

SIEMENS EDA

Calibre® nmOPC™ 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Unpublished work.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 nmOPC	19
Calibre nmOPC Key Concepts	20
CM1 Modeling	21
Variable Etch Bias	21
Flare and Shadow Modeling	22
Double and Multi-Patterning Correction	23
Tag Set Properties	24
Calibre nmOPC Usage	25
Calibre nmOPC Workflow	25
Calibre nmOPC Iteration Flow	26
Syntax Conventions	27
Chapter 2	
Calibre nmOPC Quick Start	29
Calibre nmOPC Prerequisites	29
Calibre nmOPC Configuration Summary	30
Variable Etch Bias (VEB) Model Process Summary	31
Calibre OPCVerify in the Calibre nmOPC Setup File	31
The Calibre nmOPC Setup File	31
What the Setup File Controls in a Calibre nmOPC Run	33
What the Rule File Controls in a Calibre nmOPC Run	34
How Does Calibre nmOPC Take EPE Measurements?	35
Fragment Movement With Calibre nmOPC	36
Customized Correction Algorithms in Calibre nmOPC	37
Variables, Keywords, and Commands Supported in the Setup File	37
Calibre nmOPC Modes of Operation	39
Considerations for Calibre FullScale Runs	41
Acronyms and Terminology	42
Chapter 3	
Configuring Calibre nmOPC	43
Import Model Information	47
Load Models Using Commands	47
Load Model Information With a Litho Model	48
Models in EUV Lithography	50
Topography Modeling	50
Define Layers	52
Input Layers	53
Layers Passed to OPC	54
Output Layers	55
Layer Types	55

Define an Imaging Background	58
Define an Image	59
Optional Setup File Operations	60
Select Correction Method	61
Automatic Fragmentation	61
Custom Fragmentation	63
Define Mask Rule Constraints	65
Define Process Windows	67
PV Band PWOPC	70
Direct Etch Effect Correction	71
Resist Top Loss Correction	75
Check Enclosure With Image Cost Functions	77
Define No OPC Regions	79
Smooth Biased Target Layers	80
Adjust for Mask Corner Rounding	83
Simulate Layer-Based Images Using Calibre OPCverify	83
Create Custom Scripts for Calibre nmOPC	84
SRAF Operations in Calibre nmOPC	85
Understanding the Transcript	88
Chapter 4	
Calibre nmOPC Reference Syntax	91
Commands Grouped by Function	94
Calibre nmOPC SVRF Commands	97
LITHO DENSEOPC	98
LITHO CELLARRAY DENSEOPC	100
LITHO CL DENSEOPC	102
LITHO EUV DENSEOPC	105
LITHO ML DENSEOPC	107
RET FLARE CONVOLVE	109
RET OPTIMIZE_EUV_HDB	113
LITHO DENSEOPC Setup File Commands	115
Calibre OPCverify Setup File Commands	117
clone_transformed_cells	119
ddm_model_load	121
denseopc_options	122
etch_imagegrid	124
etch_model_load	125
flare_longrange	126
flare_map_shift	129
flare_model_load	130
global_lithomodel_path	132
layer	133
lintmodel	135
ml_featurevector_load	139
ml_model_load	140
modelpath	141
optical_model_load	142

Table of Contents

output_text_filepath	143
png	144
resist_model_load	145
setlayer curve	146
setlayer curve_target	152
setlayer denseopc	160
setlayer layer_simplify	162
setlayer tilegen	164
setlayer veb_retarget	167
simulation_consistency	171
svrf_var_import	172
veb_simulate	173
denseopc_options Keywords	174
active_layer	179
adjust_moved_target_sites	182
adjust_target_control_sites	183
algorithm	184
allow_single_site_correction	186
background (for denseopc_options block)	187
check_movement	190
chiplet_placement_flare_tolerance	191
corner_control	192
displacement_limit_mode	193
curvilinear_opc	194
effective_epe_regulation	195
enhance_corner_to_corner_site_pairing	196
epe_spacing	197
etch_model	198
feedback	199
feedback_balance (Deprecated in 2016.2)	200
feedback_mode	201
flare_model	203
fragment_coincident	204
fragment_consistency_mode	206
fragment_corner	207
fragment_flaglayer	217
fragment_grid	218
fragment_inter	219
fragment_interlayers	234
fragment_island	236
fragment_layer	242
fragment_layer_order	248
fragment_marker	252
fragment_max	253
fragment_min	257
fragment_minjog	259
fragment_nearly_coincident	260
fragment_nop	262
fragment_optimize	263

fragment_preserve_length	265
fragment_ripple	266
fragment_visible	271
freeze_skew_edges	272
grids_for_angle_45_snap	273
image	274
jog_freeze	279
jog_ignore_metric	280
lapi_exec_tcl	282
layer (for denseopc_options Block)	285
line_end_fragment	292
lint	293
mask_chip_corner	297
max_iterations	300
max_iter_movement	301
max_opc_move	302
min_dog_ear_length	303
mpopc	304
mrc_consistency	309
mrc_mode	310
mrc_rule	311
mrc_rule_area	318
mrc_rule_priority	319
no_opc_region	320
nominal_epe_limit	321
one_time_site_pairing	322
opc_grid_multiplier	323
pband_tolerance	324
pw_condition	325
pwopc_mode (Deprecated in 2016.2)	332
read_layer_properties	333
retarget_layer	334
retargeting_options	338
scale	339
spa_promotion	340
sraf_promotion_distance	341
step_size	342
topo_model	343
topo_model_load	344
veb_corner_site_placement	345
veb_evalpts_per_fragment	347
veb_pseudo_nyquist	349
version	350
wafer_enclose	351
wafer_enclosedby	354
wafer_exclude	357
Calibre nmOPC File Formats	358
Lithomodel (Litho Model Format)	359

Table of Contents

Chapter 5	
Calibre nmOPC Custom	
Tcl Scripting Reference	367
Custom Tcl-Based Script Creation	369
Variables in Scripting	371
Outputs in Calibre nmOPC Scripts	371
Tag Scripting	372
Creating New Tag Sets	372
NEWTAG Output	373
Tag Set Operations	373
Tag Set Variables	373
Visible Layers and Tagging	374
Tagging Controls	374
Fragmentation Controls	376
Fragment-Based Operations	376
Measurement Controls	378
Dense Simulation Controls	378
Output Controls	379
Target Modification	379
OPC Controls	380
Unit Conversions and Miscellaneous	380
Site Controls	381
Calibre nmOPC Tcl Scripting Commands	382
auto_target_control	389
CLASSIFY_FRAGMENTS	392
DBU2UU	397
DENSE_CD	398
DENSE_EPE	400
DENSE_SIMULATE	402
DISPLACEMENT	404
EUV_MODEL_REDUCTION	409
FEEDBACK	413
fragmaxedge	416
FRAGMENT delete	418
FRAGMENT end	419
FRAGMENT modify	420
FRAGMENT split	422
FRAGMENT_CONFORM	424
FRAGMENT_EPE_MEASURE_METHOD	426
FRAGMENT_GET	427
FRAGMENT_GET_DISPLACEMENT	429
FRAGMENT_ITERATOR first/next/release	431
FRAGMENT_MAX_DISPLACEMENT	433
FRAGMENT_MOVE	435
FRAGMENT_SET_DISPLACEMENT	437
fragtypetag	439
GET_FRAGMENT_EPE_INFO	441
MEASURE	442

MEASURE_PER_FRAGMENT	444
MEASURE_PER_FRAGMENT_QUERY_COUNT	447
MEASURE_PER_FRAGMENT_QUERY_FIRST	448
MEASURE_PER_FRAGMENT_QUERY_NEXT	450
ML_OPC	452
ML_VECTOR_CAPTURE_END	454
ML_VECTOR_CAPTURE_INIT	458
MRC	460
MRC_RULE (Custom Scripting Command)	463
NEWTAG all	466
NEWTAG area	467
NEWTAG blocked	469
NEWTAG cd	470
NEWTAG complete	472
NEWTAG corner	474
NEWTAG correction_map	476
NEWTAG displacement	477
NEWTAG edge	479
NEWTAG epe	483
NEWTAG expression	485
NEWTAG external internal enclose enclosing	490
NEWTAG facing_pattern	496
NEWTAG fragment	498
NEWTAG line_end space_end	503
NEWTAG mrcViolation	504
NEWTAG neighbor	505
NEWTAG property	507
NEWTAG sadp	509
NEWTAG sequence	512
NEWTAG sites	514
NEWTAG topological	518
NEWTAG vertical horizontal all_angle 45_angle	520
NMBIAS_MEASURE_TAG	521
NYQUIST	528
OPC_ITERATION	529
OPC_XMEEF_ITERATION	541
OUTPUT_IMAGE	549
OUTPUT_MEASUREMENT_LOCATIONS	550
OUTPUT_OPCT	551
OUTPUT_RETARGET	553
OUTPUT_SHAPE fragment	554
OUTPUT_TARGET_CONTROL	558
PW_RETARGET	559
SET_EPE	560
SET_MEDIAN_EPE	564
SITES_CREATE	565
SITES_DELETE	570
SITES_DUMP	571
SITES_SHIFT (Deprecated in 2020.3)	573

Table of Contents

sraffragalign	575
sraf_print_avoidance	577
sraf_sites_create	584
TAG add remove ismember	586
TAG and or subtract	588
TAG fromlist	590
TAG size	591
TAG tolist	592
TAG_FRAG_SRC	593
TAG_MRC_SRC	595
TARGET_CONTROL	597
target_curve	600
TARGET_FUNCTION	606
TDBIAS	609
UU2DBU	611
Chapter 6	
Calibre nmOPC Best Practices	613
Recommended Settings for Best OPC Results	614
Recommended Settings for Best Runtime	615
Impact Reduction on Quality of Jogs	617
EPE Measurement Method Reset	617
Small Jog Movement Prevention	618
Process Window OPC Improvement	619
Using PV Band PWOPC	621
Sites for Minimum CD Control	623
Output Violation Tags	624
Optimize the Number of Process Window Iterations	624
Optimize the Number of PW Conditions	626
Optimize Simulation Time	626
Additional Recommendations for Process Window OPC	626
Enclosure of Contacts and Vias	627
Process Window Violations	627
Optimize Results From setlayer curve	629
Retarget Layers and setlayer curve	632
Setup File Best Practices	634
Recommended Settings for tilemicrons per Node	634
General Setup File Recommendations	635
MRC Rule Best Practices	636
Fragmentation Best Practices	638
Long Edge Fragmentation	638
Asymmetric Intrafeature Fragmentation for Long Edges	639
Fragmentation Consistency and Tile Boundaries	641
Asymmetric Intrafeature Fragmentation on the Short Edge	643
Asymmetric Interfeature Fragmentation on the Short Edge	645
Effect of Jogs on Fragmentation and OPC Consistency	647
Layer-Based Best Practices	649
POLY/ACTIVE Layers	649

Contact Layers	649
OPC Output Consistency	650
Practices to Improve VEB Results	653
Use Automatic Fragmentation	653
Set the Image Grid to 10 nm	655
Perform Convergence Analysis to Find Optimal Iterations	655
Use a Hint Offset	656
Set Final feedback to -1	657
Use Resist Constraints When Needed	657
Use Corner Chipping	658
VEB and notchfill	660
SRAF Print Avoidance Best Practices	661
Starting Recommendations for SRAFs	661
Negative SRAF Handling	662
Iterations for sraf_print_avoidance	664
SRAF Fragmentation	664
SRAFs and MRC Rules	664
Wafer-Level Data and Printing Threshold	665
SRAFs and Process Window OPC	666
Simplifying a Calibre mpOPC Setup File	667
Appendix A	
The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC	671
What is VEB Retargeting?	671
VEB Requirements	672
Add Retargeting to Your Process	672
Stage 1 - Calibrating the Optical, CM1 Resist, and VEB Models	673
Stage 2 - Retargeting the Design	674
Stage 3 - Using the Retargeted Design	676
Stage 4 - Verifying the OPC Output in Calibre OPCverify	678
Frequently Asked Questions	679
Appendix B	
Calibre OPC EUV Flows	681
Comparison of EUV Flows	682
Generating a Flare Map	684
Automatically Analyzing the Hierarchy	685
Adding Hierarchy to the OPC Recipe	686
Reading Reticle Information Files	688
EUV Information in the Transcript	690
Appendix C	
Calibre mlOPC	691
Calibre mlOPC Requirements	691
Collecting Model Data	692
Using the Trained Model	694

Table of Contents

Glossary

Index

Third-Party Information

List of Figures

Figure 1-1. Sparse OPC Versus Dense OPC	20
Figure 1-2. CM1 Modeling With Calibre nmOPC, Calibre OPCVerify, and Calibre LFD	21
Figure 1-3. EUV Correction Modules	23
Figure 1-4. Stitching Overlaps	23
Figure 1-5. Preventing Bridging in Mask-to-Mask	24
Figure 1-6. Complete Post-Tapeout Flow on the Calibre Platform	26
Figure 1-7. Calibre nmOPC Iteration Flow	27
Figure 2-1. Dense Grid Simulation	34
Figure 2-2. Cutlines Example	36
Figure 2-3. EPE Measurement Methods	36
Figure 3-1. Annotated Breakdown of a Minimum Setup File	43
Figure 3-2. Setup File Command Locations	45
Figure 3-3. Import Model Information	47
Figure 3-4. Define Layers	52
Figure 3-5. Layer Mapping Example	54
Figure 3-6. Wafer Enclosure on a Contact	57
Figure 3-7. Define an Imaging Background	58
Figure 3-8. Define an Image	59
Figure 3-9. Automatic Fragmentation With User Settings Example	62
Figure 3-10. Intrafeature Fragmentation (Corner-Based)	63
Figure 3-11. Interfeature Fragmentation (Ripple Fragments)	64
Figure 3-12. Island-Layer Fragmentation	64
Figure 3-13. Define Mask Rule Constraints	65
Figure 3-14. Define Process Windows	67
Figure 3-15. Nominal Versus Process Window Conditions	69
Figure 3-16. Using PV Band PWOPC	71
Figure 3-17. Etch and Litho Target on a 10 nm M1 Layer Error	72
Figure 3-18. Comparison of VEB and VEB + PWOPC Results	73
Figure 3-19. Top Loss Effects	75
Figure 3-20. Check Enclosure With Image Cost Functions	77
Figure 3-21. Wafer Enclosure Example	78
Figure 3-22. Define No OPC Regions	79
Figure 3-23. No OPC Region	79
Figure 3-24. Smooth Biasing on the Target Layer	80
Figure 3-25. Retargeting the Input Layer	82
Figure 3-26. Smoothed Target Layer	82
Figure 3-27. Adjust for Mask Corner Rounding	83
Figure 3-28. Polygon With Corner Rounding	83
Figure 3-29. Simulate Layer-Based Images Using Calibre OPCVerify	84
Figure 3-30. Create Custom Tcl Scripts for Calibre nmOPC	85

