two projection points have the same distance, the one with a smaller shielding count “wins.”

With -distancePriority 0, the “winner” depends on the order of processing and cannot be determined ahead of time. The default is 1.

- -externalOnly

Corners will only generate interfeature fragments if they are outside the polygon. Corners that are across the polygon from an edge will not generate fragmentation. (This is the opposite of the internalOnly option.) Note that while the “-shield” option is fixed and therefore not available for “internalOnly”, it is available for externalOnly. This is very useful for controlling fragmentation from the externalOnly option, which can cause fragmentation to occur in correct, yet unexpected, locations.

- -interdistance *y*

An optional argument specifying the maximum distance at which interfeature fragmentation will occur. This parameter is specified in microns. The default value for -interdistance is $1.5 * (\lambda / NA)$.

- -internalOnly

Corners will only generate interfeature fragments if they are within the polygon. Corners from one polygon will not generate interfeature fragmentation on another polygon. Also, corners that are separated from a polygon by a space will not generate any fragment breaks.

Note

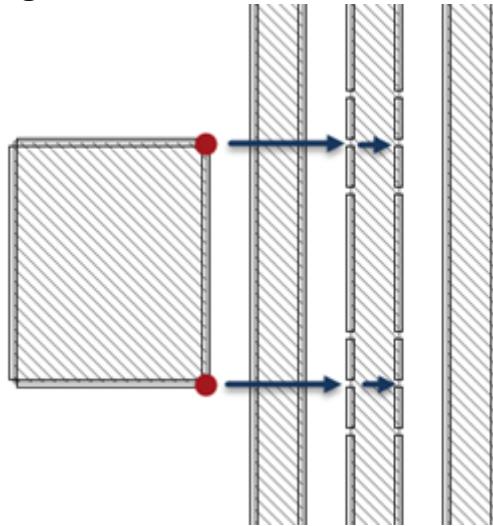
 The shield option defaults to 1 for internalOnly, and no other value may be specified.

- -minshield *distance*

An optional argument that specifies the minimum shielding value that must be reached before fragmentation occurs. The *distance* must be less than the -shield *n* value. The default *distance* is 0.

For example, a value of -minshield 2 -shield 4 means that fragmentation (assuming sufficiently large -interdistance) propagates on the third and fourth edges from a corner. The nearest and next nearest neighboring edges do not receive a fragmentation point.

Figure 5-24. fragment_inter -minshield 2 -shield 4 -ripples 1



If -shield and -minshield are set to the same value it is an error.

- -rem *r*

An optional argument specifying the minimum allowed remainder. If the interfeature fragmentation would produce an edge with length smaller than *r*, then that fragmentation will not occur. If not specified, the default value used is fragment_min.

- **-remext *r***

An optional argument that specifies the minimum remaining distance between an interfeature fragmentation point and any external (pre-existing) fragmentation points. The distance must be greater than or equal to minedgelength. This argument is similar to -rem but applies only to this one case.

The -remext argument cannot be specified with -ripplestyle 0. It generates a setup file error.

The default is the value of -rem *r*. If -rem is not specified, the default is minedgelength.

- **-ripplelen *z***

An optional argument specifying the length of a single ripple edge in microns. The default for ripplelen is [fragment_min](#) in microns. It is possible to specify the length of each ripple created by interfeature fragmentation. Up to 8 ripples can be listed.

- **[-ripples *x* | -num *x*]**

An optional argument specifying the number of ripples. The default value for -ripples or -num is $0.5 * (\text{lambda/NA}) / \text{ripplenlen}$, rounded to the nearest whole number. The maximum number of ripples you can create is ten (10).

- **-ripplestyle {0 | 1 | 2}**

An optional argument used to resolve conflicts regarding ripple generation. (You can also specify your own solution with the -conflictPriority *type* argument.) The default is 1, but changing it to 2 is recommended.

It is an error to specify -ripplestyle 0 with -sym or -symbisect.

Choose one of the following:

0 — All conflicts between fragmentation points are resolved according to the -distancePriority argument and ripples will not be generated by the interfeature fragmentation points that could not be created due to the -rem constraint.

If 0 is specified, the statement cannot also use -remext.

1 — This option allows for ripples to be generated not only by successfully-inserted interfeature fragmentation points, but also by those interfeature fragmentation points that could not be inserted due to -rem constraints, as shown in [Figure 5-25](#).

The following rules are used to uniquely resolve -rem conflicts between fragmentation points:

- Intrafeature fragmentation points and their ripples have the highest priority.
- Interfeature fragmentation points have priority over ripples from all other interfeature fragmentation points. The priority of an interfeature point is determined (in order of importance) by:
 - The distance between the fragmentation point and the fragmentation-causing corner.
 - The number of shielding edges between the fragmentation point and the fragmentation causing corner.

- The distance from the fragmentation point to the middle of the fragmented edge.
- The distance from the fragmentation point and the first point of the fragmented edge.

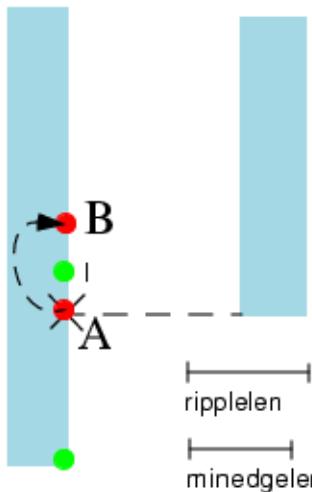
Thus, if the distance between two interfeature fragmentation points is less than the -rem argument, the fragmentation point that is closer to the corner that caused the fragmentation will be inserted. If the distances between the fragmentation points and the corresponding fragmentation causing the corners are equal, the fragmentation point with a smaller number of shielding edges will be inserted.

- The priority of ripples from interfeature fragmentation points is determined (in order of importance) by:
 - The distance from a ripple to the interfeature fragmentation point that the ripple originated from.
 - The priority of the fragmentation point that the ripple originated from based on the originating point's distance to the fragmenting corner, number of shielding edges, and whether the ripple originating point was inserted or not.

Ripples may be generated by the fragmentation points that could not be created due to the -rem constraint.

Multiple ripples can have the same priority under the rules for ripples defined previously. The conflicts between ripple points with the same priority that do not have a conflict with any higher priority points are resolved as follows:

- Ripple points of the same priority are divided into sets such that there are no -rem conflicts between ripple points in different sets and there are -rem conflicts between neighboring ripple points inside each set.
- Each set is processed according to the size of the set:
 - If set size is 1, the ripple point in the set is inserted.
 - If set size is 2 and -split parameter is set to 0, neither point in the set is inserted. If -split is set to 1, a new ripple point that is in the middle of the two points in the set is inserted.
 - If set size is greater than 2, the tool inserts each point in the set based on the distance from the first point in the edge.

Figure 5-25. Example -ripplestyle

In this example, fragmentation point A will not be inserted due to a fragment_min violation with intrafeature fragmentation point I.

Fragmentation point B, however, ripples out from point A and is inserted when -ripplestyle 1 is specified.

If -ripplestyle 0 is specified, then fragmentation point B will not be inserted.

`fragment_inter -ripplestyle 1 -ripples 1`

2 — This option is recommended. It is similar to 1 (the default) in that it also generates ripples from both successfully inserted interfeature fragmentation points and those that could not be inserted due to -rem constraints. The following rules are used to uniquely resolve -rem conflicts between fragmentation points.

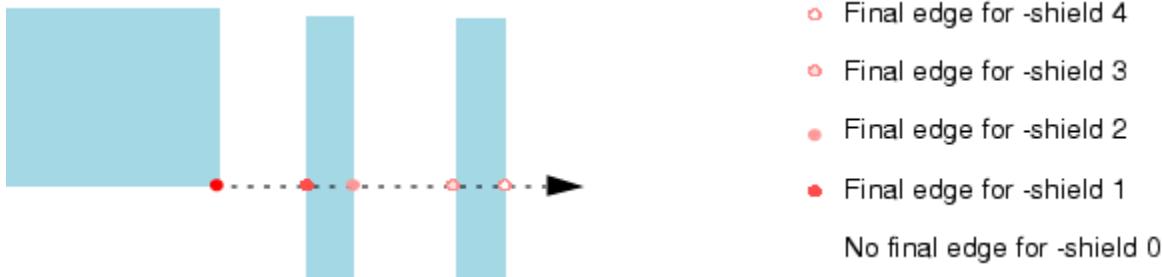
- Any preceding fragmentation caused by commands listed before fragment_inter.
- Interfeature (fragment_inter) fragmentation points have priority over ripples from all other interfeature fragmentation points. The priority of an interfeature point is, from highest to lowest,
 - The number of shielding edges between the fragmentation point and the fragmentation-causing corner
 - The distance between the fragmentation point and the fragmentation-causing corner
 - The wider of the polygons with the fragmentation-causing corner
 - The projection of the fragmentation-causing edge on to the fragmented edge
 - The distance from the fragmentation point to the middle of the fragmented edge
 - The distance from the fragmentation point to the first point of the fragmented edge
- -shield *n*

An optional argument used to ensure that interfeature fragmentation can only occur if the edge undergoing fragmentation is shielded by fewer than *n* edges. When -shield is set to a

number greater than 0, edges must meet this shielding constraint *and* the distance constraint in order to be affected by interfeature fragmentation.

An edge is “shielded” by any other edges which lie between it and the interfeature fragmentation vertex. If n is 0, then shielding is turned off. The default value for -shield is 0. [Figure 5-26](#) shows an example of shielding.

Figure 5-26. Interfeature Shielding Example



- **-shift z**

An optional argument specifying the distance that an interfeature fragmentation break should be shifted, in microns. Normally, a fragmentation break will occur on a fragment at a location that is exactly perpendicular to a corner. The shift option allows this break to be shifted away from the corner by the distance indicated by z (this is similar to the concept of opposite adjacent). A positive distance moves the fragment break point outward along the generating edge.

- **-skipCorners**

Interfeature fragmentation can interfere with intrafeature fragmentation. Fragmentation from the [fragment_corner](#) command can be segmented with subsequent interfeature commands. When the -skipCorners option is specified, interfeature projections on previously formed fragments are not imposed (see [Figure 5-29](#) on page 209). Use this option to preserve corner fragmentation.

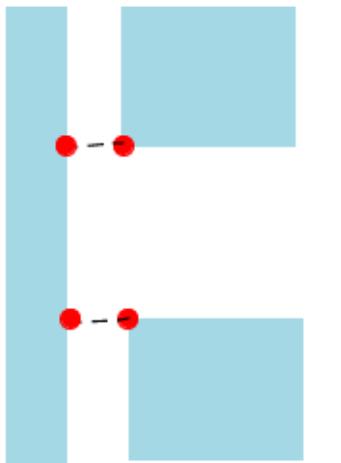
- **-split {0 | 1}**

An optional argument controlling the splitting of fragments. The following values are possible:

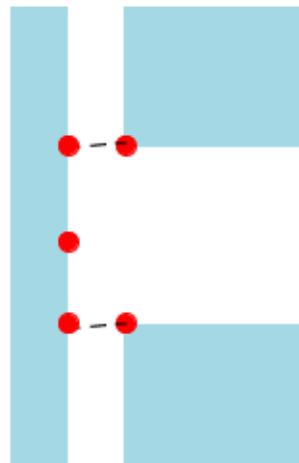
- 0 — specifies maintaining a single fragment for any fragment between interfeature projection points with length $\leq (2 * \text{ripplelen} + \text{remainder})$. Ripplestyle must be 0 for -split 0 to occur. This is the default. If ripplestyle is 1, -split 0 is not allowed.
- 1 — specifies splitting into two equal fragments any fragment between interfeature projection points with:
 - $(2 * \text{remainder}) \leq \text{length} < (2 * \text{ripplen} + \text{remainder})$ if -ripplestyle 0 is selected.
 - $(2 * \text{ripplen}) \leq \text{length} < (2 * \text{ripplen} + \text{remainder})$ if -ripplestyle 1 is selected.

Note

 If the -ripplestyle is set to 1, -split will be set to 1, overriding the specified -split user setting in the setup file.

Figure 5-27. -split Example

(A) with -split 0



(B) with -split 1

- -sym | -symbisect

Two optional arguments that handle fragmentation created by thin polygons in different ways. These solve the problem of fragmentation on an edge from the tip of a scattering bar less than fragment_min in width. The two corners both attempt to insert interfeature fragmentation points on the edge, but are blocked by being closer than the minimum fragment length.

Note

 These options only apply to Manhattan (horizontal or vertical) edges. No adjustments are made on edges that are not Manhattan.

-sym — Creates a minimum length fragment with the same center as the thin polygon.

Fragments are fragment_min in size and only occur on horizontal or vertical edges. See [Figure 5-30](#) and [Figure 5-31](#) for an example of how this option changes output.

-symbisect — Inserts a single fragmentation point instead of a minimum-length fragment. The inserted point is halfway between the two unsuccessful points. Ripples are generated from this vertex if specified.

Both options have limitations. The algorithm will only insert one of these shifted point(s) to avoid a minedgelength violation if there is no fragment_min violation on the opposite side of the point. This means that groups of points that violate fragment_min receive no corrections. Also, neither -sym nor -symbisect can be used with -ripplestyle 0.

Description

This keyword defines the fragmentation parameters that influence interfeature fragmentation.

Fragmentation is the process of breaking a layout polygon's edges (or parts of edges) into smaller segments by adding vertices. The purpose of fragmentation is to prepare the layout for correction by creating more edge fragments wherever the layout needs more correction.

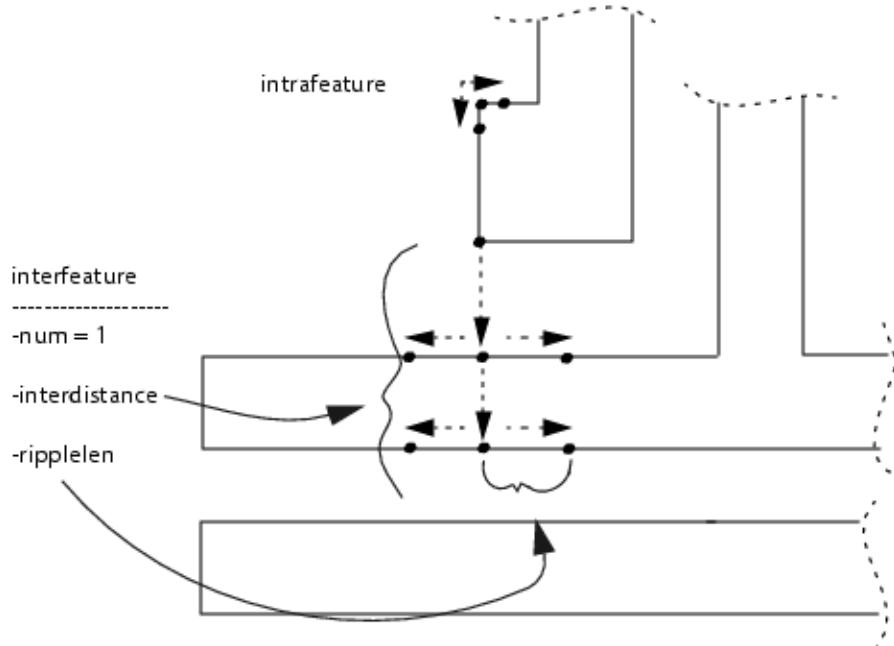
Interfeature fragmentation adds vertices based on proximity to other features. The interfeature fragmentation algorithm adds new vertices to edges that meet the following criteria:

- The distance from the edge to the corner causing the fragmentation is less than or equal to the distance specified by -interdistance.
- None of the fragments that would result from fragmentation is shorter than the distance specified by -rem.
- The edge is separated from the corner by fewer than n edges, where n is specified by -shield.

After the initial breakpoint is inserted, additional vertices are inserted on both sides of the original point with the length of the fragments formed defined by the -ripplelen parameter.

If none of the optional arguments are specified, the interfeature fragmentation algorithm calculates appropriate values. The diagram in [Figure 5-29](#) shows how the interfeature fragmentation takes place.

Figure 5-28. Using Shield With Interfeature Fragmentation



Fragmentation Point Priority

When placing fragmentation points using -cornerDist, priority is established using the following rules:

- All `fragment_corner` fragmentation points and their ripples have the highest priority.
- Interfeature fragmentation points have priority over ripples from all other interfeature fragmentation points. The priority of an interfeature point is determined (in order of importance) by:
 - The distance between the fragmentation point and the fragmentation causing corner.
 - The number of shielding edges between the fragmentation point and the fragmentation causing corner.
 - The distance from the fragmentation point to the middle of the fragmented edge.
 - The distance from the fragmentation point and the first point of the fragmented edge.

Thus, if the distance between two interfeature fragmentation points is less than the -rem argument, the fragmentation point that is closer to the corner that caused the fragmentation is inserted. If the distances between the fragmentation points and the corresponding fragmentation-causing corners are equal, the fragmentation point with a smaller number of shielding edges will be inserted.

- The priority of ripples from interfeature fragmentation points is determined (in order of importance) by:
 - The distance from a ripple to the interfeature fragmentation point that the ripple originated from.
 - The priority of the fragmentation point that the ripple originated from based on an originating point at Y's distance to the fragmenting corner, the number of shielding edges, and whether the ripple originating point was inserted or not.

Ripples may be generated by the fragmentation points that could not be created due to the -rem constraint.

Note that multiple ripples can have the same priority under the rules for ripples as defined previously. The conflicts between ripple points with the same priority that do not have a conflict with any higher priority points are resolved as follows:.

- a. First, ripple points of the same priority are divided into sets such that there are no -rem conflicts between ripple points in different sets and there are -rem conflicts between neighboring ripple points inside each set.
- b. Each set is processed according to the size of the set:
 - If the set size is 1, the ripple point in the set is inserted.

- If the set size is 2, and -split parameter is set to 0, neither point in the set is inserted. If -split is set to 1, a new ripple point that is in the middle of the two points in the set is inserted.
- If the set size is greater than 2, Calibre nmMPC attempts to insert each point in the set based on its distance from the first point in the edge. When -ripplestyle 0 is specified, all conflicts between fragmentation points are resolved according to the -distancePriority argument, and ripples will not be generated by the interfeature fragmentation points that could not be created due to the -rem constraint.

Examples

Example 1 - skipCorners and cornerDist

In this example, the fragment_corner command forms two fragments on the convex corner. The following interfeature command places two additional fragment points in between the original fragment formed with fragment_corner:

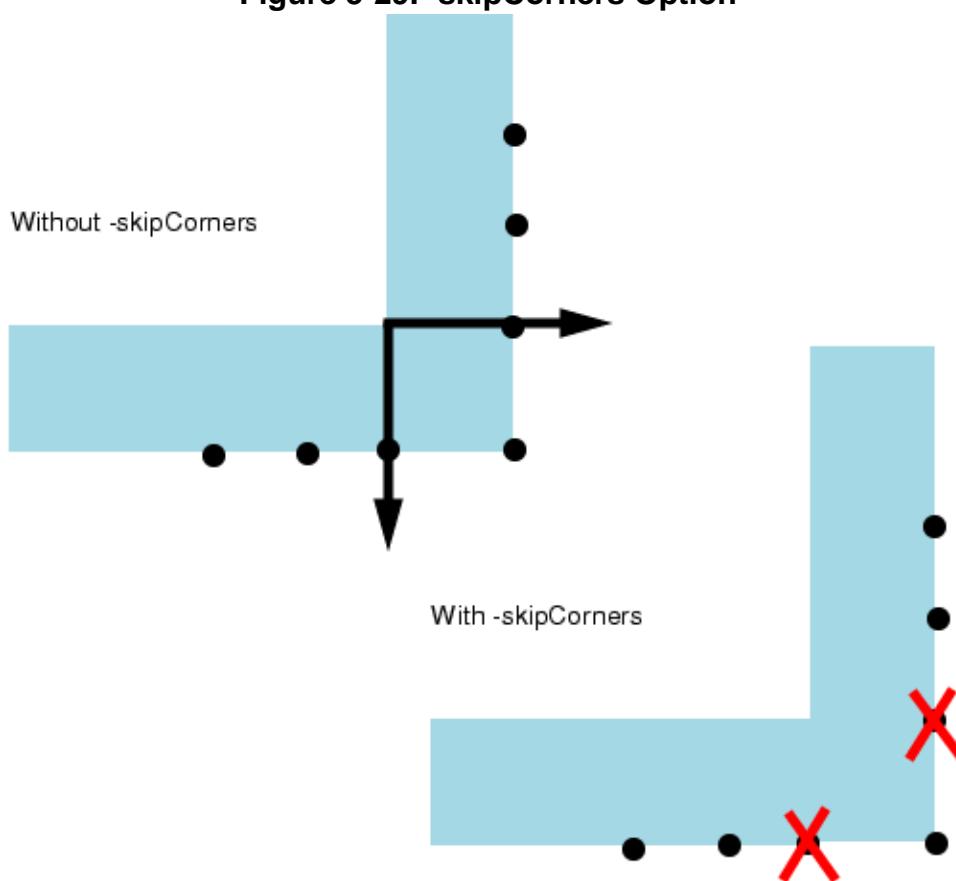
```
fragment_corner convex 0.1 0.05
fragment_inter -interdistance 0.13 -skipCorners
```

However, the -skipCorners option is enabled in this case to prevent from breaking the initial set of intrafeature fragments.

Another way to achieve the same effect in this specific instance is by setting -cornerDist to the *len1* of fragment_corner:

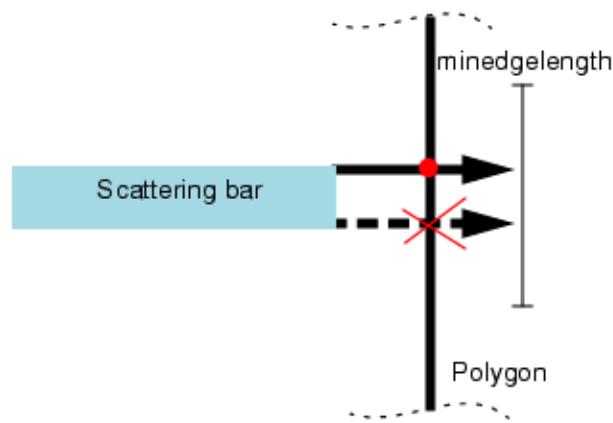
```
fragment_corner convex 0.1 0.05
interfeature -interdistance 0.13 -cornerDist 0.1
```

In general, use -skipCorners if intrafeature fragmentation runs first, because it ignores any fragments touching either convex or concave corners. Use -cornerDist if performing interfeature fragmentation before intrafeature, or when there might be multiple fragments within an area in which you do not want further fragmentation.

Figure 5-29. -skipCorners Option

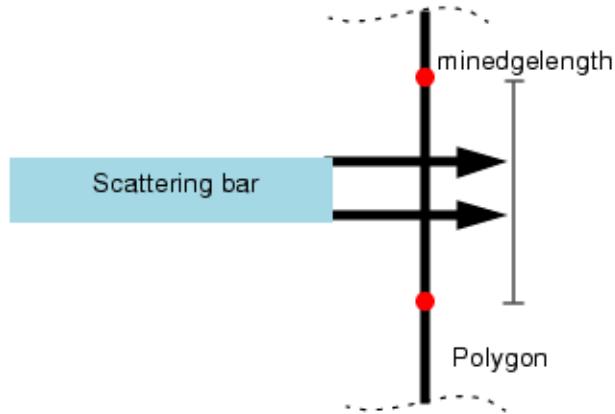
Example 2 - Scattering Bars

Scattering bars are generally quite thin, usually less than the minimum edge length. This means that when the corners of a scattering bar fragment the edge of a polygon, one corner generates a break point but the other does not because the fragment would violate the minimum length as shown in Figure 5-30.

Figure 5-30. Typical Fragmentation Caused by a Scattering Bar

Specifying -sym in the interfeature command reduces this sort of fragmentation. Instead of attempting to split the edge at fragments in line with each corner, it attempts to create a single fragment of minedgelength centered on the scattering bar end as shown in [Figure 5-31](#). Because of the many causes of fragmentation, some other cause may prevent -sym from adjusting points as needed to create a symmetric fragment of sufficient size. Also, non-90 degree edges are not adjusted.

Figure 5-31. Symmetric Fragmentation Caused by a Scattering Bar and -sym



Example 3 - Managing Conflict Between Multiple Layers

By default, when fragment_interlayers specifies multiple layers (for example, A, B, and C) the interfeature fragmentation from each layer is completed one layer at a time; that is, all the points from layer A are inserted, then layer B, then layer C. If a shape is receiving points from more than one layer, the one to run first inserts its points no matter what the conflict resolution scheme.

By adding the -combined option, the layers in fragment_interlayers are treated as one layer for the purposes of generating points. (The shapes in the layers are not merged.) This allows the conflict resolution scheme to consider all potential insertions as a set and does not give priority to any particular layer.

[Figure 5-32](#) shows two layers, both of which generate fragmentation on the shapes in the middle. The following lines were used to generate the results, differing only in -combined (in green):

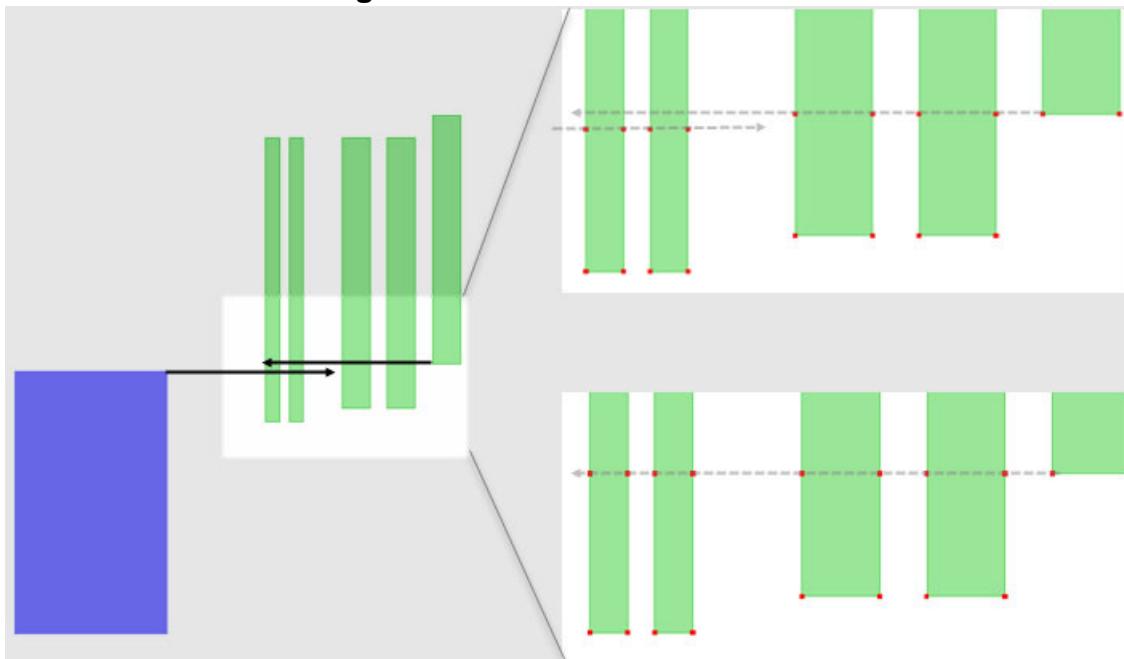
```
layer Green opc
layer Blue island

fragment_layer_order full

fragment_layer Green {
    fragment_interlayers Green Blue
    fragment_inter -interdistance 0.9 -ripples 0 -shield 0 \
        -cornerDist 0.08 -rem 0.038 -remext 0.07 -ripplestyle 2 -combined
}
```

The results are shown on the right. The top resolution uses the -combined option, allowing -ripplestyle 2 to give priority to the points from Blue, as there were fewer shielding edges than the points generated by Green. The lower resolution was generated without -combined. As fragment_interlayers specified Green before Blue, the interfeature points from Green were inserted. When fragment_inter processed Blue, the earlier points prevented them from being inserted.

Figure 5-32. Effect of -combined



fragment_interlayers

[MPC denseopc_options Commands](#)

Fragmentation control: Specifies layers for interfeature fragmentation.

Usage

fragment_interlayers *layer1* [*layer2...layerN*]

Arguments

- ***layer1* [*layer2 ... layerN*]**

Specifies layers to consider for interfeature fragmentation.

Description

This keyword defines an ordered list of layers to consider for interfeature fragmentation. The layers are each used, one-at-a-time, starting with ***layer1***, to generate interfeature fragmentation on the current [fragment_layer](#). Either layer names or numbers may be used, with numbers supported only for backward compatibility.

This keyword can only be used inside of a [fragment_layer](#) block. Using it outside of a [fragment_layer](#) block generates an error. This keyword may only be specified one time inside a [fragment_layer](#) block. Specifying it more than once generates an error.

Using [fragment_interlayers](#) can have the side-effect of causing interfeature fragmentation to occur. This is because interfeature fragmentation defaults to “on,” and has default parameters. While this usage is not recommended, it is still legal for backward compatibility.