Figure 3-31. SRAF Correction Example	86
Figure 3-32. sraf_print_avoidance Example	87
Figure 3-33. Simulation Transcript Segment	88
Figure 4-1. Setup File Location for LITHO DENSEOPC Commands	115
Figure 4-2. Effect of minStepTanDist	153
Figure 4-3. Effect of colinearAngleTolerance 1	154
Figure 4-4. Curvature Example	156
Figure 4-5. Comparisons of linearRatio True Versus False	157
Figure 4-6. Setup File Location for denseopc_options Commands	174
Figure 4-7. Gate CD Prioritization	180
Figure 4-8. Active Layer Enclosure	180
Figure 4-9. Gate Flare Reduction Using active_layer	181
Figure 4-10. Algorithm 1 Effects	184
Figure 4-11. Multiple-Mask Exposures	189
Figure 4-12. External Corner-to-Corner	196
Figure 4-13. Internal Corner-to-Corner	196
Figure 4-14. Changing the Coincident Layer to Affect Fragmentation	205
Figure 4-15. Intrafeature Fragmentation Example	212
Figure 4-16. corner_next and length_next	215
Figure 4-17. Effect of any on length2	216
Figure 4-18. fragment_inter -minshield 2 -shield 4 -ripples 1	222
Figure 4-19. Example -ripplestyle	224
Figure 4-20. Interfeature Shielding Example	225
Figure 4-21. -split Example	226
Figure 4-22. Using Shield With Interfeature Fragmentation	228
Figure 4-23. -skipCorners Option	231
Figure 4-24. Typical Fragmentation Caused by a Scattering Bar	231
Figure 4-25. Symmetric Fragmentation Caused by a Scattering Bar and -sym	232
Figure 4-26. Effect of -combined	233
Figure 4-27. Without -dontcross Option	237
Figure 4-28. With -dontcross Option	237
Figure 4-29. Island Layer Fragmentation Example	240
Figure 4-30. Interfeature Fragmentation With full_edge	246
Figure 4-31. fragment_max	254
Figure 4-32. fragment_ripple corner Versus frag_corner	270
Figure 4-33. jog_ignore_metric Behavior	280
Figure 4-34. Mask Chipping Process	298
Figure 4-35. The Four Metrics	314
Figure 4-36. Effects of retarget_layer emulate	335
Figure 4-37. EPE Measurements at Corners (Example 1)	336
Figure 4-38. EPE Measurements at Corners (Example 2)	337
Figure 4-39. wafer_enclose Example	352
Figure 4-40. Corner Rounding (wafer_enclose)	352
Figure 4-41. wafer_enclosedby Example	355
Figure 4-42. Corner Rounding	355

List of Figures

Figure 5-1. Setup File Location for Tcl Scripting Commands	382
Figure 5-2. Symmetrical X and Y Groups	394
Figure 5-3. The Four Metrics for Tag-Based MRC	464
Figure 5-4. Corner and Len Definitions	480
Figure 5-5. Effect of -perpendicular_only and -corner_block	491
Figure 5-6. Enclosing -c2c Constraints	492
Figure 5-7. Example of mcorner and rdisp	502
Figure 5-8. NEWTAG sadp Layers and Tag Sets	509
Figure 5-9. projlen Measurement Resolution	523
Figure 5-10. Enclosing -c2c Constraints	524
Figure 5-11. Projecting Length (projlen)	525
Figure 5-12. Space2, Space3, Width2, and Width3	525
Figure 5-13. XMEEFs Matrix	542
Figure 5-14. XMEEF Heat Map	545
Figure 5-15. OPC Output Per Iteration Example	552
Figure 5-16. SITES_DUMP Example	572
Figure 5-17. Effect of minStepTanDist	601
Figure 5-18. Effect of colinearAngleTolerance 1	602
Figure 5-19. TARGET_FUNCTION Example 1	607
Figure 5-20. TARGET_FUNCTION Example 2	607
Figure 5-21. TARGET_FUNCTION Example 3	608
Figure 6-1. Jogs and EPE Measurement Method	617
Figure 6-2. Resetting EPE Measurement Method for Jogs	618
Figure 6-3. Important Steps of PWOPC Implementation	620
Figure 6-4. PV Band PWOPC Example	622
Figure 6-5. Sites for Minimum CD Control	623
Figure 6-6. Example of PWOPC With Minimum CD	623
Figure 6-7. Optimizing PW Iterations	625
Figure 6-8. Enclosure of Contacts and Vias	627
Figure 6-9. Violation Tags	628
Figure 6-10. Example of Line/Space End Scaling	630
Figure 6-11. Length and Width Representation for setlayer curve	631
Figure 6-12. Effects of retarget_layer emulate and setlayer curve	633
Figure 6-13. Using Length-Based MRC Rules	637
Figure 6-14. Long Edge Fragmentation Example 1	638
Figure 6-15. Long Edge Fragmentation Example 2	639
Figure 6-16. Effect of fragment_corner With Four Ripples	640
Figure 6-17. Effect of fragment_corner With Five Ripples	641
Figure 6-18. Effect of Smaller -interdistance Values at Tile Boundaries	642
Figure 6-19. Effect of Larger -interdistance Values at Tile Boundaries	643
Figure 6-20. Asymmetric Fragmentation for Short Edges	644
Figure 6-21. Symmetrical Fragmentation	645
Figure 6-22. Asymmetric Interfeature Fragmentation	646
Figure 6-23. Symmetric Interfeature Fragmentation	647
Figure 6-24. Jog Impact on Fragmentation	647

Figure 6-25. Fragmentation Corrected for Jog Corners.....	648
Figure 6-26. I-Shape OPC Output	649
Figure 6-27. Dense Simulation Inconsistency Illustration.....	650
Figure 6-28. MRC-Related Inconsistency	651
Figure 6-29. Incorrect Manual Fragmentation Example	654
Figure 6-30. Automatic Fragmentation Example	654
Figure 6-31. veb_pseudo_nyquist Optimization Example	655
Figure 6-32. Convergence Analysis.....	656
Figure 6-33. Hint Offset.....	656
Figure 6-34. Unrealistic Target Example	657
Figure 6-35. mrc_rule Adjustment.....	658
Figure 6-36. Corner Rounding Effects	658
Figure 6-37. mask_chip_corner Example	659
Figure 6-38. SRAF Line End Fragment Example	665
Figure A-1. VEB Retargeting Flow.....	672

List of Tables

Table 1-1. Syntax Conventions	28
Table 2-1. Example SVRF Commands Used With Calibre nmOPC	34
Table 2-2. Required Calibre nmOPC Setup File Commands	38
Table 3-1. Topography Modeling Development Flow	51
Table 3-2. Layer Descriptions	56
Table 3-3. Optional Calibre nmOPC Setup File Operations	60
Table 3-4. Calibre nmOPC Fragmentation Defaults	61
Table 3-5. Process Window Conditions	68
Table 4-1. EUV-Specific Commands	94
Table 4-2. Machine Learning Commands	95
Table 4-3. DP- and MP-Specific Commands	96
Table 4-4. VEB-Specific Keywords	96
Table 4-5. Calibre nmOPC SVRF Commands	97
Table 4-6. LITHO DENSEOPC Setup File Commands	116
Table 4-7. Calibre OPCverify Setup File Commands	117
Table 4-8. Custom Lint Checks	137
Table 4-9. Keywords Accepted in veb_retarget OPTIONS Block	167
Table 4-10. denseopc_options Summary	175
Table 4-11. Available Flag Names	217
Table 4-12. Supported Fragmentation Keywords in fragment_layer Block	244
Table 4-13. Fragmentation Allowed in a fragment_layer Block	249
Table 4-14. Examples of Fragment Splitting	254
Table 4-15. layer Command layer_types (denseopc_options)	286
Table 4-16. layer Command transmission Argument Settings (denseopc_options)	288
Table 4-17. Calibre nmOPC Fragmentation Defaults	339
Table 4-18. Calibre nmOPC File Formats	358
Table 5-1. NEWTAG Operations	372
Table 5-2. Tag Set Operations	373
Table 5-3. Calibre nmOPC Tcl Custom Script Tagging Commands Summary	374
Table 5-4. Fragmentation Controls Summary	376
Table 5-5. Fragment-Based Operations Commands Summary	377
Table 5-6. Calibre nmOPC Tcl-Based Measurement Commands Summary	378
Table 5-7. Dense Simulation Commands Summary	378
Table 5-8. Output Commands Summary	379
Table 5-9. Target Function Controls Summary	379
Table 5-10. OPC Controls Summary	380
Table 5-11. Unit Conversion and Miscellaneous Summary	381
Table 5-12. Site Controls Summary	381
Table 5-13. Calibre nmOPC Custom Scripting Commands Summary	382
Table 5-14. Special Variables for SET_EPE	560

Table 5-15. Expressions for Setting EPE	561
Table 6-1. Recommended Settings for Best OPC Results	614
Table 6-2. Recommended Settings for Best Runtime	615
Table 6-3. General Recommended Settings for setlayer curve	629
Table 6-4. Line/Space End-Specific Recommended Settings for setlayer curve	630
Table 6-5. Recommended Jog-Specific Settings for setlayer curve	632
Table 6-6. Hardware Requirements and tilemicrons Settings (Calibre nmOPC)	634
Table A-1. The VEB Model-based Retargeting Process	673
Table A-2. Calibre nmOPC and OPCVerify Keyword Requirements for VEB Retargeting	674

Chapter 1

Introduction to Calibre nmOPC

Calibre® nmOPC™ is a full-chip batch OPC tool that corrects potential distortions on a mask input layer introduced by the optical and resist processes.

The corrections are performed using a pixel-based or dense simulation engine to calculate the wafer image contour (referred to as “Dense OPC”). This is in contrast to the shape-based simulation sites that are used in the Calibre® OPCpro™ tool (referred to as “Sparse OPC”). Calibre nmOPC was developed for the 65 nm technology node and below and can be integrated into the design to mask flow in the same way as Calibre OPCpro.

Calibre nmOPC uses the SVRF and TVF controls and command language formats common to all Calibre tools. This manual explains and describes the key concepts, setup file commands, and scripting commands for Calibre nmOPC.

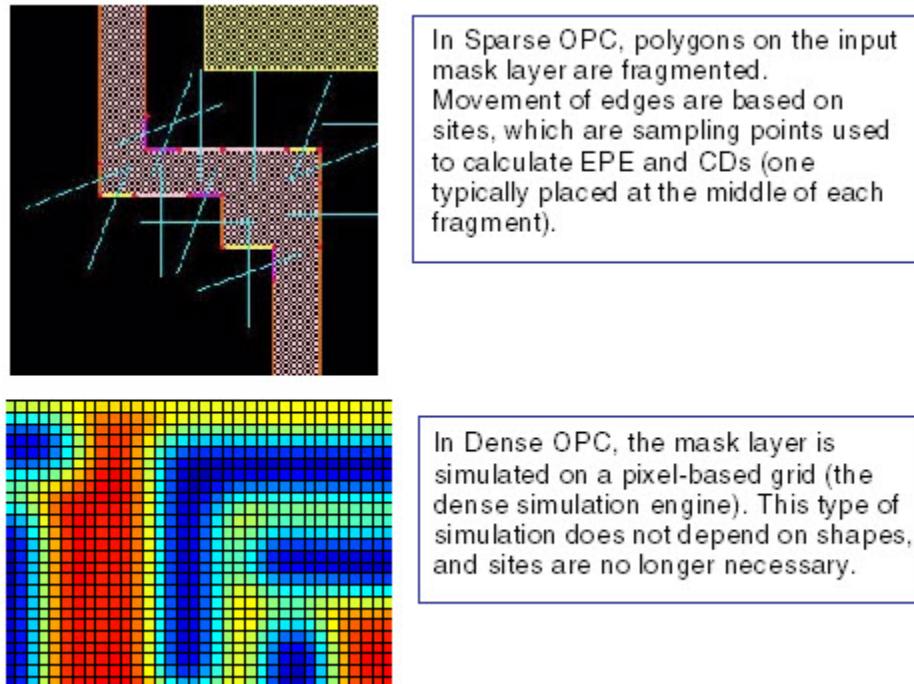
Calibre nmOPC Key Concepts	20
CM1 Modeling	21
Variable Etch Bias	21
Flare and Shadow Modeling	22
Double and Multi-Patterning Correction	23
Tag Set Properties	24
Calibre nmOPC Usage	25
Calibre nmOPC Workflow	25
Calibre nmOPC Iteration Flow	26
Syntax Conventions	27

Calibre nmOPC Key Concepts

The Calibre nmOPC tool provides several extensions and enhancements to Calibre OPCpro technology that are geared for the challenges implicit in the 65 nm technology nodes and below.

The Calibre nmOPC tool performs OPC corrections on a mask input layer using a pixel-based or dense simulation engine to calculate the wafer image contour. Calibre nmOPC can be integrated into the design-to-mask flow in the same way as Calibre OPCpro.

Figure 1-1. Sparse OPC Versus Dense OPC



Calibre nmOPC expands on the functionality offered in Calibre OPCpro with these enhancements and additional features:

- Advanced modeling features and capabilities to provide accurate process window simulation accuracy
- Simplified run scripts and setup files integrated with Calibre® OPCVerify™ syntax
- Pixel-based dense simulation removes need for sites
- Circuit-aware OPC cost functions including contact and via coverage, and gate control constraints during OPC
- Process-window OPC capabilities to optimize CD control through varying process conditions
- Integrated support for retargeting using rule or model-based pre-biasing (also known as “target smoothing”)

- A resist model specifically designed for the dense simulation engine, CM1 (see “[CM1 Modeling](#)” on page 21)
- A new modeling step to account for etch biasing (see “[Variable Etch Bias](#)” on page 21)

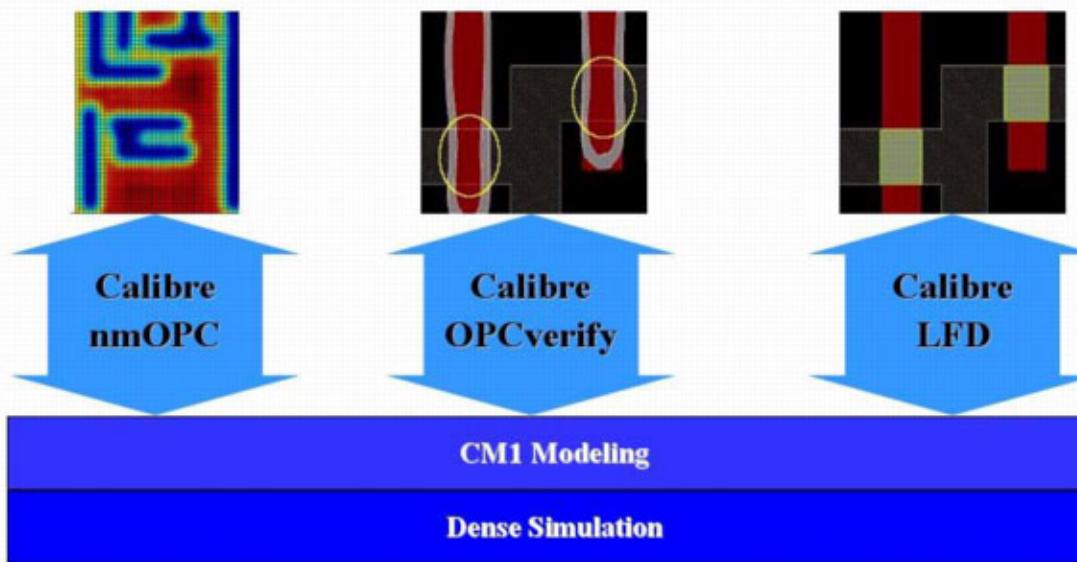
CM1 Modeling	21
Variable Etch Bias	21
Flare and Shadow Modeling	22
Double and Multi-Patterning Correction	23
Tag Set Properties	24

CM1 Modeling

The Compact Model 1 (CM1) resist model has been specifically developed for use with Calibre nmOPC. The Calibre nmOPC tool takes advantage of the data available for resist simulation with a dense simulation engine.

CM1 modeling also works seamlessly with Calibre® OPCVerify™ and Calibre® LFD™. For more information on CM1 modeling, see the [Calibre WORKbench User’s and Reference Manual](#).