Unless you want interfeature fragmentation to occur with the default parameters, the [fragment_inter](#) command should be specified along with the [fragment_interlayers](#) command. Additionally, these two commands should be specified one-after-the-other in the [fragment_layer](#) block to ensure that interfeature fragmentation is applied only one time with the correct parameters. This ensures that interfeature fragmentation occurs when you want it to occur, and does not occur when you don't want it to occur.

Examples

These two example fragment_layer blocks will perform in the same fashion, since the fragment_inter and fragment_interlayers commands are specified consecutively:

```
fragment_layer m1.decom.maskx {
    fragment_corner convex 0.06 0.07
    fragment_corner concave 0.05 0.06
    fragment_inter -interdistance 0.3 -ripplelen 0.08 -num 2
    fragment_interlayers m1.decom.maskx m1.decom.masky
    fragment_corner convex 0.21 0.09 0.1
    fragment_corner concave 0.18 0.08 0.09
}

fragment_layer m1.decom.maskx {
    fragment_corner convex 0.06 0.07
    fragment_corner concave 0.05 0.06
    fragment_interlayers m1.decom.maskx m1.decom.masky
    fragment_inter -interdistance 0.3 -ripplelen 0.08 -num 2
    fragment_corner convex 0.21 0.09 0.1
    fragment_corner concave 0.18 0.08 0.09
}
```

If fully-ordered fragmentation is being used, and the fragment_inter command is needed more than one time along with fragment_interlayers, it should be specified adjacent to one of the fragment_inter commands. The list of layers should include all of the layers needed for interfeature fragmentation. Note that both fragment_inter commands will be applied using all of the layers listed in the fragment_interlayers list at their respective times during fragmentation.

```
fragment_layer_order full
fragment_layer m1.decom.maskx {
    fragment_corner convex 0.06 0.07
    fragment_corner concave 0.05 0.06
    fragment_interlayers m1.decom.maskx m1.decom.masky
    fragment_inter -interdistance 0.3 -ripplelen 0.08 -num 4
    fragment_corner convex 0.21 0.09 0.1
    fragment_corner concave 0.18 0.08 0.09
    fragment_inter -interdistance 0.6 -ripplelen 0.08 -num 2
}
```

fragment_island

MPC denseopc_options Commands

Fragmentation control: Used to define parameters for island-layer fragmentation.

Usage

```
fragment_island layer_name [-ripples x] [-ripplelen z] [-rem r] [-enforce {0 | 1}]  
[-rippleinside | -rippleboth] [-dontcross | -allowcross] [-offset offset_value]  
[-stampdistance y [-stampshield w] [-stampall]] [-bisect] [-cornerdist minimum]
```

Arguments

- ***layer_name***

Specifies the layer name.

- **-enforce 0 | 1**

When set to 1, a fragment is broken at intersection points only if no fragments of lengths less than [fragment_min](#) are created. If not present, fragmentation can result in [fragment_min](#) violations. The default value is [0](#).

- **-ripples *x***

An optional argument specifying the number of ripples. The default value is [0](#) (no ripples are created). The maximum number of ripples you can create is 10.

- **-ripplelen *z***

An optional argument specifying the length of a single ripple edge in microns. The default value is [fragment_min](#).

- **-rem *r***

An optional argument specifying the minimum allowed remainder for the purposes of ripple generation. If the ripple would produce an edge with length less than *r*, the vertex will not be inserted. The default value is [fragment_min](#).

- **-rippleinside**

Generates ripples toward the inside of the fragment_island layer polygons at any intersection, not just where coincident edges are present. For example:

```
fragmentLayer POLY {  
    fragment_island DIFF -ripples 2 -ripplelen 0.04 -enforce 1 \  
        -rippleinside  
}
```

This example generates ripples in the edges on the POLY layer wherever they cross the DIFF layer. Two ripples of length 0.04 microns towards the inside of the DIFF layer polygon will be generated, and no fragments less than [fragment_min](#) in length will be created.

Either **-rippleinside** or **-rippleboth** may be specified, but not both.

- -rippleboth

Generates ripples toward both the inside and the outside of the island fragment layer polygons at any intersection, not just where coincident edges are present. For example:

```
fragment_layer POLY {  
    fragment_island DIFF -ripples 2 -ripplelen 0.04 -enforce 1 -  
        rippleboth  
}
```

This example generates ripples in the edges on the POLY layer wherever they cross the DIFF layer. Two ripples of length 0.04 microns towards the inside of the DIFF layer polygon will be generated, two ripples of length 0.04 microns towards the outside of the DIFF layer polygon will be generated, and no fragments less than fragment_min in length will be created in either direction.

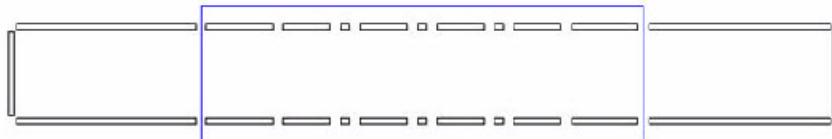
Either -rippleinside or -rippleboth may be specified, but not both.

- -dontcross | -allowcross

The -dontcross option prevents ripples from crossing an island layer, or crossing one another. In [Figure 5-33](#), the island layer shown in blue has fragmented the layer shown in black resulting from the following code using the default setting -allowcross:

```
fragment_island island_layer -ripples 4 -ripplelen 0.07 -rem 0.0005  
\\  
-enforce 0 -rippleinside
```

Figure 5-33. Without -dontcross Option



Toward the center of the island layer, the polygon has been broken into a series of very short fragments. This is because the ripples from the opposite sides of the island have crossed at the middle, and are breaking one another up into very small fragments. While it is possible to prevent this problem by limiting the number of ripples or using minedgelength, that may not be possible in all situations. Instead, the -dontcross option can be used to prevent the undesired fragmentation. [Figure 5-34](#) shows the result of the same code using the -dontcross option.

Figure 5-34. With -dontcross Option



The undesirable short fragments at the center of the island are no longer present.

Note

 In cases where no ripple crossings occur, this option has no effect.

The `-allowcross` option is the inverse of the `-dontcross` option, as well as the default setting. This means that ripples that meet and cross at the center of an edge are allowed, even if that is undesirable.

- `-offset offset_value`

An optional argument specifying an offset for all ripples. Normally, an island break occurs at the point where the island layer intersects the opc layer. There are occasions when it is desirable to have the break offset from the intersection point. This might allow a site to be placed on the island layer crossing.

This option causes all ripples to be offset from the island layer crossing by specifying an `offset_value` in microns. All produced ripples occur at $(\text{ripplelen} * n) + \text{offset_value}$ locations.

If an offset is specified, the initial break at the island crossing does not occur. Instead, the first break is displaced by the specified amount in `offset_value` from the island crossing. The second break occurs at `offset_value + ripplelen` microns from the island crossing, with all further breaks occurring at `ripplelen` intervals. This means that the total number of ripples is reduced by one ripple when you specify an offset.

- `-stampdistance y [-stampshield w] [-stampall]`

An optional family of arguments that cause interfeature fragmentation to be generated for each island crossing. Unlike `fragment_corner` interfeature fragmentation, the inserted fragmentation point is at the *crossing*. Island layer corners do not create fragmentation. The default is to do no interfeature fragmentation at island crossings.

`-stampdistance y` — Turns on interfeature fragmentation from island crossings. The number of ripples and their lengths are set by other arguments. The maximum distance that the fragment crossing points can project is `y` microns.

`-stampshield w` — Stops the fragmentation from propagating after `w` edges. Has no effect unless `-stampdistance` is specified.

`-stampall` — Causes ripples to be generated even from fragmentation points that were not inserted. Even if a particular island crossing was rejected, it and its ripples can be propagated and points inserted where no conflict exists. Has no effect unless `-stampdistance` is specified.

- `-bisect`

An optional keyword that resolves conflicts between ripples by bisecting the location of the two conflicting ripples.

If there are conflicting ripples and this keyword is not specified, neither ripple is inserted.

- -cornerdist *minimum*

An optional argument that specifies the minimum distance at which a fragmentation point can be inserted near a corner. Island fragmentation only adds fragmentation points to an edge if the points are at least *minimum* distance from the corner.

Description

This keyword controls island layer fragmentation. Island layer fragmentation inserts fragmentation points where edges on the island layer intersect edges on an opc or correction layer.

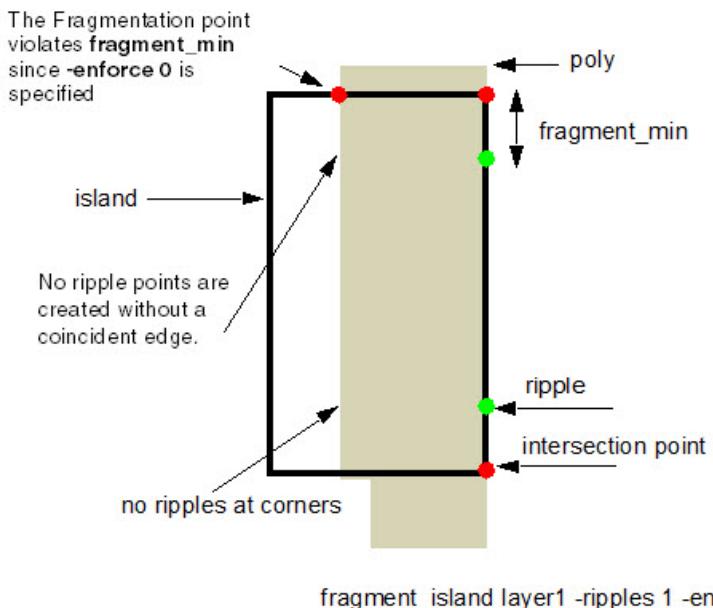
If **-enforce 0** is specified, fragmentation points are created unconditionally at intersections as described previously.

If **-enforce 1** is specified, fragmentation points are inserted only if no [fragment_min](#) violations are created.

Additionally, the **-ripples** and **-ripplelen** parameters define the number of ripples and the distance between consecutive ripple points to be created at each intersection. Ripples are created toward the interior of the island edge only.

Ripples are created only if no edges of length smaller than the **-rem** argument are produced. Fragmentation is performed with each island layer specified with the **layer** keyword, in the order of definition. No ripple points are created without a coincident edge unless the rule includes **-rippleinside**, **-rippleboth**, or **-stampdistance**.

Figure 5-35. Island Layer Fragmentation Example



fragment_layer

[MPC denseopc_options Commands](#)

Fragmentation control sub-command block: Used to reset global fragmentation settings for special-case fragmentation.

Usage

```
fragment_layer layer direction{  
    settings  
    ...  
}
```

Arguments

- ***layer***

The layer name to which the fragmentation parameters within this keyword apply.

- ***direction***

This argument causes fragmentation to be applied only to edges in the specified direction. Edges in any other direction are not fragmented. The ***direction*** can be: horizontal, vertical, or non90. If it is omitted, it means all directions (the default). A layer may be listed multiple times to do fragmentation in different directions. For example:

```
fragment_layer POLY horizontal {  
    ...  
}  
fragment_layer POLY vertical {  
    ...  
}  
fragment_layer POLY non90 {  
    ...  
}
```

Directional fragmentation does have some limitations. It only applies to interfeature fragmentation, corner fragmentation and maxedgelength fragmentation. Any of the layer based commands, like coincident layer, apply to all edges irrespective of any direction specification.

- ***settings***

A sequence of lines containing fragmentation keywords with parameters. A single keyword is allowed per line.

The following fragmentation keywords are allowed in a fragment_layer block:

Table 5-9. Supported Fragmentation Keywords in fragment_layer Block

fragment_corner	fragment_min	fragment_max
fragment_inter	fragment_island	fragment_coincident
fragment_visible	fragment_nearly_coincident	

Description

This keyword defines a fragment layer sub-command block, which customizes the fragmentation for the specified *layer*. Any fragmentation parameters included within this keyword apply only to the specified *layer*, overriding the global settings.

```
fragment_layer CORRECTION {  
    fragment_island DIFF -ripples 1 -ripplelen 0.04 -enforce 1 -rippleboth  
}
```

The fragment_layer block modifies the island -ripples default, setting the ripples to 1 rather than the default 0, as shown in the following example (the underlined section indicates where a change occurred):

```
fragment_island DIFF -ripples 1 -ripplen 0.1 -interdistance 0.45
```

fragment_layer_order

[MPC denseopc_options Commands](#)

Fragmentation control: Used to change the default processing order of fragmentation schemes in a fragment_layer command block.

Usage

fragment_layer_order {standard | partial | full}

Arguments

- **standard**

Specifies that **fragment_layer** fragmentation will be carried out in its standard order.

- **partial**

Specifies that fragmentation within a fragment_layer block is set by the order of the first appearance of command types. This is the default value.

- **full**

Specifies that fragmentation within a fragment_layer block are followed in explicit order. This differs from partial in the following ways:

- Intrafeature and interfeature fragmentation will be performed repeatedly, as specified in the fragmentation block (for example, specifying intrafeature, interfeature and intrafeature will cause fragmentation to be done in three phases, once with the first set of intrafeature parameters, next with the interfeature parameters and then with the last set of intrafeature parameters).
- Only the forms of fragmentation listed in the fragment_layer block are performed. The default forms of fragmentation are not performed. So, for example, if coincident layer or island layer fragmentation are not specified, they simply will not be performed.

Description

Changes the default processing order of the fragmentation schemes (intrafeature, interfeature, and so on) to instead use the order as specified in the fragment_layer block. The default value for this variable is **partial**.

For example, if intrafeature fragmentation is specified before island fragmentation, then intrafeature fragmentation will be performed first, followed by island fragmentation. Any form of fragmentation not specified in the fragment_layer command is then handled using default values after all of the forms of fragmentation listed in the fragment_layer command have been performed. The order of the unspecified methods of fragmentation is the same as the default order, but after the explicitly specified forms of fragmentation.

[Table 5-10](#) shows the forms of fragmentation allowed within a fragment_layer command block along with the keywords that control them.

Table 5-10. Allowed Fragmentation in a fragment_layer Command

Default Order	Method	Keyword(s)
1	island layer	fragment_island
2	intrafeature	fragment_corner
3	interfeature	fragment_inter
4	maxedgelength	fragment_max

The `fragment_min` keyword has no effect on the order of fragmentation if `fragment_layer_order` is set to “1” and may be specified at any point in the `fragment_layer` block.

Examples

The following example results in all corner fragmentation being performed prior to interfeature fragmentation:

```
fragment_order_layer partial

fragment_corner convex 0.06 0.07
fragment_inter -interdistance 0.4 -ripples 2 -ripplelen 0.06
fragment_corner concave 0.06 0.08
```

Note

 Once `fragment_layer_order` is set, all `fragment_layer` blocks will be processed in the order described in the file. It is not possible to specify that one `fragment_layer` block be processed in user-defined order while another is processed in default order. If you need to have a block processed in default order while the `fragment_layer_order` flag is set, you must manually specify the operations in the same order as the default.

fragment_max

[MPC denseopc_options Commands](#)

Fragmentation control: Used to set maximum fragment length constraint.

Usage

```
fragment_max maxedge [-rem r] [-minlen d]
```

Arguments

- **maxedge**

A required argument specifying the maximum length of fragments generated by **fragment_max** fragmentation, specified in microns. The default value, if fragmentation is not explicitly defined, is $4 * \text{Nyquist}$.

The length can range from a minimum of [fragment_min](#) to a maximum of [tilemicrons/6](#). If a value is specified higher than that, then it will be automatically reduced to [tilemicrons/6](#).

- **-rem r**

An optional argument specifying the minimum allowed remainder. No edges shorter than r are output. If not specified, the default value is [fragment_min](#).

- **-minlen d**

An optional argument specifying the shortest edge (an unbroken fragment with two corners) that can be fragmented. The limit does not apply to long fragments that are part of a fragmented edge.

Description

This keyword is a fragmentation parameter. It is used during **fragment_max** fragmentation, which breaks up very long edges into shorter fragments.

Note

 The value of **fragment_max** affects the interaction distance, so longer **fragment_max** lengths can have a negative impact on runtime.

The **fragment_max** fragmentation algorithm will attempt to break all edges equal to or longer than $2 * \text{maxedge} + \text{fragment_min}$. Subsequently, these edges will be split into two **maxedge** parts and one part greater than or equal to [fragment_min](#). Fragmentation will be symmetric with respect to the center of the edge. [Figure 5-36](#) shows the results of fragmenting an edge of length $2 * \text{maxedge} + N$, where $N > \text{fragment_min}$.

No fragment smaller than **fragment_min** will be generated by this operation. In [Figure 5-36](#), if N had been shorter than **fragment_min**, the edge would not have been fragmented.

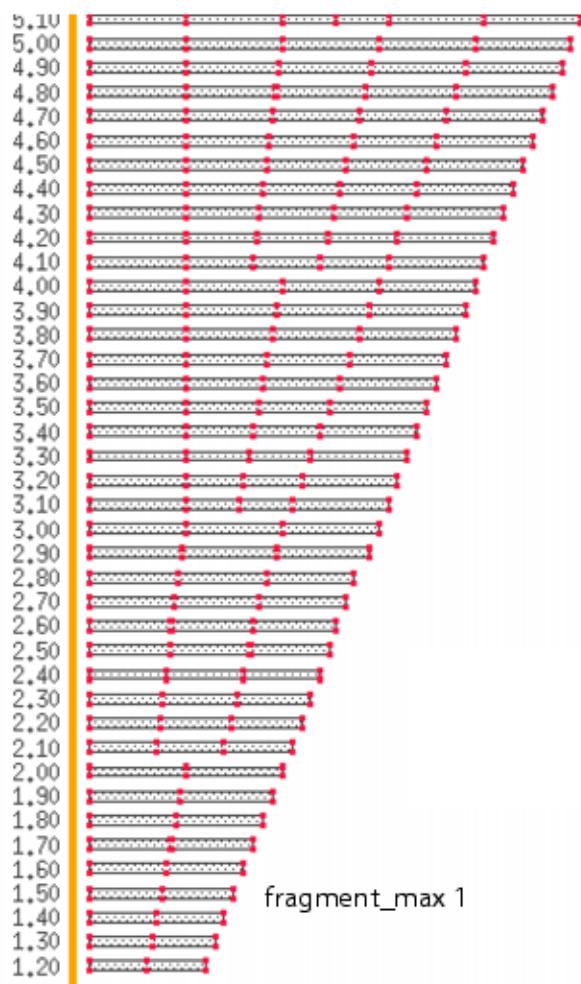
Note that the **fragment_max** is not the maximum fragment length. The true maximum fragment length is $(2 * \text{maxedge} + \text{fragment_min} - \text{epsilon})$. [Table 5-11](#) shows several ranges of values of

possible fragment lengths and shows how nmMPC splits the fragment. Figure 5-36 is another example of how **fragment_max** implements splitting.

Table 5-11. Examples of Fragment Splitting

Conditions	Effect
< 2MIN or < MAX	Do not break
$\geq 2\text{MIN}$, $> \text{MAX}$, and $\leq 2\text{MAX}$	Split into two equal fragments (+/- 1 dbu)
$> 3\text{MIN}$ and $> \text{MAX}$	Split into three equal fragments (+/- 1 dbu)
$> (3\text{MAX})$	Fragments of length <code>fragment_max</code> , with a centered remainder split into two or three equal fragments that are as long as possible with none longer than <code>fragment_max</code> , and none shorter than <code>fragment_min</code> . Note: if the original long fragment is $<(3\text{MAX})$ there will be no fragments of <code>fragment_max</code> .

Figure 5-36. fragment_max



Note

 By default, global fragmentation settings for `fragment_max` and `fragment_min` are calculated automatically by Calibre nmMPC. However, if you are using custom fragmentation using `fragment_layer`, then these global settings are not passed to the `fragment_layer` blocks. You must explicitly set `fragment_max` and `fragment_min` outside the block.

fragment_min

[MPC denseopc_options Commands](#)

Fragmentation control: Used to set minimum fragment length constraint.

Usage

fragment_min *minfrag* [enforce]

Arguments

- ***minfrag***

Minimum allowed length of a fragment edge in microns. The default value is 2 * Nyquist.

- **enforce**

An optional keyword that causes **fragment_layer** blocks to honor the **fragment_min** defined inside the layer block. For backwards compatibility, this is not done by default. Only use **enforce** when the **fragment_min** keyword is inside a **fragment_layer** block.

Description

This keyword is a fragmentation parameter. It defines the smallest fragment that can be produced by fragmentation. An edge is not fragmented if it results in a new edge shorter than **minfrag**. The **minfrag** value refers to fragments created by the application, not created by the user, so while this keyword defines the minimum length of a fragment, smaller edges will still exist if the user defined them elsewhere in the setup file.

Note

 By default, global fragmentation settings for **fragment_max** and **fragment_min** are calculated automatically by Calibre nmMPC. However, if you are using custom fragmentation using **fragment_layer**, then these global settings are not passed to the **fragment_layer** blocks. You must explicitly set **fragment_max** and **fragment_min** outside the block

fragment_minjog

[MPC denseopc_options Commands](#)

Fragmentation control: Controls jog definition.

Usage

`fragment_minjog minjogsز`

Arguments

- ***minjogsز***

Specifies the size of a jog in microns. Specifying a value smaller than this size and an edge with a convex corner on one side and a concave corner on the other is considered a jog.

Description

This command allows you to control the definition of a jog, which controls fragmentation near small jogs. Any edge less than ***minjogsز*** is a jog, and fragmentation due to a jog can be handled differently. For example, the following command:

```
fragment_flaglayer poly intrafeature_smalljogs disable
```

followed by this command in the fragment_layer block for poly:

```
fragment_minjog 0.070
```

prevents intrafeature fragmentation of edges adjacent to jogs less than 0.070 microns in length.

fragment_nearly_coincident

MPC denseopc_options Commands

Copies fragmentats from one layer to another where the layer edges are within a specified distance.

Usage

```
fragment_nearly_coincident source_layer [-dist microns] [-nocorners] [-nonparallel]
```

Arguments

- ***source_layer***
A required argument specifying the layer containing fragments.
- **-dist *microns***
An optional argument indicating the maximum distance in microns separating the edges from the source layer and current layer. Only edges within the distance receive fragmentation.
The default is 0.010 microns.
- **-nocorners**
An optional argument that prevents corners on the source layer from generating fragments on the current layer.
- **-nonparallel**
An optional argument that copies fragmentation to an edge that is 45 to 90 degrees to the source edge. By default, these edges are excluded.

Description

This optional command copies fragments from the source layer to the current layer when the layer edges are within a specified distance and at no more than 45 degrees to each other. In addition, the command creates fragments only, if they do not violate the fragment_min limit. This command differs from fragment_coincident, which only copies fragments where the edges have no separation. Use fragment_nearly_coincident for multidoze corrections and if the edges of one layer have already been fragmented and adjusted, or otherwise have non-coincident segments.

This keyword must be used inside a [fragment_layer](#) block.

Keep -dist small enough that only one edge on the source layer corresponds to the current layer, and the edges of source and current are parallel. The fragment_nearly_coincident operation does not perform any shielding if multiple source edges are detected; the operation attempts to copy all selected source edge fragmentation to the current edge and then resolves conflicts. Results can be unexpected when unrelated layers are paired.

When using `fragment_nearly_coincident` when the dose assignment occurs within one layer, an alternate method is required. In the following example, the correction layers have their fragments copied and applied to a target layer using the `fragment_nearly_coincident` commands.

```
LITHO FILE MPC.recipe [
...
    layer MpcDL0
    layer MpcDL1
    layer MpcTarget
...
denseopc_options mpcOptions {
...
    layer MpcDL0      correction      clear
    layer MpcDL1      correction      clear
    layer MpcTarget   opc            clear
...
    image etch_model etch0 ebeam_model ebeam0 dose $DV0 layer MpcDL0 \
          ebeam0 dose $DV1 layer MpcDL1

    max_iterations      3
    feedback           -0.8 -0.9 -0.7
...
    fragment_layer MpcTarget {
        fragment_nearly_coincident MpcDL0
        fragment_nearly_coincident MpcDL1
    }
...
}
setlayer mpc_out_d10 = mpc MpcDL0 MpcDL1 MpcTarget MAP MpcDL0 OPTIONS \
    mpcOptions
setlayer mpc_out_d11 = mpc MpcDL0 MpcDL1 MpcTarget MAP MpcDL1 OPTIONS \
    mpcOptions
]
```

Typically, verification recipes mirror many of the same layer specifications. However, if dose layers are specified as correction layers in the verification recipe (layers MpcDL0 and MpcDL1),

```
layer MpcOutDL0      external      clear
layer MpcOutDL1      external      clear
layer MpcDL0         correction    clear
layer MpcDL1         correction    clear
layer MpcTarget       opc          clear
```

The opc layers don not have their edges moved, since the correction layers are for that purpose. Island layers, and not opc layers, must be used as the dose layers for the `fragment_nearly_coincident` operation, as shown in the following example recipe:

```
LITHO FILE MPC.recipe [
...
layer MpcOutDL0
layer MpcOutDL1
layer MpcDL0
layer MpcDL1
layer MpcTarget
...
denseopc_options mpcOptions {
...
    layer MpcOutDL0      external      clear
    layer MpcOutDL1      external      clear
    layer MpcDL0          island       clear
    layer MpcDL1          island       clear
    layer MpcTarget        opc         clear
...
    image etch_model etch0 ebeam_model ebeam0 dose $DV0 layer MpcOutDL0 \
          ebeam0 dose $DV1 layer MpcOutDL1
...
    max_iterations 1
    feedback 1
fragment_layer MpcTarget {
    fragment_nearly_coincident MpcDL0
    fragment_nearly_coincident MpcDL1
}
...
}

setlayer EPE_Layer = mpc MpcOutDL0 MpcOutDL1 MpcDL0 MpcDL1 MpcTarget \
MAP MpcTarget OPTIONS mpcOptions
```

Examples

The following example copies the fragmentation from the m1.preclean layer to the m1.fill layer when edges are within 0.005 microns:

```
fragment_layer m1.fill {
    fragment_inter off
    fragment_nearly_coincident m1.preclean -dist 0.005
}
```

fragment_visible

[MPC denseopc_options Commands](#)

Defines layers for visible layer fragmentation.

Usage

fragment_visible [layer1 layer2 ... layerN]

Arguments

- [layer1 layer2 ... layerN]

The layers whose corners are to be used to define visible layer fragmentation, listed from highest priority to lowest priority, on the target. If no arguments are given, then no visible layers will affect fragmentation or define false edges on the target layer. All layers must be of type **visible**. This statement is required if you do not want visible layers to introduce interfeature fragmentation.

Description

An *optional* user control for visible layer fragmentation. This keyword can only be used within a fragment_layer block.

This keyword defines a prioritized list of layers whose geometries will be used to control generation of fragmentation vertices on the layer to which the fragment_layer block applies. The order of the layers in the keyword defines the priority. The first layer has the highest priority, the second layer has the second highest, and so on.

Fragmentation vertices are added wherever the edges of the target layer intersect edges or vertices on these visible layers, as long as the fragments created are not smaller than fragment_min. Any fragments which are coincident with, or overlapped by active visible layers are marked as “false edges” that do not move during MPC.

fragment_small

[MPC denseopc_options Commands](#)

Specifies the upper size limit for fragments using a high-accuracy EPE measurement algorithm.

Usage

fragment_small *val*

Arguments

- ***val***

A required argument that specifies the size, in microns, of what is considered a small fragment by Calibre nmMPC, and overrides the default small fragment value (the very short range e-beam kernel length).

Description

In MPC correction, small fragments (especially jogs) can move by large amounts depending on the movement of neighboring fragments. One way to avoid this is using the jog_freeze parameter that prevents jogs below a certain size limit to move. However, this can lead to inaccurate correction results, especially for very advanced EUV masks. This parameter sets the upper size limit for fragments using an advanced moving algorithm. With this algorithm, the correction accuracy of small fragments is largely improved, but because of the higher computational effort, this algorithm should not be applied to large fragments.

By default, all fragments less than the very short range e-beam kernel length are considered as small fragments. This option can be used to override the small fragment value. This allows you to control the fragments moved with higher correction accuracy.

freeze_skew_edges

MPC denseopc_options Commands

Freezes skew edges and neighboring fragments.

Usage

freeze_skew_edges [off | on] [is_error | no_error] [*max_log_limit*]]

Arguments

- off | on

Deactivates (off) or activates (on) freezing skew edges. On is implied (does not need to be explicitly present). The default setting is on.

- is_error | no_error

If is_error is specified, Calibre nmMPC sets skew edges as an error condition and exits the program. If no_error is specified, the execution continues even if skew edges exist, though warnings are issued. The default setting is no_error.

- *max_log_limit*

Limits the maximum number of warnings per skew edge encountered in the log per tile. The warning log message contains the skew edge coordinates and layer. The default is set to log all skewed edges.

Description

This command freezes skew edges and their immediate neighboring fragments from Calibre nmMPC movement.

Note

 By default, **freeze_skew_edges** freezes skew edges in place (on) and continues the MPC run while issuing warning messages (no_error).

Performance

This setting adds O(*N*) to runtime where *N* is number of fragments.