Figure 1-2. CM1 Modeling With Calibre nmOPC, Calibre OPCVerify, and Calibre LFD



Variable Etch Bias

Variable Etch Bias (VEB) can be used as a modeling step to separately account for etch effects.

The VEB (Variable Etch Bias) model-based retargeting flow:

- Separates out the modeling of etch bias effects from optical and resist effects
- Separates out the correction of etch effects and correction of optical and resist effects
- Derives an etch-corrected design based on the VEB model. It uses fragmentation and edge movement calculations, similar to making OPC corrections to a layout.

Separating out the post-etch data and using it to retarget the design:

- Reduces the run-time of etch bias correction, since the corrections are applied to the fragment edges on the original target rather than applied to the resist contours.
- Allows you greater control over your OPC corrections by separating the resist and etch effect requirements.
- Allows you to verify post-OPC corrections at the resist level against the resist target, enabling detection of problems in the resist phase.

Refer to “[The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC](#)” appendix for complete information.

Flare and Shadow Modeling

Flare effects are a significant concern for extreme ultraviolet (EUV) lithography. Flare in EUV lithography is a random effect caused by light scattering due to optics surface roughness and defects in the multilayered optical surfaces.

Shadowing mainly occurs due to oblique incidence of light on the mask. In EUV, not all the light incident on the mask will be reflected back to the wafer, which leads to a noticeable difference in width/space of the printed patterns with respect to the designed ones.

The Calibre RET tool suite can be used to compensate for density-dependent fluctuations in conventional DUV lithography. This can be as simple as making corrections using rules for the variations of isolated and dense lines in an environment with prescribed flare, or a more complex correction incorporating flare and shadow biasing into model-based OPC.

Calibre nmOPC is a critical part of the correction flow for EUV lithography, consisting of several module/pieces that can be put together in any order to complete a working correction flow.

Figure 1-3. EUV Correction Modules



Related Topics

[Calibre OPC EUV Flows](#)

Double and Multi-Patterning Correction

Calibre mpOPC is designed for double and multiple mask scenarios where a layout can be split into separate masks for correction, and then “stitched” back together by overlapping the split masks. Calibre nmOPC provides the ability to correct for mask-to-mask issues, such as bridging or a lack of proper overlapping between masks, ensuring that the stitching process will succeed.

The condition where mask-to-mask correction is most critical is the process option of Litho Freeze Litho Etch (LFLE) or Litho Process Litho Etch (LPLE) where the two masks have the potential to interact. The risk of bridging or near-bridging becomes the primary concern in that situation. Thus, the ability to do interlayer bridge checks between the two contours is crucial.

Figure 1-4 shows an example of using Calibre mpOPC to ensure good stitching overlap between masks. The blue and red shapes indicate separate masks. The areas where they overlap are where the masks will “stitch” them back together.

Figure 1-4. Stitching Overlaps

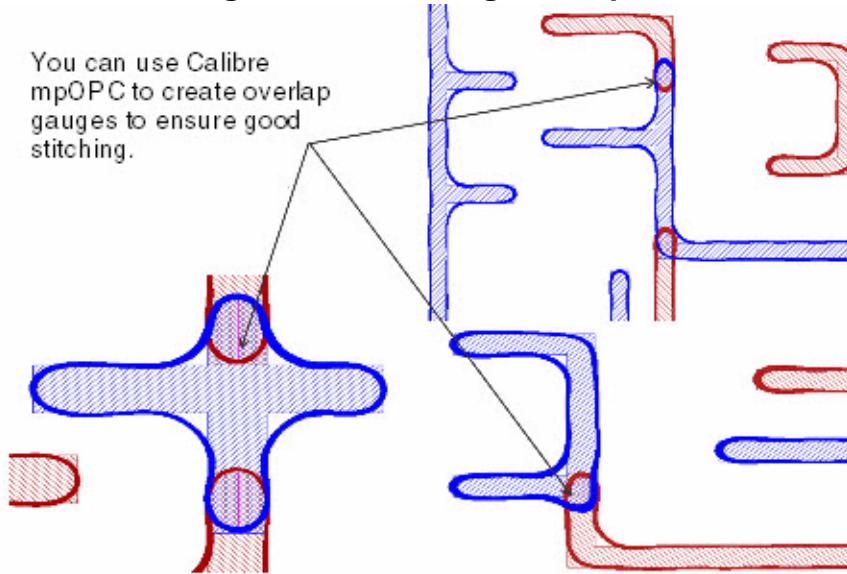
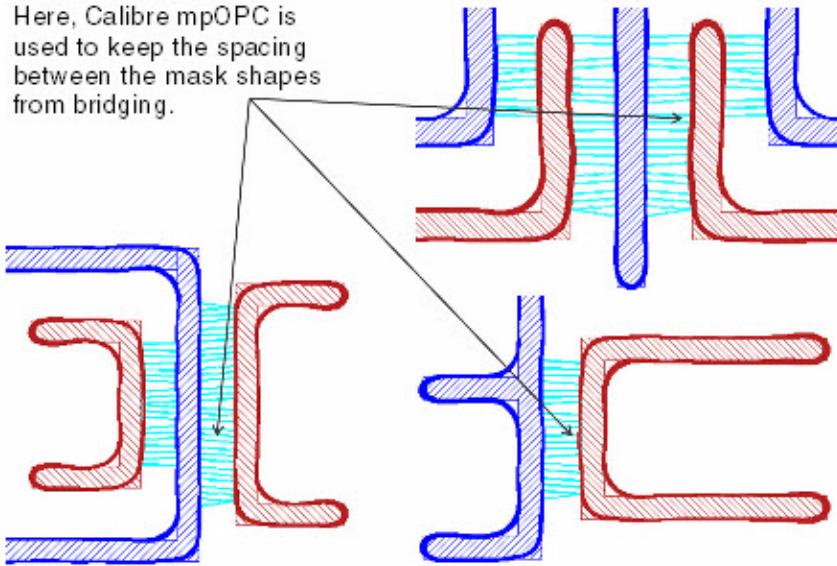


Figure 1-5 shows an example of using Calibre mpOPC to create inter-mask gauges to prevent bridging between mask-to-mask shapes.

Figure 1-5. Preventing Bridging in Mask-to-Mask



Tag Set Properties

Some Calibre nmOPC commands can make measurements on a layout and save the values in a tag set. These values can be passed to the layout using the Calibre OPCverify property mechanism. There are some additional limitations, however.

Unlike Calibre OPCverify, most Calibre nmOPC commands can output no more than one property at a time. The property is saved to a tag set along with its associated fragment.

The properties can be written to layers using the OUTPUT_SHAPE command. Each property is annotated to a separate marker on the layer. Layers are output to the results database using the standard SVRF commands.

The property values are strictly associated with a tag set. If a fragment has three different tags (that is, it is a member of three different tag sets), the value returned depends on the tag set specified. For example, consider a fragment that is a member of the sets all_frags, line_end, and a subset of line_end called short_line_end. If the length was saved to the tag set line_end, attempts to access it using the subset short_line_end will not work even though the short_line_end members are all also members of line_end.

Related Topics

[Writing Properties to Outputs \[Calibre OPCverify User's and Reference Manual\]](#)

Calibre nmOPC Usage

Calibre nmOPC is intended for use at 65 nm and below. Depending on layer critical dimensions and pattern density, Calibre nmOPC will provide faster run times for critical layers than Calibre OPCpro due to the computational efficiencies inherent in performing dense over sparse simulations.

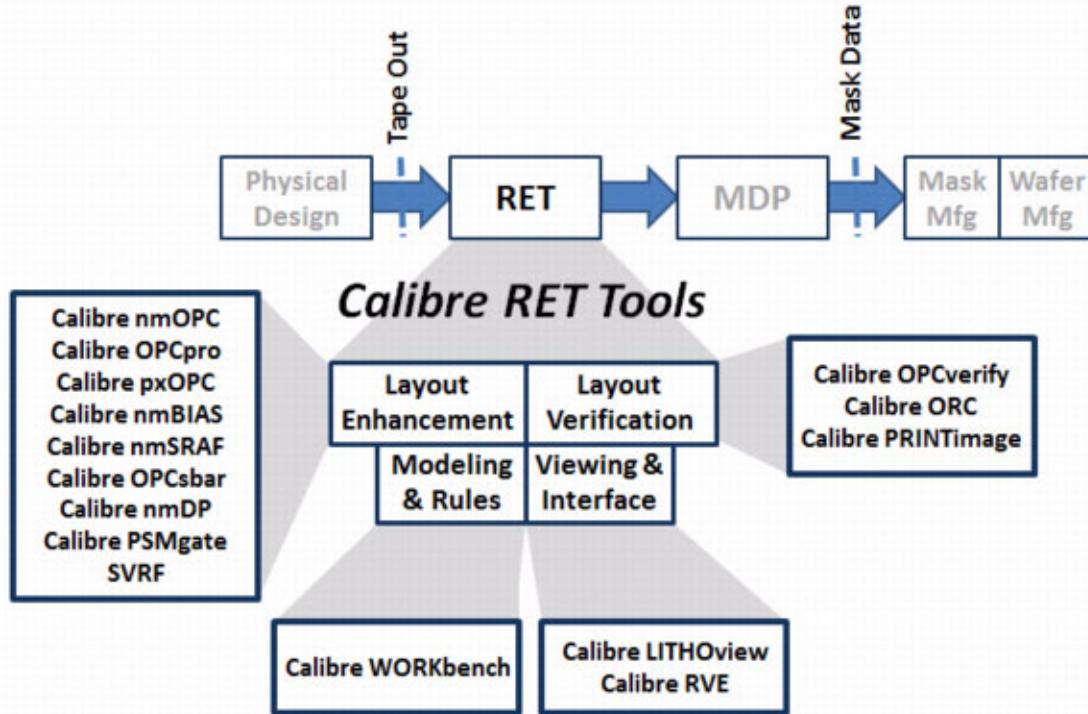
A crossover point exists between the 65 nm and 45 nm half-pitch technology nodes where dense or pixel-based simulation becomes much more efficient than a sparse or site-based OPC solution. Calibre nmOPC improves computational efficiencies necessary to manage both the technical challenges at 45 nm and below and the cost containment challenges for leading edge processes and chip designs. The advantage of Calibre nmOPC is to drive a more cost-effective process.

Calibre nmOPC Workflow

Calibre nmOPC is fully integrated with the complete set of Calibre RET tools. Calibre nmOPC and Calibre OPCverify provide dense OPC and OPC verification capabilities in an integrated flow. Together with Calibre® nmDRC™, Calibre® OPCsbar™, and Calibre® FRACTURE™, a complete hierarchical post-tapeout flow can be built on the Calibre platform.

Figure 1-6 shows the complete design to mask flow based on the Calibre platform. Calibre nmOPC is part of the post-tapeout flow and can also be used with Calibre® Local Printability Enhancement (LPE).

Figure 1-6. Complete Post-Tapeout Flow on the Calibre Platform

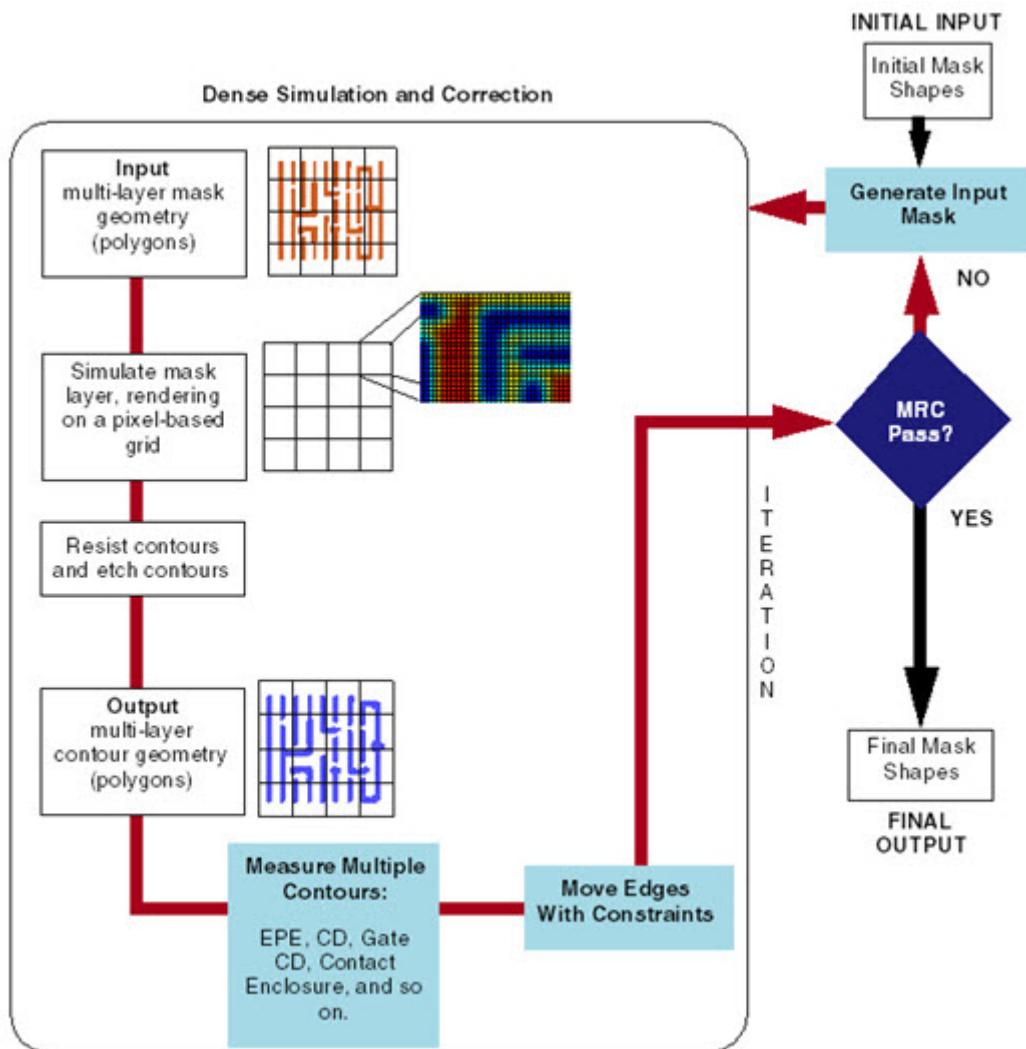


Calibre nmOPC Iteration Flow

Calibre nmOPC corrects for potential distortions introduced by the optical and resist processes through an iterative process. It repeats the process until the printed image closely approximates the original layout.

A basic overview of the iterative process is illustrated in [Figure 1-7](#).

Figure 1-7. Calibre nmOPC Iteration Flow



Syntax Conventions

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

Table 1-1. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	Underlining indicates either the default argument or the default value of an argument.
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.
‘ ’	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
or	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
...	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.

Example:

```
DEvice {element_name [‘(‘model_name‘)’]}

  device_layer {pin_layer [‘(‘pin_name‘)’] ...}

    [‘<‘auxiliary_layer‘>’ ...]

    [‘(‘swap_list‘)’ ...]

    [BY NET | BY SHAPE]
```

Chapter 2

Calibre nmOPC Quick Start

Calibre nmOPC can be utilized quickly using some basic processes, including the creation of a minimum setup file.

Calibre nmOPC Prerequisites	29
Calibre nmOPC Configuration Summary	30
Variable Etch Bias (VEB) Model Process Summary	31
Calibre OPCverify in the Calibre nmOPC Setup File	31
The Calibre nmOPC Setup File	31
What the Setup File Controls in a Calibre nmOPC Run.....	33
What the Rule File Controls in a Calibre nmOPC Run.....	34
How Does Calibre nmOPC Take EPE Measurements?.....	35
Fragment Movement With Calibre nmOPC.....	36
Customized Correction Algorithms in Calibre nmOPC	37
Variables, Keywords, and Commands Supported in the Setup File.....	37
Calibre nmOPC Modes of Operation	39
Considerations for Calibre FullScale Runs.....	41
Acronyms and Terminology	42

Calibre nmOPC Prerequisites

If you intend to run Calibre nmOPC, you should be familiar with Calibre nmDRC and Calibre OPCverify® procedures. An understanding of programming SVRF and TVF rule files is very useful.

To use Calibre nmOPC, you must have the following items:

- **Platform support** — Calibre nmOPC 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** — To run any of the Calibre layout viewers, you must have the license file required by the tool. This includes the following:
 - Calibre version 2006.4 or higher.
 - Licenses for Calibre nmOPC, Calibre nmDRC, and Calibre WORKbench. Siemens EDA recommends that you have Calibre OPCverify to verify your corrections.

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

- **Required files** — The following files are required to perform Calibre nmOPC operations:
 - SVRF file.
This rule file contains specification and operation statements with verification rules that are applied to your designs.
 - Setup file (start with “[Calibre nmOPC Configuration Summary](#)” on page 30).
The setup file is used to configure Calibre nmOPC for simulations. A setup file can be a separate file or it can be placed into a SVRF script.
 - GDS or OASIS®¹ layout.
 - Model information: optical model and resist model.
Refer to the [Calibre WORKbench User's and Reference Manual](#) for information on creating optical and resist models for dense simulations.

Calibre nmOPC Configuration Summary

You run Calibre nmOPC as a batch run using the Calibre Hierarchical Engine.

You control how the tool functions through a setup file and an SVRF file. The basic steps are as follows:

1. Create a Calibre nmOPC setup file manually using a text editor (referencing it in the SVRF file).
2. Create an SVRF rule file that invokes Calibre nmOPC. This rule file references the setup file created in step 1 in one of the following ways:
 - a. As an external file, in which case the setup file must be saved as a separate file.
 - b. As an internal block. In this case the entire setup file must be placed inside the SVRF rule file as a uniquely-named LITHO FILE block, and referenced by the name of that block.
3. Run Calibre on the SVRF rule file.
4. View the results in Calibre® WORKbench™ or your preferred viewer.

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.

Variable Etch Bias (VEB) Model Process Summary

If you wish to implement a flow that accounts for the VEB model, there is a particular process to follow in creating a Calibre nmOPC setup file.

VEB model-based retargeting uses the following main steps:

1. Calibrate the Optical, CM1 resist, and VEB models on a test pattern.
2. Using the VEB model, retarget the original design file in Calibre nmOPC to generate an etch-corrected design file.
3. Using the etch-corrected design file, run your simulations through Calibre nmOPC to generate an OPC-corrected file.
4. Using the OPC-corrected file from Calibre nmOPC, inspect the design file in OPCverify.

When Calibre nmOPC corrects for the VEB model, it uses an alternate setup file (as opposed to the typical Calibre nmOPC setup file that references optical and resist models) with several etch model-specific commands. Refer to “[The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC](#)” appendix for complete information.

Calibre OPCverify in the Calibre nmOPC Setup File

You can add Calibre OPCverify commands that work on setlayer image commands in a Calibre nmOPC setup file. For Calibre nmOPC, this means that you can combine Calibre nmOPC and Calibre OPCverify verification in a single setup file.

Refer to “[Simulate Layer-Based Images Using Calibre OPCverify](#)” on page 83 for further information.

The Calibre nmOPC Setup File

Setup files for Calibre nmOPC vary from simple (leveraging the built-in algorithms to streamline corrections) to extremely complex (defining custom correction recipes based on the dense simulations).

The following is an example of a minimal Calibre nmOPC setup file:

```
##### SVRF layer derivation #####
m1_out = LITHO DENSEOPC metall sraf
FILE m1_opc MAP m1_opc
```

```
LITHO FILE m1_opc [
#####
# Simulation Models #####
modelpath ./models
optical_model_load opt m1.opt
resist_model_load res m1.cm1

#####
# Input Layers/Background #####
layer m1 hidden clear
layer sraf hidden clear

denseopc_options m1_opt {
#####
# Version #####
version 1
#####
# Background/OPC Layers #####
background poly opc atten 0.06
layer m1 opc clear
layer sraf visible clear
#####
# Nominal Image#####
image optical opt resist res
}

setlayer m1_opc = denseopc m1 sraf MAP m1 OPTIONS m1_opt
]
```

As mentioned previously, one option is to place the setup file inside the SVRF rule file used to invoke Calibre nmOPC.

The following example shows an excerpt from such a rule file. Notice that, in this case, the setup file is slightly more complex, with a few additional controls for Calibre OPCVerify and for mask rule checking.

```
LAYOUT PATH "$LAYOUT_INPUT"
LAYOUT SYSTEM OASIS
LAYOUT PRIMARY "*"
PRECISION 1000
RESOLUTION 1
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "result.oas" OASIS PSEUDO
DRC SUMMARY REPORT "result.report"
LAYOUT MAGNIFY 1.0

# Input Layers are defined with LAYER command

LAYER poly 3
poly { COPY poly } drc check map poly 3
LAYER sbar 5

# Derived Layers are defined using LITHO DENSEOPC command

dense_opc = LITHO DENSEOPC FILE MY_DENSE_FILE poly MAP dense_opc
dense_opc {COPY dense_opc} DRC CHECK MAP dense_opc 30
```

```

# A nmOPC setup file command block, MY_DENSE_FILE, is placed within a
# LITHO FILE statement

LITHO FILE MY_DENSE_FILE /*

modelpath ./models:$env(DESIGN_DIR)/models
optical_model_load opticsfile $env(OPTICAL_MODEL)
resist_model_load resistfile $env(RESIST_MODEL)

layer poly visible atten 0.06

# A DENSEOPC sub-command block, opt, creates the OPC algorithm

denseopc_options opt {
    version 1
    max_iterations 5
    step_size 0.001
    image optical opticsfile resist resistfile
    background clear
    layer poly opc atten 0.06

# Mask rule checks are made
    mrc_rule external poly {
        use 0.063 euclidean
        length 0.02 use 0.04 euclidean
    }
}
# opt is passed to a setlayer command
# A new output layer, dense_opc is created

setlayer dense_opc = denseopc poly MAP poly OPTIONS opt
*/]

```

The OPC algorithm results generate a new layer, `dense_opc`, as specified by the `setlayer denseopc` command. The LITHO DENSEOPC command then specifies the layer with the `MAP` value. It is written to the SVRF derived layer, `dense_opc`. Finally, the results are written to the results database (RDB) with the DRC CHECK MAP command. If the OPC layer is not passed to the output, the Calibre run includes a warning in the Lint Check section of the transcript:

```

//-- WARN: None of the OPC or correction layer(s) are mapped out through
LITHO DENSEOPC. Output may not be valid.

```

When this occurs, check your rule file and litho file. (The layer names are case-sensitive and must match.)

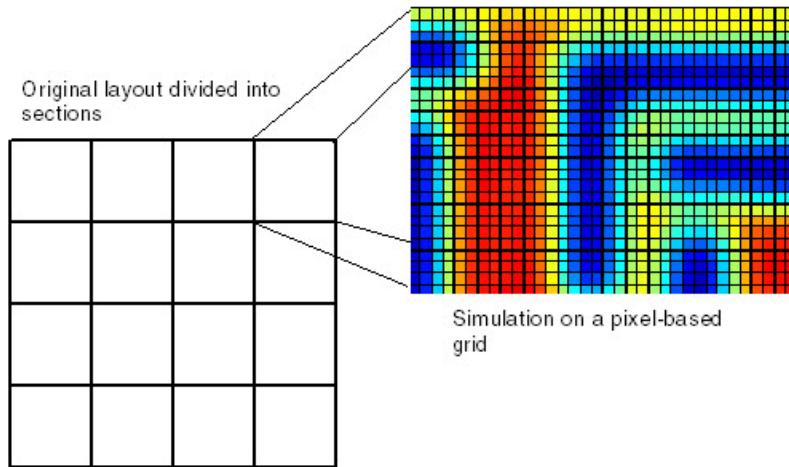
What the Setup File Controls in a Calibre nmOPC Run

In the Calibre nmOPC setup file, you control how simulations are performed, how corrections are applied, and any post processing or verification required.

You can design a setup file with minimal simulation controls because dense simulation samples data at evenly spaced grid points on a dense simulation grid. You do not need to specify the size

of this grid because the optimal grid size, or Nyquist sampling distance (also referred to as “Nyquist value” in this document), is calculated based on the optical conditions.

Figure 2-1. Dense Grid Simulation



Correction controls can also be minimal because the tool fragments the polygon edges for you based on the NSD, calculates the edge placement error (EPE), and moves edges based on the EPEs.

Refer to “[Variables, Keywords, and Commands Supported in the Setup File](#)” on page 37 to see what types of controls are available for the Calibre nmOPC setup file.

What the Rule File Controls in a Calibre nmOPC Run

In the SVRF rule file, you control what and how the Calibre Hierarchical Engine reads layer data in from the original layout database and writes results out to the results database. In addition, you use SVRF commands to perform pre- and post-processing on the Calibre nmOPC data and to invoke Calibre nmOPC.

Table 2-1 lists several SVRF commands used with Calibre nmOPC.

Table 2-1. Example SVRF Commands Used With Calibre nmOPC

Calibre nmOPC Commands Used	Task
LITHO FILE (SVRF link)	A specification command that defines a setup file.
LAYER (SVRF link)	Declare the name of a layer with a layer number.

Table 2-1. Example SVRF Commands Used With Calibre nmOPC (cont.)

Calibre nmOPC Commands Used	Task
LITHO DENSEOPC	<p>Generate derived layers. The order you give to input layers determines the order of assignment in the Calibre nmOPC setup file. For example:</p> <pre>out1 = LITHO DENSEOPC m1 m2 FILE nmopc MAP m1_opc out2 = LITHO DENSEOPC m1 m2 FILE nmopc MAP m2_opc</pre> <p>In this example, m1 and m2 are input layers; m1 will map to the first layer listed in the setup file, m2 will map to the second, and m1_opc and m2_opc are the outputs created in the setup file.</p>

For a complete description of the SVRF rule file, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).

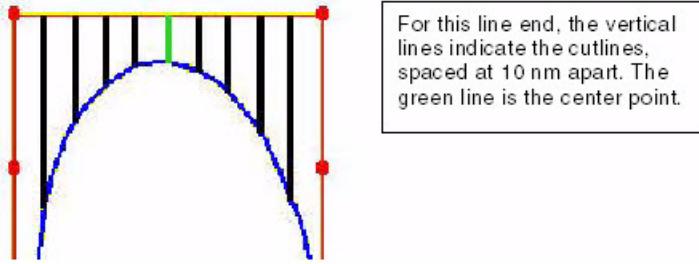
How Does Calibre nmOPC Take EPE Measurements?

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. Sampling EPE for simulation in OPC is one of the most important factors in getting accurate output.

Since Calibre nmOPC uses a dense simulated contour, every fragment has multiple EPE measurements, taken at a number of cutlines (sampling points). Each fragment is fitted with N measurement cutlines (where N is the number of cutlines that can fit in the fragment) given the following guidelines:

- The first cutline starts at the center of the fragment
- All subsequent cutlines ripple outward from the center
- The spacing between cutlines are a default 10 nm
- Measurement points are never placed on fragment endpoints
- The maximum search distance from target to contour is $(8/3 * \text{Nyquist sampling distance (NSD)})$

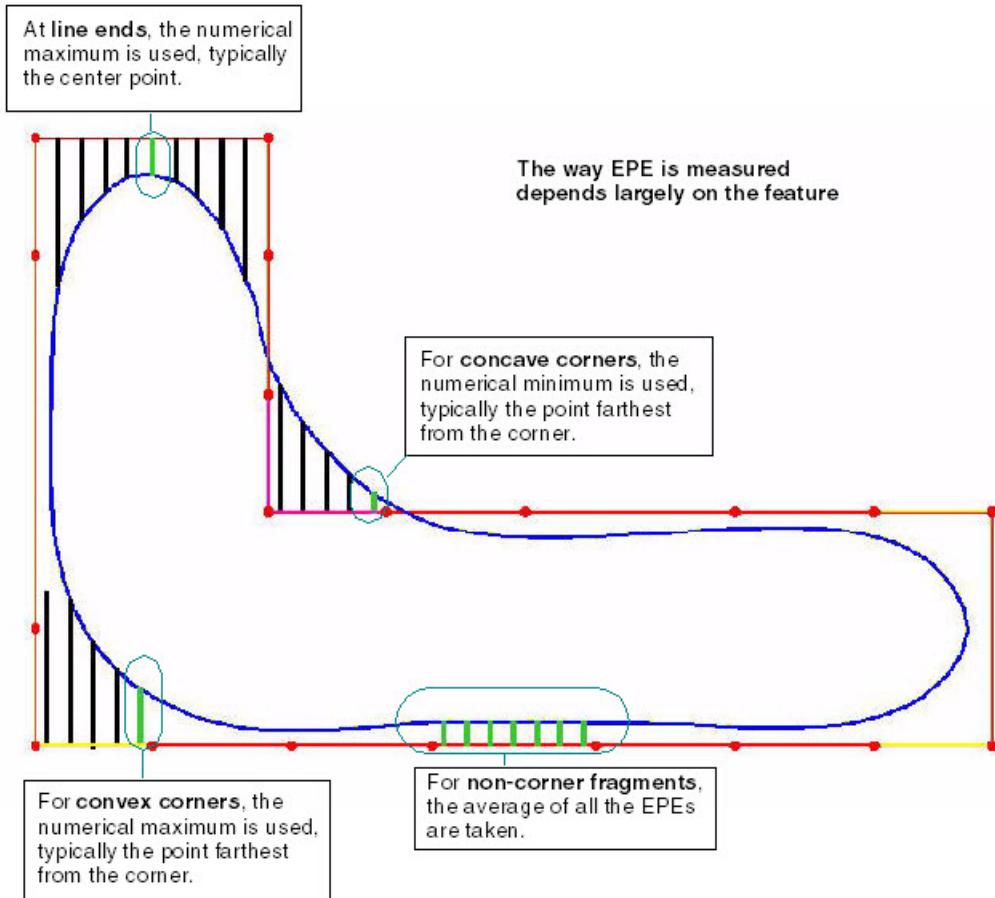
Figure 2-2. Cutlines Example



For this line end, the vertical lines indicate the cutlines, spaced at 10 nm apart. The green line is the center point.

More specifically how EPE is measured will vary according to the feature, as shown in Figure 2-3.

Figure 2-3. EPE Measurement Methods



Fragment Movement With Calibre nmOPC

The Calibre nmOPC tool uses the simulated EPE values to calculate how far to move an edge.

Fragments are moved based on the following formula:

$$\text{EPE}_{\text{effective}} * \text{feedback}$$

where *feedback* is a factor that can be adjusted for every fragment during each iteration.

The tool performs multiple iterations of the simulation-correction cycle until it achieves convergence with the original design. “Convergence” occurs when the results of OPC edge correction reflect the closest possible representation of the original mask feature.

For edges that are jogs, movement is constrained by the following special rules:

- A jog’s total movement will never be more than its length.
- If a jog’s $\text{EPE}_{\text{max}} - \text{EPE}_{\text{min}}$ is $> \text{Nyquist}/2$, it will not move.
- However, if a very short jog with only a single EPE measurement point, it will not be moved at all. Any edge with a length $< 2 * \text{epc_spacing}$ will have only a single point and will not move. Since *epc_spacing* defaults to 0.010, any edge shorter than 0.02 will not move.