Examples

`freeze_skew_edges is_error`

grids_for_angle_45_snap

MPC denseopc_options Commands

Used to specify how 45-degree angles are moved during MPC.

Usage

`grids_for_angle_45_snap val`

Arguments

- *val*

Specifies the snapping grid for 45-degree angle lines. The grids are specified in microns. This forces 45 degree edges to snap to a final grid of this value. The default is 0.

Description

An optional command that allows you to specify how 45-degree angle lines are moved during MPC. You use this variable to specify the grid used for snapping 45-degree edges. When set, the endpoints of 45-degree angle lines snap to the specified grid. This ensures that edges that were input as 45-degree angles maintain that angle and also line up with a manufacturable grid.

The snapping grid for 45-degree angles should be greater than or equal to the size of the grid on which the data was originally created AND greater than or equal to value of the stepsize keyword.

If the variable is not set, then 45-degree edges maintain that angle after movement, but their endpoints are on a grid equal to the size of the database unit.

Examples

The following example causes 45-degree edges to remain 45-degrees and also remain on a 5 nm grid after MPC.

```
grids_for_angle_45_snap 0.005
```

image

MPC denseopc_options Commands

Creates the nominal image for an E-beam or etch model.

Usage

For single layer E-beam input:

```
image ebeam_model ebeam_name [dose val] [layer name] [etch_model etch_name]
```

For multilayer E-beam input:

```
image {ebeam_model ebeam_name dose val layer name} [ebeam_name dose val layer  
name]... [etch_model etch_name]
```

For etch-only input (without an E-beam model):

```
image etch_model etch_modelname [sigma sigma_value]
```

Arguments

- **ebeam_model *ebeam_name***
A required parameter specifying the E-beam model used for this operation.
- **etch_model *etch_name***
An optional parameter that specifies the etch model.
- **dose *val***
An optional parameter that specifies the dose. The default value is **1**.
- **layer *name***
An optional parameter that specifies the input layer name.
- **sigma *sigma_value***
An optional parameter that specifies the sigma value. This is only supported for etch models. If specified, it is used to determine the EPE measurement location identical to those used with a single simulation-type E-beam model with the same sigma parameter.

Description

A required command that specifies the nominal image for an E-beam or, optionally, an etch model. You can define multiple input layers for “image ebeam_model.” Each input layer has its applied E-beam model and dose. The dose is optional, and the default value is 1. Multiple layers are not supported for etch model image statements.

All simulation-type models as well as etch-only correction require that the algorithm 1 option be specified in the corresponding denseopc_options command. For example, the following is an expanded syntax of the single-input layer declaration:

```
denseopc_options <srcOptions> {
    image ebeam_model <ebeam model name> [dose <dose value>] \
        [layer <input layer name>] [etch_model <etch model name>]
    algorithm 1
}
<output_layer_name> = LITHO MPC <list_of_input_layer_name> \
    FILE <parameter_block_name> MAP <output_layer>
```

Examples

```
image ebeam_model fastEB etch_model EM_Nominal
```

image_options

MPC denseopc_options Commands

Creates a named option block. This image_options block is used with the image command in litho model mode. These options are passed using the specified name as an ease-of-use substitution.

Usage

```
image_options name {  
    layer name {visible | hidden}  
    litho_model litho_model_name  
    ...  
}
```

Arguments

- ***name***

Specifies a user-defined name that is used to refer to this block from the image command. The option block must be enclosed in braces ({}).

- **layer *name*{visible | hidden}**

Specifies a visible or hidden layer. At least one visible layer is required.

- **litho_model *litho_model_name***

A required argument that specifies a directory name. This directory must contain all model components (e-beam model and VEB model) to be used for the image options block, and a Lithomodel file with a filename “Lithomodel” (see “[Lithomodel \(Litho Model Format\)](#)” on page 341).

Examples

```
modelpath @MODELPATH@  
layer M0  
layer M2  
layer M3  
layer M4  
layer M5  
layer M6  
layer M7  
tilemicrons 100  
image_options MDF {  
    layer M0 visible ebeam_dose 1.0  
    layer M2 visible ebeam_dose 0.9  
    layer M3 visible ebeam_dose 1.1  
    layer M4 visible ebeam_dose 1.2  
    layer M5 visible ebeam_dose 1.3  
    layer M6 visible ebeam_dose 1.4  
    layer M7 visible ebeam_dose 1.5  
    litho_model @MODELDIR@  
}  
setlayer im = ebeam_simulate MDF
```

jog_freeze

[MPC denseopc_options Commands](#)

Prevents a fragment from moving during MPC if it is a jog edge.

Usage

jog_freeze *value*

Arguments

- *value*

Specifies a length that if an edge is less than or equal to (and it is bounded by one convex and one concave corner), the edge will not be moved during MPC iterations.

Description

The jog_freeze command will prevent an edge from moving during MPC iterations if the edge is a jog (bounded by 1 convex and 1 concave corner) and it's length is <= *value*.

jog_ignore_metric

[MPC denseopc_options Commands](#)

Changes fragment corner classification when shared with a short jog.

Usage

jog_ignore_metric *len*

Arguments

- *len*

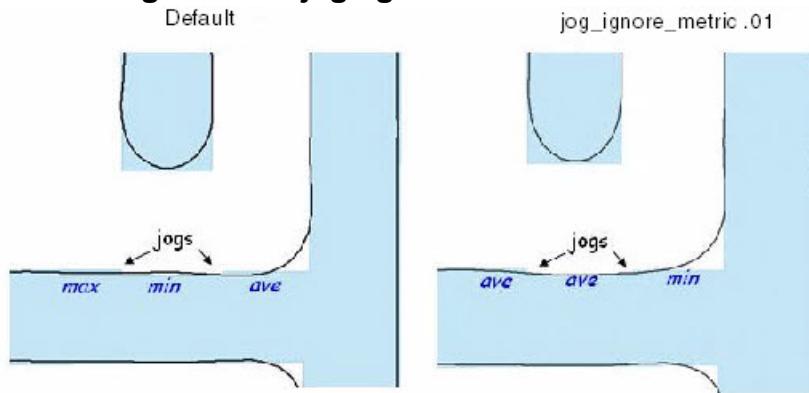
Define jog fragments with length $\leq \text{len}$ as jogs for this command. Valid values are > 0 .

Description

This command changes a fragment's corner classification when it is shared with a short jog (in other words, a jog with length $\leq \text{len}$) from the usual concave or convex to “not-a-corner” when assigning EPE measurement metrics for that fragment.

This jog_ignore_metric change carries a lower precedence than the normal convex or concave corner. For example, if a fragment shares one end point with a short jog and the other end point on a normal convex corner, that fragment is classified as “convex” for its EPE measurement metrics. [Figure 5-37](#) shows how the metrics are evaluated using this command.

Figure 5-37. jog_ignore_metric Behavior



The figure on the right shows how jogs can effect EPE sampling by introducing small corners in what was meant to be a single, unbroken fragment. At convex corners, the numerical maximum is used as the EPE sampling method, and the numerical min is used as the EPE method for convex corners. An average of the EPEs are calculated for non-corner edges.

In the second figure, because the jogs are now defined as 0.01 or less, the small corners are ignored. Thus, the calculations are more accurate (at the non-corner edges, an average of the EPEs are calculated, and a numerical minimum at the concave corner), resulting in MPC output that more closely match the original target.

Performance

$O(N)$ to runtime where N is number of fragments.

layer (denseopc_options Command)

MPC denseopc_options Commands

Defines layer characteristics such as name, type, transmission, and mask.

Usage

layer *layer_name* *layer_type*

Arguments

- ***layer_name***

Specifies the name of the layer. Calibre nmMPC uses this name in the rest of the setup file to refer to this layer.

Layer names must be alphabetic and/or numeric strings less than 32 characters in length.

- ***layer_type***

Specifies how the layer is used by Calibre nmMPC and the aerial image simulator. The following table lists the valid layer_types and their related function.

Table 5-12. layer Command layer_types (denseopc_options)

Layer Type	Function
opc (required)	<p>The opc layer serves as the target layer for LITHO operations, except when retargeting is used. The retarget layer instead becomes the target under these circumstances. The shapes on the opc layer represent the actual geometry that you intend to transfer onto the wafer. All EPEs are calculated with respect to opc layers.</p> <p>During processing, an opc layer is impacted as follows:</p> <ul style="list-style-type: none">• Its edges are fragmented.• Its edges are moved during MPC, unless correction layers are present.• The output (the MPC modifications to this layer) represents the final mask, unless correction layers are present. <p>At least one opc layer is required. You can pass multiple opc layers during a LITHO operation and define the opc layers in a setup file. When the opc layer is used for retargeting, the opc layer is fragmented and corrected in order to make a contour that matches the retarget layer.</p> <p>Multiple opc layers are allowed. This allows layer operations such as fragmentation and MRC enforcement to be tailored to specific needs. During simulation and correction, the OR of all the opc layers is used as the input to the simulator and to determine which edges are movable.</p>
hidden (optional)	<p>By default, any layer not specified in the setup file is marked as a hidden layer. Hidden layers are used for a number of functions in Calibre nmMPC, including serving as retargeting layers and marker layers, or for re-opc and “no opc” regions.</p>

Table 5-12. layer Command layer_types (denseopc_options) (cont.)

Layer Type	Function
island (optional)	Island layers contain geometries used to force fragmentation and tagging on the opc and correction layers. The contents of these layers do not factor into simulation results.

Description

A **required** command that names and describes optical properties of an input layer. Every layer to be used in setlayer denseopc commands must be declared in the setup parameters using this command.

Note the following:

- Calibre nmMPC does not simulate layers designated as the hidden layer type, but these layers can otherwise be used in all OPCverify commands.

Examples

```
layer POLY opc dark
```

line_end_fragment

[MPC denseopc_options Commands](#)

Activate or deactivate line-end fragmentation.

Usage

line_end_fragment {off | on}

Arguments

- **{off | on}**

If set to on, `line_end_fragment` explicitly specifies using the legacy definition of a line end. Turning off the legacy behavior turns off the legacy definition of a line end.

Description

This command affects `fragment_corner` operation. By default, `fragment_corner` uses the legacy line end definition. This is equivalent to “`line_end_fragment on`”, which explicitly specifies using the legacy definition of a line end. The legacy line end definition can interfere with the operation of `fragment_corner` on edges less than the line end width. Turning off the legacy behavior with “`line_end_fragment off`” will prevent this problem from occurring.

max_angle_tolerance (LITHO CLMPC Only)

MPC denseopc_options Commands

Specifies the angle tolerance between two edge segments considered when smoothing edges of curvilinear and skew-edge layouts.

Usage

max_angle_tolerance *angle*

Arguments

- *angle*

A required argument that specifies the angle tolerance considered when calculating average EPE. Apart from the distance specified by [max_smoothing_range \(LITHO CLMPC Only\)](#), a fragment should be within the angle specified of max_angle_tolerance to be considered for smoothing. The default value is 20 degrees.

Description

This parameter is used in conjunction with the [max_smoothing_range \(LITHO CLMPC Only\)](#) in a LITHO CLMPC setup file, targeted specifically for curvilinear and skew-edge layouts.

Examples

```
LITHO FILE "clmpc.recipe" [
    denseopc_options clmpcOptions {
        algorithm 1
        max_smoothing_range 0.025
        max_angle_tolerance 40
        max_iterations 2
        feedback -0.8 -0.6
        image ebeam_model ebeam0 etch_model etch0
    }
    setlayer mpcOut = clmpc mpcIn MAP mpcIn OPTIONS clmpcOption
]
mpcOut = LITHO CLMPC poly MAP mpcOut FILE "clmpc.recipe"
```

max_iterations

[MPC denseopc_options Commands](#)

Limits number of MPC iterations.

Usage

max_iterations *m* [nopw *number_nopw_it*]

Arguments

- ***m***

Specifies maximum number of MPC iterations to be performed. If *m* is not specified, the default is 16.

- **nopw *number_nopw_it***

If nopw is specified, the first *number_nopw_it* iterations will be performed without process window simulations. This means that everything related to process window conditions (wafer cost functions, process window tolerances, and so on) will not work for the first iterations.

Description

This optional command controls the number of iterations of MPC movement to perform on the design. During each iteration, each fragmented edge in the design is moved by a multiple of the step_size keyword's **val** argument in microns, according to the simulations models.

Note that if you specify both max_iterations and OPC_ITERATION in the setup file, the number of iterations specified in OPC_ITERATION takes precedence.

In this example, Calibre nmMPC will perform 3 non-process window iterations, followed by 4 more process window iterations, for a total of 7 iterations:

```
max_iterations 8 nopw 3
OPC_ITERATION 7
```

Examples

Run 7 iterations without process window conditions and 2 with:

```
max_iterations 9 nopw 7
```

max_smoothing_range (LITHO CLMPC Only)

MPC denseopc_options Commands

Specifies the local range along edges used for smoothing. Larger values result in smoother output contours.

Usage

max_smoothing_range *val*

Arguments

- *val*

A required argument that specifies an arc length used for contour smoothing. The default value is 0.020 microns.

Description

This command, used specifically for a **LITHO CLMPC** setup file, sets the smoothing range of curvilinear edges. Larger values result in smoother edges.

Examples

```
LITHO FILE "clmpc.recipe" [
    denseopc_options clmpcOptions {
        algorithm 1
        max_smoothing_range 0.025
        max_angle_tolerance 40
        max_iterations 2
        feedback -0.8 -0.6
        image ebeam_model ebeam0 etch_model etch0
    }
    setlayer mpcOut = clmpc mpcIn MAP mpcIn OPTIONS clmpcOption
]
mpcOut = LITHO CLMPC poly MAP mpcOut FILE "clmpc.recipe"
```

max_iter_movement

[MPC denseopc_options Commands](#)

Limits edge movement distance for an MPC iteration.

Usage

max_iter_movement *val*

Arguments

- *val*

Specifies the maximum edge movement distance, expressed in microns. This value must be greater than or equal to 0. The default distance is Nyquist/2.

Description

This optional command controls the maximum distance an edge can be moved during an iteration of MPC. Use this keyword to:

- improve MPC stability
- control convergence

max_opc_move

MPC denseopc_options Commands

Specifies maximum total distance all fragments are allowed to be moved by the entire MPC run.

Usage

max_opc_move {*val* | *vin* *vout*}

Arguments

- ***val***

Specifies the maximum edge movement distance in either direction (inside and outside the shape). This value must be ≥ 0 . The default distance is 80 nm.

- ***vin* *vout***

The ***vin*** option specifies how much the edge is allowed to move inside, while ***vout*** specifies how much it can move outside the shape. This value must be ≥ 0 . The default distance is 80 nm.

Description

This optional command controls the total distance a fragment is allowed to move (inside or outside the polygon) during biasing for the entire nmMPC run. Specifying the smallest value possible can improve performance because it will speed up mrc_rule checking.

measure_metric_mode

Specifies an average metric to be used for all fragment operations.

Usage

measure_metric_mode average

Arguments

- **average**

Specifies an average metric is to be applied to fragments instead of min or max metrics.

Description

Fragments near corners typically use the max or min metric, although spline re-targeting may require a different metric. When this command is specified, fragments use an average metric for all operations and normal computation is deactivated.

mpc_contour_search_radius

[MPC denseopc_options Commands](#)

Overrides default contour search distance during EPE calculations.

Usage

mpc_contour_search_radius *val*

Arguments

- *val*

A required argument that specifies a non-zero positive value (in microns) that overrides the default search distance used to locate contours during EPE calculation. The default value is $3 * \text{ebeam diff_length}$.

Description

This command extends the default search radius. Calibre nmMPC typically uses a default distance value to search for contours during EPE calculation: $3 * \text{ebeam diff_length}$.

mpc_fragment_type_projection

[MPC denseopc_options Commands](#)

Projects fragment types from opc layers to correction layers for multilayer MPC.

Usage

```
mpc_fragment_type_projection opc_layer_name... {correction_layer_name... |  
  all_correction_layers}
```

Arguments

- ***opc_layer_name...***

A required argument that specifies the name of the opc layer specifies one or more opc layers containing the fragment types to be projected onto one or more corresponding correction layers. More than one layer can be specified.

- ***correction_layer_name...***

A keyword in a required argument pair that specifies the name of the correction layer where fragment types are to be projected from the opc layer. More than one layer can be specified.

- ***all_correction_layers***

A keyword in a required argument pair that specifies that the fragment types from the opc layer(s) be applied to all correction layers.

Description

This keyword is used while performing multilayer MPC in Calibre nmMPC. In multilayer MPC, one or more opc layers define the mask target and one or more correction layers must be specified to achieve the mask target. Jog and corner fragment types are determined across the opc layers. The fragments on correction layers do not always have the same fragment type as the corresponding fragment on the opc layer.

This keyword instructs the Calibre nmMPC engine to modify fragment types on the correction layers by using the types from corresponding fragments on the opc layers. One or more opc layers must be specified that define the source layers from which to copy fragment information. This information is then projected onto the corresponding fragments in correction layers. For convenience, you can specify *all_correction_layers* to project the fragments onto all correction layers. This keyword can only be specified once.

mpc_poi_measurement (LITHO CLMPC Only)

MPC denseopc_options Commands

Optimizes EPE measurement time for CLMPC by reducing EPE measurements.

Usage

```
mpc_poi_measurement -range range_in_um -angle angle_in_degrees
```

Arguments

- **-range range_in_um**

A required argument that ensures that the distance between two consecutive EPE measurement locations is not more than *range_in_um*.

- **-angle angle_in_degrees**

A required argument that specifies an angle tolerance (in degrees) to ensure that measurement locations are closer in high curvature regions and, conversely, ensures those regions have closely-placed measurement locations.

Description

EPE measurement sites are also referred to as points of interest (POI). In CLMPC, EPE measurement is optimized by measuring EPE on a reduced set of fragments. Based on the arguments of this command, EPE measurement can be skipped for sites on some fragments. This command reduces the fragment set by qualifying EPE measurement locations using range (the distance between two consecutive EPE measurement locations) and angle (to ensure measurement locations are closer in high curvature regions). The fragments skipped for EPE measurement are assigned an EPE computed by interpolating EPEs of neighboring fragments.

Examples

Example 1

```
mpc_poi_measurement -range 0.020 -angle 15
```

mpc_preserve_angles (LITHO CLMPC Only)

MPC denseopc_options Commands

Preserves edge angles in CLMPC movement.

Usage

```
mpc_preserve_angles -edgeLen edge_length_in_micron -angleTol  
    angle_tolerance_in_degrees [-octangular] [-smoothJog]
```

Arguments

- **-edgeLen *edge_length_in_microns***
A required argument that specifies a length (in microns) that qualifies an edge for angle preservation. Any edge longer than this value is considered for angle preservation.
- **-angleTol *angle_tolerance_in_degrees***
A required argument that specifies an angle tolerance value (in degrees) to identify octangular edges. This option also identifies fragments that belong to the same input edge.
- **-octangular**
An optional keyword that, when specified, considers octangular edges only for angle preservation.
- **-smoothJog**
An optional keyword that, when specified, smooths any jogs retained for angle preservation.

Description

In CLMPC, octangular edges (edges that are horizontal, vertical, or any 45-degree increment) become skewed due to jog removal. This command is used to preserve the angle of these edges in CLMPC movement.

Examples

Example 1

```
mpc_preserve_angles -edgeLen 0.4 -angleTol 1.0 -octangular -smoothJog
```

mpc_pseudo_nyquist

MPC denseopc_options Commands

Sets the Nyquist value for MPC flows.

Usage

mpc_pseudo_nyquist *val*

Arguments

- *val*

A required argument that sets the Nyquist value in MPC flows in microns. If this value is not user-specified the Nyquist value is set using one of the following conditions:

- The EBEAM component is present in the model: The smallest EBEAM kernel size is used as Nyquist value.
- Only the VEB model is present: The smallest VEB Gaussian kernel is used as the Nyquist value.

All other fragmentation related parameters (including line end, space end, corner fragment) are derived from the Nyquist (fragmentation unit) value similar to Calibre nmOPC fragmentation.

Description

A Nyquist value represents the largest sampling distance of aerial intensity that does not introduce aliasing errors. When determining a basic unit of length for fragmenting polygon edges into fragments (the fragmentation unit), Mask Process Correction uses the Nyquist value and scale factor:

fragment unit (frgu) = **scale** x Nyquist

This fragmentation unit is then used to derive other fragment parameters such as line end, space end, corner fragments. For Calibre nmMPC, the Nyquist value is specified by the mpc_pseudo_nyquist command.

Examples

```
mpc_pseudo_nyquist 0.06
```

mpc_sites_control

[MPC denseopc_options Commands](#)

Specifies distance measurement sites are to be placed away from corners.

Usage

mpc_sites_control -min_cornerdist *distance*

Arguments

- **-min_cornerdist *distance***

A required argument that specifies the distance (in microns) that measurement sites are to be placed away from corners.

Description

The mpc_sites_control command specifies the distance (specified in microns) that measurement sites are to be placed away from corners. This command works only if [mpc_sites_mode](#) is set to 1. A corner is any non-collinear fragment in the neighborhood of a given fragment. By default, all non-collinear fragments are qualified as corners. However, if [jog_ignore_metric](#) is set, then fragments shorter or equal to the jog_ignore_metric value are ignored with regard to EPE measurement site placement.

mpc_sites_mode

[MPC denseopc_options Commands](#)

Specifies the site placement algorithm.

Usage

mpc_sites_mode {0 | 1}

Arguments

- **0 | 1**

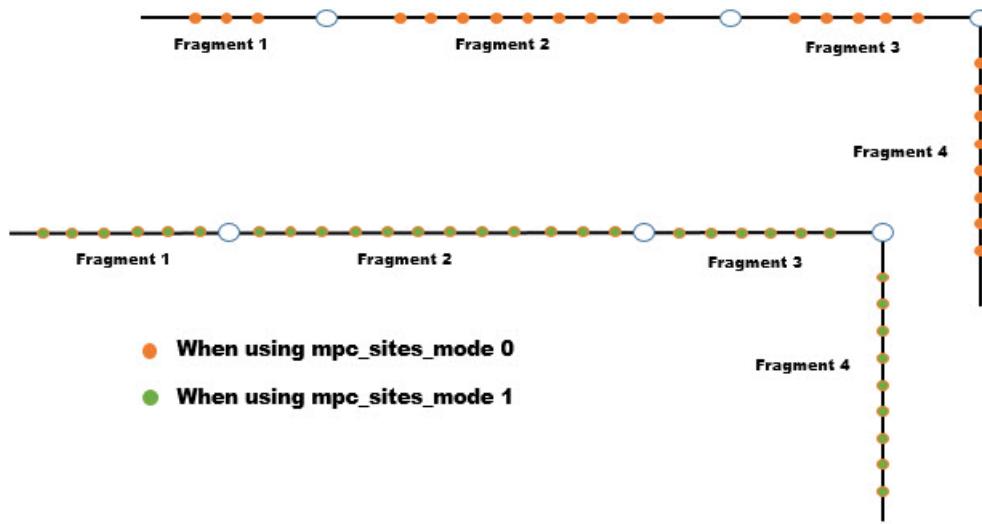
A required argument set that specifies if an alternate site placement algorithm is to be used for advanced nodes to produce improved accuracy at corner fragments. If set to 1, the alternate algorithm is used. If set to 0 (the default), the default algorithm is used.

Description

This command specifies an alternate algorithm for site placement with better accuracy at corner fragment sites. Typically used for advanced nodes, the [mpc_sites_control](#) command is enabled when mpc_sites_mode is set to 1.

When using a very small value for epe_spacing (for example, 10 nm) and mpc_sites_mode is set to 0, sites are not placed all the way across fragments; there are always certain lengths that are blocked out that do not contain EPE sites. This is illustrated in the following figure (the red dots indicate site placement).

Figure 5-38. Site Placement Using mpc_sites_mode



The reason for the block-out lengths is that mpc_sites_mode 0 (the default setting) does not distinguish between fragment ends and edge ends (for example, a corner). For all fragments, a range derived from the smallest e-beam kernel size is used as a block-out length for placing EPE

measurement sites. This behavior applies to MPC only and is different from site placement behavior in OPC.

When mpc_sites_mode 1 is set, the algorithm distinguishes between fragment ends and real corners and only uses a block-out length around corners. In this mode, the block-out length is not derived from model parameters implicitly, but is set using the command mpc_sites_control -min_cornerdist *value*.

In this mode, sites are placed the entire length of fragments unless a corner is encountered. Corners smaller than the jog_ignore_metric value are not treated as corners. This is illustrated in [Figure 5-38](#).

mpc_skip_fragments

[MPC denseopc_options Commands](#)

Skips specified fragments from EPE measurement computation.

Usage

```
mpc_skip_fragments -iterationStart N -iterationEnd M -epeLimit convergence_EPE  
-searchRange range
```

Arguments

- **-iterationStart *N***

A required argument that specifies the initial MPC iteration (as an integer value) where fragments are skipped for EPE measurement.

- **-iterationEnd *M***

A required argument that specifies the ending MPC iteration (as an integer value) where fragments are skipped for EPE measurement.

- **-epeLimit *convergence_EPE***

A required argument that specifies the EPE tolerance limit to identify fragments that are to be skipped from EPE measurement calculations.

- **-searchRange *range***

A required argument that specifies the range (in microns) to search for fragments and qualifying adjacent fragments to be skipped for EPE measurement calculations. The search range is applied to either side of the fragment.

Description

This command specifies that for a specified range of MPC iterations, fragments that have an EPE less than or equal to a specified tolerance (for any fragment along with adjacent fragments within a specified search range) are omitted from EPE measurement computation.

By default, if a fragment is found to be within tolerance, it is moved for that iteration and frozen thereafter. If the environment variable CALIBRE_MPC_SKIP_FRAGMENT_MODE is set to 1, then the fragment is frozen in the same iteration where its EPE is within the specified tolerance.

Examples

Example 1

```
mpc_skip_fragments -iterationStart 2 -iterationEnd 6 -epeLimit 0.0005 \  
-searchRange 0.050
```

mpc_use_nearest_epe

Selects the nearest possible valid EPE for fragments that do not have a well-defined EPE.

Usage

mpc_use_nearest_epe {on | off}

Arguments

- **on | off**

When enabled, Calibre nmMPC uses the nearest valid EPE for fragments that do not otherwise have a well-defined EPE. The default is off.

Description

This command is used to specify EPE for fragments that have conditions that prevent well-defined EPE calculations. This is performed by selecting the nearest valid EPE.

mrc_rule

MPC denseopc_options Commands

Used to create external and internal spacing constraints.

Note

 Using MRC constraints in MPC is possible but not recommended. If mask level corrections are limited by MRC constraints because the MPC input data is not MRC clean, optical performance of the mask can be affected and you might detect this only after wafer evaluation.

Instead, Siemens EDA recommends that you enforce MRC clean output during OPC runs and allow unrestricted shape correction through MPC based on a calibrated mask model.

Usage

```
mrc_rule {external | internal | enclosure}
layer1 [{inside | outside | not_inside | not_outside} reglayer]
[layer2 [{inside | outside | not_inside | not_outside} reglayer]] '{'
    [[projecting [dist] | not_projecting [dist]]
     use d1 {metric | d2}
     [{length len use d1 {metric | d2}}...]]...
'}'
"}
```

The curly braces ({{}}) with quotes ("") around them are meant as literals (without the quotes).

Arguments

- **external | internal | enclosure**

Specifies the check type. These allow specific mask constraint rules to be applied to fragments between two layers. Any opc, correction, and visible layers can be used.

By default, layers defined on the same mask will always be compared using the MRC rule that applies. When *layer1* and *layer2* are layer names, the rule is applied to polygon edges, which may consist of multiple fragments or be equal to a single generated jog between two fragments.

The enclosure rule restrains fragment movement if it violates the specified enclosure rule. For example:

```
mrc_rule enclosure tags_on_enclosed tags_on_enclosing {
    use num
    ...
}
```

where:

- tags_on_enclosed — Specifies contact or via layer fragments that are enclosed by the enclosing layer.
- tags_on_enclosing — Specifies the enclosing layer fragments.
- use *num* — Measures the enclosure between tags_on_enclosed and tags_on_enclosing
- ***layer1*** [{inside|outside|not_inside|not_outside} *reglayer*]
Specifies the first layer of the check. Optional specification of a region of *layer1* for which this MRC rule is valid. You can optionally specify a condition that applies to fragments that meet one of the following conditions for the specified *reglayer*: completely “inside” the layer, completely “outside” the layer, completely “not_inside,” or completely “not_outside.”
- ***layer2*** [{inside|outside|not_inside|not_outside} *reglayer*]
Optional specifies the second layer of the check. If not specified, the first layer is also used as the second layer. You can optionally specify a condition that applies to fragments that meet one of the following conditions for the specified *reglayer*: completely “inside” the layer, completely “outside” the layer, completely “not_inside”, or completely “not_outside.” When using several mrc_rule statements, the last statement in the sequence should be without an inside/not_inside/outside/not_outside declaration.
- **projecting** [*dist*]
An optional keyword that instructs Calibre to measure the separation between two edges only when one edge projects onto the other edge. When specifying projecting and not_projecting, use the following guidelines:
 - Do specify a rule without one or the other. For example, the first rule in the following example conflicts with the other two, which can cause violations:

```
mrc_rule external m1 {  
    use 0.0750  
    projecting use 0.0825 opposite  
    not_projecting use 0.0725 euclidean  
        length 0.032 use 0.025  
}
```

The following does not produce conflicts:

```
mrc_rule external m1 {  
    projecting use 0.0825 opposite  
    not_projecting use 0.0725 euclidean  
        length 0.032 use 0.025  
}
```

You can optionally specify a projection distance limit (*dist*). If the projecting distance is $\leq dist$, then the fragment is considered not projecting. Note that the following example:

```
mrc_rule ...{  
    projecting 0.01  
    ...  
    not_projecting  
    ...  
}
```

is equivalent to:

```
mrc_rule ...{  
    projecting  
    ...  
    not_projecting 0.01  
    ...  
}
```

and they both mean that the common projecting length should be strictly larger than 0.01 for the edges to be considered projecting.

- **not_projecting [dist]**

An optional keyword that instructs Calibre to measure the separation between two edges only when neither edge projects onto the other edge. You can optionally specify a projection distance limit (*dist*).

- **use d1 {metric | d2}**

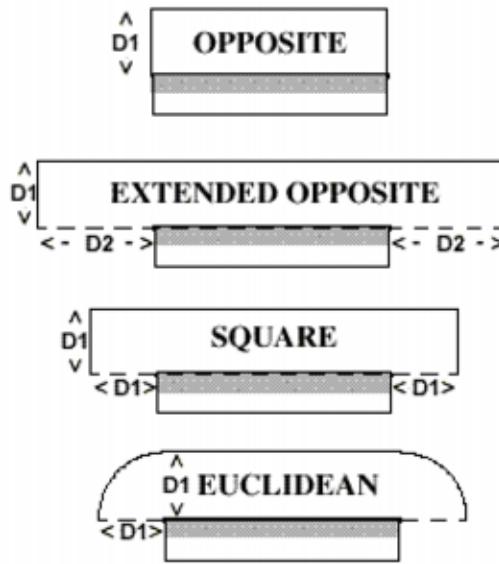
Specifies the minimum distance to allow for the rule. Valid distances are $\geq d1$. The value for *d1* is required and is the default value used if none of the length-constraint rules apply (specified by either *metric* or *d2*). You must specify either *metric* or *d2*.

The possible values for *metric* are **euclidean**, **opposite**, and **square**.

When *d2* is specified, it implies **opposite extended** with *d2* as the extension value.

[Figure](#) shows the measurement regions created for each of the different metrics, and the dimensions used with each.

Figure 5-39. The Four Metrics



- length *len*

An optional keyword that specifies a length criteria, *len*. Typically, “length” keywords specified before “use” keywords indicate that the subcheck is applied only when the length criteria is met by the shortest of the two edges being compared.

Description

The mrc_rule command block allows you to specify external/internal constraint checks between layers based on the distance between fragments as well as fragment length.

The rules establish conditions to find edges for correction (or to prevent correction in certain cases). If an mrc_rule “passes”, then the edges are classified by the constraint.

For example, if you set up a rule check to identify a set of edges as a jog, typically, you do not want corrections to be applied to those edges to make the jog even larger. So if the rule passes, then the edges are flagged as a jog and MPC does not move the edges out.

On the other hand, you could have a check stating that if there are certain length constraints met, you want to move the edges outward, then in that case, MPC moves the edges as a result of the rule.

Examples

The mrc_rule command block allows the flexibility to perform a internal/external check operations for a variety of unique situations.

One Layer Constraint

The following is a simple one layer constraint example:

```
mrc_rule m1 { use 0.0525 }
```

When length is specified, it is applied to edges with length \leq to the input value.

```
mrc_rule external m1 { use 0.0525
    length 0.032 use 0.025
    length 0.002 use 0
}
```

Two Layer Constraint

The mrc_rule command can be used to control spacing to SRAF layers. The following example uses length to set a rule for SRAF ends.

```
mrc_rule external m1 sraf { use 0.0725
    length 0.040 use 0.06
}
```

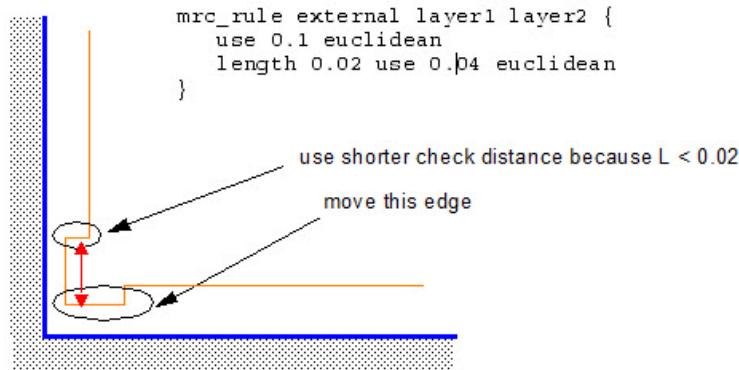
Area Constraints

The mrc_rule allows you to specify rules for specific areas. Marker layers are used to define regions. Valid choices are inside, not_inside, outside, not_outside, and the marker layers are hidden type.

```
mrc_rule external poly not_outside SRAM_MARKER {
    use 0.060
    length 0.025 use 0.030
}
mrc_rule external poly outside SRAM_MARKER {
    use 0.065
    length 0.025 use 0.030
}
```

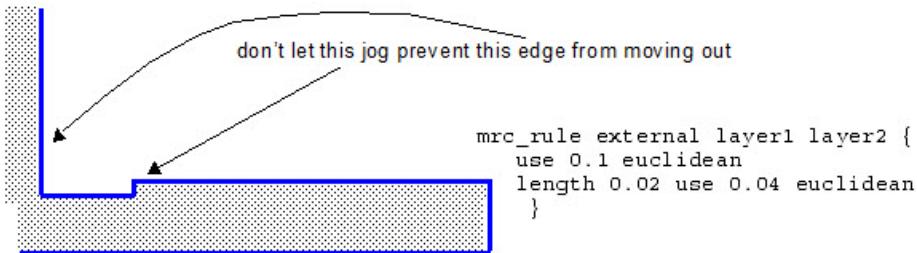
Moving an Edge from a Concave Corner

The following example allows an edge at a concave corner to move outwards.



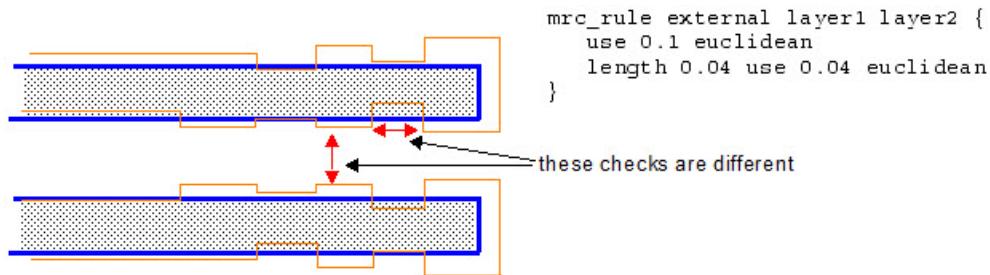
Jogs

In this example, a rule is set to ignore a small jog on the input layout when preventing an edge from moving during MPC.



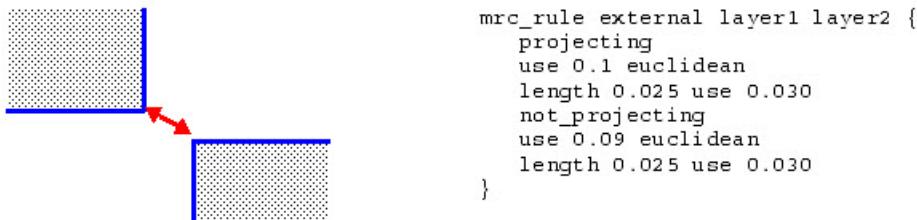
Notches and Feature-To-Feature Checks

In the following example, a distinction is made between notches and feature-to-feature checks.



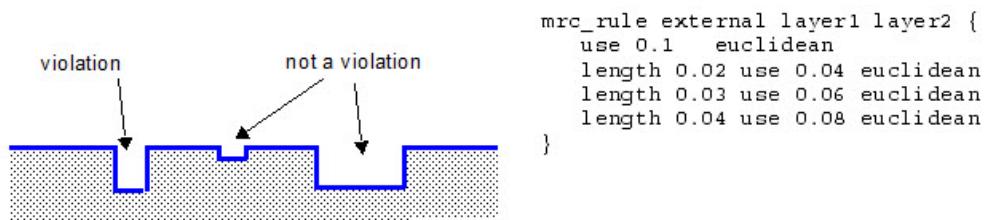
Corner-to-Corner Checks

The NOT_PROJECTING option can distinguish features such as corner-to-corner checks, which are not projecting onto each other, as in the following example.



Stair-Stepping

The following example implements a “stair-stepped aspect ratio” of 2:1 when features are below a certain size.



no_opc_region

MPC denseopc_options Commands

Creates a “non-opc” layer that designates areas where edges will not be moved by MPC (areas that intersect the “non-opc” layer).

Usage

```
no_opc_region
layer
```

Arguments

- *layer*

Specifies a polygon layer where fragments that touch this layer will not be moved by MPC.

Description

An optional command that specifies a “do not MPC” layer where fragments that touch the specified layer are not moved during MPC. This defines a region for which any fragments which are topologically not outside will be turned off for MPC, and therefore will not move. The default is an empty no_opc_region.

Though this command restricts the movement of fragments in the layer, no_opc_region does not prevent simulation occurring on that layer.

Examples

```
no_opc_region poly1
```

opc_grid_multiplier

[MPC denseopc_options Commands](#)

Refines the correction grid by a specified factor.

Usage

opc_grid_multiplier {on | off}

Arguments

- **on|off**

Specifies the refinement factor to set the correction grid. Setting opc_grid_multiplier to “off” sets the grid to a multiplier of 1. The default is on.

Selecting on means that the highest integer opg_grid_multiplier that results in an internal precision of less than or equal to 4000. For example, with opc_grid_multiplier set to on:

PRECISION 100 means the highest integer opc_grid_multiplier is 400

PRECISION 1000 means the highest integer opc_grid_multiplier is 40

PRECISION 4000 means the highest integer opc_grid_multiplier is 10

PRECISION 8000 means the highest integer opc_grid_multiplier is 5

Description

An optional command that sets correction grid to a finer setting by a specified factor. This can help reduce inconsistencies resulting from small differences that accumulate into large ones over many iterations. The final displacement of manhattan edges will automatically snap to [step_size](#).

Examples

```
opc_grid_multiplier on
```

retarget_layer

MPC denseopc_options Commands

Defines a new target layer, leaving the fragmentation on the opc layers.

Usage

```
retarget_layer layer... [{emulate | spline} [concave_adjacent val] [convex_adjacent val]]
```

Arguments

- *layer*

Specifies a new target layer(s). The default is no retarget layer.

- emulate

An optional flag that is used for target smoothing in which the input retarget layer is a smooth contour (generated by the [setlayer curve](#) command). This is similar to the spline option, except that it allows retarget edges to differ from the original Calibre nmMPC run. This helps reduce ripples near corners especially at concave corners. The setlayer curve command produces a more realistic target, which often results in less aggressive corner correction. However, you must still carefully tune your fragmentation and Calibre nmMPC recipes.

Note

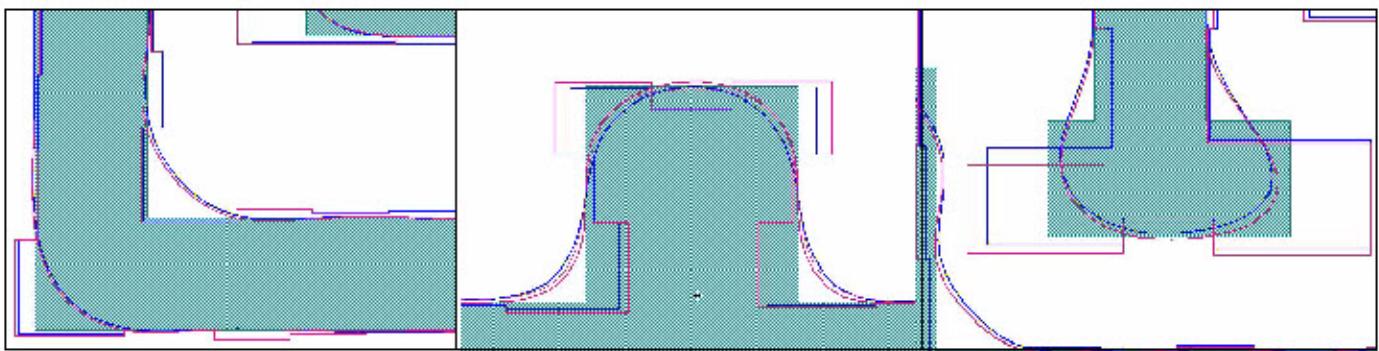


The emulate option is particularly useful in scenarios where the spline option incorrectly handles the retarget edges and MPC might not even converge.

In general, retargeting should not modify the opc layer by more than the minimum feature width of MPC. Also, Calibre nmMPC will attempt to correct a retarget layer (you will need to ensure that PVbands do not worsen because of it). The emulate option uses a continuous angle-weighted distance function, which means there is no limitation to degree changes.

Because spline searches for the nearest curve target segment the emulate option is best used where the curve target lies significantly outside the opc target. For example, when the curve is outside a line end, the nearest curve target segment for spline is a segment at the corner, rather than at the line end tip. This results in a sub-optimal site placement slanted to the line end fragment but perpendicular to the curve target. In this same situation, emulate considers the intersection angle and picks the optimal segment at the line end tip for site placement, perpendicular to the line end.

[Figure 5-40](#) shows the effects of using `retarget_layer emulate`.

Figure 5-40. Effects of retarget_layer emulate

- **spline**
An optional flag that is used for target smoothing in which the input retarget layer is a smooth contour (generated by the `setlayer curve` command). This differs from the emulate option in that it restricts retarget edges to stay close to the original biased edges.
- **concave_adjacent *val***
An optional keyword that uses the distance from a concave corner specified by *val* to tag fragments as concave adjacent.
- **convex_adjacent *val***
An optional keyword that uses the distance from a convex corner specified by *val* to tag fragments as convex adjacent.

Description

An optional command that defines a new target layer (which should be defined as a hidden layer with the `layer` command), but leaves the fragmentation on the opc layers. This allows a simple implementation of a pre-bias correction flow.

Retarget layers are used to separate the target from the layer being corrected and help when the target is unsuitable for correction. For example, layers with many jogs from biasing can prevent good fragmentation from occurring.

When attempting to create a retarget layer, note that:

- The original layer is the opc layer
- The correction target is the retarget layer
- EPE is measured from the image to the retarget layer
- Fragments on the opc layer are then moved based on the EPE measured from the retarget layer

Typically, if `retarget_layer` is not used, different metrics are used for different fragments. For example, on line ends, EPE is determined using the numerical max, typically the center point.

When retarget_layer is used, the contour of the spline is determined by averaging all EPEs on a fragment, except for the following exceptions:

- For fragments near convex corners that are NOT near concave corners, the numerical max is used (instead of averaging EPEs) to converge the image to an imaginary line tangent to the spline and parallel to the fragment.
- For fragments near concave corners that are NOT near convex corners as well, the numerical min is used to converge image to an imaginary line tangent to the spline and parallel to the fragment.

[Figure 5-41](#) and [Figure 5-42](#) illustrate how EPE is measured to determine how a fragment will be moved.

Figure 5-41. EPE Measurements at Corners (Example 1)

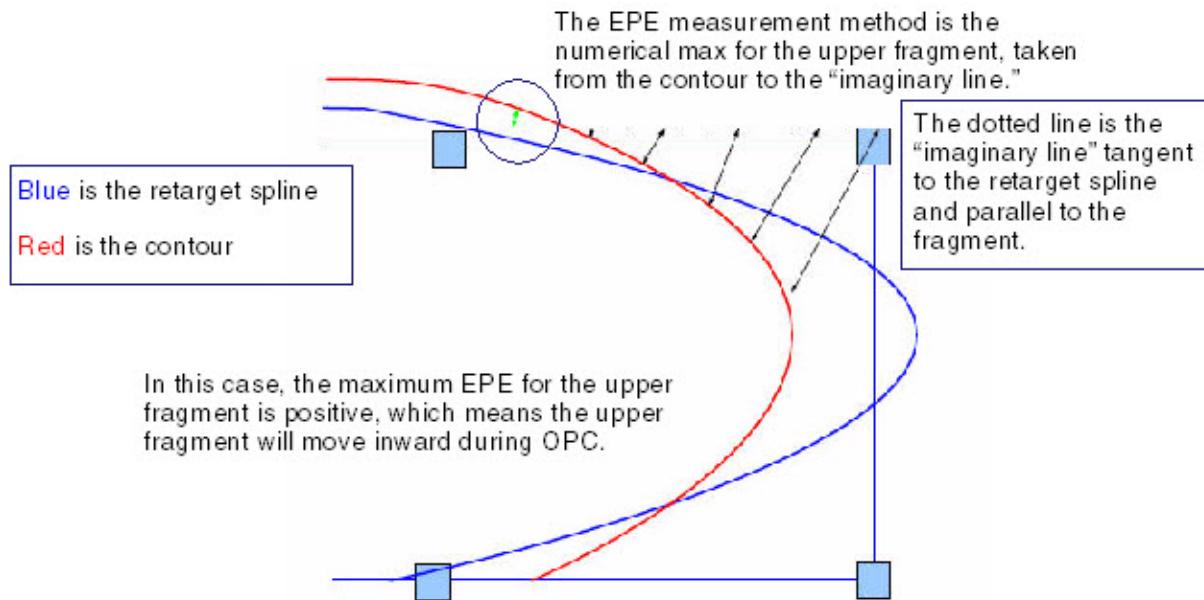
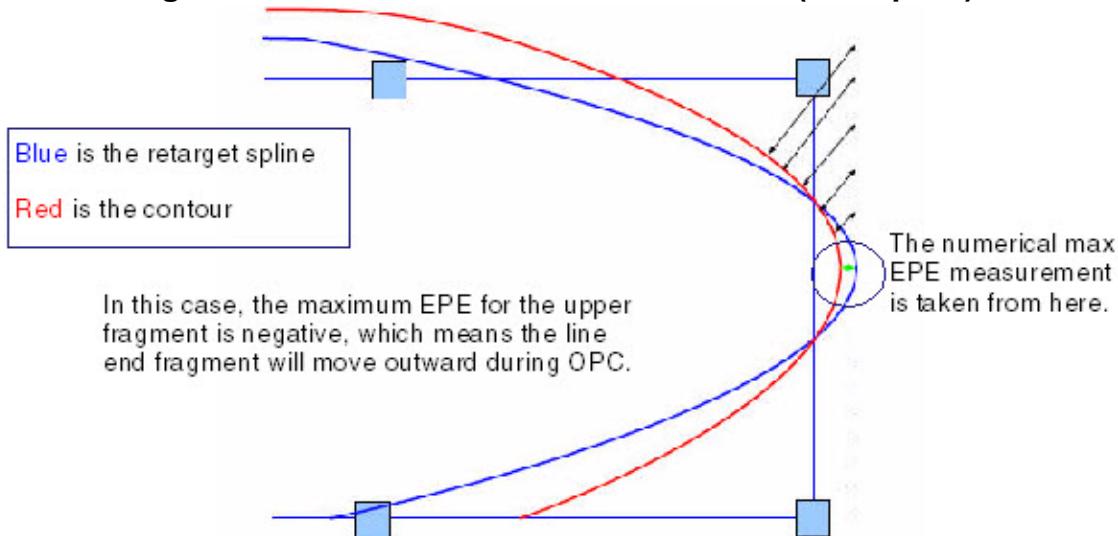


Figure 5-42. EPE Measurements at Corners (Example 2)



Examples

In this example, the original drawn layer is the layer whose fragments move during correction.

```
layer POLY      visible ... # original poly target
layer PREBIASED hidden ... # red layer above
retarget_layer  PREBIASED
```

In this example, fragments at specified distances from corners are tagged as convex adjacent and concave adjacent.

```
retarget_layer L1 spline concave_adjacent 0.05 convex_adjacent 0.08
```

scale

MPC denseopc_options Commands

Used to resize fragmentation lengths according to a scale multiplier.

Usage

scale *factor*

Arguments

- *factor*

Specifies the scaling factor. The range is 0.9 to 1.9. The default is 1.0.

Description

This command is a multiplier that scales the initial fragmentation lengths. The scale command determines the final unit of fragmentation. The fragmentation lengths are determined by a “pseudo-Nyquist” value (also referred to as “frgu”). This is defined as: frgu = scale * Nyquist (the Nyquist value is set by the [mpc_pseudo_nyquist](#) command). The default value for scale is 1.0.

The default values for Calibre nmMPC keywords that utilize the fragmentation unit are listed in the following table.

Table 5-13. Calibre nmMPC Fragment Command Defaults

Calibre nmMPC Command	Defaults
fragment_corner convex	fragment_corner convex not_end_adjacent (2+corner_control)*frgu 2*frgu 2*frgu 2*frgu end_adjacent 2*frgu 2*frgu 2*frgu 2*frgu
fragment_corner concave	fragment_corner concave not_end_adjacent (2+corner_control)*frgu 2*frgu 2*frgu 2*frgu end_adjacent 2*frgu 2*frgu 2*frgu 2*frgu
fragment_inter	interdistance 1.5*(?/NA) -num 0.5*(?/NA)/ ripplenlen -ripplelen 2*frgu
fragment_max	4*frgu
fragment_min	2*frgu
NEWTAG line_end space_end	4*frgu

sparse_ebeam_veb

Specifies a sparse Gaussian simulation for e-beam and Variable Etch Bias (VEB) simulators.

Usage

sparse_ebeam_veb {ebeam | both | off}

Arguments

- **ebeam | both | off**

A required argument set that activates or deactivates a sparse Gaussian simulation mode for e-beam and VEB simulators.

- **ebeam** — Specifies a sparse simulation mode for e-beam Gaussians and dense simulation for VEB Gaussians. However, even in sparse mode, some of the tiles and frames might be switched to dense mode if that appears to achieve better runtime for those tiles and frames.
- **both** — Enables sparse simulation for e-beam Gaussians and sparse and dense simulation for VEB Gaussians based on the edge density.
- **off** — Enables dense simulation for both e-beam and VEB Gaussians. This is the default setting.

Description

This command enables you to specify a sparse simulation mode for e-beam and Variable Etch Bias (VEB) Gaussians. By default, both are processed using a dense simulator. This command is valid only if [ebeam_model_load](#) has specified “sequence LUMPED”, enabling a lumped model mode. In this mode, instead of applying the e-beam and VEB models sequentially (which would require saving the intermediate results of both models at the expense of run time), both models are treated as one model calculating the signals of both models in a single step.

By default, Calibre nmMPC processes fragments in “dense mode”, simulating on a pixel-based grid. “Sparse mode” simulations are based on evaluation points and can be more efficient for low-density designs. If you have a design with a lower density of fragments, the sparse simulation may often have a shorter run time than dense mode. However, as more fragments are added (increasing density), run times increase. For dense designs, pixel-based calculations are more efficient.

step_size

[MPC denseopc_options Commands](#)

Sets the minimum edge movement distance for MPC.

Usage

step_size *val*

Arguments

- *val*

Specifies the minimum movement distance. The default is 1 dbu.

Description

An optional keyword that specifies smallest movements of an MPC fragment. This value is specified in user units and must be an integer multiple of the database unit.

Examples

```
step_size 0.00025
```

tile_fragmentation_only

MPC denseopc_options Commands

Deactivates all default fragmentation for an MPC run.

Usage

tile_fragmentation_only [0 | 1]

Arguments

- **0 | 1**

If set to 1, tile_fragmentation_only will turn off all default fragmentation and perform only fragmentation at tile boundaries. The default is 0.

Description

Deactivates all default fragmentation for an MPC run. As post-correction data already has fragmented edges, in most cases, no additional fragmentation is needed or warranted. The only fragmentation needed for tile boundaries is performed. Few, if any, new jogs should be introduced in the output.

You can also use tile_fragmentation only inside of a fragment_layer block. To do so:

1. Set fragment_layer_order to fully-ordered. This will allow types of fragmentation and their order to specified. If a form of fragmentation is not listed, it will not occur.
2. Set fragment_layer_order to full.
3. Set defaults for fragment_max and fragment_min.
4. Make a fragment_layer block for the layer to be fragmented that has normal settings.
5. Make a fragment_layer block for the layer not being fragmented that contains only “fragment_inter off.”

Examples

```
tile_fragmentation_only 1
```

version

MPC denseopc_options Commands

Specifies the nmMPC version number.

Usage

```
version
{
N
| {
major.minor
}}
```

Arguments

- *N*
Specifies the version number. Currently, only version 1 is supported.
- **major.minor**
Specifies the Calibre release version (for example, **2010.3**).

Description

A required keyword that specifies the version number of the nmMPC algorithm to use. Since this is a required parameter, all setup files contain this keyword. This version applies only to the MPC algorithm and not the hierarchical database constructor.

Over time, changes in setting defaults take place. Set this to 1 to have the tool use the defaults for the current version (the version of the Calibre executable that is currently running).

Set this to a particular Calibre version to use the default settings for that release. No versions earlier than 2009.2 are supported.

Examples

```
version 2009.3
```

This command specifies that the default settings for the 2009.3 release version of the MPC algorithm be used.

```
version 1
```

This command specifies that the defaults for the current running version of Calibre be used (it may or may not be 2009.3). Note that specifying “version 1” can produce changes in MPC behavior if defaults have changed between releases.

Performance

No impact

MPC Custom Tcl Scripting Commands

The following table lists the nmMPC custom scripting commands.

Table 5-14. nmMPC Custom Scripting Commands Summary

Command	Description
DISPLACEMENT	Sets FROZEN/FIXED status, displacement, desired displacement, and displacement limits for all fragments in the given tag.
FEEDBACK	This command sets the feedback for one or more fragments.
fragalign	Aligns fragment break points on a specified layer.
FRAGMENT delete	Deletes either a single fragment, or set of fragments.
FRAGMENT modify	Shifts the ordered endpoint of a given fragment by the given amount while preserving minimum resulting fragment lengths.
FRAGMENT split	Splits a single or set of fragments into two fragments by the ratio specified.
FRAGMENT_EPE_MEASURE_METHOD	Allows you to select between minimum, maximum, or average values of EPE for a tag set.
FRAGMENT_MOVE	Moves all fragments to their desired displacements simultaneously, while obeying mask constraints.
MRC	Creates MRC constraints, used in conjunction with mrc_rule.
MRC_RULE (Custom Scripting Command)	Creates external and internal spacing constraints based on tag sets.

Table 5-14. nmMPC Custom Scripting Commands Summary (cont.)

Command	Description
NEWTAG all	Creates a fragment set consisting of all fragments.
NEWTAG blocked	Checks for fragments that had movement blocked by an MRC rule.
NEWTAG cd	Measures the CD and outputs a given bin.
NEWTAG complete	Outputs all fragments on the edge (if the edge meets length and corner constraints).
NEWTAG displacement	Returns a set of fragments whose displacements are in the given range which is specified in user units.
NEWTAG edge	Searches an edge for fragments within a given set of constraints.
NEWTAG epe	Outputs EPE for verification.
NEWTAG external internal enclose enclosing	Computes distance as internal, external, or enclosure from fragments in the <i>fromLayer</i> to fragments in the <i>toLayer</i> .
NEWTAG fragment	Searches for fragments within a given set of constraints.
NEWTAG line_end space_end	Computes pre-defined line-end and space-end definitions.
NEWTAG neighbor	Creates a new set of fragments containing the previous, next, or both neighbors.
NEWTAG sequence	Creates new tag set(s) holding fragments from a sequence of edges.
NEWTAG topological	Performs the indicated topological operation on the given input layers.
NEWTAG vertical horizontal all_angle 45_angle	The output set contains fragments whose edge slopes match the specification.
OPC_ITERATION	Performs a “standard” iteration as defined by all setup file user settings.
OUTPUT_MEASUREMENT_LOCATIONS	Outputs EPE measurement locations.
OUTPUT_OPCT	Outputs MPC layout of specified layer.
OUTPUT_SHAPE fragment	Outputs one or more simple shapes.

Table 5-14. nmMPC Custom Scripting Commands Summary (cont.)

Command	Description
TAG and or subtract	Boolean set operations on multiple tag sets.

DISPLACEMENT

MPC Custom Tcl Scripting Commands

Sets FROZEN/FIXED status, displacement, desired displacement, and displacement limits for all fragments in the given tag.