Setup file controls let you constrain edge movements using mask rule checks or other cost factors.

Customized Correction Algorithms in Calibre nmOPC

Calibre nmOPC supports a Tcl-based scripting language that provides a highly customizable environment for Calibre nmOPC with a Tcl interface.

“[Calibre nmOPC Custom Tcl Scripting Reference](#)” on page 367 provides complete descriptions of the commands available to you.

Variables, Keywords, and Commands Supported in the Setup File

The setup file for Calibre nmOPC can contain any of the variables, keywords, or commands described in this document.

The following table shows the commands that *are required* to be in the setup file.

Table 2-2. Required Calibre nmOPC Setup File Commands

Calibre nmOPC Commands Used	Task
<code>modelpath</code> <code>optical_model_load</code> <code>resist_model_load</code>	Import model information.
<code>layer</code> (input and opc)	Define input layers to the setup file and pass all necessary layer data to the tool. The definitions map input layers from SVRF to simulation names. This mapping is strictly by order (this is a different layer than declared using the SVRF LAYER command).
<code>background</code> (for <code>denseopc_options</code> block)	Define an imaging background.
<code>denseopc_options</code>	Define the OPC algorithm in the <code>denseopc_options</code> sub-command block. These are used to create the Calibre nmOPC algorithm and will be processed through a setlayer command. Further expansion on what operations are available are documented in the chapter “ Calibre nmOPC Reference Syntax ” on page 91.
<code>version</code>	Specify the Calibre nmOPC version.
<code>image</code>	Create a nominal image to generate the initial contour.
<code>setlayer denseopc</code>	Define output layers.

The complete set of LITHO DENSEOPC setup file commands is documented in “[LITHO DENSEOPC Setup File Commands](#)” on page 115 of this manual. The setup file can also contain the following:

- Optional Tcl-based nmOPC custom scripting commands to create alternative OPC algorithms. See “[Calibre nmOPC Tcl Scripting Commands](#)” on page 382.
- Optional Calibre OPCverify commands used to generate images based on layers created using the setlayer command. You can combine Calibre nmOPC and Calibre OPCverify commands in a single setup file. See “[Calibre OPCverify Setup File Commands](#)” on page 117.

Note

 Note that many keywords, environment variables, and commands have the same names as parallel controls for Calibre OPCpro. However, the syntax for the Calibre nmOPC versions is slightly different, reflecting the different simulation processes.

Calibre nmOPC Modes of Operation

Calibre nmOPC is primarily run in batch mode, invoked from a shell tool command line. Calibre nmOPC can also be run on small sections of layout from within Calibre WORKbench.

From Within Calibre WORKbench

Most Calibre nmOPC operations can be performed through the RET Flow Tool in Calibre WORKbench. This tool is accessed either by clicking the **Litho** button in the Calibre WORKbench button bar, or selecting **Litho > RET Flow Tool**. For complete information on Calibre nmOPC operations in the RET Flow Tool, refer to the *Calibre WORKbench: RET Flow Tool User's Manual*.

From the Command Line

Calibre nmOPC can be run with either the Calibre nmDRC-H hierarchical engine or with the Calibre FullScale platform. Both require an SVRF file to be processed.

The complete Calibre nmOPC invocation syntax is as follows:

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

where:

- *calibre_path*
Specifies the path to your Calibre executables.
- *-drc -hier*
Selects hierarchical Calibre nmDRC checking. Cannot be specified with *-pto*.
- *-pto*
Specifies Calibre® FullScale™. Cannot be specified with *-drc -hier*. See “*calibre -pto*” in the *Calibre Post-Tapeout Flow User's Manual*.
- *svrf_filename*
Specifies the SVRF filename.
- *-turbo [number_of_processors]*
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_litho [number_of_processors]**

Specifies the number of processors to use for RET and MDP applications. Only 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.

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

- **-remotefile filename**

This option is part of the Calibre 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 or -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 primary 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](#).

Several example invocations are as follows:

- A basic invocation in batch mode that outputs a summary to a log file, *drc.log*:

```
calibre_path/calibre -drc -hier litho.svrf > drc.log
```

You can output to a terminal using the tail -f *drc.log* command in a separate window.

Alternatively, the following command also outputs to a log file (though it may slow performance):

```
calibre_path/calibre -drc -hier litho.svrf | tee.log
```

- A more typical invocation, making use of multithreaded parallel processing:

```
calibre_path/calibre -drc -hier litho.svrf -turbo -turbo_litho  
-remotefile remote > drc.log
```

Considerations for Calibre FullScale Runs

Calibre nmOPC 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 designs. 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 operations². 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. It also cannot be used 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 the [Calibre Administrator’s Guide](#) for more information on RDS.

RDS benefits from high bandwidth. To avoid saturating the network, the bandwidth should be greater than 10 Gbit/sec.

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

Calibre nmOPC in Calibre FullScale

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

- MDP EMBED is not supported.
- LITHO OPCVERIFY has limited support for properties.

Acronyms and Terminology

This manual uses several acronyms and terms to refer to product features or technical processes.

Acronym/Term	Expansion/Description
CD	Critical Dimension
DRC	Design Rule Checking
DUV	Deep Ultraviolet
EPE	Edge Placement Error
EUV	Extreme Ultraviolet
MRC	Mask Rule Constraint
OPC	Optical and Process Correction
RVE	Results Viewing Environment
SRAF	Sub-Resolution Assist Feature
SVRF	Standard Verification Rule Format

Chapter 3

Configuring Calibre nmOPC

Calibre nmOPC is configured primarily through the setup file. The setup file contains all the key operations to correct lithographic errors in your layout file.

A setup file can be defined using one of the following methods:

- Using a text editor to create a separate setup file and then referencing it in an SVRF file.
- Placing a command block inside of an SVRF rule file. In the SVRF file, create a uniquely-named LITHO FILE block.

Figure 3-1 shows an example minimum Calibre nmOPC setup file, annotated to indicate which operations are required. It uses the older, non-litho model style.

Figure 3-1. Annotated Breakdown of a Minimum Setup File

```
/* Derive SVRF Layers using LITHO DENSEOPC */
m1_out = LITHO DENSEOPC metall sraf FILE m1_opc MAP m1_opc

/* Begin the setup file with an SVRF LITHO FILE statement */
LITHO FILE m1_opc /*

# Import model information (see "Import Model Information" on page 47)
modelpath ./models
optical_model_load opt m1.opt
resist_model_load res m1.cm1

# Define layers to pass to the tool (see "Define Layers" on page 52)
layer m1 hidden clear
layer sraf hidden clear

# Define the OPC algorithm in the denseopc_options block.
denseopc_options m1_opt {
    # Specify the Calibre version
    version 1

    # Define an imaging background in the denseopc_options block
    # (see "Define an Imaging Background" on page 58)
    background poly opc atten 0.06

    # Define at least one layer (see "Define Layers" on page 52)
    layer m1    opc    clear
    layer sraf  visible clear
```

```
# Define an image (see "Define an Image" on page 59)
image optical opt resist res

# Add other controls such as fragmentation, process windows,
# mask rule constraints, and so on.
# (See "Optional Setup File Operations" on page 60.)

}

# Pass the denseopc_options block to a setlayer denseopc command
# to create an output layer.
setlayer m1_opc = denseopc m1 sraf MAP m1 OPTIONS m1_opt

# End the LITHO FILE statement
*/]
```

There are specific groups of commands that are utilized for the setup file. [Figure 3-2](#) illustrates a map of an example setup file with the corresponding command locations.

Figure 3-2. Setup File Command Locations

```

m1_out = LITHO DENSEOPC met1 sraf
FILE m1_opc MAP m1_opc
LITHO FILE m1_opc /*

modelpath ./models
optical_model_load opt m1.opt
resist_model_load res m1.cm1

layer m1 hidden clear
layer sraf hidden clear

denseopc_options m1_opt {

    version 1
    background poly opc atten 0.06
    layer m1 opc clear
    layer sraf visible clear
    image optical opt resist res

    NEWTAG all POLY -out A
    NEWTAG edge A len < 0.3 corner1 convex corner2 convex
        -out line_end
    NEWTAG neighbor both line_end len < 0.4 corner convex \
        -out line_end_adj
    NEWTAG neighbor both line_end_adj len < 0.4 corner no_corner \
        -out secondfrag
    NEWTAG neighbor prev line_end len < .4 corner convex \
        -out line_end_adj0
    NEWTAG neighbor next line_end len < .4 corner convex \
        -out line_end_adj1
    FRAGMENT modify pt1 line_end_adj0  0.05 0.1 0.1
    FRAGMENT delete pt1 line_end_adj0
    NEWTAG all POLY -out A
    NEWTAG edge A len < .3 corner1 convex corner2 convex -out line_end
    NEWTAG neighbor prev line_end len < 0.4 corner convex -out \
        line_end_adj0
    FRAGMENT split line_end_adj0  0.4 0.1
    OPC_ITERATION 8
    NEWTAG all POLY -out A

}

setlayer m1_opc = denseopc m1 sraf MAP m1 OPTIONS m1_opt

```

Black: LITHO DENSEOPC Setup File Commands (layers and models)

Red: denseopc_options Keywords (nmOPC correction algorithm)

Blue: Calibre nmOPC Tcl Scripting Commands (tag scripting)

Import Model Information	47
Load Models Using Commands	47
Load Model Information With a Litho Model	48

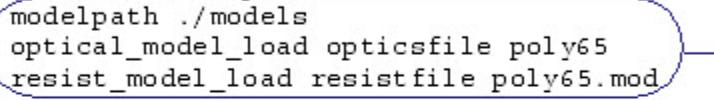
Models in EUV Lithography	50
Topography Modeling	50
Define Layers	52
Input Layers	53
Layers Passed to OPC	54
Output Layers	55
Layer Types	55
Define an Imaging Background	58
Define an Image	59
Optional Setup File Operations	60
Select Correction Method	61
Define Mask Rule Constraints	65
Define Process Windows	67
Check Enclosure With Image Cost Functions	77
Define No OPC Regions	79
Smooth Biased Target Layers	80
Adjust for Mask Corner Rounding	83
Simulate Layer-Based Images Using Calibre OPCverify	83
Create Custom Scripts for Calibre nmOPC	84
SRAF Operations in Calibre nmOPC	85
Understanding the Transcript	88

Import Model Information

One of the main steps in using Calibre nmOPC is to import model information into your setup file.

Figure 3-3. Import Model Information

```
LAYER POLY 5
POLY_OPc = LITHO DENSEOPC FILE NMOPC POLY MAP poly_opc
POLY_OPc {COPY POLY_OPc} DRC CHECK MAP POLY_OPc 105
LITHO FILE nmopc [
    modelpath ./models
    optical_model_load opticsfile poly65
    resist_model_load resistfile poly65.mod
]
layer poly hidden atten 0.06
denseopc_options dopc {
    version 1
    background clear
    layer poly opc atten 0.06
    image optical opticsfile resist resistfile
}
setlayer poly_opc = denseopc poly MAP poly OPTIONS dopc
]
```



This segment points Calibre nmOPC to the model information and maps the files to unique names.

You should have your models prepared prior to creating the setup file. 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
```

Certain advanced OPC features such as machine learning and CellArray OPC require litho models and will exit with an error if the setup file loads individual optical and resist models.

Load Models Using Commands	47
Load Model Information With a Litho Model	48
Models in EUV Lithography.....	50
Topography Modeling.....	50

Load Models Using Commands

One of the methods to load models is through a set of Calibre nmOPC setup file commands.

Use the [optical_model_load](#) and [resist_model_load](#) commands to import the optical model file and resist model file, respectively. The model data should have been previously calibrated to the process used for simulation, as described in the [Calibre WORKbench User's and Reference Manual](#). Assign a unique name to each model (one or more of each type can be defined). This allows the models to be referenced later by the image and [pw_condition](#) commands to create a nominal and variant process window conditions (described in “[Define Process Windows](#)” on page 67).

For example:

```
optical_model_load opticsfile poly65
resist_model_load resistfile poly65.mod
```

The optical model is imported from poly65 and assigned to the name opticsfile. The resist model poly65.mod is assigned to resistfile.

There are additional models available that provide different extensions for specific conditions:

- **Domain Decomposition Method (DDM)** — The DDM model describes the light scattering for three-dimensional mask topography. Each DDM model is associated with one or more layers in the setup file. You can load the DDM model using the [ddm_model_load](#) command.
- **EUV** — The flare and shadow bias models are specific to EUV lithography. Flare models capture long-range effects caused by mirror scattering. Shadow bias models are dependent on the position of the EUV slit and illumination axis. Refer to “[Models in EUV Lithography](#)” on page 50 for more information.
- **Topography** — The topography model allows OPC to include the effects of underlying layers where the resist stack is not uniform, such as above the periphery of implant. Refer to “[Topography Modeling](#)” on page 50 for more information.

Load Model Information With a Litho Model

An alternative method to load model information (as well as mask information) is using a litho model.

A litho model is a directory whose name must be found in the model path. (The directory also contains a file called *Lithomodel*.) There is no command required to explicitly load it or its sub-models. It is automatically loaded when the litho model name is detected in the

`denseopc_options` block, specified with the [image](#) command. In the following example, the litho model name is “32nm-POLY”:

```

denseopc_options opts {
    layer poly.main    opc      mask_layer 0
    layer poly.sraf   visible  mask_layer 1
    layer layer.area  hidden
    ...
    image 32nm-POLY  # all images are generated with this litho model
    pw_condition focus 20nm dose 1.1 inside layer.area noopc
    pw_condition focus -20nm dose 0.9 inside layer.area noopc
}

```

If you use a litho model:

- You will not need to use the [background](#) (for `denseopc_options` block), [resist_model_load](#), and [optical_model_load](#) commands in the setup file. Their information is taken instead from the litho model. If specified, they will be ignored.
- The [layer](#) (for `denseopc_options` Block) command uses an alternative syntax as it no longer needs to specify options such as transmission, instead it maps the layer to a component of the litho model.
- The [image](#) and [pw_condition](#) options also use alternative syntaxes. The [image](#) command specifies the name of the litho model to be used.
- For Calibre nmOPC, the `denseopc_options` block must either use the litho model entirely, or not at all. You cannot use both model methods at the same time. However, you can specify several litho models in a single setup file.

The following is a basic example of the *Lithomodel* contents and directory structure:

```

% cat duv10/Lithomodel
version 1
resist mymodel.mod
mask 0 {
    background clear
    mask_layer 0 TRANS dark CATEGORY main
    optical duv193_nom
}
% ls duv10
Lithomodel
duv193_nom
mymodel.mod

```

For further information on the litho model format, refer to “[Lithomodel \(Litho Model Format\)](#)” on page 359.

Models in EUV Lithography

Extreme ultraviolet (EUV) lithography models are an alternative to deep ultraviolet (DUV) models more commonly used for optical models. An EUV system uses a much lower wavelength (in the 13.5 nm range, which is about a 10x reduction in wavelength) compared to a standard DUV system. At this wavelength, optics must be all-reflective, and therefore two new effect characteristics must be accounted for:

- Flare effects, which are caused by scattering from the surface roughness of the mirrors.
- Shadowing, which is caused by off-axis illumination and makes some features print differently, depending on the orientation and position of the slit.

The basic flow to model shadow and flare effects in EUV can be summarized as follows:

1. Create a flare model file specification. This is a text file that describes the flare effects.
2. Generate the calculated flare map data using either [RET FLARE CONVOLVE](#).
3. Create a nominal EUV optical model.
4. Create a DDM model to accompany the optical model.
5. Create a starting CM1 resist model.
6. Either run Calibre nmModelflow or configure Calibre WORKbench CM1 Center to calibrate the optical and resist models, using the flare model file as an additional model input, then either calibrate or export the run script.
7. Create a shadow bias model file based on the test pattern values for later use (optional).
8. Apply a mask bias to the design file using CM1 Center (optional).
9. Use the calibrated model as input to Calibre nmOPC and Calibre OPCverify as described in “[Calibre OPC EUV Flows](#)” on page 681.

Flare model information is loaded into Calibre nmOPC using the [flare_model_load](#), [flare_model](#), and [flare_longrange](#) commands, or by loading a litho model that contains the flare model.

The full process is documented in “[EUV Optical Model Creation](#)” in the *Calibre WORKbench User’s and Reference Manual*.

Topography Modeling

Topography modeling methodology, for specific processes, is aware that the composition of underlying layers can significantly influence OPC results. In particular, the materials stacked on top of the silicon wafer up to the resist-air interface (also known as a “film stack”) are dependent on X and Y coordinates laterally across the wafer, where the resist stack is non-uniform across the wafer.

Topography models (also known as topo models) attempt to simulate the effects of non-uniform resist stacks for implant models. The implant layer is defined in an optical mask and is usually a dark layer. The result of this process is a set of modified optical models that have been optimized to take into account the varying density caused by the transition between regions, as well as some compensation for reflections from optical waves on the material.

Calibre nmOPC supports the topography modeling process, enabling you to create a topography model within a litho model file.

Table 3-1. Topography Modeling Development Flow

Step	Task
1	<ul style="list-style-type: none">• Create the oxide optical model.• Create the silicon optical model. FINFET users will need to specify a two-film optical model containing an oxide and silicon stack.
2	<ul style="list-style-type: none">• Create a model-layer file containing a topo model definition for use with modelflow_v2. The model-layer file must also have underlying layer keywords added to it.
3	<ul style="list-style-type: none">• Calibrate the optical models.• Calibrate the topo model.• Calibrate the resist model.
4	<ul style="list-style-type: none">• Use the model set in Calibre OPCverify or Calibre nmOPC.

Implementing topo model support in Calibre nmOPC only requires minor additions to existing Calibre nmOPC code. The details described in the *Calibre WORKbench Topography Modeling User's and Reference Manual*.

Related Topics

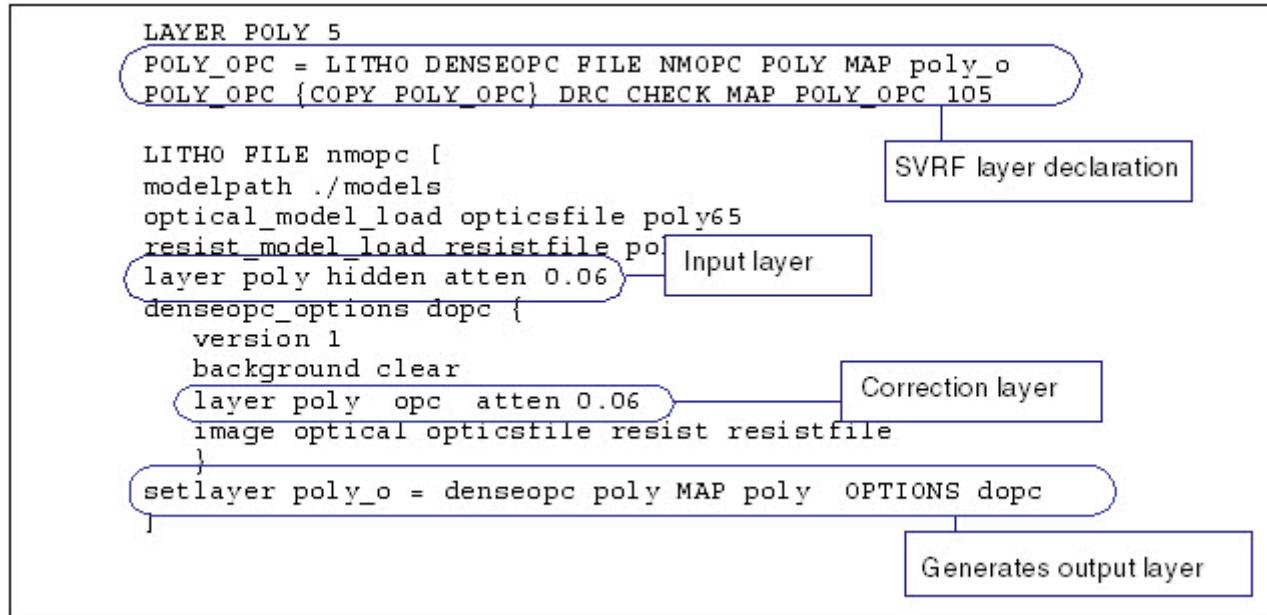
[image](#)

[topo_model](#)

Define Layers

You must define layers in the Calibre nmOPC setup file.

Figure 3-4. 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 actually defined in two places (with two different commands):

- Input layers passed from LITHO DENSEOPC SVRF command into the setup file (see “[Input Layers](#)” on page 53)
- Layers passed to OPC by the denseopc setlayer command (see “[Layers Passed to OPC](#)” on page 54)

Two layer declarations are required due to the fact that layers used in OPC may not be the same as the input layers defined in SVRF. Any layer derived inside setup file with a setlayer command can be passed to OPC.

Input Layers	53
Layers Passed to OPC	54
Output Layers	55
Layer Types	55

Input Layers

Each layer passed from LITHO DENSEOPC 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 type transmission [mask_num dose]*

- ***name***: A user-specified alphanumeric string that names the layer.
- ***type***: A parameter that defines the layer type (see “[Layer Types](#)” on page 55).
- ***transmission***: An argument that defines the mask layer’s optical transmission type. Values accepted are dark, clear, attenuated x, phase60, phase90, phase120, phase180, and phase270.
- ***mask_num dose***: An optional parameter pair used with phase shifting masks to allow the simulators to support multiple masks. The *mask_num* option is the number of the mask to which the layer belongs, and the *dose* option is the layer’s light energy dose, expressed as a percentage of the full energy dose for the mask.

The input layers are matched to SVRF LAYER statements strictly by order, which means the layer names do not necessarily need to match (see [Figure 3-5](#) for an example). If you are simply passing layers from SVRF to the DENSEOPC options block (you only need to pass the name, not transmission, type, or dose), you can set all layers can be set as type hidden (layers are referenced by name in the setup file).

Figure 3-5. Layer Mapping Example

```
### SVRF LAYER DECLARATION

ml_out = LITHO DENSEOPC met1 met1sraf
FILE ml_opc MAP ml_opc
LITHO FILE ml_opc [
    ### SIMULATION MODELS
    modelpath ./models
    optical_model_load opt ml.opt
    resist_model_load res ml.cml
]

### INPUT LAYERS */
layer m1 hidden clear
layer sraf hidden clear

denseopc_options ml_opt {
    ### VERSION */
    version 1
    background atten 0.06

    ### OPC INPUT LAYERS */
    layer ml_opc clear
    layer sraf visible clear

    ### NOMINAL IMAGE */
    image optical opt resist res
}

### OPC OUTPUT */
setlayer ml_opc = denseopc ml sraf MAP ml OPTIONS ml_opt
]
```

As the layers are mapped by declaration order, the layers names can be different. In this example, **met1** maps to **m1** and **met1sraf** is mapped to **sraf**.

Layers Passed to OPC

Layers passed to OPC are defined within a `denseopc_options` block. All layers passed on the `denseopc setlayer` command line must be defined. The number of layers passed must equal the number defined.

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

layer *name* *type* *transmission* [mask *n*]...

- ***name***: A user-specified alphanumeric string that names the layer.
- ***type***: A parameter that defines the layer type (see “[Layer Types](#)” on page 55).
- ***transmission***: An argument that defines the mask layer’s optical transmission type. Values accepted are dark, clear, attenuated x, phase60, phase90, phase120, phase180, and phase270.
- ***mask n***: The mask number is optional for a single exposure.

Layer definitions must be first in the [denseopc_options](#) block. Any layer name used as a keyword argument must be already defined.

Output Layers

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

setlayer *name* = *operation inputs*

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

```
setlayer poly_opc = denseopc poly sraf MAP poly OPTIONS dopc
```

In this example, “poly_opc” is the name of the output target layer. Two input layers are defined, “poly” and “sraf”. 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](#) sub-command block of LITHO DENSEOPC setup file commands. Output layers using the setlayer commands are passed out by layer name, not by order.

Layer Types

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

There are several different layer types supported by Calibre nmOPC. These layer types have meaning only with respect to the RET toolset.

Table 3-2. Layer Descriptions

Layer Type	Function
opc	<p>The opc layer serves as the target layer (the layer that will have OPC performed on it) for LITHO operations, except when retargeting is used (see “Smooth Biased Target Layers” on page 80). 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 OPC, unless correction layers are present. • The output (the OPC 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 input to the simulator and to determine which edges are movable.</p>
correction	<p>Correction layers are special layers whose edges are moved during OPC. Typically, you use correction layers when you need multiple masks to create the desired image, as with phase shifting masks. They are used only with the Calibre nmOPC batch tool or Calibre WORKbench application.</p> <p>During processing, a correction layer has:</p> <ul style="list-style-type: none"> • Its edges are fragmented. • Its fragments are mapped to fragments on an opc layer. • Its edge fragments are 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. Correction layers can also be used with retarget layers.</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 nmOPC, including serving as retargeting layers and marker layers, or for “re-opc” and “no opc” regions.</p>

Table 3-2. Layer Descriptions (cont.)

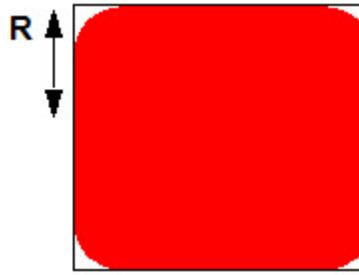
Layer Type	Function
contact	<p>The contact layer is a specific layer for only contacts. This layer is exactly like a hidden layer, with the additional property that if it is used with a wafer_enclose or wafer_enclosedby command (part of the denseopc_options block), its corners will be rounded.</p> <p>Wafer enclosure on contacts are never square on a real chip. It would be dangerous to try to achieve coverage on the drawn contact, because that would overcorrect. Instead, consider a corner-rounded contact for coverage (shown in Figure 3-6).</p> <p style="text-align: center;">Figure 3-6. Wafer Enclosure on a Contact</p> 
target (optional)	<p>Target layers are for retargeting. Use them when the opc layer and desired target are too far apart.</p> <p>If a target layer is defined, the sites are shifted to it. This layer type supports fragmentation.</p>
visible	<p>Visible layers contain geometries that will not be 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 does not need 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>
island	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.

Table 3-2. Layer Descriptions (cont.)

Layer Type	Function
sraf	This layer behaves like an opc layer with respect to simulation and fragmentation, but the layer will not be affected by regular opc iterations. This is intended to be used for SRAF optimization commands such as sraf_print_avoidance .
asraf (optional)	This layer represents holes on the mask and behaves like an opc layer with respect to simulation and fragmentation. However, it will not be affected by regular OPC iterations. This is intended to be used for SRAF optimization commands such as sraf_print_avoidance .
etch_underlying (VEB only) and underlying (topography models only)	<p>These are underlying layers that are used for VEB (etch_underlying layer) or topography models (underlying layers). These layers are like a hidden layer because they are not fragmented and have no optical properties.</p> <p>Etch_underlying layers are included in VEB simulations. Refer to “The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC” appendix for information on VEB.</p> <p>Underlying layers are used for performing OPC with a topography model. Refer to the Calibre WORKbench Topography Modeling User’s and Reference Manual for information on topography models.</p>

Define an Imaging Background

You can define an imaging background in the Calibre nmOPC setup file.

Figure 3-7. Define an Imaging Background

```

LAYER POLY 5
POLY_OPC = LITHO DENSEOPC FILE NMOPC POLY MAP poly_opc
POLY_OPC {COPY POLY_OPC} DRC CHECK MAP POLY_OPC 105
LITHO FILE nmopc [
modelpath ./models
optical_model_load opticsfile poly65
resist_model_load resistfile poly65.mod

layer poly hidden atten 0.06
denseopc_options dopc {
    version 1
    background clear
    layer poly opc atten 0.06
    image optical opticsfile resist resistfile
}
setlayer poly_opc = denseopc poly MAP poly OPTIONS dopc
]

```

A background is the part of the mask that is outside the drawn polygons. A Calibre nmOPC setup file requires an imaging background be defined, one background type per mask exposure. For Calibre nmOPC, a background statement is only required in the [denseopc_options](#) block.

background {dark | clear | {attenuated *factor*} | {real *imag*} }...

Backgrounds can be of type dark, clear, or attenuated *factor* (a phase shifting background). The *real imag* pair is an alternative way of specifying the layer transmission value and background value numerically. Refer to the background command description in the “[Calibre nmOPC Reference Syntax](#)” chapter for complete information.

Related Topics

[background \(for denseopc_options block\)](#)

Define an Image

You must define an image in the Calibre nmOPC setup file.

Figure 3-8. Define an Image

```
LAYER POLY 5
POLY_OPCT = LITHO DENSEOPC FILE NMOPC POLY MAP poly_opc
POLY_OPCT {COPY POLY_OPCT} DRC CHECK MAP POLY_OPCT 105

LITHO FILE nmopc [
    modelpath ./litho_models

    layer poly
    denseopc_options dopc {
        version 1
        layer poly opc mask_layer 0