Usage

```
DISPLACEMENT tag [-applyLimitIn | -applyLimitOut | -applyProperty]
[{-fixedOffset | -fixedOffsetIncr | -hintOffset | -hintOffsetIncr} val]
[-freeze | -unfreeze] [-limit {inner outer} | clear]
[-moveLimitsFromTarget] [-unlock]
```

Arguments

- ***tag***
A required argument that specifies the tag set to operate upon.
- **-applyLimitIn**
An optional keyword that sets the inner displacement limit to the value annotated on ***tag***. The annotation value must be ≤ 0 . Positive values generate a warning, and the inner limit is set to 0.
This option cannot be combined with -applyProperty or -applyLimitOut.
- **-applyLimitOut**
An optional keyword that sets the outer displacement limit to the value annotated on ***tag***. The annotation value must be ≥ 0 . Negative values generate a warning, and the outer limit is set to 0.
This option cannot be combined with -applyProperty or -applyLimitIn.
- **-applyProperty**
An optional keyword that sets the desired displacement from property annotations on the fragments in ***tag***. The properties are treated as absolute bias values.
This option cannot be combined with -applyLimitIn or -applyLimitOut.
- **-fixedOffset *val***
An optional argument that sets the displacement to *val*, even if it violates mask constraints. The fragment is moved and its desired displacement is set to the current (after move) displacement.
- **-fixedOffsetIncr *val***
An optional argument that sets the displacement to *current_disp + val*, even if it violates mask constraints. The fragment is moved and its desired displacement is set to the current (after move) displacement.

- -freeze | -unfreeze

Optional arguments that allow or disallow the fragments to be moved by FRAGMENT_MOVE or OPC_ITERATION.

Note



Do not manually unfreeze fragments that were frozen by commands such as sraf_print_avoidance. Some commands algorithmically freeze sensitive fragments to preserve litho quality.

When using -freeze, be sure the coordinates are in multiples of database units. After MPC, fragments are snapped to the layout grid and the final position may be different.

- -hintOffset *val*

An optional argument that sets the desired displacement to *val*. The fragment is not moved until the next call to FRAGMENT_MOVE.

- -hintOffsetIncr *val*

An optional argument that sets the desired displacement to *current_disp + val*. The fragment is not moved until the next call to FRAGMENT_MOVE.

- -limit *inner outer*

-limit clear

An optional argument that sets limits in user units for the inner or outer displacements for all fragments in the tag set. The inner value must be ≤ 0 , and the outer value must be ≥ 0 .

- -moveLimitsFromTarget

An optional keyword that sets values for the correction layer based upon the opc layer. When -moveLimitsFromTarget is specified, all other parameters except for **tag** are ignored. This keyword is relevant only to correction layers.

For correction fragments in **tag**, displacement limits are set using displacement limits of mapped opc fragment. If no limits are set on the opc fragment, no limits are set on correction fragment. If the correction layer fragment is already outside the displacement limits of the opc fragment, no restrictions are placed on the displacements of correction fragment.

- -unlock

An optional keyword for use in Calibre® nmBIAS™ setup files. It unlocks fragments locked by BIASRULE ... -lockAll.

Tcl Result

None

Description

This command sets FROZEN/FIXED status, displacement, desired displacement, and displacement limits for all fragments in the given tag set.

After moving, the actual displacement might differ from the desired displacement because of various setup file commands, such as MRC constraints (mrc_rule), max_opc_move, and so on.

DISPLACEMENT does not conform to max_iter_movement. The max_iter_movement command only applies to an MPC iteration-directed edge movement.

Precedence

These are the rules by which conflicting arguments are resolved:

1. -applyProperty and -moveLimitsFromTarget both cause all other optional arguments to be ignored.
2. -limit, -applyLimitIn, and -applyLimitOut take precedence over offsets. They are mutually exclusive.
3. -fixedOffset, -fixedOffsetIncr, -hintOffset, and -hintOffsetIncr are mutually exclusive, and take precedence over -unlock, -freeze, and -unfreeze.

For example:

```
DISPLACEMENT tag1 -freeze -fixedOffset 0.20 -limit -0.01 0.01
```

sets the displacement to 0.01, moves the fragment, and then freezes it. (Fragments in tag1 that are already frozen only have their displacement set.)

You can get different behavior depending on whether hintOffset or fixedOffset is called first, and whether FRAGMENT_MOVE is called or not. The behavior of hintOffset and fixedOffset can be described as follows:

- hintOffset just sets the desired displacement, which only affects the next FRAGMENT_MOVE.
- fixedOffset sets the desired displacement *and* moves the fragment to its new location.

Then:

- If you specify hintOffset followed by fixedOffset, MPC behaves as if hintOffset was not specified.
- If you specify fixedOffset followed by hintOffset, OPC behaves as if hintOffset was not specified (because hintOffset without FRAGMENT_MOVE doesn't affect the locations of the fragments).
- If you specify fixedOffset followed by hintOffset, then FRAGMENT_MOVE behaves as if fixedOffset was not specified.

Property Annotations

For -applyProperty, -applyLimitIn, and -applyLimitOut, it is your responsibility to ensure the value in the property annotation is appropriate. Because the values are computed at runtime, it is not possible to check them when the setup file is parsed.

Numeric annotations can be computed with NMBIAS_MEASURE_TAG and NEWTAG expression.

Tcl Results

None

Performance

O(N), where N is the number of fragments in the input.

Rating = fast

Examples

Example 1

The following lines freeze the tag set line_ends and perform one iteration of MPC on all other fragments. The fragments of the line_ends tag set are then unfrozen and have their desired displacement increased by 0.02.

```
DISPLACEMENT line_ends -freeze
OPC_ITERATION 1
DISPLACEMENT line_ends -unfreeze -hintOffsetIncr 0.02
```

Example 2

This example calculates context-specific values for fragments in the tag_short group and saves the values by creating annotated tag sets. DISPLACEMENT reads the values in the annotated tag sets to create variable limits for the fragments.

```
NMBIAS_MEASURE_TAG m1 -tag tag_short -overlapLayer ovl
    -measurement_type enclosing -dist 0.2 -aout tag_short_encl

NEWTAG expression tag_short {
    subexpression nbr_enc1 = neighbor(tag_short_encl, corner:convex,
        resolution: max);
    expr(nbr_enc1) * 1.2} -aout tag_short_outer

NEWTAG expression tag_short {
    subexpression nbr_enc2 = neighbor(tag_short_encl, corner:convex,
        resolution: min);
    expr(nbr_enc2) * -0.8} -aout tag_short_inner

NEWTAG expression tag_short_encl {
    property(tag_short_encl) * 0.1
} -aout tag_short_disp
```

```
DISPLACEMENT tag_short_outer -applyLimitOut
DISPLACEMENT tag_short_inner -applyLimitIn
DISPLACEMENT tag_short_disp -applyProperty
FRAGMENT_MOVE
```

Unlike the case of constant limits or offsets, each annotation-based setting must be in a separate DISPLACEMENT call.

FEEDBACK

MPC Custom Tcl Scripting Commands

Sets feedback factor (used to determine edge movement).

Usage

FEEDBACK [-keep] [-control] **tag** *feedbackvalue* [*feedbackvalue...*]

FEEDBACK [-keep] [-control] **gid** *feedbackvalue*

FEEDBACK -remove **tag**

Arguments

- **-keep**

Locks the last *feedbackvalue* to the user-set value. The MPC algorithm will not adjust the user-set feedback. To unlock, issue FEEDBACK again without -keep switch.

- **-control**

When the user-set feedback has expired or cancelled, instead of starting with the MPC system default feedback values, use the last user-set feedback value as seed.

- **gid**

Specifies a generalized fragment ID, which can be a single fragment ID or a tag variable name.

- **tag**

A tag variable name.

- **feedbackvalue**

Sets the feedback for the next iteration on the fragments indicated. This value should be between -2 and +2. Typically, the feedback would be a negative number between -1 and 0.

Each *feedbackvalue* corresponds to an MPC iteration (for example, MPC iteration N uses *feedbackvalue N*). If the MPC iteration count is more than the given number of *feedbackvalues*, then further iterations use the last given feedback value. When -keep is in effect, the last *feedbackvalue* is not adjusted between iterations for further iterations.

- **-remove**

Removes the feedbacks previously assigned to the tag set.

TCL Result

None

Description

This command sets the feedback for one or more fragments. The feedback will be used to calculate fragment movement on the next opc iteration. On the subsequent iteration, the feedback is reset to the default value.

Setting a fragment's feedback to 0 will freeze it in its current location.

Performance

$O(N)$, where N is the number of fragments in the input.

For performance, recommend to use set entire fragment tag sets at once, instead of setting single fragment at a time.

Rating = fast

Examples

```
FEEDBACK line_end -0.2
FEEDBACK convex    -0.4
FEEDBACK gate      -0.6
OPC_ITERATION      5
```

fragalign

MPC Custom Tcl Scripting Commands

Aligns fragment break points on a specified layer.

Usage

```
fragalign layer tag maxmove max_move maxpolywidth max_width
    minedgelen minedgelength [priorityTag inTagName] [copy] [alignToCorners]
    [unalignedTag outTagName] [allowDelete]
```

Arguments

- ***layer***
Specifies the layer to align (fragments on layers that do not match are not touched).
- ***tag***
Specifies the tag name of fragments to align (use “all” to align all fragments, otherwise only a particular tag set is considered for alignment and all other fragments are left alone).

Note

 Since all of the fragments to be aligned must be contiguous, it is recommended that you specify all of the tags to be aligned at one time and not to use multiple calls to fragalign. This might mean combining some tags together before calling fragalign.

- **maxmove *max_move***
Specifies the maximum distance that a fragment break is to be moved from its original location.
- **maxpolywidth *max_width***
Specifies the maximum distance across a polygon for which alignment is to be performed. This is measured perpendicular to the direction that a breakpoint is moved during alignment. To prevent unintended alignment, this should be set to the minimum acceptable value.
- **minedgelen *minedgelength***
Specifies the shortest edge that may be created during alignment. This is not the same as the value set by fragment_min.
- **priorityTag *priorityTagName***
An optional parameter that specifies a set of tags whose edges take priority over all others. These fragments will not be lengthened or shortened, but other fragments may be aligned to them. Only the non-priority fragment will be lengthened or shortened, the priority fragment will remain the same length. If an attempt is made to align two priority fragments with one another, then neither will be moved.

```
newTag corners -how ORTAGS convex_corner concave_corner
fragalign opcLayer all 0.005 0.005 0.040 0.025 corners
```

There is no default priorityTag value, so if none is specified, there are no priority fragments.

- **copy**

Duplicates fragment breaks on one side of a polygon to the other side of the polygon if the constraints *max_width* and *minedgelength* are met.

- **alignToCorners**

By default fragments opposite a corner are not aligned to a corner. This type of alignment will generally be done by interfeature fragmentation, if needed. In cases where that is not possible, alignToCorners can be specified where needed. This option aligns the fragments opposite a corner to a corner, as long as the search constraints and minedgelength are met.

Note

 It is possible to specify alignToCorners with priorityTag. Fragments which are in the priorityTag tag set will not be aligned to corners (the priorityTag takes priority over alignToCorners).

- **unalignedTag *unalignedTagName***

After fragalign is finished, some fragments may still not be aligned. Fragments that were not aligned can be output to the tag set *unalignedTagName*. For a fragment to be in this tag set, it must be alignable (within the maxmove and maxpolywidth constraints) and one of its endpoints must not line up with the fragment across the polygon. If a fragment has both endpoints aligned, or is too far away to ever be considered for alignment, it will not be in this tag set.

- **allowDelete**

This is available when fragalign is used with the “copy” option. It deals with a very specific case where alignment is needed, but not possible: one side of a polygon has promoted fragments which cannot be moved, so only the opposite side can be moved. If alignment absolute must be achieved, and only one side can be adjusted, then it may be necessary to delete a fragment from the one side which can be adjusted. This option allows a deletion to occur, if the delete will result in correct alignment afterwards.

Description

This command aligns fragment break points on a specified layer with tag name so that fragment breaks across a feature are in line.

A fragment break lines up with its neighbor across the polygon if it is within $+/- \text{max_move}$ when projected across the polygon. It does not need to be moved more than *max_move* and the polygon is no wider than *max_width* between the two break points. No edges will be created less than *minedgelength* in length.

The break points to be aligned are generally lined up on the midpoint between the two breaks. This may not be possible in some cases without violating *minedgelength*. Under those conditions, other points may be tried in an attempt to get an alignment without violating

minedgelength. If this cannot be done without violating *minedgelength* for either fragment, then neither end point will be moved.

In some cases, fragment alignment may desirable, but some fragments should not be changed (they should only used to move other fragments). When this occurs, a tag name should be specified for priorityTag to give priority to edges whose break point should not be moved. For example, a fragment next to a corner may be critical, and should not be lengthened or shortened:

```
newTag corners -how ORTAGS convex_corner concave_corner
fragalign opcLayer all maxmove 0.005 maxpolywidth 0.040 minedgelen 0.025
priorityTag corners
```

This permits most fragments to be aligned to the midpoint between fragments. The fragments on corners are not changed, but the fragment across the polygon from them may be lengthened or shortened to line up with end of a corner fragment.

Note the following limitations for fragalign:

- Only Manhattan fragments may be aligned. All other angles, including 45 degree angles, are ignored.
- Fragments opposite a corner are not aligned by default.
- Fragment breaks will not be lined up if aligning either side will violate *minedgelength*.

Examples

The following example finds fragments on the opc layer which are on lines greater than 65nm wide and further than 65nm from other features. All fragments which met both of these requirements will be aligned. This approach uses less CPU time than trying to align twice, and is more effective at identifying long fragment spans which can be aligned:

```
newTag wideLayer -how INTERNAL > 0.065 OPPOSITE INPUT1 opc_layer all
newTag wideGap -how EXTERNAL > 0.065 OPPOSITE INPUT1 opc_layer all
newTag alignTags -how andTags wideLayer wideGap
fragalign opc_layer alignTags maxmove 0.005 maxpolywidth 0.040
minedgelen 0.025
```

FRAGMENT delete

MPC Custom Tcl Scripting Commands

Deletes fragments.

Usage

FRAGMENT delete {pt1 | pt2 | both} *gid*

Arguments

- **pt1 | pt2 | both**

Specifies which endpoint of the fragment to delete. This can be either one endpoint (pt1 or pt2) or both endpoints (both).

- ***gid***

Specifies a generalized fragment ID, which can be a single fragment ID or a tag variable name.

TCL Result

None

Description

This command can delete either a single fragment, or set of fragments. Due to the complexity of this operation, it is advised that you use this command with extreme caution (it is recommended for experts only). This command will invalidate all previously computed tag sets.

Performance

$O(N)$ where N is the number of fragments to delete.

Rating = slow, use rarely

FRAGMENT modify

MPC Custom Tcl Scripting Commands

Shifts the ordered endpoint of a given fragment.

Usage

FRAGMENT modify {pt1 | pt2} *gid shiftby min min_neighbor*

Arguments

- **pt1 | pt2**

Specifies a first or second endpoint (pt1 or pt2) to modify (in clockwise order).

- **gid**

Specifies a generalized fragment ID, which can be a single fragment ID or a tag variable name.

- **shiftby**

Specifies the distance to move in user units. A negative value shortens the fragment, and a positive value lengthens the fragment.

- **min**

Specifies the minimum length of the resulting fragment after modify cannot be smaller than this value. The fragment endpoint will move as much as possible while avoiding a length violation.

- **min_neighbor**

When modifying a fragment endpoint, this argument will not allow the neighboring fragment length to become smaller than this value. The operation will move the endpoint as much as possible, while preserving this length of the neighbor.

Note

 Specify only numerical values for *shiftby*, *min*, and *min_neighbor*. Entering keywords such as tag names will generate an error.

TCL Result

None

Description

This command shifts the ordered endpoint of a given fragment by the given amount while preserving minimum resulting fragment lengths.

Performance

O(N) where N is the number of fragments to modify.

Rating = medium

FRAGMENT split

MPC Custom Tcl Scripting Commands

Splits fragments.

Usage

```
FRAGMENT split {gid location min | -3fragments gid dist_pt1 dist_pt2 min}  
[{gid2  
location2 min2 | -3fragments gid2 dist_pt1_2 dist_pt2_2  
min2}]...
```

Arguments

- ***gid***
Specifies a generalized fragment ID, which can be a single fragment ID or a tag variable name.
- ***location***
Specifies a relative distance between two points at which to split the fragment. The value is between 0 and 1 (setting ***location*** to 0.5 means splitting the fragment in half).
- ***min***
Specifies the minimum length of resulting fragments after the split. The operation will not split if it results in a length violation.
- **-3fragments**
Splits the fragment into 3 fragments (as opposed to 2). Note that in the -3fragments case, the parameter specifying the split location is distance-based rather than a percentage.
- ***dist_pt1***
Specifies the distance from point1 to split.
- ***dist_pt2***
Specifies the distance from point2 to split.

TCL Result

None

Description

This command splits a single fragment (or set fragments) into two fragments by the ratio indicated. The ratio indicates the percentage distance between two points (pt1 and pt2) to place the split.

When multiple sets of [***gid location min***] are given, ***gid*** must be tag names and duplicate fragments take on the first tagset parameters.

Caution

 Due to the complexity of this operation, it is advised that you use this command with extreme caution. This command will invalidate all previously computed tag sets.

Performance

$O(N)$ where N is the number of fragments to split.

Rating = slow, use rarely

FRAGMENT_EPE_MEASURE_METHOD

MPC Custom Tcl Scripting Commands

Allows you to select between minimum, maximum, or average values of EPE for a tag set.

Usage

FRAGMENT_EPE_MEASURE_METHOD -tag *tagname* -epe *val*

Arguments

- **-tag *tagname***

Specifies a tag name.

- **-epe *val***

Specifies an EPE type: min, max, or ave.

TCL Result

None

Description

This command allows you to select between minimum, maximum, or average values for EPE for a tag set.

Performance

O(1), constant time to get single fragment.

Rating = fast

FRAGMENT_MOVE

MPC Custom Tcl Scripting Commands

Moves fragments.

Usage

FRAGMENT_MOVE [-freeze_unmoving] [-freeze_tag *tagname*]

Arguments

- -freeze_unmoving

An optional argument that temporarily freezes everything with the desired displacement.
This behavior is identical to DISPLACEMENT.

- -freeze_tag *tagname*

An optional argument that temporarily freezes all the fragments in *tagname*.

TCL Result

None

Description

This command moves all fragments to their desired displacements simultaneously, while obeying mask constraints. This command only moves fragments of a desired displacement differing from the current displacement.

There are two ways to set desired displacements:

- OPC_ITERATION
- FRAGMENT_SET_DISPLACEMENT (without the -force option)

Using either method requires a subsequent call to FRAGMENT_MOVE. The maximum movement per iteration is also enforced (the default is Nyquist/2), which may result in a fragment moving less than expected. Note that FRAGMENT_MOVE conforms to max_iter_movement when the command is used with OPC_ITERATION.

Performance

O(*T*), where *T* is the total number of fragments for all layers. This command is considered slow because it moves fragments while checking mask constraints. The mask constraint checking algorithm required non-trivial runtime.

Rating = slow

Examples

```
FRAGMENT_SET_DISPLACEMENT line_end 0.03
FRAGMENT_SET_DISPLACEMENT lea 0.02
FRAGMENT_SET_DISPLACEMENT convex_corner 0.015
FRAGMENT_SET_DISPLACEMENT concave_corner -0.015
# All goals set, now do the movement
FRAGMENT_MOVE
```

MRC

MPC Custom Tcl Scripting Commands

Establishes tag-specific adjustment of width or spacing between fragments after MPC movement.

Usage

```
MRC {internal | external} dist tag1 tag2 priority1 priority2 [opposite [extended_dist]] [projecting dist | not_projecting dist]
```

Arguments

- **internal | external**
Specifies an internal or external check.
- ***dist***
Specifies the minimum distance between fragments for this check.
- ***tag1 tag2***
For the check to apply to any 2 fragments, one fragment must be in the *tag1* set and the other must be in the *tag2* set.
- ***priority1 priority2***
Assigns a priority for which a fragment can move more, when the fragments are constrained. The fragments move in the following proportion:
 - The *tag1* fragment moves by $\text{priority1}/(\text{priority1+priority2})$
 - The *tag2* fragment moves by $\text{priority2}/(\text{priority1+priority2})$
- **opposite**
Optionally specifies the “opposite” measurement metric.
- **extended_dist**
Optionally specifies the “opposite extended” metric.
- **projecting *dist***
Optionally specifies that if fragments are projecting, use this *dist* as minimum distance.
- **not_projecting *dist***
Optionally specifies that if fragments are not projecting, use this *dist* as minimum distance.

TCL Result

None

Description

This command creates tag-specific mask rule constraints specification. It is used in conjunction with the built-in mrc_rule checks. Mask rule constraints created using the nmMPC custom

scripting capability do not affect the way other commands (like FRAGMENT_MOVE) behave (as opposed to the setup file mrc_rule statement). Instead, it is a stand-alone command that moves back fragments.

This command sets constraints for EXTERNAL/INTERNAL rule checks for specific tags. With MRC:

- You can control the MRC limits for each set of tags independently. You can specify from one tag to another tag.
- A “default” check can be defined for all structures on the same mask. Tags from exposures on different masks are not compared by default, but can be forced to be compared using MRC.
- You can use it in conjunction with the mrc_rule syntax (the mrc_rule syntax allows you to create length-specific checks between layers).
- Unlike mrc_rule, this command applies only after the edge movement is completed.

MRC enforces constraints by detecting edge violations and moves the edges back until the constraint is met. Thus MRC must always be declared following the FRAGMENT_MOVE command.

Performance

O(N) where N is the bigger of *tag1* or *tag2*.

Rating = medium

Examples

```
OPC_ITERATION
FRAGMENT_MOVE
MRC external 0.07 line_end line_end 1 1
```

Alternative Commands

The mrc_rule setup file keyword sets MRC rules between layers, with length-specific checks. This command is to be used for tag-specific checks.

MRC_RULE (Custom Scripting Command)

MPC Custom Tcl Scripting Commands

Creates external and internal spacing constraints based on tag sets.

Usage

```
MRC_RULE {external | internal} tag1 tag2 {""
{
  [[projecting [dist] | not_projecting [dist]]
  {
    [length len] use d1 {metric}
    ...
  }...
"}"]}
```

The curly braces ({{}}) with quotes ("") around them are meant as literals (without the quotes).

Arguments

- **external | internal**

Specifies the check type. These allow specific mask constraint rules to be applied to fragments specified in 2 tag sets.

- **tag1 tag2**

Specifies tag sets to be used by the MRC rule.

- **projecting dist**

An optional keyword that instructs Calibre to measure the separation between two edges only when one edge projects onto the other edge. You can optionally specify a projection distance limit (*dist*).

- **not_projecting dist**

An optional keyword that instructs Calibre to measure the separation between two edges only when neither edge projects onto the other edge. You can optionally specify a projection distance limit (*dist*).

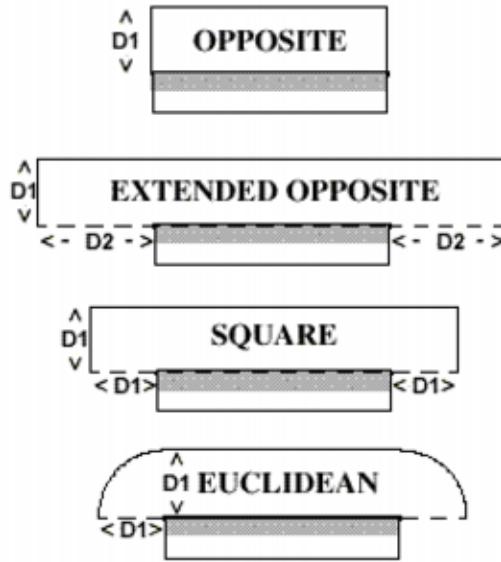
- **use d1 metric**

Specifies the minimum distance to allow for the rule. Valid distances are $\geq d1$. The value for *d1* is required and is the default value used if none of the length-constraint rules apply (specified by *metric*).

The possible values for *metric* are **euclidean**, **opposite**, **square**, or specify an extended opposite value.

[Figure](#) shows the measurement regions created for each of the different metrics, and the dimensions used with each.

Figure 5-43. The Four Metrics for Tag-Based MRC



- **length *len***

An optional keyword that specifies a length criteria, *len*. Typically, “length” keywords specified before “use” keywords indicate that the subcheck is applied only when the length criteria is met by the shortest of the two edges being compared.

Description

The MRC_RULE Tcl scripting command block allows you to add new MRC rules to the existing layer-based set of MRC rules (specified by the setup file command mrc_rule) using tag sets.

Performance

Very fast, but will slow down [FRAGMENT_MOVE](#) and [OPC_ITERATION](#).

NEWTAG all

MPC Custom Tcl Scripting Commands

Tags all fragments.

Usage

NEWTAG all *layer* -out *tag*

Arguments

- *layer*

Specifies an output layer to place into the output tag set. This argument currently accepts the layer index number (for example, 0, 1, 2, 3, and so on).

- **-out** *tag*

Specifies a name for the variable to hold the output tag set.

TCL Result

None

Description

Creates a fragment set consisting of all fragments.

Performance

O(N) where N is either the total number of fragments in the output tag set.

Examples

The following example gets all fragments on the layer L1:

```
NEWTAG all L1 -out T0
```

NEWTAG blocked

MPC Custom Tcl Scripting Commands

Tags fragments blocked by an MRC rule.

Usage

NEWTAG blocked *tin* -out *tag* [-outRest *tag2*]

Arguments

- ***tin***
Specifies the tag variable name to perform the check.
- **-out *tag***
Specifies a name for the variable to hold the output tag set.
- **-outRest *tag2***
An optional argument that creates a second tag set containing all fragments from ***tin*** that are not blocked.

TCL Result

None

Description

Outputs fragments that have movement blocked by an MRC rule. The output is valid only if **NEWTAG blocked** is called after **OPC_ITERATION** or **FRAGMENT_MOVE**.

Performance

$O(N)$ where N is the size of the set *tin*.

Examples

The following example checks fragments blocked by an MRC rule and tags them with “mrc_tag”.

```
OPC_ITERATION 1
NEWTAG blocked ALL -out mrc_tag
```

NEWTAG cd

MPC Custom Tcl Scripting Commands

Measures CDs from a contour.

Usage

```
newtag cd tag_in contourID {-out tag mode range constr}... [-outRest tag2]
```

Arguments

- ***tag_in***

Specifies an input set of fragments variable name.

- ***contourID***

Specifies a valid simulation contour ID.

- **-out *tag mode range constr***

A required set of arguments that specify the output. This argument set may appear more than once.

“**-out *tag***” specifies a name for the output tag set. The next part defines what fragments are included. The values for ***mode*** are listed following. “**range *constr***” is in user units.

- spacemin — Minimum CD space or negative number if invalid
- spacemax — Maximum CD space or negative number if invalid
- spaceav — Average CD space or negative number if invalid
- spaceexception — CD space exceptions
- widthmin — Minimum CD width or negative number if invalid
- widthmax — Maximum CD width or negative number if invalid
- widthav — Average CD width or negative number if invalid
- widthexception — CD width exceptions

- **-outRest *tag2***

An optional argument that creates a second tag set containing all fragments from ***tag_in*** that were not assigned to at least one ***tag***.

TCL Result

None

Description

Given a pre-simulated contour with the input ***contourID***, which takes on the values of IM0...IM32, this command measures the CD and outputs a given bin.

Performance

O(N) where N is the size of *tagin*

Rating = slow

Examples

The following example compares the runtime speed of two methods to perform the same check:

Method 1: Multiple NEWTAG cd declarations (3x slower):

```
NEWTAG cd TAG IM0 -out T1 spacemin < 0.05
NEWTAG cd TAG IM0 -out T2 widthav >= 0.05 < 0.06
NEWTAG cd TAG IM0 -out T3 widthmax >= 0.04 < 0.06
```

Method 2: Multiple -out declarations:

```
NEWTAG cd TAG IM0 -out T1 spacemin < 0.05 -out T2 widthav >= 0.05 < 0.06
-out T3 widthmax >= 0.04 < 0.06
```

NEWTAG complete

MPC Custom Tcl Scripting Commands

Outputs all fragments on the edge (if the edge meets length and corner constraints).

Usage

```
NEWTAG complete tin len constr [dist constr] [corner1 type] [corner2 type] [jog_tol dist]  
      -out tag [-outRest tag2]
```

Arguments

- ***tin***
A required argument that identifies the tag variable name referencing fragments to check.
- ***len constr***
A required argument that specifies the length constraint on the complete edge that contains the fragment.
- ***dist constr***
An optional argument that specifies a distance constraint for filtering the output fragments. The default distance is the interaction distance.
- ***corner1 type***
An optional argument that specifies the type constraint for the first corner. Valid types are convex, concave, no_corner, and any. The default is any.
- ***corner2 type***
An optional argument that specifies the type constraint for the second corner.
- ***jog_tol dist***
An optional argument that ignores (that is, tolerates) jogs less than or equal to *dist* when computing edge length for the **len** constraint.
- **-*out tag***
A required argument that specifies a name for the variable to hold the output tag set.
- **-*outRest tag2***
An optional argument that creates a second tag set containing all fragments from **tin** that did not satisfy the constraints.

TCL Result

None

Description

This command returns all the fragments belonging to an edge if any fragment in that edge satisfies the specified length and corner constraints.

This differs from NEWTAG edge, where all fragments on the edge are output only if the whole edge (not all the fragments in the edge, but the whole edge) satisfies the condition.

Performance

$O(N)$ where N is the size of *tin*.

NEWTAG displacement

MPC Custom Tcl Scripting Commands

Returns fragments within a specified displacement range.

Usage

NEWTAG displacement *tin* [-desired] {-out [-annotated] *tag range constr*}... [-outRest *tag2*]

Arguments

- ***tin***
A required argument that specifies the name of the tag set to classify into groups.
- **-desired**
An optional keyword that specifies the value used for displacement is the desired displacement instead of the current actual displacement.
- **-out [-annotated] *tag range constr***
A required argument set that can be specified multiple times but must occur at least once.
The set specifies that each fragment in the input tag whose displacement matches the constraint (in user units) is placed into the output *tag*.
The -annotated option causes the fragments of the tag set to be annotated with the displacement value.
- **-outRest *tag2***
An optional argument that creates a tag set containing all fragments from *tin* that did not satisfy any of the range constraints.

TCL Result

None

Description

Returns a set of fragments whose displacements are in the given range, specified in user units. For “binning” displacements into multiple tag sets, it is recommended that you define a single newtag displacement with multiple -out options rather than multiple calls to newtag displacement.

Performance

O(*N*) where *N* is size of input set *tin*

Rating = fast

Examples

The following compares the speed of two methods used to perform the same check:

Method 1: Multiple newtag displacement declarations (3x slower):

```
NEWTAG displacement TAG -out T1 range < 0.02
NEWTAG displacement TAG -out T2 range 0.02 >= < 0.04
NEWTAG displacement TAG -out T2 range 0.04 >= < 0.06
```

Method 2: Use multiple -out declarations:

```
NEWTAG displacement TAG -out T1 range < 0.02 -out T2 range >= 0.02 < 0.04
-out T3 range >= 0.04 < 0.06
```

NEWTAG edge

MPC Custom Tcl Scripting Commands

Searches an edge for fragments within a given set of constraints.

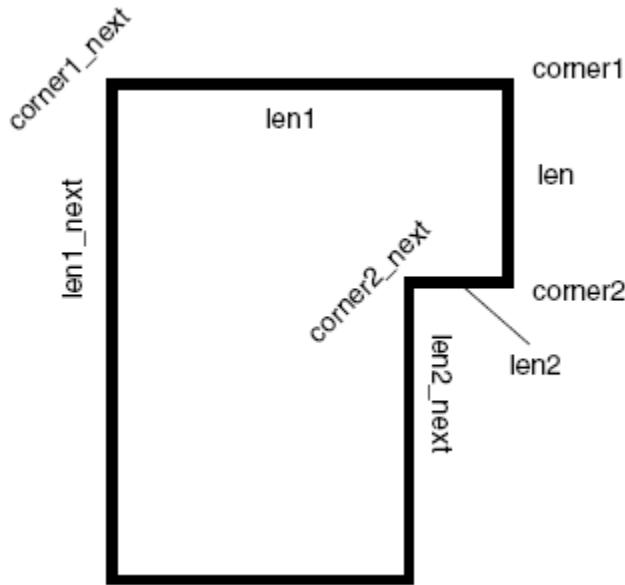
Usage

```
NEWTAG edge tin len constr [corner1 angle] [corner2 angle]  
[corner1_next type] [corner2_next type]  
[len1 constraint] [len1_next constraint] [len2 constraint] [len2_next constraint]  
[mcorner1 type] [mcorner2 type]  
[mlen constraint] [mlen1 constraint] [mlen2 constraint]  
[rdisp1 constraint] [rdisp2 constraint]  
[fragcount constraint] [fragcount1 constraint] [fragcount2 constraint]  
[measure {min | max | ave}] [-ordered | -unordered] [-jog_tol value] [-jog_tol_main value]  
-out tag [-outRest tag2]
```

Arguments

- ***tin***
A required argument that specifies the tag variable name to perform the calculations.
- ***len constr***
A required argument that specifies the length criteria for current edge. You can use comparison operators (<, >, <=, >=, and so on) to specify the criteria in *constr*. For example, len < 5.0.
- **corner1 *angle***
Specifies the corner for the first endpoint of the edge. The angle can be convex, concave, or an angle constraint such as >=90.
- **corner2 *angle***
Specifies the corner for the second endpoint of the edge. The angle can be convex, concave, or an angle constraint such as >=90.
- **corner1_next *type***
Specifies the corner type (one of convex or concave) for the first endpoint of the preceding edge. If not specified, the corner can be any type. Angle constraints are not accepted.
- **corner2_next *type***
Specifies the corner type (one of convex or concave) for the second endpoint of the following edge. If not specified, the corner can be any type. Angle constraints are not accepted.

Figure 5-44. Corner and Len Definitions



- **len1 constr**

An optional argument that specifies the length criteria for the previous edge. You can use comparison operators ($<$, $>$, \leq , \geq , and so on) to specify the criteria in *constr*. For example, $\text{len1} = 0.5$.

- **len1_next constr**

An optional argument like len1 except that it applies to the edge before the len1 edge.

- **len2 constr**

An optional argument that specifies the length criteria for the next edge. You can use comparison operators ($<$, $>$, \leq , \geq , and so on) to specify the criteria in *constr*. For example, $\text{len2} < 0.5$.

- **len2_next constr**

An optional argument like len2 except that it applies to the edge after the len2 edge.

- **mcorner1 type**
mcorner2 type

Optional arguments that apply to the new context of an edge after it and its neighbors have moved. The arguments function similarly to corner1 and corner2 and can be specified individually or together.

Unlike corner1 and corner2, valid values do not include angle constraints. Also, either neighbor may be “mcorner1” unless -ordered is set.

- *mlen constraint*
mlen1 constraint
mlen2 constraint

Optional arguments that apply to the new context of an edge after it and its neighbors have moved. The arguments function like **len**, **len1**, and **len2** in all other respects.

- *rdisp1 constraint*
rdisp2 constraint

Optional arguments that specifies a relative displacement constraint measured at the corners. The relative displacement is measured between a moved edge and the new locations of the edges at corner1 (*rdisp1*) or corner2 (*rdisp2*).

Displacement is determined by moving out from the corner until finding a fragment with a different displacement. If the **mcorner** and **mlen** constraints are met, displacement is then measured. For example, if from corner1 three fragments all moved inwards, but the fourth fragment did not, this creates a concave corner. (The concave corner may not be the same as the original corner2.) The displacement value is measured between the edge formed by the first three fragments and the fourth, unmoved, fragment, if **mlen** and **mcorner** constraints are met.

The *rdisp* measurement is not affected by the measure setting.

- *fragcount constraint*
fragcount1 constraint
fragcount2 constraint

Optional arguments that constrain the fragment count based on the number of fragments on an edge before fragment movement. The **fragcount** argument applies to the current edge, while **fragcount1** and **fragcount2** apply to the corner1 and corner2 adjacent edges respectively. The options can be specified individually or in combination.

- **measure {min | max | ave}**

Specifies the measurement method. The method can be changed using **FRAGMENT_EPE_MEASURE_METHOD**.

- **-ordered | -unordered**

Specifies the amount of directional dependency. The default behavior (neither flag set) is to check the pattern match in both directions, but with directional dependency for determining corner1 and corner2.

-ordered — The pattern match is performed only in the clockwise direction.

-unordered — The pattern match is performed in both directions, eliminating the occasional directional dependency of the default.

- **-jog_tol *value***

An optional argument that ignores, or tolerates, jogs shorter than *value* when computing **fragcount1**, **fragcount2**, **len1**, and **len2** (that is, constraints on the neighbors of the primary edge). For **fragcount**, jogs are ignored unless two concave or two convex corners are specified. To ignore jogs on those as well, use **-jog_tol_main**.

You must specify either two identical corner constraints or a single constraining convex or concave length.

- **-jog_tol_main value**

An optional argument that ignores jogs for len and fragcount (primary edge measurements) in all cases. This is not recommended when tagging line ends and space ends.

- **-out tag**

Specifies that each fragment in the input tag that meets the criteria be placed into the output tag.

- **-outRest tag2**

An optional argument that creates a second tag set containing all fragments from *tin* that did not meet the criteria.

Description

This command performs a search for a fragment whose length and corner types match a constraint. The neighbor's lengths can also be considered.

Note

 Do not use unbounded NEWTAG edge constraints with corners. These are likely to lead to inconsistent MPC output.

The smallest angle that can be resolved is 0.01 degrees. Slight skew may not be detected.

Tcl Results

None

Performance

O(N) where N is the size of the set *tin*

Examples

```
NEWTAG edge ALL len < 0.065 corner1 convex corner2 convex len1 > 0.065 \
len2 > 0.065 -out MYLINEEND; # customized line end tag
```

NEWTAG epe

MPC Custom Tcl Scripting Commands

The output is used for verification.

Usage

```
NEWTAG epe tag_in [contourid] {-out tag mode range constr}... [-outRest tag2]
```

Arguments

- **tag_in**

A required argument that identifies the initial tag set of fragments.

- **contourid**

Specifies the ID of an output register containing the simulated contour. The value must be one of the predefined IM0...IM32 set.

This argument is optional when the mode is effective_epe, and required for all others.

- **-out tag mode range constr**

A required set of arguments that specify the output. This argument set may appear more than once.

“**-out tag**” specifies a name for the output tag set. The next part defines what fragments are included. The values for **mode** are listed following. “**range constr**” is in user units.

- epemin is the minimum EPE.
- epemax is the maximum EPE.
- epeav is the average EPE.
- epeexception specifies that fragments for which all the measurements are invalid are output.
- effective_epe is a combination of measurements using nominal image and process window conditions, wafer constraints, and any set pinch/bridge tolerances. It is the value that MPC uses to derive the fragment movement amount, and is set by OPC_ITERATION. *contourid* is not required with effective_epe, because this mode does not require a contour.
- epeopc **constraint** creates a tag whose EPE meets the constraint. If epeopc is used, EPE will be measured using the same method (ave, min, or max) used by MPC.

Note



When using **OUTPUT_SHAPE fragment** with NEWTAG epe, note that the OUTPUT_SHAPE generates a contour based on the EPE from the *prior* MPC iteration, while NEWTAG epe obtains its EPE from the *final* iteration of the MPC run.

- **-outRest tag2**

An optional argument that creates a second tag set containing all fragments from **tag_in** that were not assigned to at least one **tag**.

Description

The output of this command is used for verification. Given a pre-simulated contour with the input *contourID*, which takes on the values of IM0...IM32, this command measures EPE and outputs a given bin.

For “binning” into multiple tag sets, it is recommended that you define a single newtag definition with multiple -out options rather than multiple calls to this command.

Performance

O(N) where N is the size of *tagin*

Examples

The following example compares the runtime speed of two methods to perform the same check:

Method 1: Multiple -out declarations:

```
NEWTAG epe TAG IM0 -out T1 epeav < 0.02 -out T2 epeav >= 0.02 < 0.04  
-out T3 epeav >= 0.04 < 0.06
```

Method 2: Multiple newtag epe declarations (3x slower):

```
NEWTAG epe TAG IM0 -out T1 epeav < 0.02  
NEWTAG epe TAG IM0 -out T2 epeav >= 0.02 < 0.04  
NEWTAG epe TAG IM0 -out T2 epeav >= 0.04 < 0.06  
  
NEWTAG all poly -out tags_all  
NEWTAG epe tags_all -out tags_range effective_epe > -0.01 < 0.01  
NEWTAG epe tags_all -out tags_neg effective_epe < 0 -out \  
tags_pos effective_epe >= 0
```

NEWTAG external | internal | enclose | enclosing

MPC Custom Tcl Scripting Commands

Computes distance from one layer to another using one of three metrics.

Usage

NEWTAG {external | internal | enclose | enclosing} *from to metric*

[*-not_projecting* | *-projecting* [*all*] [*constr*]]
 [-perpendicular_only [-corner_block *length*]] [-moved] [-drc_style]
 [-shielding_check {none | simple | all}]
 {-out *tag range constr*} ... [-outRest *tag2*]

Arguments

- **external | internal | enclose | enclosing**

Specifies if the area to be measured is to be external, internal, enclose or enclosing.

Setting “enclosing” specifies that the layers or tags specified in *from* enclose layers or tags specified by *to*. For example:

```
NEWTAG enclosing metal1 contact1 ...
```

outputs fragments from metal1, where the metal1 to contact1 enclosure is within the given constraint.

Setting “enclose” specifies that tags or layers specified in *from* are **enclosed by** *to*. For example:

```
NEWTAG enclose contact1 metal1 ...
```

outputs fragments from contact1, where the metal1 to contact1 enclosure is within the given constraint.

- ***from to***

Specifies layers or tags to measure.

- ***metric***

Specifies a measurement metric: opposite, euclidean, or opposite extended *value*.

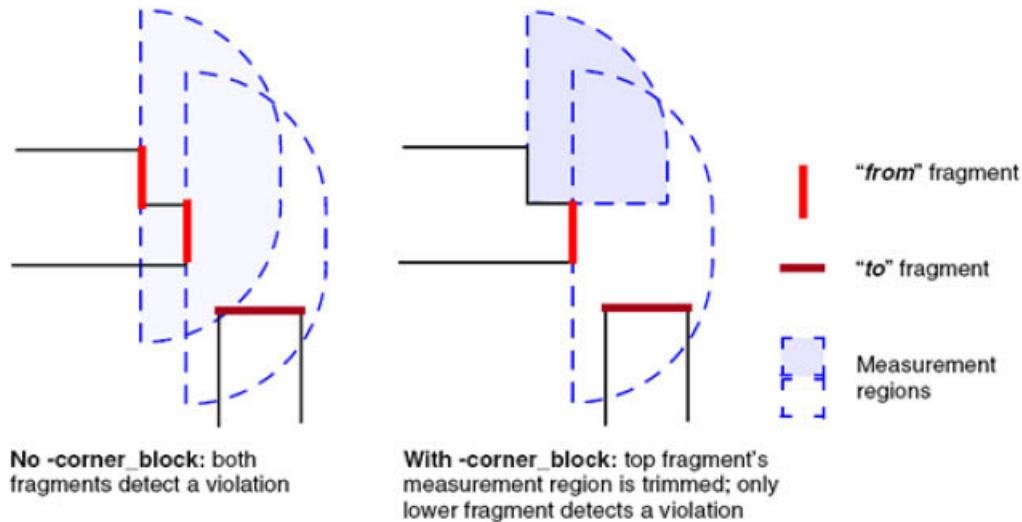
- **-perpendicular_only [-corner_block *length*]**

An optional argument that specifies measurement between *from* and *to* fragments that are at right angles to each other, such as for corner-to-corner measurements.

The -corner_block option can be used to further restrict what is measured. By default, if any part of a fragment is within the measurement region of the *from* fragment, the *from* fragment is returned in the output tag set, even if there are intervening fragments. When -corner_block is specified, a concave corner with neighbor $\geq length$ on the *from* fragment clips the region, giving a rectangular region on that side similar to the “opposite” metric.

In the following figure, the *from* fragments marked in red are in violation and are in the **-out** tag set.

Figure 5-45. Effect of -perpendicular_only and -corner_block



- **-moved**

An optional flag that measures to/from the moved fragments.

- **-drc_style**

Specifies that the MPC operation mimics a corresponding Calibre nmDRC operation. If present, the `not_projecting` condition is not satisfied if no fragments are found within the range constraint, and the `projecting` condition is not satisfied if projecting length is 0.

If no projecting constraint is specified, a fragment is classified if any fragments meet the range constraint. A DRC-style shielding check is performed as specified with the `shielding_check` mode.

- **-shielding_check {none | simple | all}**

Specifies the level of shielding check to be performed. Valid modes include the following:

`none` — No shielding check.

`simple` — Tag-level shielding check. This is the default.

`all` — Layer-level shielding check.

- **-not_projecting**

Keep (filter) only non-projecting fragments.

- **-projecting [constr]**

Keep (filter) only projecting fragments. If `constr` is given, then further filter fragments by keeping only fragments passing `constr`. The constraint checks for the length of the projection (not the fragment length).

- **-out tag range constr**

An argument set that must be specified at least once. The set specifies a name for the tag set of fragments that meet the range constraint. The constraint must be specified in user units.

This argument set must be specified after all other settings. A single NEWTAG command can specify multiple -out sets with different range constraints.

- **-outRest tag2**

An optional argument that creates a tag set of all *from* fragments that did not satisfy any of the **constr** conditions.

TCL Result

None

Description

This command computes distance as internal, external, or enclosure from fragments in the *from* layer or tagset to fragments in the *to* layer or tagset. This command places fragments whose distance meets the constraint into the output tag sets. More than one output tag set can be computed at the same for efficiency.

For “binning” into multiple tag sets, it is recommended that you define a single newtag definition with multiple -out options rather than multiple calls to this command.

When projection filtering is used, a fragment is filtered out (i.e. not included) when it violates the constraint with any fragment. Shielding is not considered during projection compares.

Performance

$O(N)$ where N is the number of fragments in the *fromLayer*.

Examples

The following example performs a simple check:

```
NEWTAG external L0 opposite -out dense range < 0.065 -out iso range >
0.065 < 0.2
```

The following example compares the runtime speed of two methods to perform the same check:

Method 1: Multiple newtag external declarations (3x slower):

```
NEWTAG external L1 L2 opposite -out T1 range < 0.1
NEWTAG external L1 L2 opposite -out T2 range >= 0.1 < 0.2
NEWTAG external L1 L2 opposite -out T3 range >= 0.2 < 0.3
```

Method 2: Multiple -out declarations:

```
NEWTAG external L1 L2 opposite -out T1 range < 0.1 \
-out T2 range >= 0.1 < 0.2 -out T3 range >= 0.2 < 0.3
```

NEWTAG fragment

MPC Custom Tcl Scripting Commands

Searches for fragments within a given set of constraints.

Usage

```
NEWTAG fragment tin len constr [corner1 angle] [corner2 angle]  
[corner1_next type] [corner2_next type]  
[len1 constraint] [len2 constraint] [len1_next constraint] [len2_next constraint]  
[tgname1 tag] [tgname2 tag]  
[mcorner1 type] [mcorner2 type] [mcorner1_next type] [mcorner2_next type]  
[mlen constraint] [mlen1 constraint] [mlen2 constraint]  
[mlen1_next constraint] [mlen2_next constraint]  
[mside1 constraint] [mside2 constraint] [mside1_next constraint] [mside2_next constraint]  
[rdisp1 constraint] [rdisp2 constraint]  
[measure {min | max | ave}] [-ordered | -unordered]  
-out tag [-outRest tag2]
```

Arguments

- ***tin***

A required argument that specifies the tag variable name to perform the calculations.

- ***len constr***

A required argument that specifies the length criteria for current fragment. You can use comparison operators (<, >, <=, >=, and so on) to specify the criteria in *constr*. For example, len < 0.05.

- ***corner1 angle***

Specifies the corner for the first endpoint of the fragment. The angle can be an angle constraint such as >=90 or one of no_corner, convex, or concave.

- ***corner2 angle***

Specifies the corner for the second endpoint of the fragment. The angle can be an angle constraint such as >=90 or one of no_corner, convex, or concave.

- ***corner1_next type***

Specifies the corner type (one of no_corner, convex, or concave) for the first endpoint of the preceding fragment. If not specified, the corner can be any type. Angle constraints are not accepted.

- ***corner2_next type***

Specifies the corner type (one of no_corner, convex, or concave) for the second endpoint of the following fragment. If not specified, the corner can be any type. Angle constraints are not accepted.

- *len1 constr*

An optional argument that specifies the length criteria for the previous fragment. You can use comparison operators ($<$, $>$, \leq , \geq , and so on) to specify the criteria in *constr*. For example, $\text{len1} = 0.05$.

- *len2 constr*

An optional argument that specifies the length criteria for the next fragment. You can use comparison operators ($<$, $>$, \leq , \geq , and so on) to specify the criteria in *constr*. For example, $\text{len2} < 0.05$.

- *len1_next constr*

An optional argument like *len1* except that it applies to the fragment before the *len1* fragment.

- *len2_next constr*

An optional argument like *len2* except that it applies to the fragment after the *len2* fragment.

- *tgname1 tag*
tgname2 tag

Optional arguments that further restrict the selection based on the tagging of the previous (*tgname1*) or next (*tgname2*) fragments when the shape is traversed in a clockwise direction when *-ordered* is set. (The default is *-unordered*, allowing neighbors to be evaluated in any order.) The arguments can be specified individually or together.

- *mcorner1 type*
mcorner2 type
mcorner1_next type
mcorner2_next type

Optional arguments that apply to the new context of a fragment after it and its neighbors have moved. (See “[Example 2: Relative Displacement](#)” on page 322.) The arguments function similarly to *corner1*, *corner2*, *corner1_next*, and *corner2_next* and can be specified individually or together. They cannot be specified with *-jog_tol*.

Unlike *corner1* and *corner2*, valid values do not include angle constraints. Also, these options may be evaluated in either order unless *-ordered* is set.

- *mlen constraint*
mlen1 constraint
mlen2 constraint
mlen1_next constraint
mlen2_next constraint

Optional arguments that apply to the new context of a fragment after it and its neighbors have moved. The arguments function similarly to *len*, *len1*, *len2*, *len1_next*, and *len2_next* and can be specified individually or together. They cannot be specified with *-jog_tol*.

- *mside1 constraint*
mside2 constraint

`mside1_next constraint`
`mside2_next constraint`

Optional arguments that combine the functionality of the mlen and rdisp options. Like them, these cannot be specified with -jog_tol.

If a corner evaluates to “no_corner”, then mside is equivalent to the relative displacement between the fragments; if it is a concave or convex corner, mside constraints apply to the length of the moved fragment.

- `rdisp1 constraint`
`rdisp2 constraint`

Optional arguments that specify a relative displacement constraint measured at the corners. The relative displacement is measured between a moved fragment and the new locations of the previous fragment (rdisp1) or the next fragment (rdisp2) when -ordered is set. (The default is -unordered, allowing neighbors to be evaluated in any order.) For example, if a fragment moved outward 10 nm and its neighbor moved outward 7 nm, the relative displacement is 3 nm.

- `measure {min | max | ave}`

Specifies the measurement method. The method can be changed using [FRAGMENT_EPE_MEASURE_METHOD](#).

- `-ordered | -unordered`

Specifies the amount of directional dependency. The default behavior (neither flag set) is to check the pattern match in both directions, but with directional dependency for determining corner1 and corner2.

`-ordered` — The pattern match is performed only in the clockwise direction.

`-unordered` — The pattern match is performed in both directions, eliminating the occasional directional dependency of the default.

- `-out tag`

Specifies that each fragment in the input tag that meets the criteria be placed into the output tag.

- `-outRest tag2`

An optional argument that creates a second tag set containing all fragments from `tin` that did not meet the criteria.

Description

This command performs a search for a fragment whose length and corner types match a constraint. The neighbor's lengths can also be considered.

The smallest angle that can be resolved is 0.01 degrees. Slight skew may not be detected.

Tcl Results

None

Performance

$O(N)$ where N is the size of the set *tin*

Examples

Example 1

The following example tags short fragments that form a corner with a long fragment.

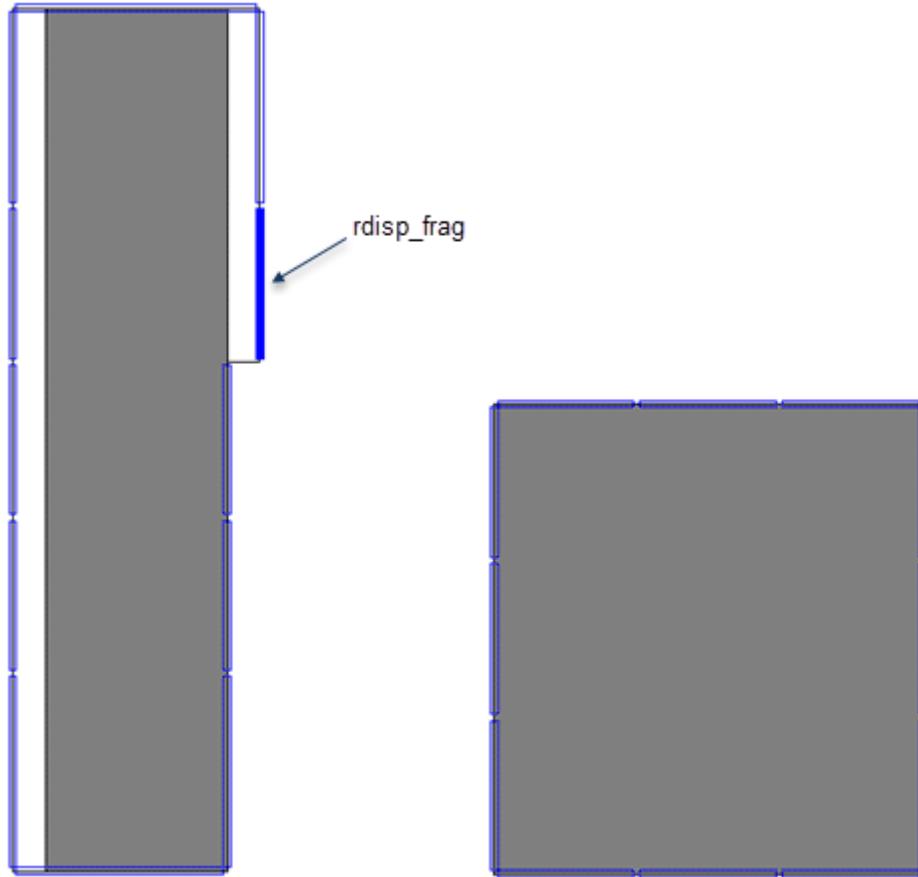
```
NEWTAG fragment ALL len < 0.01 corner1 convex len1 > 0.065 -out tag1
```

Example 2: Relative Displacement

This example demonstrates tagging fragments based on moved corners and relative displacements. The figure shows an original polygon (grey), the shape after biasing, and in blue the results of this command:

```
NEWTAG fragment all_frags len > 0.047 < 0.048 \
           mcorner1 no_corner mcorner2 convex \
           rdisp1 < 0.001 rdisp2 > 0.005 < 0.011 \
           -out rdisp_frag
OUTPUT_SHAPE fragment all_frags all_frags -moved 0.001 0.001
OUTPUT_SHAPE fragment rdisp_frag rdisp_frag -moved 0.001 0.001
```

Figure 5-46. Example of mcorner and rdisp



The fragment in blue has one endpoint on a collinear edge (mcorner1 no_corner) and the other on a convex corner (mcorner2 convex), with a relative displacement of 0 on the edge ($rdisp1 < 0.001$) and the other forming a short edge ($rdisp2 > 0.005 < 0.011$).

NEWTAG line_end | space_end

MPC Custom Tcl Scripting Commands

Computes line ends and space ends.

Usage

```
NEWTAG {line_end | space_end} tin -out tag [-outRest tag2]
```

Arguments

- **line_end | space_end**
Specifies either to tag the line-end or space-end fragments.
- ***tin***
Specifies the tag variable name to perform the calculations.
- **-out *tag***
Specifies an output name for the variable to hold the output tag set.
- **-outRest *tag2***
An optional argument that creates a tag set containing all fragments from *tin* that are not part of *tag*.

TCL Result

None

Description

This command is a fast method to compute pre-existing line-end and space-ends.

A line end is an edge that is greater less than or equal to lineEndLength, has two convex corners, and the lengths of the adjacent edges are greater than or equal to lineEndLength. A space end is the same except there are two concave corners.

Performance

O(N) where N is the size of *tin*

Examples

```
NEWTAG line_end all -out line_end
```

NEWTAG neighbor

MPC Custom Tcl Scripting Commands

Returns a set of fragments containing previous, next, or both neighbors.

Usage

NEWTAG neighbor {prev | next | both} *tin* *len constr* [corner *constr*] -out *tag*

Arguments

- **prev | next | both**

Specifies if the neighbor fragment returned is the previous fragment, the next fragment, or both neighbors.

- ***tin***

Specified tag variable name to perform the calculations.

- ***len constr***

Specifies the length constraint for the neighbor fragment.

- **corner *constr***

An optional argument that specifies a corner constraint for the neighbor fragment, either no_corner, convex, or concave. If omitted then the corner type can be any of the corner types (no_corner, convex, concave).

- **-out *tag***

Specifies the output name for the variable to hold the output tag set.

TCL Result

None

Description

Given a set of fragments, this command will create a new set which contains the previous, next, or both neighbors. The operation is ordered such that the previous fragment is the pt1 side of the fragment and the next fragment is on the pt2 side. The length and corner criteria are checked on the neighbor before adding it to output.

Performance

O(*N*) where *N* is the size of *tin*.

Examples

In the following example, the “second fragment” if found away from a line-end.

```
NEWTAG line_end POLY -out line_end
NEWTAG neighbor both line_end len <= 0.08 corner convex -out line_end_adj
NEWTAG neighbor both line_end_adj len <= 0.08 corner no_corner -out
secondfrag
```

NEWTAG sequence

MPC Custom Tcl Scripting Commands

Creates new tag set(s) holding fragments from a sequence of edges.

Usage

```
NEWTAG sequence tin
{len length_constraint angle angle_constraint }...
    -out tag [-ordered] [-output_all] [-outRest tag2]
```

Arguments

- ***tin***
Specifies the tag name to perform the check.
- **len *length_constraint* angle *angle_constraint***
Creates pairs of length and angle constraints that define the sequence edges. You must specify at least one length-angle pair and end the sequence with a final length; the angle following the sequence is irrelevant. Up to 100 pairs can be specified.
- **-out *tag***
Specifies the variable name to hold the output tag set.
- **-ordered**
If this optional flag is set, then the pattern match is performed only in the clockwise direction and not the counter-clockwise direction. The default behavior is to check the pattern match in both directions.
- **-output_all**
Creates extra output tag sets holding fragments belonging to the same edge in the sequence. These extra tag set names are formed by appending “_*n*” to “-out” tagset name, where *n* is a positive integer representing edge number of the fragment (for example, fragments of edge 1 go into _1, and so on).
- **-outRest *tag2***
An optional argument that generates a tag set containing all fragments from *tin* that did not match the sequence.

Tcl Result

None

Description

This command creates new tag set(s) holding fragments from a given (initial) tag set that are part of a sequence of edges (multiple fragments per edge). All sequence fragments must be in a given (initial) tag set for those fragments to be output.

Performance

$O(N)$ where N is the size of the set *tin*.

Examples

```
NEWTAG sequence all_frags len == 0.07 angle == 90 len == 0.07\  
-out long_corners  
NEWTAG sequence all_frags len == 0.07 angle == 90 len == 0.07\  
-out long_corners -output_all
```

NEWTAG topological

MPC Custom Tcl Scripting Commands

Performs a topological operation the specified input layers.

Usage

NEWTAG topological [standard] ***operation*** ***layer1*** ***layer2*** **-out** ***tag*** [-outRest *tag2*]

Arguments

- standard

An optional argument that excludes edges not completely complying with inside_edge and outside_edge. This behavior is consistent with other uses of inside abd outside edges in Calibre. This argument only applies to inside_edge and outside_edge.

- ***operation***

Specifies one of the following topological conditions:

- inside_edge
- outside_edge
- touch_edge
- coincident_edge
- coincident_inside_edge
- coincident_outside_edge
- interact_edge
- not_inside_edge
- not_outside_edge
- not_touch_edge
- not_coincident_edge
- not_coincident_inside_edge
- not_coincident_outside_edge
- not_interact_edge

When ***operation*** is inside_edge or not_inside_edge and standard is not specified, you can set a threshold the edges must meet to be passed to **-out** ***tag*** with “-pct value”. The value setting must be between 0 and 100.

- ***layer1 layer2***

Specifies the layer indices. The *layer1* edges are always the selected edges and *layer2* is used to perform the selection based on the topological condition specified.

- **-out *tag***

Specifies the variable name for the variable to hold the output tag set.

- **-outRest *tag2***

An optional argument that creates a tag set containing all fragments from *tin* that do not match the sequence.

TCL Result

None

Description

This command performs the indicated topological operation on the specified input layers. The first layer is always the “seed” layer from which the output tag set is derived. These commands differ from the Calibre-equivalent commands because these topologicaLs select any fragments which even partially satisfy the condition. This can result in situations, for example, where the `inside_edge` and `not_inside_edge` tags are not mutually exclusive.

This command allows you to identify those fragments which partially satisfy a topological condition by AND-ing the results of an operation and the results of its complement operation (such as `inside_edge` and `not_inside_edge`).

Performance

$O(N)$ where N is the number of fragments in the first layer.

Rating = fast

Examples

The following example finds edges inside a filter by using `not_outside_edge`, then finds all edges which are partially inside the filter.

```
NEWTAG topological not_outside_edge L0 L1 -out s1
NEWTAG topological outside_edge L0 L1 -out s2
TAG and s1 s2 -out s3; # s3 contains those partially inside/outside
```

NEWTAG vertical | horizontal | all_angle | 45_angle

MPC Custom Tcl Scripting Commands

Generates a set of fragments that match an angle specification.

Usage

```
NEWTAG {vertical | horizontal | all_angle | 45_angle} tin -out tag [-outRest tag2]
```

Arguments

- **vertical | horizontal | all_angle | 45_angle**

Specifies that the fragments match one of the following angle specifications: vertical, horizontal, all_angle, 45_angle.

- ***tin***

Specifies the tag variable name to perform the calculations.

- **-out *tag***

Specifies the name for the variable to hold the output tag set.

- **-outRest *tag2***

An optional argument that creates a tag set containing all fragments from *tin* that do not match the sequence.

TCL Result

None

Description

This command generates an output set containing fragments with edge slopes that match the angle specification (vertical, horizontal, all_angle, or 45-degree angle).

Performance

O(*N*) where *N* is the size of *tin*.

Rating = fast

OPC_ITERATION

MPC Custom Tcl Scripting Commands

Instructs Calibre nmMPC to perform one or more MPC iterations.

Usage

OPC_ITERATION *count* [-registers *reglist*]

OPC_ITERATION -set_epe

OPC_ITERATION -apply_feedback

Arguments

- *count*

An optional keyword that enables the “iteration count” mode of this command, specifying a number of iterations to be performed.

- -registerlist *reglist*

An optional keyword that saves the computed images into the registers specified by *reg*.

- -set_epe

An optional keyword that will trigger the refragmentation and EPE measurement steps. After this command has been executed, it is then possible to create tags, and perform tag-based operations.

- -apply_feedback

Sets set the specified fragment displacements. The fragments will not be moved, however. The setup file must explicitly call **FRAGMENT_MOVE** in order to move the fragments to their final location.

TCL Result

None

Description

This command performs a “standard” MPC iteration as defined by all setup file user settings. All options in the setup file which apply to iterations will be obeyed.

There are four modes for this command:

- No arguments mode

Upon output, the desired displacements of the fragments will be set to the values computed during the iteration. However, the fragments will not be moved until the next call to **FRAGMENT_MOVE**.

- Iterations count mode (specified with the *count* option).

In this mode, there is no need to call FRAGMENT_MOVE. The requested number of iterations is performed.

- set_epe mode

Only simulation, re-fragmentation (if enabled) and epe calculation is done in this mode.

- apply_feedback mode

Feedbacks are applied and desired displacements are set in this mode.

The last two modes offer finer control over iterations, enabling, for example, creation of tag sets and setting feedbacks after simulation and after epes are calculated, but before the feedbacks are applied.

Note that if you specify both **max_iterations** and OPC_ITERATION in the setup file, the number of iterations specified in OPC_ITERATION take precedence.

In this example, Calibre nmMPC will perform 3 non-process window iterations, followed by 4 more process window iterations, for a total of 7 iterations:

```
max_iterations 8 nopw 3
OPC_ITERATION 7
```

Examples

The following example shows 4 iterations.

```
for {set i 0} {$i < 4} {incr i} {
    OPC_ITERATION
    FRAGMENT_MOVE
}
```

The following example is a simpler method of iterating 4 times.

```
OPC_ITERATION 4
```

The following is an example of fine-grained control.

```
OPC_ITERATION -set_epe
NEWTAG . . .
FEEDBACK . . .
OPC_ITERATION -apply_feedback
FRAGMENT_MOVE
```

OUTPUT_MEASUREMENT_LOCATIONS

MPC Custom Tcl Scripting Commands

Outputs EPE measurement locations.

Usage

OUTPUT_MEASUREMENT_LOCATIONS *tag* *layer* *half_width* [*length*]

Arguments

- ***tag***
The name of the tag set where the measurement locations will be output.
- ***layer***
The name of the output layer to place the measurement locations.
- ***half_width***
The half-width of boxes representing the measurement locations in the user units.
- ***length***
An optional argument that specifies the length of the boxes. The default is 16/3 of the Nyquist sampling interval (the length used in the actual EPE calculations).

TCL Result

None

Description

This command is used to output one box per EPE measurement location.

Performance

O(N) where N is the number of measurements.

Rating = depends on the tagset. Outputting all measurement locations will be slow due to their sheer number.

Examples

```
NEWTAG all M1 -out all
OUTPUT_MEASUREMENT_LOCATIONS all sites 0.002 0.2
```

OUTPUT_OP

MPC Custom Tcl Scripting Commands

Outputs correction layout of specified layer.

Usage

```
OUTPUT_OP {-opc | -epe | -nomepe} opc_layer1 [... opc_layerN] -out output_layer1 [... output_layerN]
```

Arguments

- **-opc | -epe | -nomepe**
Selects how to offset the edges: -opc is an offset by correction displacement; -epe is an offset by effective EPE; -nomepe is an offset by nominal EPE.
- **opc *opc_layer1* [... *opc_layerN*]**
Specifies the opc input layer(s) for which the current layout state is to be saved to its corresponding output layer.
- **out *output_layer1* [... *output_layerN*]**
Specifies the destination output layer(s) to copy the opc layer's layout to. Each output layer is cleared of layout before copying the opc layout.

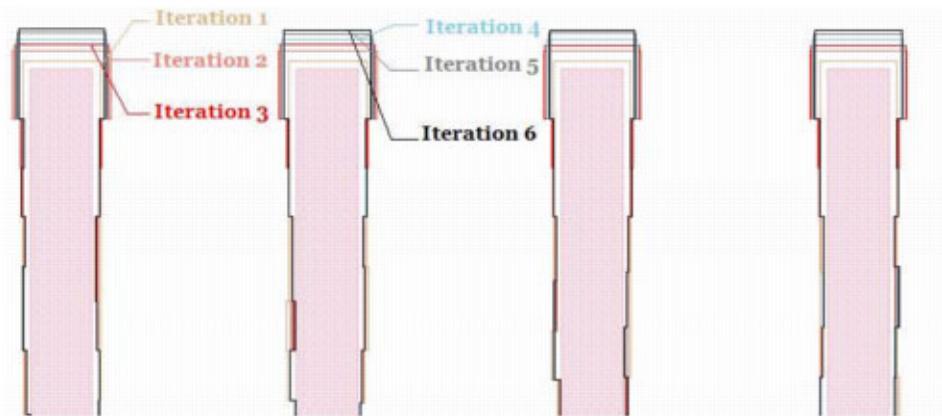
TCL Result

None

Description

The command outputs the corrected layout of the specified layer. This can be useful to output the MPC results per iteration, as shown in [Figure 5-47](#). You can then analyze how edges are fragmented and moved each time the MPC recipe iterates.

Figure 5-47. Output Per Iteration Example



Performance

Theta (N) where N is the number of edges in design.

OUTPUT_SHAPE fragment

MPC Custom Tcl Scripting Commands

Outputs simple shapes from a layer.

Usage

Syntax 1: Geometric Output

```
OUTPUT_SHAPE fragment name gid
  [-moved | -epe | -nomepe | -pwepe name | -point | -point1]
  halfwidth [trimends [offset]]]
```

Syntax 2: Annotated Output

```
OUTPUT_SHAPE fragment name gid -property name type operation
  [-add_property tag name type operation]... [halfwidth [trimends [offset]]]]
```

Arguments

- ***name***
Specifies the name of the output layer to place the shape.
- ***gid***
Specifies a generalized fragment ID, which can be single fragment ID or a tag variable name. Used when not in array mode.
- **-moved**
An optional flag that centers the output shape at the moved fragment's location. If -moved is enabled, fragments that cross hierarchical or tile boundaries may contain jogs at tile or cell boundaries that will not be present in the final MPC result.
- **-epe**
An optional flag that offsets the output shape by the effective epe. A negative EPE value offsets inwards; a positive EPE values moves shapes outwards.
- **-nomepe**
An optional flag that offsets the output shape by the nominal EPE. A negative EPE value offsets inwards; a positive EPE values moves shapes outwards.
- **-pwepe *name***
An optional argument set that offsets the output shape by the EPE of the specified process window condition. Negative EPE values offset inwards; positive EPE values move shapes outward.
- **-point**
An optional flag that marks each endpoint with a box. The box is 2***halfwidth** on each side and centered on the endpoint.

- **-point1**
An optional flag that marks the start endpoint of each fragment with a box.
- **-property *name type operation***
A required keyword set for annotated output (Syntax 2) that specifies the format of the property.
 - name*** — Specifies a name that can be used by other commands such as setlayer mpc to read the specified property.
 - type*** — Specifies the numeric type. (Only numbers are supported.) Use one of **int**, **float**, or **double**.
 - operation*** — Specifies what to do if two or more shapes created by OUTPUT_SHAPE touch. (Only one property can be attached because the touching shapes get merged.) Use one of **min**, **max**, or **sum**.
- **-add_property *tag name type operation***
An optional argument that attaches a property from a different tag set, specified with ***tag***, to the output shape layer. The other arguments function as with **-property**.
- ***halfwidth***
Specifies half the width of the output box, in user units. This is ignored when **-property** is used.
- ***trimends***
Specifies the distance to trim the ends when generating the output boxes in user units.
- ***offset***
Specifies an offset from the nominal center in user units.

Description

This command can be used to output one or more simple shapes. Typically, these simple shapes can be used for debugging or visualization of fragmentation results. The shapes can either be defined by fragments or general rectangular boxes.

In the fragment mode, an entire tag set can be output to shapes.

Note

 When using OUTPUT_SHAPE with **NEWTAG epe**, note that the OUTPUT_SHAPE generates a contour based on the EPE from the **prior** MPC iteration, while NEWTAG epe obtains its EPE from the **final** iteration of the MPC run.

Performance

O(N) where N is the number of shapes to output.

Rating = fast

Examples

The following example outputs all line-ends:

```
OUTPUT_SHAPE fragment DEBUG line_end 0.05 0.004
```

TAG and | or | subtract

MPC Custom Tcl Scripting Commands

Performs boolean operations on tag sets.

Usage

TAG {and | or | subtract} *t1 ... tN* -out [-annotated] *tag*

Arguments

- **and | or | subtract**

Specifies boolean AND, OR, or SUBTRACT operation to be performed on the specified tags.

- ***t1 ... tN***

Specifies several input tag variable names. At least two tag inputs are required.

- **-out *tag***

Specifies output tag variable names.

- **-annotated**

An optional argument specified to write fragment properties to the tag. The “-out - annotated” argument can be abbreviated as “-aout”.

When a fragment appears in multiple sets, the value in the first listed tag set is used.

When using -annotated, all tag sets must be annotated with the same type of value (integer or floating point number).

TCL Result

None

Description

This command is a set of boolean operations (AND, OR, and SUBTRACT) performed on multiple tag sets. The subtract operation treats the first input as the set from which to subtract the other sets. These commands support “in-place” operations in which the output tag set overwrites one of the inputs.

Performance

O(*N*) where *N* is the sum of the sizes of all input sets.

Rating = fast

Examples

```
TAG subtract A B C -out D; # subtracts elements in B and C from A
```

Calibre nmMPC File Formats

Calibre nmMPC supports file formats listed in the following table.

Table 5-15. Calibre nmOPC File Formats

Command	Description
Lithomodel (Litho Model Format)	Loads modeling information to Calibre nmMPC that describes all aspects of a simulation.

Lithomodel (Litho Model Format)

Calibre nmMPC File Formats

Loads modeling information to Calibre nmOPC that describes all aspects of a simulation including multiple exposure.

The litho model process (see “[Import and Define Models](#)” on page 56) uses a directory that describes all aspects of a simulation including multiple exposure. The litho model directory contains a file called “Lithomodel” that defines the specifics.

Keywords in the Lithomodel can be in any order, except **version** must be the first line.

The e-beam dose must be explicitly defined in the denseopc_options block of the setup file using an [image_options](#) block.

Format

```
version 1
lithotype EBEAM
[etch file]
[ebeam file]
```

Parameters

- **version 1**

A required keyword that specifies the version. This must always be 1. This keyword must always be the first parameter specified.

- **lithotype EBEAM**

A required keyword that specifies that the litho model file is specifically for e-beam simulation.

- **etch file**

An optional keyword that specifies the name of an etch model file, without path information. The etch model must be located in the litho model directory (or may be a link in that directory).

- **ebeam file**

An optional keyword that specifies the name of an e-beam model file without path information. The model must be located in the litho model directory (or may be a link in that directory).

Examples

The following is a basic example of an e-beam Lithomodel:

```
version 1
lithotype EBEAM
etch etch.mod
ebeam ebeam.mod
```

The following is an example of the dose specification in the setup file using an [image_options](#) block:

```
modelpath ./myModels
layer M0
layer M2
layer M3
layer M4
layer M5
layer M6
layer M7
tilemicrons 100
image_options MDF {
    layer M0 visible ebeam_dose 1.0
    layer M2 visible ebeam_dose 0.9
    layer M3 visible ebeam_dose 1.1
    layer M4 visible ebeam_dose 1.2
    layer M5 visible ebeam_dose 1.3
    layer M6 visible ebeam_dose 1.4
    layer M7 visible ebeam_dose 1.5
    litho_model myLatestEuvModel
}
setlayer im = ebeam_simulate MDF
```

Appendix A

Calibre nmMPC in Calibre nmModelflow

A subset of the Calibre nmMPC development flow can be performed in the Calibre nmModelflow tool, primarily the development of the E-Beam and VEB etch (short range) models.

Key Concepts for Calibre nmModelflow	343
Calibre nmMPC in Calibre nmModelflow Workflow	344
Creating a Litho Model in Calibre nmModelflow	345
Loading Gauge Data into Calibre nmModelflow	349
Creating a Stage for Calibre nmMPC	350
E-Beam Calibration Stage.....	352
Etch Calibration Stage	353
Creating a Calibration Job in Calibre nmModelflow (Calibre nmMPC)	354

Key Concepts for Calibre nmModelflow

Calibre nmModelflow is a tool designed around creating calibration jobs for models in a persistent database for reuse purposes. The following list is a subset of the key concepts most relevant to Calibre nmMPC users of the tool.

- **Active Data** — The currently displayed dataset (gauges, layout, litho model, metrics, plots, filters, and so on). The GUI applies the commands to active data; items in the database must be activated before they can be used. An active data item can be added to the database.
- **CLI** — Command Line Interface. Calibre nmModelflow includes an integrated console window where you can enter commands directly.
- **Cost Objective** — The value that the optimization attempts to minimize when testing permutations of variables.
- **Database** — A collection of items managed by Calibre nmModelflow. All models, simulations, stages, and simulation results are stored in a database that persists through Calibre WORKbench sessions. You can select items from the database to reuse elements like litho models and filters.

Database items are unique, and have referential integrity; you cannot delete or modify an item without removing all references to it first.

- **GID** — Gauge ID. A GID is a numeric value that corresponds to a row in the **Gauge Analysis** tab. The GIDs are stored in the third column of a gauge or super gauge file.
- **Job** — A run of a Stage. You can have multiple jobs running, and jobs can be dependent on the results of other jobs.
- **Lithomodel** — A file comprising mask, optical, resist, etch, EUV, and topo models. Calibre WORKbench operates on the models contained in a litho model as a set, instead of the modelflow_v2 method of needing to load and manage each model type separately.

Note

 The litho model file is always named “*Lithomodel*”, therefore this document and the Calibre nmModelflow interface refers to the actual file as “*Lithomodel*” but to the repository of data in general as a “litho model”.

- **MDF** — An abbreviation for Calibre nmModelflow.
- **Stage** — One step of the modeling process. Stages include calibration of the E-Beam model and etch model; other stages are used for simulation and bias table creation.

Creating stages forms the building blocks of an optimization run.

Calibre nmMPC in Calibre nmModelflow Workflow

The workflow for the relevant Calibre nmMPC tasks in Calibre nmModelflow can be summarized as follows:

Table A-1. Calibre nmMPC in Calibre nmModelflow Steps

Step	Process	Description
1	<ul style="list-style-type: none">• Stage 1 - Long Range Density Convolve• Stage 2 - Build Uniformity Polynomial• Stage 3 - Remove LR Effects From Short Range Data	Perform tasks that require MPC Center before invoking Calibre nmModelflow.
2	Creating a Litho Model in Calibre nmModelflow	Create a litho model in Calibre nmModelflow.
3	Loading Gauge Data into Calibre nmModelflow	Load the gauge file results from Step 1, Stage 3 into Calibre nmModelflow.
4	Creating a Stage for Calibre nmMPC	Create a stage for calibration.
5	Creating a Calibration Job in Calibre nmModelflow (Calibre nmMPC)	Create and run calibration jobs for the stage you created.

Table A-1. Calibre nmMPC in Calibre nmModelflow Steps (cont.)

Step	Process	Description
6	Correct and Verify the Design	When you have successfully calibrated E-Beam and VEB Etch models, you can use them to correct and verify designs.

Creating a Litho Model in Calibre nmModelflow

Calibre nmModelflow contains functionality to create a Calibre nmMPC-relevant litho model file via a GUI wizard. At least one litho model must be present in the database to perform calibration.

Prerequisites

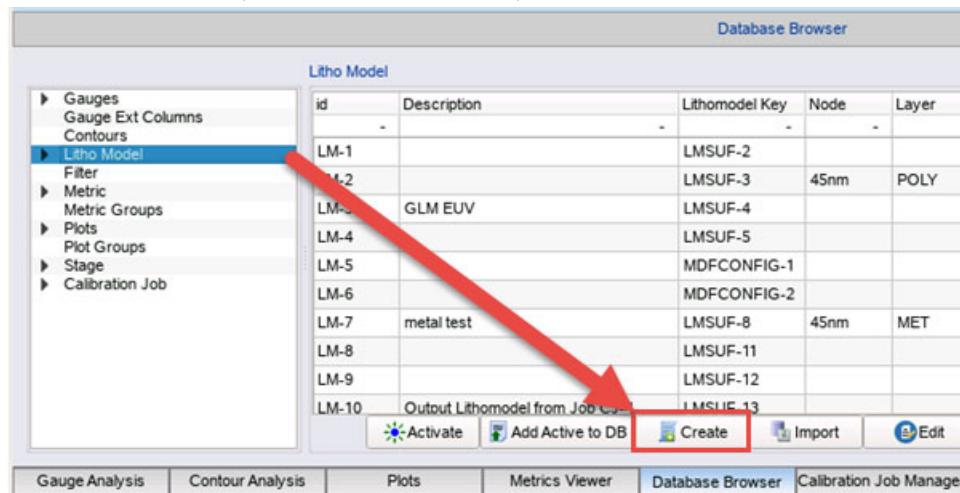
- Information about your mask process

Procedure

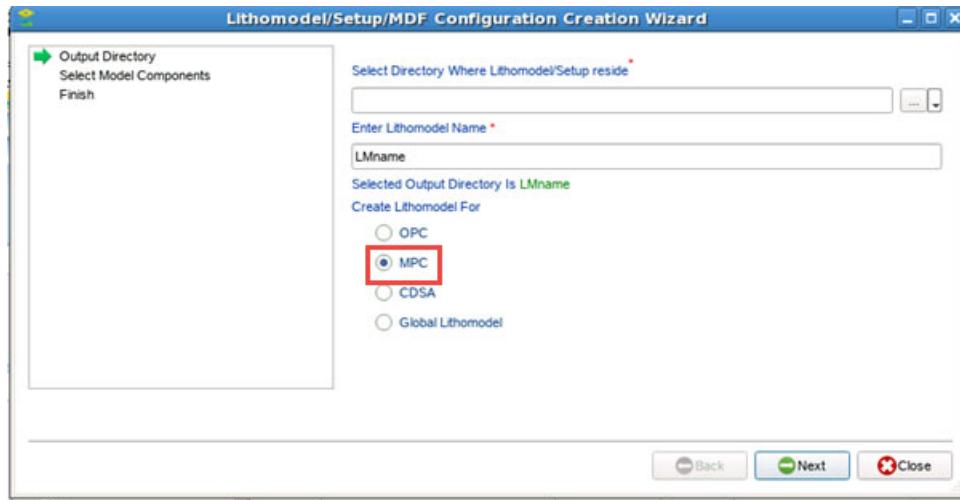
- Invoke Calibre WORKbench.
- Open Calibre nmModelflow (**Litho > nmModelflow**). The tool typically opens on the **Gauge Analysis** tab.
- Click the **Database Browser** tab to activate the browser.



- In the list on the left, choose Litho Model, then click Create.



5. In the Output Directory screen of the MDF Configuration Creation Wizard, select Create Lithomodel For MPC.



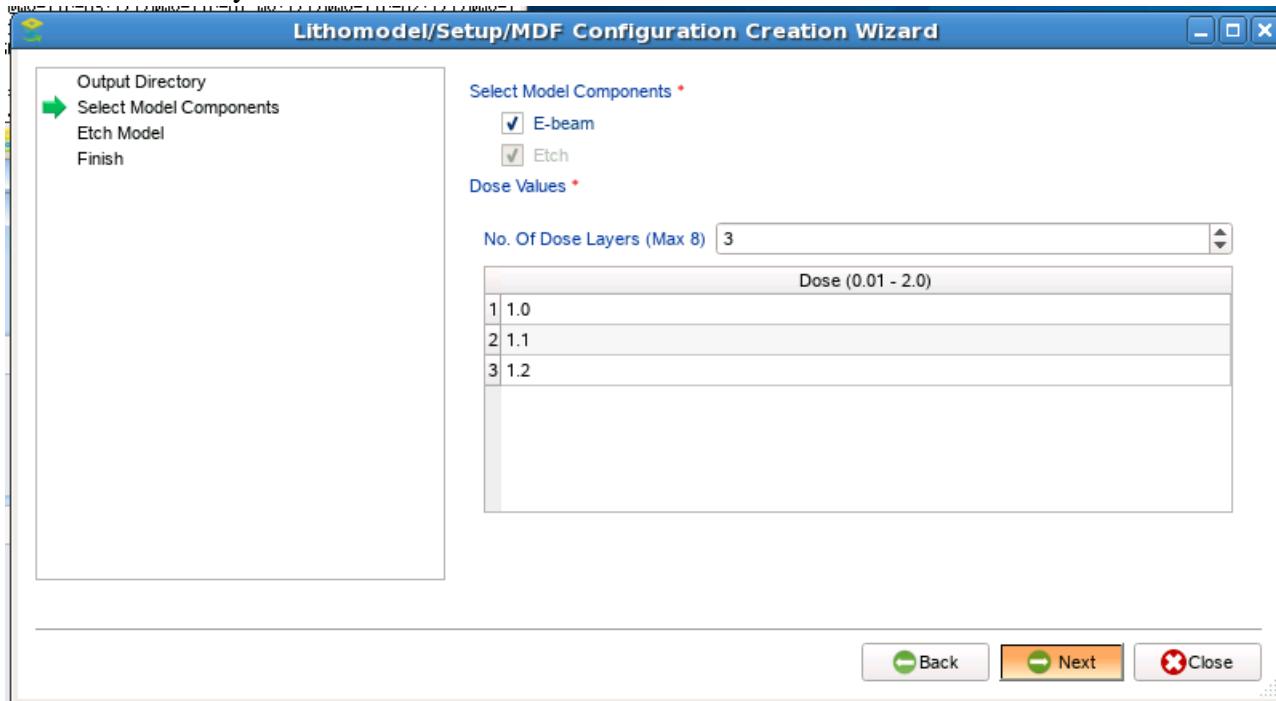
Additionally, set the other fields in this screen as follows:

- **Select Directory Where Lithomodel/Setup Reside** — Specify the absolute path to a /working directory to contain a litho model directory. You can use the ... button on the right to bring up a File Chooser to navigate to or create directories.
- **Enter Lithomodel Name** — Specify a unique name that will be used as the name of the litho model directory.

Click **Next** to continue.

6. In the Select Model Components screen, select E-Beam if you are calibrating E-Beam models with this litho model. (VEB Etch models are always required.)

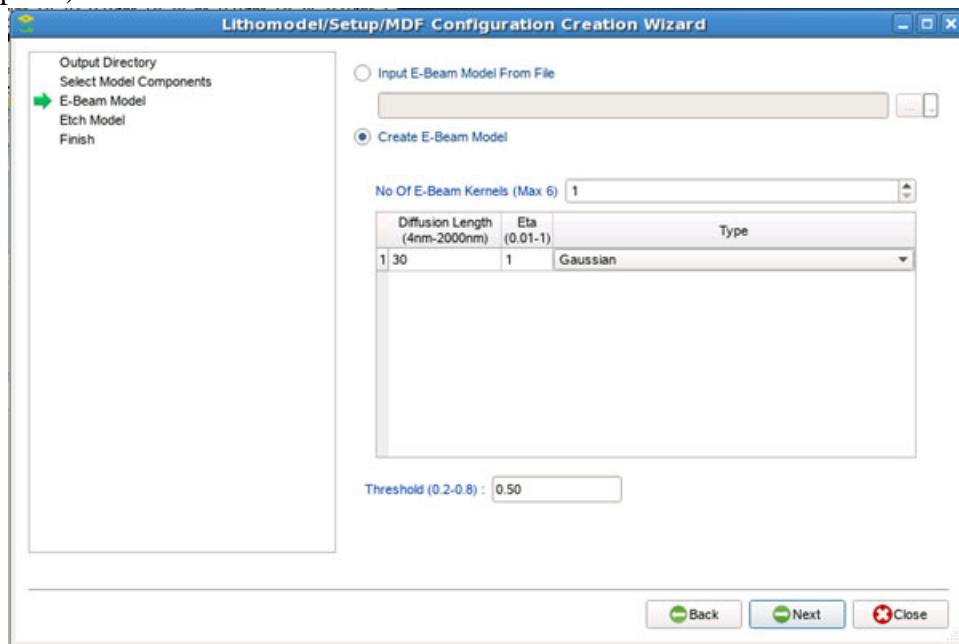
Selecting E-Beam opens up the Dose Values section, which lets you define up to eight dose layers.



Click **Next** to continue.

7. If you did not select an E-Beam model, skip to step 8.

Specify the E-Beam model in the next screen, either by specifying an existing file (Input E-Beam Model From File option) or defining one on this screen (Create E-Beam Model option).

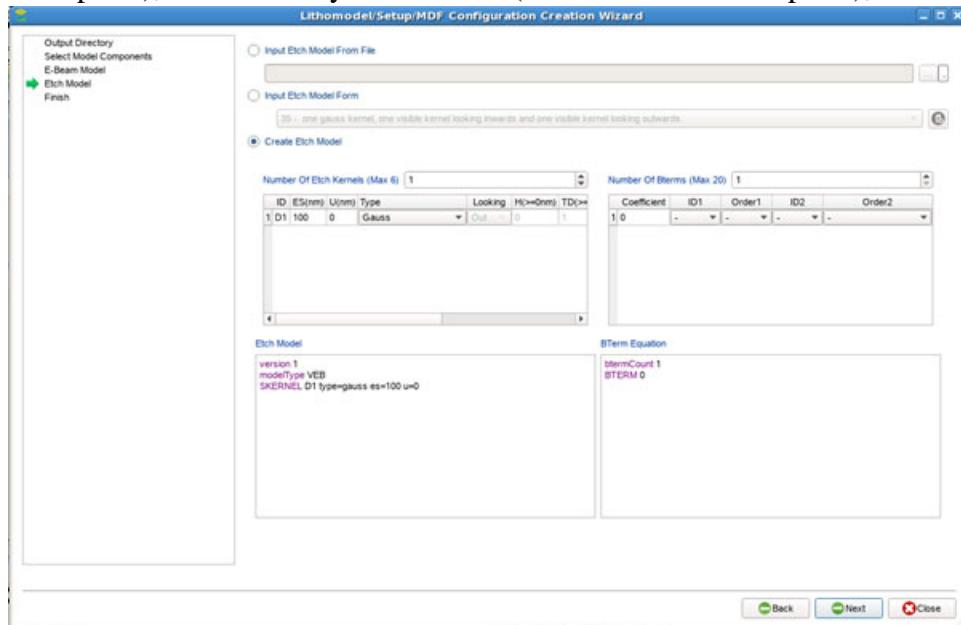


Up to six kernels can be defined in an E-beam model, with each having a different diffusion length, eta value, and type. In addition, you can set one threshold value for the model.

Note

 For information on the E-Beam format, see “[ebeam_model_load](#)” on page 141.

8. Add a VEB etch model to the litho model on the Etch Model screen, using an existing model (Input Etch Model From File option), a selected modelform (Input Etch Model Form option), or a manually defined one (Create Etch Model option), then click **Next**.



Note

 Only a limited set of etch modelforms (35, 46, 55, 66, and 67) are supported for Calibre nmMPC modeling.

9. The Finish screen shows the CLI command that Calibre nmModelflow executes to create the Calibre nmMPC litho model file. Before clicking **Finish**, perform the following actions:
 - Label the new litho model using the Description, Node, and Layer fields.
 - Leave the Import Lithomodel to Database checkbox selected. Calibration tasks can only be effectively run on litho models in the database.

Results

The new litho model is added to the database.

Loading Gauge Data into Calibre nmModelflow

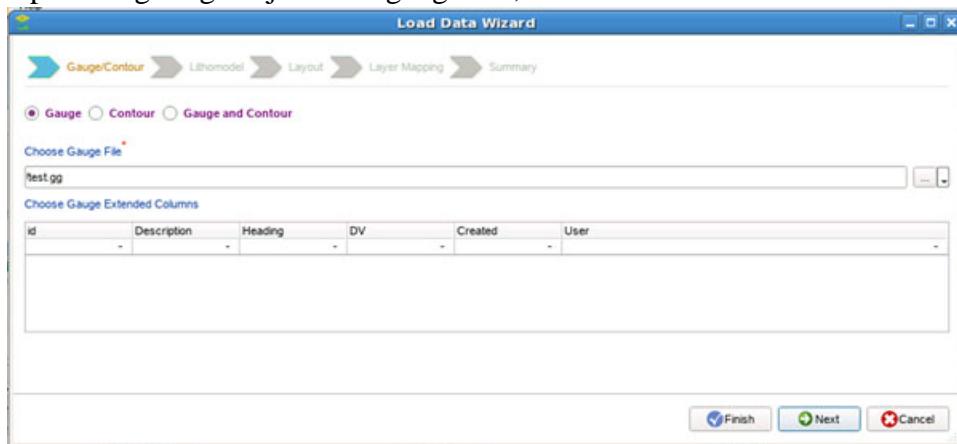
After removing long range effects in MPC Center, you load the resulting gauge file into the Calibre nmModelflow database.

Prerequisites

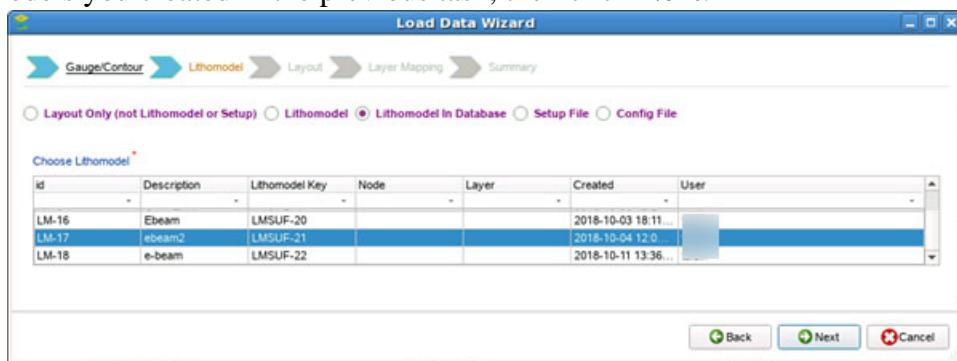
- Completed the task, “[Creating a Litho Model in Calibre nmModelflow](#)”

Procedure

- In the upper set of tabs, switch to the **Gauges** tab and click **Load**.
- In the Load Data Wizard, leave the selector at Gauge, and use the file navigator to select the post-long range adjustment gauge file, then click **Next**.

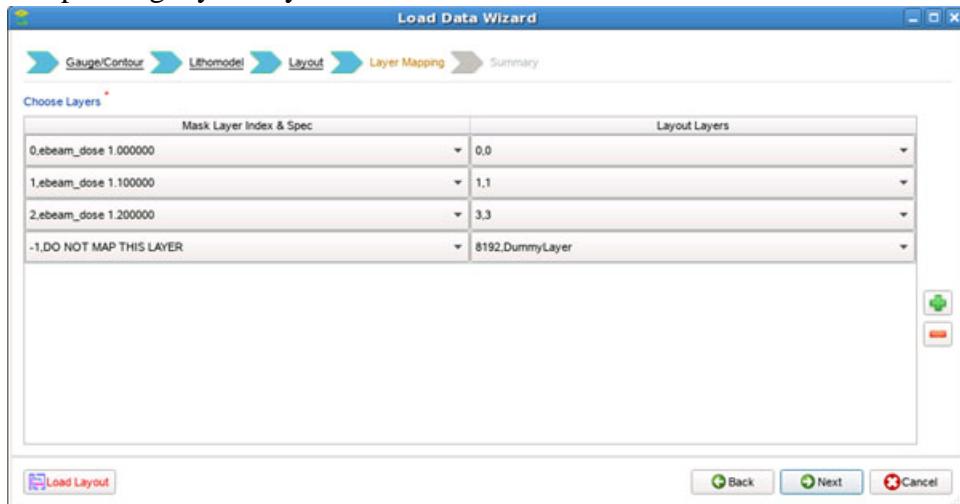


- In the Lithomodel page, choose Lithomodel in Database, and select one of the litho models you created in the previous task, then click **Next**.

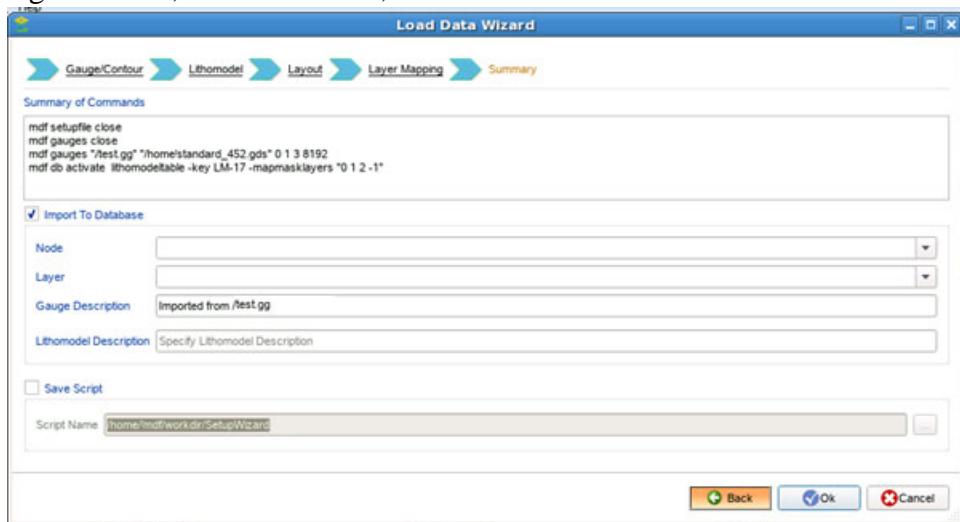


- In the Layout page, navigate to the relevant layout file, select it, then click **Next**.

5. In the Layer Mapping page, select the layers in the litho model and match them to the corresponding layout layers.



6. In the final Summary screen, enter any uniquely identifying labels for the node, layer, gauge data file, and lithomodel, then click **OK**.



Results

The new gauge file is given an entry in the database, and can be used as an input to a calibration job.

Creating a Stage for Calibre nmMPC

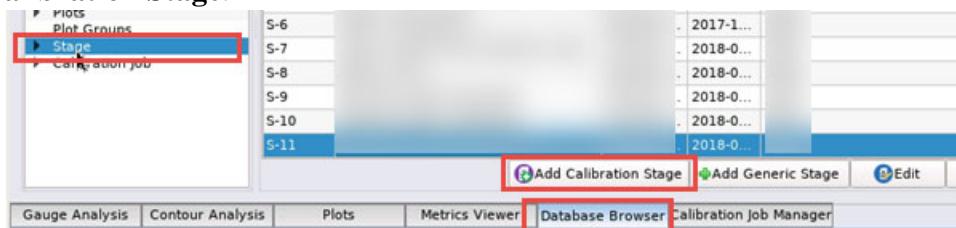
Stages define operations to be performed on a litho model. For Calibre nmMPC, you must create at least one calibration stage; other stages can be created for supporting information.

Prerequisites

- Calibre nmModelflow invoked

Procedure

- Switch to the **Database Browser** tab, and select the Stages list, then click **Add Calibration Stage**.



- In the Flow Stage Wizard, switch the selector on the left to MPC to show the Calibre nmMPC-specific choices.



Table A-2. Flow Stage Wizard Choices for Calibre nmMPC Mode

Option	Description
Calibrate	Creates a calibration stage. Checkboxes are available to calibrate an EBeam and Etch model, individually or together.
Simulate	Runs a simulation using an associated litho mode without calibration.
Build Etch	Builds the etch model in the associated litho model, calibrating only the threshold.
Etch Bias Table Generation	Creates a VEB etch bias table from the associated gauge file. Not used by Calibre nmMPC at this time.

Choose an option, then click **Next**.

- Fill out the Stage Wizard page depending on the options you chose:

- [E-Beam Calibration Stage](#) (optional)
- [Etch Calibration Stage](#) (required)

Click **Next** to continue.

- In the Optimizer Settings screen, select the following:

- **Cost Function** — Sets the objective for the calibration search.

- **Optimization Search** — Selects the type of algorithm used to search the optimization parameter space.
- **Optimization Parameters** — Contains parameters that can adjust the gauge selection criteria and other options that are not search-related.
- **SEM Box** — Not presently used for E-Beam or VEB Etch model calibration. It is advised that you deactivate this control.
- **Regularization Parameters** — If selected, this option averages the bias value if it is less than the specified tolerance value.

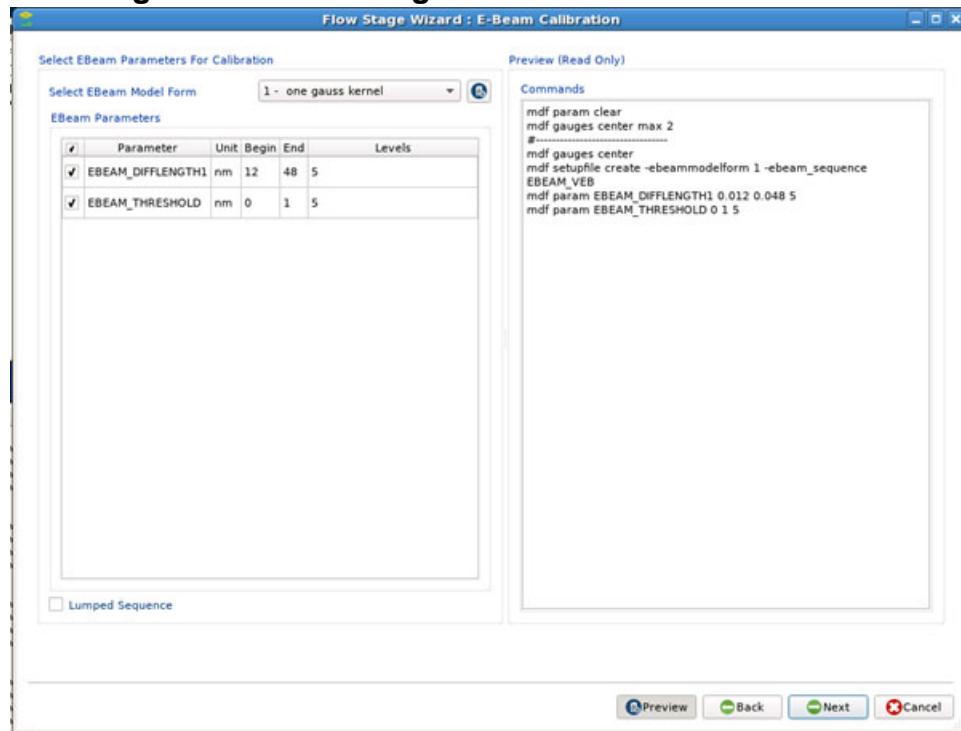
Click **Next** when you have finished making your selections.

5. The final wizard screen contains the list of CLI commands that Calibre nmMDF uses to create the stage. If you need to modify any of the commands, click **Edit**. Otherwise, click **Finish** to add the Stage to the database.

E-Beam Calibration Stage

Configure the E-Beam calibration stage when you select the **Calibrate > EBeam** option from the Flow Stage Wizard.

Figure A-1. Flow Stage Wizard: E-Beam Calibration



Options for Calibration

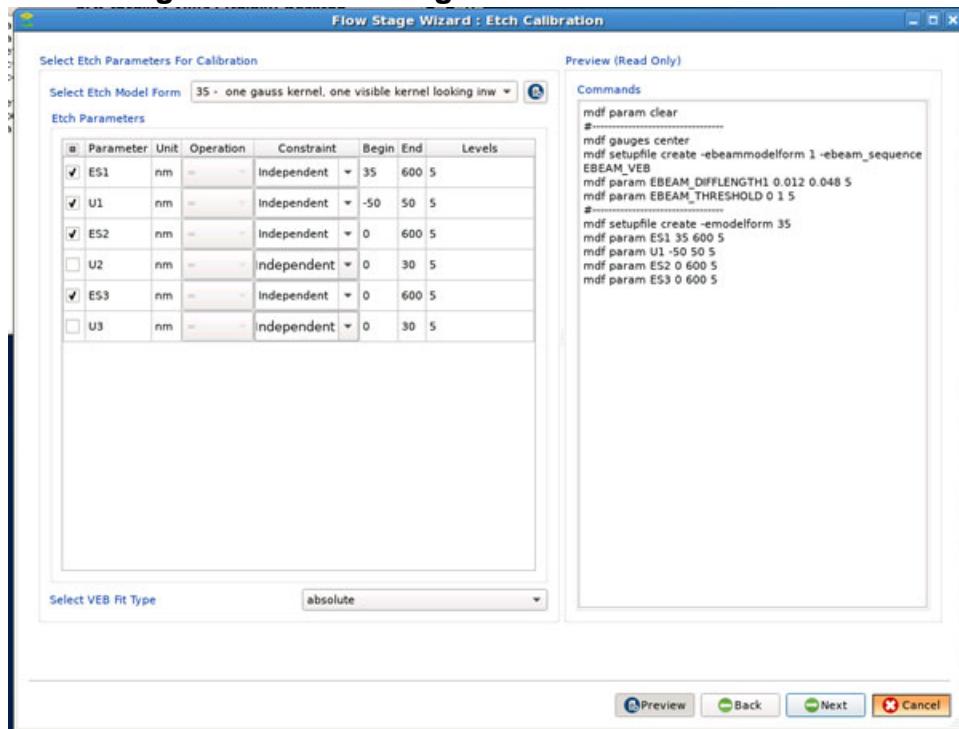
- **Select EBeam Model Form** — Selecting a modelform automatically populates the EBeam Parameters list with relevant parameters. However, you must choose the modelform that matches the EBeam model in your litho model.
If you have an active litho model, the option Active Input Lithomodel is available to generate the calibration parameters based on the loaded model.
- **Lumped Sequence** — Selecting the Lumped Sequence checkbox toggles between calibrating the EBeam and VEB model together (checked, which shows ‘ebeam_sequence LUMPED’ in the Preview window) and separately (unchecked, which shows ‘ebeam_sequence EBEAM_VEB’).

Click **Next** once you have set your parameters, and continue with the task “[Creating a Stage for Calibre nmMPC](#)” on page 350.

Etch Calibration Stage

Configure the E-Beam calibration stage when you select the **Calibrate > EBeam** option from the Flow Stage Wizard.

Figure A-2. Flow Stage Wizard: Etch Calibration



Options for Calibration

- **Select Etch Model Form** — Select the VEB etch model form to use to calibrate the model. This modelform must match the model that is in the litho model you run with this stage.
If you have an active litho model, you have an additional choice, Active Input Lithomodel - Tune All Available Parameters, which automatically sets up the parameter list based on the etch model.
- **Etch Parameters** — Set the parameters to be calibrated, along with search ranges for each parameter. The Constraint column is optionally used to have one or more calibration parameters be related to another the dependent parameter must have a test value that matches the operation (either equal to or an inequality expression).
- **Select VEB Fit Type** — Sets whether the etch error statistics are performed on absolute values or relative to the target edge.

Click **Next** once you have set your parameters, and continue with the task “[Creating a Stage for Calibre nmMPC](#)” on page 350.

Creating a Calibration Job in Calibre nmModelflow (Calibre nmMPC)

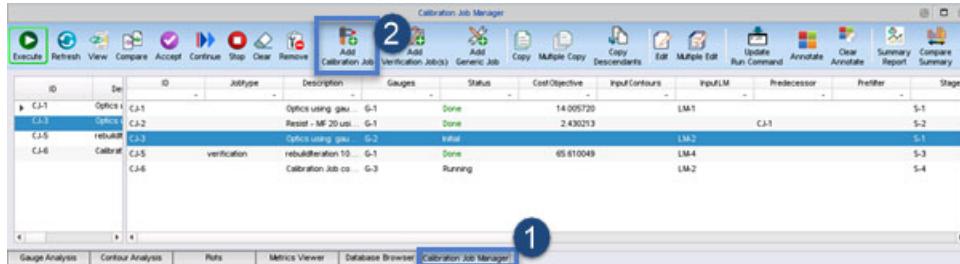
For Calibre nmMPC, use the standard Calibre nmModelflow steps to create a calibration job.

Prerequisites

- Completed the task, “[Creating a Stage for Calibre nmMPC](#)”.
- Completed the task, “[Loading Gauge Data into Calibre nmModelflow](#)”

Procedure

1. In Calibre nmModelflow, switch to the **Calibration Job Manager** tab.



2. Click **Add Calibration Job**.
3. On the Gauge/Contour page, select the gauge file you added to the database from the list, then click **Next**.

4. On the Lithomodel/Predecessor page, select the litho model you added to the database from the list, then click **Next**.
5. On the Stage page, select the stage you added to the database from the list, then click **Next**.
6. On the Layer Mapping & Run Command page:
 - Check that the layer mapping matches the layout layers in the list.
 - If you have a custom run script, navigate to it and select it in place of the default (*run_rsh_ssh.sh*).

Click **Finish** to continue.

Note

 This task skips the Optional Components screen, since we have not created any filters, metrics, or plots in this flow.

7. On the Summary screen, enter a description of the job and click **OK**.
The new calibration job is added to the list.
8. To run a calibration job, select it from the list and click the **Execute** button in the button bar.

Results

When a calibration job is executed, it creates a [Calibration Job Record](#) of the run. The Calibration Job Record contains the results of the run, and also contains error information in the **Transcripts > Job** section if the run did not complete successfully. You can access it by selecting the job and clicking **View**.

Index

— Symbols —

[]²²
{ }²²
|²²

— A —

algorithm, 174
allow_jog_full_move, 175
Angle constraints, 309, 319
Angle snapping, 233
Attenuated masks
 background, 176

— B —

background, 176
Background density model, 109
Bold words, 22

— C —

Calibrating long-range effects, 31
Calibre nmDRC-H, 19
Calibre WORKbench, 16
check_movement, 177
Configuration
 etch_model_load command, 145
 modelpath command, 133, 147
Contour tab, 46
Controlling OPC
 concurrency, 210
 maximum fragment length, 222
 maximum movement, 219
 minimum fragment length, 225
Convergence, 185
convex_concave_adj_len, 178
convex_concave_correction, 179
convex_concave_count, 180
convex_concave_limit, 181
convex_concave_metric, 182
Corner fragmentation, 193
corner_control, 183

Correcting masks, 51
Correction, 51
correction layer, 61
Courier font, 22

— D —

Dark
 background, 176
Define an image, 62
Define input layers, 59
denseopc_options, 138
DENSITY CONVOLVE, 73
Density loading model, 109
direct_input, 140
Double pipes, 22

— E —

E-beam calibration
 MPC Center, 45
 mpcflow_v2, 129
E-beam correction, 74
E-beam effects, 13
E-beam model, 40
 creating, 45, 71
ebeam_model, 234
ebeam_model_load, 141
Edge placement errors, 72
 calculating, 240
epc_spacing, 184
Errors
 minedgelengthviolation, 190
etch_model, 234
etch_model_load setup command, 145
external layer, 61

— F —

FEEDBACK, 285
Feedback, 63
feedback, 185
fragalign, 287
FRAGMENT delete, 290

-
- FRAGMENT modify, 291
 FRAGMENT split, 292
 fragment_coincident, 186
 fragment_corner, 188
 FRAGMENT_EPE_MEASURE_METHOD, 294
 fragment_flaglayer, 197
 fragment_inter, 198
 fragment_interlayers, 210, 212
 fragment_island, 214
 fragment_layer, 218
 fragment_layer_order, 220
 fragment_max, 222
 fragment_min, 225
 fragment_minjog, 226
 FRAGMENT_MOVE, 295
 fragment_visible, 230
Fragmentation
 concave corner, 193
 disabling, 63
 interfeature, 206, 216
 maximum fragment length, 222
 minimum fragment length, 225
 priority, 201
Fragments
 adjacent space ends, 190, 193
 ripple, 206
 freeze_skew_edges, 232
 FullScale, 20
- G**
- Gauges tab, 40
 MPC Center
 Gauges, 34
 Generating a test layout, 23
 Generating output layers, 61
 grids_for_angle_45_snap, 233
- H**
- Heavy font, 22
 Hidden layers, 60
- I**
- image, 234
 image_options command, 236
 Import mod els, 57
- Input layers, 59
 Interfeature fragmentation, 201
 island crossing, 216
Intrafeature
 split, 190
 Island layer fragmentation, 214
 Italic font, 22
 Iterations, 62
- J**
- jog_freeze, 237
 jog_ignore_metric, 238
- L**
- layer
 denseopc_options, 240
 LITHO DENSEOPC, 146
 Layers, 58
 hidden, 60
 names, 146, 240
 OPC, 60
 visible, 60
 Limits
 on OPC movement, 219
 line_end_fragment, 242
 Litho model, 57
 format, 341
 Lithomodel, 57, 341
 Long-range correction, 73
 Long-range effect, 13
 Long-range model file format, 124
 LR tab, 34, 36
- M**
- Mask layer file, 36
 Mask Process Correction (MPC)
 workflow, 14
 Mask rule constraints, 63
 Mask test pattern specification file, 23, 107
 Masks
 background illumination, 176
 max_iter_movement, 246
 max_iterations, 244
 max_opc_move, 247
 Maxedgelength fragmentation, 222
 MDP MAPSIZE, 73

-
- measure_metric_mode, 248
Minimum keyword, 22
Misc tab, 36, 44
modelpath setup command, 133, 147
MPC Center
 Contour, 46
 Gauges, 40
 LR, 34, 36
 Misc, 36, 44
 Optimize, 34
 Optimize tab, 44
 Remote, 48
 SVRF, 38
 VEB, 41
mpc_contour_search_radius, 249
mpc_fragme nt_type_projection, 250
mpc_preserve_angles, 252
mpc_pseudo_nyquist, 253
mpc_sites_control, 254
mpc_sites_mode, 255
mpc_skip_fragment, 257
mpc_use_nearest_epe, 258
mpcCompile, 93
mpceval, 122
mpcflow_v2, 126
mpcgConvert, 114
mpclr, 115
mpclropt, 119
mpcssConvert, 114
MRC, 297
mrc_rule, 259, 299
mtpCompile, 105
Multilayer fragmentation, 218
Multiple masks
 defining background, 176
- N —
- NEWTAG 45_angle, 330
NEWTAG all, 301
NEWTAG all_angle, 330
NEWTAG blocked, 302
NEWTAG cd, 303
NEWTAG complete, 305
NEWTAG displacement, 307
NEWTAG edge, 309
NEWTAG enclose, 315
- NEWTAG enclosing, 315
NEWTAG epe, 313
NEWTAG external, 315
NEWTAG fragment, 319
NEWTAG horizontal, 330
NEWTAG internal, 315
NEWTAG line_end, 324
NEWTAG neighbor, 325
NEWTAG space_end, 324
NEWTAG topological, 328
NEWTAG vertical, 330
no_opc_region, 266
No-correction region, 64
- O —
- OPC layer, 60
opc_grid_multiplier, 267
OPC_ITERATION, 331
Optically visible data, 60
Optimize tab, 34
Output layers, 61
OUTPUT_MEASUREMENT_LOCATIONS, 333
OUTPUT_SHAPE box, 336
OUTPUT_SHAPE fragment, 336
- P —
- Parentheses, 22
Passing layers to MPC, 59
Pipes, 22
processing_mode command, 148
Proximity correction, 74
Proximity error, 13
Proximity model, 23, 110
Proximity model file, 45
Proximity test pattern specification file, 96, 107, 124
pto switch, 20
ptpCompile, 94
- Q —
- Quotation marks, 22
- R —
- Remote tab, 48
retarget_layer, 268
Ripple fragments

controlling, 206
interfeature fragmentation, 206

— S —

scale, 272
Set feedback, 63
Set iterations, 62
setlayer curve_target, 161
setlayer denseopc, 167
setlayer ebeam_simulate, 141, 149
Setup file, 53
Shielding
 in fragmenting features, 206
Short-range effects
 see Proximity errors
Slanted words, 22
sparse_ebeam_veb, 273
Specification files, 23
Spl it
 intrafeature, 190
Square parentheses, 22
step_size, 274
SVRF tab, 38

— T —

TAG and, 339
TAG or, 339
Tag scripting, 70
TAG subtract, 339
Test layout
 generating, 23
tile_fragmentation_only, 275
Total mask CD error, 13

— U —

Underlined words, 22
uniCompile, 103
Uniformity models, 23
Uniformity test pattern specification file, 23,
 104

— V —

VEB tab, 41
Verifying the corrections, 81
version, 276
Visible layers, 60

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.