        # The next line generates the initial contour based on ideal conditions
        image poly65.lm
    }

    setlayer poly_opc = denseopc poly MAP poly OPTIONS dopc
]
```

One [image](#) must be defined based on an ideal condition, including dose, mask size, resist, and aerial threshold. This is defined using the [image](#) command, which creates the initial contour that will be used for EPE calculations.

Optional Setup File Operations

There are a number of additional optional operations that can be performed from the Calibre nmOPC setup file.

Table 3-3 provides a list of the supported optional setup file operations available and where they are described in detail.

Table 3-3. Optional Calibre nmOPC Setup File Operations

Operation	Notes
“Select Correction Method” on page 61	If no method is specified, Calibre nmOPC defaults to automatic fragmentation.
“Define Mask Rule Constraints” on page 65	A mask rule constraint is a type of spacing-based limitation where the rules are enforced on the photomask.
“Define Process Windows” on page 67	Process windows are the tolerable change in dose, focus, and mask sizing until the design fails to yield on silicon.
“Check Enclosure With Image Cost Functions” on page 77	Check for contact or via enclosure on the wafer with images.
Set maximum OPC iterations and maximum edge movement distance per iteration. (See the <code>max_iterations</code> and <code>max_iter_movement</code> command descriptions in the “ Calibre nmOPC Reference Syntax ” chapter.)	Define how many iterations of OPC you want to perform and how far you want edges to be biased.
“Define No OPC Regions” on page 79	Establish regions where OPC will not be performed, assuming those areas are free of MRC violations.
“Smooth Biased Target Layers” on page 80	Correct small, irregular biasing on a target layer by smoothing.
“Adjust for Mask Corner Rounding” on page 83	Adjust mask polygons for corner rounding that occurs during lithographic exposure.
“Simulate Layer-Based Images Using Calibre OPCverify” on page 83	Use Calibre OPCverify commands to generate images based on layers created using setlayer.
“Create Custom Scripts for Calibre nmOPC” on page 84 and the “Calibre nmOPC Custom Tcl Scripting Reference” chapter.	Use the Calibre nmOPC Tcl-based custom script to augment the existing nmOPC algorithms.

Select Correction Method

There are different correction methods using Calibre nmOPC.

The different correction methods include:

- [Automatic Fragmentation](#)
 - Fully Automatic (default): In this method, fragmentation is based entirely on the Nyquist value. If you place no fragmentation commands in the setup file, it will default to automatic fragmentation.
 - Automatic with user settings: In this method, limited adjustments can be made to fragment lengths.
- [Custom Fragmentation](#)

You can create your own custom fragmentation using DENSEOPC setup file commands. Note that using any of the fragmentation commands will override the automatic settings.

You can also create fragmentation algorithms using a Tcl-based Calibre nmOPC customization script (see “[Create Custom Scripts for Calibre nmOPC](#)” on page 84 for details on the customization script).

Automatic Fragmentation	61
Custom Fragmentation	63

Automatic Fragmentation

There are two methods to set up automatic fragmentation, fully automatic fragmentation or through user specification.

Fully Automatic

By default, fragmentation is generated automatically in Calibre nmOPC. Fragment lengths are based on the Nyquist sampling distance (NSD) value (the basic fragment length is 2*fragmentation unit (frgu)).

Calibre nmOPC automatic fragmentation occurs using the settings listed in the following table.

Table 3-4. Calibre nmOPC Fragmentation Defaults

Calibre nmOPC Commands	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

Table 3-4. Calibre nmOPC Fragmentation Defaults (cont.)

Calibre nmOPC Commands	Defaults
fragment_corner concave	fragment_corner concave not_end_adjacent $(2+\text{corner_control})*\text{frgu}$ $2*\text{frgu}$ $2*\text{frgu}$ $2*\text{frgu}$ end_adjacent $2*\text{frgu}$ $2*\text{frgu}$ $2*\text{frgu}$ $2*\text{frgu}$
fragment_inter	-interdistance $1.5*(\lambda/\text{NA})$ -num $0.5*(\lambda/\text{NA})/\text{ripplenlen}$ -ripplelen $2*\text{frgu}$
fragment_max	$4*\text{frgu}$
fragment_min	$2*\text{frgu}$
NEWTAG line_end space_end	$4*\text{frgu}$

Automatic Fragmentation With User Settings

Although fragmentation is generated automatically, you can also refine the operations using the following nmOPC setup file commands:

- **scale** — Scales the length of all fragments (to determine the fragmentation unit).
Fragment lengths are set by the Nyquist value, which comes from the optical model. There is an optional LITHO DENSEOPC command **scale** that can be used to determine the final unit of fragmentation (the value is referred to as “frgu”). A scale value between 0.9 and 1.9 can be specified (the default is 1.0).

For example:

```
scale 1.11
```

This value is multiplied with the Nyquist value to set the basic fragmentation length unit. The value used is printed in the transcript:

```
----- DENSEOPC fragmentation base value: 0.031000
```

- **corner_control** — Extends fragments next to corners.
By default all corner fragments are $2*\text{frgu}$ long. The corner_control keyword can modify the length of the first fragment.

Figure 3-9. Automatic Fragmentation With User Settings Example

```
...
denseopc_options dopc {
    version 1
    max_iter_movement 0.005
    max_iterations 8
    background atten 0.06
    layer M1 opc clear

    step_size 0.00025
```

```
# The scale and corner_control commands change default settings
# for automatic fragmentation, scaling the fragment lengths and
# extending the fragments next to corners.
scale 1.2
corner_control 0.8
...
}
...
```

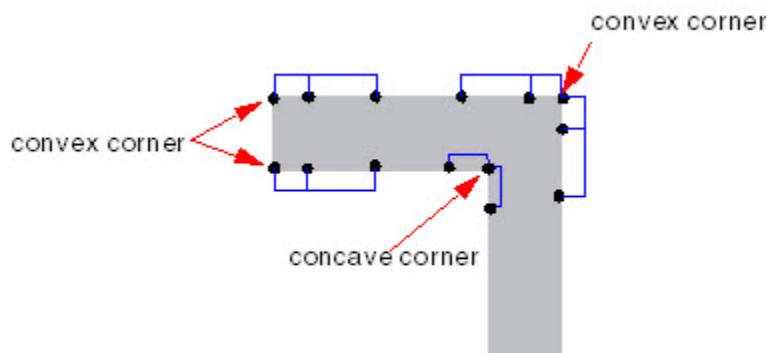
Custom Fragmentation

You can create custom fragmentation schemes using commands supported in the Calibre nmOPC setup file. These custom schemes will entirely replace the default automatic fragmentation behavior. Note that once you use any custom fragmentation scheme, automatic fragmentation no longer occurs.

There are several different types of fragmentation schemes available:

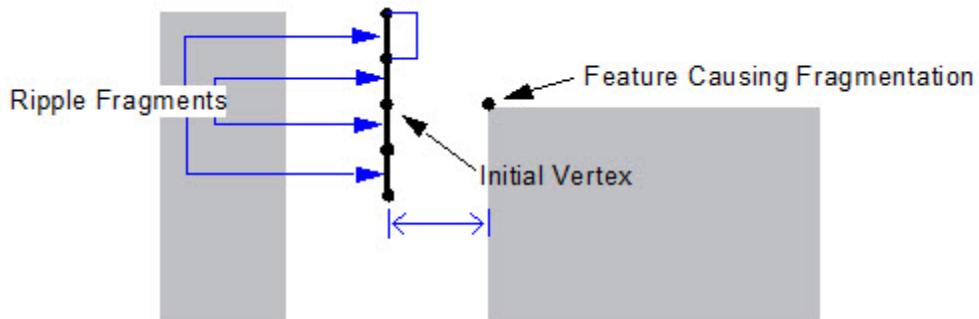
- **Global (All-Layer) Fragmentation:** The starting point for any fragmentation scheme, this strategy uses several different standardized methods to create fragments for polygon edges on your mask. The fragmentation is applied globally to all layers. These methods include:
 - Intrafeature fragmentation (using the [fragment_corner](#) command) evaluates each polygon on the layout, independent of other polygons, and adds vertices at specified distances from the figure's corners.

Figure 3-10. Intrafeature Fragmentation (Corner-Based)



- Interfeature fragmentation (using the [fragment_inter](#) command) evaluates each polygon in terms of proximity to nearby features and adds vertices to edges that are within a specified distance from these features.

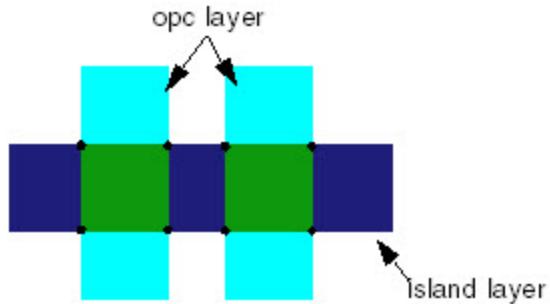
Figure 3-11. Interfeature Fragmentation (Ripple Fragments)



Interfeature fragmentation creates an initial vertex at the point on the edge that is closest to the corner. It then adds pairs of vertices on either side of the initial vertex to create symmetrical fragments on both sides of that initial vertex. The symmetrical fragments are called ripple fragments.

- Maxedgelength fragmentation breaks long fragments into shorter fragments. A long fragment is defined as greater than or equal to $(2 * \text{fragment_max}) + \text{fragment_min}$. This fragmentation breaks these longer edges into three parts: two parts equaling `fragment_max`, and one part greater than or equal to `fragment_min`. The shorter fragments into which it is broken are all greater than or equal to `fragment_min`.
- Island-layer fragmentation (using the `fragment_island` command) inserts vertices where the edges on an opc layer or correction layer intersect edges or vertices on an island layer. With the `fragment_layer` command block, you can configure island-layer fragmentation to also add ripple fragments to the side of the initial vertices.

Figure 3-12. Island-Layer Fragmentation



- **Specific-Layer Fragmentation:** Used for multilayer masks, the `fragment_layer` command block allows you to make custom settings for global fragmentation types on specified layers. Global fragmentation continues to be applied to all other layers.

These fragmentation schemes can also be augmented or even replaced by routines in the Calibre nmOPC customization script, as documented in the “[Calibre nmOPC Custom Tcl Scripting Reference](#)” chapter.

Define Mask Rule Constraints

You can define mask rule constraints in the Calibre nmOPC setup file.

Figure 3-13. Define Mask Rule Constraints

```
...
denseopc_options opt {
    version 1
    step_size 0.001
    image lithomodel
    layer poly opc

    # mrc_rule creates a mask rule to limit fragment movement during OPC
    mrc_rule external poly [
        use 0.1 euclidean
        length 0.02 use 0.04 euclidean
    ]
}

setlayer dense_opc = denseopc poly MAP poly OPTIONS opt
...
```

A constraint is a behavioral limitation observed by OPC. A mask rule constraint (MRC) is a type of spacing-based constraint where the rules are enforced on the photomask (as opposed to the wafer). These rules limit fragment movement during the OPC process. The `mrc_rule` syntax, defined in the `denseopc_options` block, allows you to create OPC constraint checks to perform a variety of internal and external check operations.

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
- Applied to only vertical or horizontal edges

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

The checks are different types of constraints that can be defined for the mask:

- **External** — This rule type checks the space outside the polygon. Space includes areas between polygons and fragments on the same polygon.
- **Internal** — This rule type checks inside polygon width.
- **Edge Topological** — This rule type restricts checks to specific areas marked by island layers. This includes areas such as inside, not inside, outside, and not outside the polygon.

Multiple mask constraints can be defined for difficult feature-to-feature combinations.

- Constraints can be specified for a single layer, or between two layers
- Constraints can be restricted to specific region using marker layers
- Constraint values can be varied for different edge lengths or directions
- Constraints can be varied for projecting and non-projecting fragments

Specific internal/external constraints on the mask can be defined based on the distance between fragments or the length of a fragment. For example:

```
mrc_rule external poly assist {  
    use 0.055  
    length 0.04 use 0.035  
}
```

In this example, an external constraint is applied to two layer values (poly and assist). These allow the specified MRC rule (the “use 0.055” constraint) to be applied to fragments between these two layers.

You can also define constraint checks for tagged fragments using the MRC_RULE or MRC nmOPC Tcl scripting commands. For example, the following code adjusts all line ends after moving fragments. (The relative priorities set the amount each edge will adjust to meet the constraint.)

```
OPC_ITERATION  
FRAGMENT_MOVE  
MRC external 0.07 line_end line_end 1 1
```

Related Topics

[mrc_rule](#)

[MRC](#)

[MRC_RULE \(Custom Scripting Command\)](#)

Define Process Windows

You must define process windows in the Calibre nmOPC setup file.

Figure 3-14. Define Process Windows

```
...  
denseopc_options options1 {  
    version 1  
    background clear  
    layer poly_main opc atten 0.06  
    layer poly_assist visible atten 0.06  
  
    image optical f0 resist cml  
    pw_condition fP1 outside active optical fP1 dose 0.97 resist cml  
  
    ...  
    wafer_enclose contact by 0.01  
}  
...
```

A callout box highlights the `image` and `pw_condition` commands with the following note:
Creates an alternative setting for dose and focus for optimization with the nominal image

A process window is defined as the tolerable change in dose, focus, and mask sizing until the design fails to yield on silicon. Process windows are used to define how much correction is required on the layout based on that range of conditions. This is useful to optimize dose and focus settings as well as CD control to yield better results from OPC. In general, process window OPC will degrade CD control at nominal dose and focus conditions, but improve CD control when measured throughout the process window.

With Calibre nmOPC, use the `pw_condition` command to create alternative process window conditions to compare with a nominal condition (created using the `image` command), varying settings for dose, focus, and mask sizing until the design fails to yield silicon. This allows you to measure EPE over a wider range of potential conditions and obtain a more comprehensive result. Process window conditions are used with image constraints to calculate the “effective EPE.”

Process window conditions can be defined using marker layers, including `inside`, `not_inside`, `outside`, and `not_outside` markers. The marker layers should be passed as type `hidden`.

Process window conditions also allow you to:

- define relative importance of each process window condition
- define enclosure and CD optimization criteria for each process condition
- correct catastrophic failures for all process conditions by default

Table 3-5 describes the variations that can be defined for alternative process windows.

Table 3-5. Process Window Conditions

Process Window Condition	Description
Dose Variations	Dose variations require less additional simulation time than focus or mask size simulations
Focus Variations	Each focus condition requires an optical model. Each focus condition requires an additional simulation
Mask size	Each mask size requires a simulation
Tolerance bands	A range of acceptable variances in dose and focus settings

The following rules apply to process window conditions:

- Currently, there is a limit of 16 process window conditions
- Focus variations require separate optical models (each optical model requires its own simulation)
- Exposure variations are modeled by scaling

You can use your defined process window conditions (with pw_condition) when creating layer constraints. For example:

```
pw_condition D1 image dose 1.05
wafer_enclose contact by -0.01 pw_condition D1
```

Process windows are applied to OPC in a process known as Process Window OPC (PWOPC). The goal of PWOPC is to try to improve the process window of the layout by adjusting the target CD within given tolerance.

PWOPC will:

- Assess the process window of every specified fragment and make adjustments to the target by shifting it outside of the polygon if pinching conditions are detected or shifting it inside the polygon if bridging conditions are detected.
- Shift the EPE distribution based on the previous assessments. It is expected for all the features corrected by PWOPC to have non-zero EPEs. This is not meant to justify the use of PWOPC on Poly gates, but it is strongly recommended for use on Poly connectors, metal interconnects and contact or via holes.

PWOPC will not:

- Enhance resolution by adjusting, adding or removing SRAFs (or use any other RET) around a target feature.

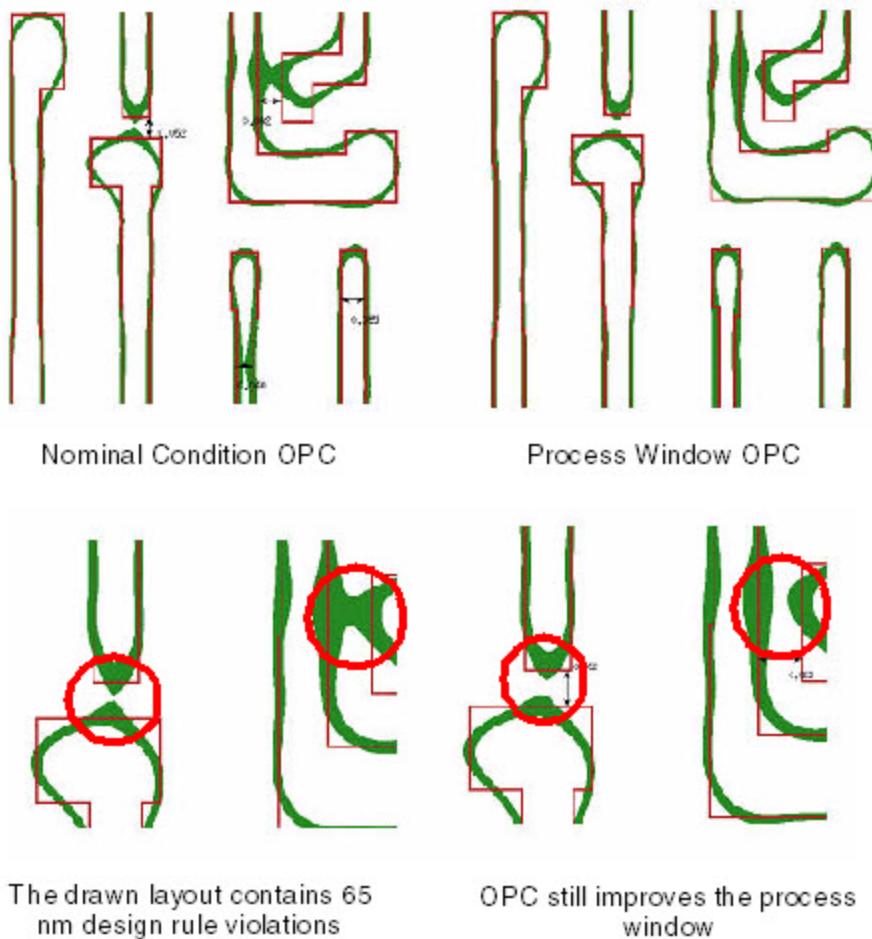
The most basic PWOPC implementation is as follows:

1. Define a process window (for example, plus or minus 1 nm mask, plus or minus 5% dose, nominal + defocus)
2. Use OPC to decrease sensitivity to: Dose, Focus, and Mask Sizing

The following example shows the effects of nominal condition OPC versus process window OPC on a 65 nm layout.

Figure 3-15. Nominal Versus Process Window Conditions

These examples show the prevention of bridging and pinching using process window OPC.



EPE is measured at process window conditions.

PV Band PWOPC	70
Direct Etch Effect Correction.....	71
Resist Top Loss Correction.....	75

PV Band PWOPC

When using the `pw_condition` command to implement process window-based OPC (or PWOPC), the recommended method of implementation for most cases is called PV Band PWOPC. This method uses physical verification bands (also known as PV bands) that are simulated using different process window conditions. This method has several key advantages.

- It is tag based, giving more flexible control.
- Minimum CD can be specified in addition to EPE-based tolerance control.
- Sites are automatically generated.

Simulating PV bands under different process window conditions are done inside sites. This means that sites are required in order to perform PV Band PWOPC. The site-based simulation makes the CD width/space cost function possible using this method.

Sites are automatically generated using PV Band PWOPC, and tag-based operations are supported for further control.

The basic process is as follows:

1. Define PWOPC (you must use the `pvband` option in the `pw_condition` command). For example:

```
pw_condition PW0    optical OPTIC    dose 1.00  pvband
```

2. Run PWOPC on a selected number of OPC iterations (using the `OPC_ITERATION` Tcl scripting command). For example:

```
OPC_ITERATION 5 -PW \
nominal_epe_limit all_frags -0.003 0.003 \
min_width all_frags 0.034 \
min_space all_frags 0.037 \
exclude tag
```

[Figure 3-16](#) summarizes several best practices for PV Band-based PWOPC.

Figure 3-16. Using PV Band PWOPC

```
...
pw_condition PW1    optical OPTICPW    dose 1.03 resist RESIST pvbnd
pw_condition PW2    optical OPTICPW    dose 0.97 resist RESIST pvbnd

OPC_ITERATION 5
OPC_ITERATION 5 -PW \
nominal_epe_limit all frags -0.003 0.003 \
min_width all frags 0.035 \
min_space all frags 0.035 \
exclude tag
...
```

Refer to “[Using PV Band PWOPC](#)” on page 621 for more information on setting up PV Band PWOPC.

Direct Etch Effect Correction

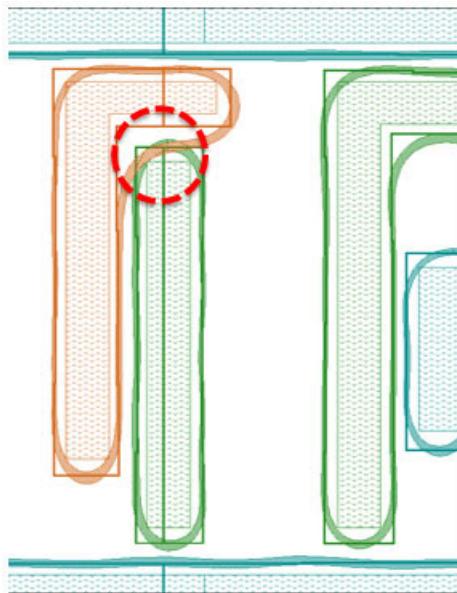
Direct etch effect correction is a method of using both a Variable Etch Bias (VEB) litho model along with Process Window OPC (PWOPC) to correct a target. More specifically, you can correct etch effects on a target directly using VEB-based process windows.

A typical VEB retargeting flow uses the following steps (refer to “[The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC](#)” on page 671 for complete details):

1. Create a calibrated optical, resist, and VEB model.
2. Using the VEB model, retarget the original design file to generate an etch-corrected design file (the litho target).
3. Using the etch-corrected design file, run Calibre nmOPC to generate an OPC-corrected file.
4. Inspect the OPC-corrected file using Calibre OPCverify.

A standard VEB retargeting flow for etch compensation can make it difficult to enforce post-etch process conditions using litho conditions alone, particularly in multi-patterning applications. Because the etch and litho targets are resolved separately, errors can occur. For example, the following figure shows a 10 nm M1 layer where the etch target (in orange) and the litho target (in green) overlap.

Figure 3-17. Etch and Litho Target on a 10 nm M1 Layer Error



Instead, use VEB-based process window conditions (the pw_condition command) to directly correct the final etch EPE. Through the etch-based process windows, you add constraints for bridging, pinching, overlays for multi-patterning, via enclosures. This method enforces the proper constraints during a Calibre nmOPC run.

To create VEB-based process windows, first create a litho model with the etch parameters. For example:

```
version 1
resist resist.mod
etch etch.mod
mask 0 {
    optical e1_litho_f+0.0000
    optical e1_litho_f+0.0410 focus 41nm
    optical e1_litho_f-0.0500 focus -50nm
    background 1 0
    mask_layer 0 TRANS -1.245 0 DDM m196d.ddm
}
```

You then create a block of pw_condition commands in the Calibre nmOPC setup file that specify the imaging constraints for your etch and litho targets. The nominal condition is not supported for a VEB model. Each image must also be explicitly marked as to whether they are a litho or etch-based condition by specifying the no_etch (for litho targets) or with_etch (for etch targets) options.

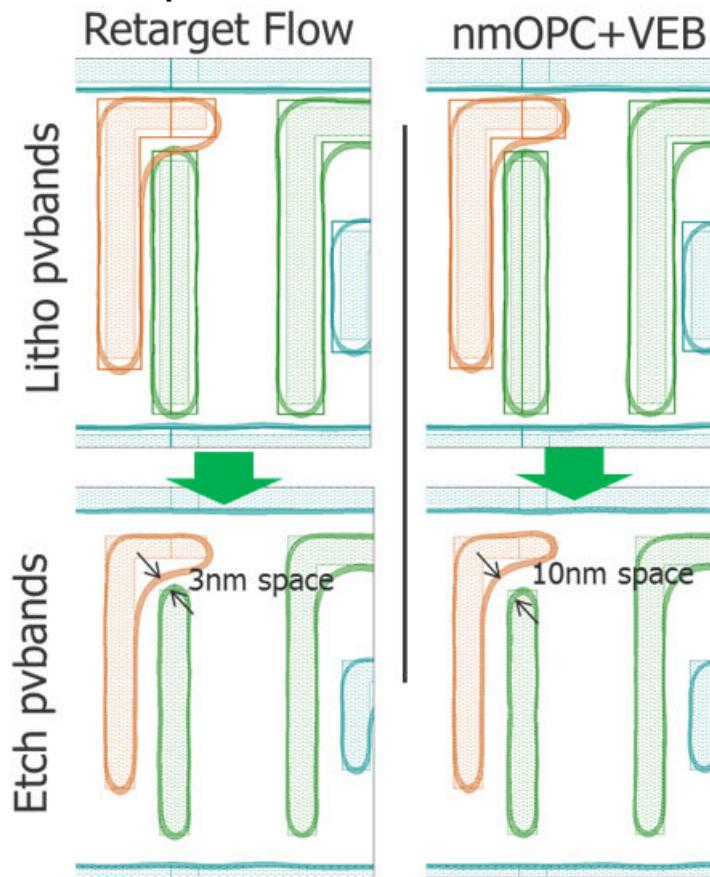
```

image m1_LM no_etch
pw_condition inner_litho m1_LM bias -0.000375 \
               focus -50nm dose 1.036 pvband no_etch
pw_condition outer_litho m1_LM bias 0.000375 dose 0.970 \
               pvband no_etch
pw_condition inner_etch m1_LM bias -0.000375 focus -50nm dose 1.036 \
               pvband with_etch
pw_condition outer_etch m1_LM bias 0.000375 dose 0.970 \
               pvband with_etch
pw_condition sraf_model m1_SPA bias 0.000250 dose 0.915 \
               pvband spa no_etch

```

These pw_condition commands generate PV bands for both the etch and litho targets, resulting in more accurate corrections. The following figure illustrates the differences between correction between the standard VEB flow (Retarget Flow) and the VEB-based PWOPC (nmOPC + VEB) flow.

Figure 3-18. Comparison of VEB and VEB + PWOPC Results



For the standard VEB retargeting flow, litho contours and even the litho target can unintentionally overlap, affecting the final output. Restrictions against these overlaps can only be properly enforced if etch contours are simulated. Without etch contour simulation, contours renderings may fail.

VEB-based process windows are supported for the following uses:

- CD-based conditions, enclosure, and overlay.
- Single and multiple patterning applications.
- Matrix OPC.
- Process window and matrix retargeting.

VEB-based process window conditions are not supported for any other type of condition (PV band optimizations and so on).

The VEB-based process window conditions optimally use the per-condition pw_condition syntax with process window or matrix retargeting. For example, the following code illustrates the creation of VEB-based process window along with the OPC_ITERATION command (with the -PW option enabled for PWOPC mode). This allows you to directly control the OPC iterations while performing etch-based corrections, enabling better run-time control.

```
image m1_LM no_etch
pw_condition inner_litho m1_LM bias -0.000375 \
    focus -50nm dose 1.036 pvbnd no_etch
pw_condition outer_litho m1_LM bias 0.000375 dose 0.970 \
    pvbnd no_etch
pw_condition inner_etch m1_LM bias -0.000375 focus -50nm dose 1.036 \
    pvbnd with_etch
pw_condition outer_etch m1_LM bias 0.000375 dose 0.970 \
    pvbnd with_etch
pw_condition sraf_model m1_SPA bias 0.000250 dose 0.915 \
    pvbnd spa no_etch
...
OPC_ITERATION 6 -PW min_width_pw inner_etch all_fragments 0.019 \
    min_space_pw outer_etch all_fragments 0.019 \
    min_width_pw inner all_fragments 0.033 \
    min_space_pw outer all_fragments 0.033 \
    min_space_interpattern all_fragments 0.005 \
    -schedule { retarget 2 3 pw 2 3}
```

Multiple Litho Model Support

There are occasions when out-of-plane effects such as SRAF print avoidance or top loss should be considered. In these cases, you must define multiple litho models.

Note that the following parameters in these litho models must be the same:

- DDM model
- CX/CC/BIAS
- Etch model

The following is an example litho model used strictly for nominal and PWOPC operations:

```
version 1
resist resist.mod
etch etch.mod
mask 0 {
    optical e1_litho_f+0.0000
    optical e1_litho_f+0.0410 focus 41nm
    optical e1_litho_f-0.0500 focus -50nm
    background 1 0
    mask_layer 0 TRANS -1.245 0 CX 0.002 CC 0.002 DDM m196d.ddm
}
```

The following is an example of a separate litho model used specifically for SRAF print avoidance:

```
version 1
resist resist.mod
etch etch.mod
mask 0 {
    optical e1_spa_f+0.0000
    background 1 0
    mask_layer 0 TRANS -1.245 0 CX 0.002 CC 0.002 DDM m196d.ddm
}
```

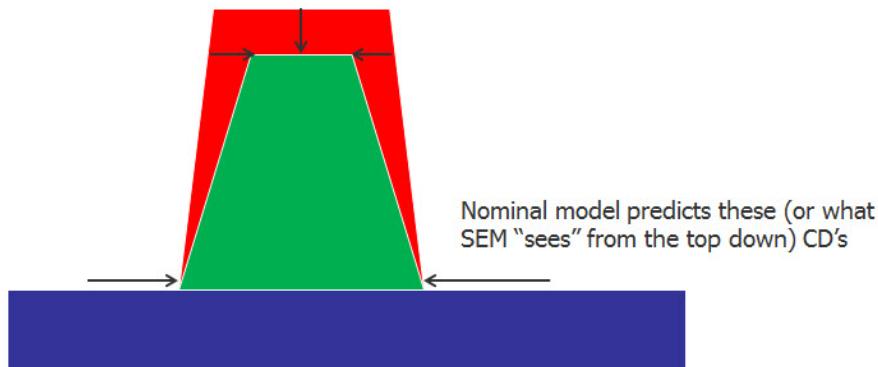
Resist Top Loss Correction

Resist top loss is a source of wafer failure in advanced technology nodes.

During the development process, some portion of the resist material is dissolved as illustrated in [Figure 3-19](#). Specific resist and development processes can cause further erosion.

Figure 3-19. Top Loss Effects

Red – healthy resist profile
Green – resist affected by the top loss phenomena



Typically, when top loss occurs, the CD at the lower resist layers is almost unchanged and the impact is localized close to the resist surface. You can correct for top loss effects by creating a

PV band process window condition (using the [pw_condition](#) command) specifically designated for top loss 3D correction. This creates a PV band with a different defocus start and beam focus that can predict what happens at the upper resist layers. This PV band can be used for both Calibre nmOPC and Calibre OPCverify.

To set up a top loss process window condition, use the 3DSlice keyword in the “[pw_condition](#) pvband” command. The 3DSlice keyword designates a process window condition as a resist top-loss specification that contributes to a PV band that OPC will improve. In particular, this process window condition will be checked only when corresponding 3D width and spacing constraints are specified during a PWOPC iteration.

For example:

```
pw_condition TL mask_size 0.0 optical OPTIC_TL dose 1.000 resist \
    RESIST_TL pvband 3DSlice
```

This generates a PV band that simulates top loss erosion based on the tolerance in terms of the resist height.

You can further adjust the PV band by experimenting with its width and height. Use the min_space_3D and min_width_3D keywords in the OPC_ITERATION command to adjust width and height for the top loss 3D effects.

The top loss corrections are active only for last few process window iterations for purposes of incorporating the top loss information into an overall CD-based correction.

The following example incorporates top loss corrections at the end:

```
pw_condition Pinch mask_size -0.00025 optical OPTIC_PW dose 0.959 pvband
pw_condition Bridge mask_size 0.00025 optical OPTIC dose 1.041 pvband
pw_condition TL mask_size 0.0 optical OPTIC_TL dose 1.000 resist \
    RESIST_TL pvband 3DSlice

...
for {set i 1} {$i <= 4} {incr i} {
    OPC_ITERATION 1
}

for { set i 5 } { $i <= 10 } { incr i } {
    OPC_ITERATION 1 -PW \
        min_space all_opc_frags 0.033 \
        min_width all_opc_frags 0.033
}

for { set i 11 } { $i <= 15 } { incr i } {
    OPC_ITERATION 1 -PW \
        min_space all_opc_frags 0.033 \
        min_width all_opc_frags 0.033 \
        min_space_3D all_opc_frags 0.010 \
        min_space_3D le_frags_80 0.030
}
```

In this example, top loss correction occurs during the last 5 OPC iterations. Conditions are separated from the bottom CD models.

In addition, you can use the min_space_3D and min_width_3D keywords in the “NEWTAG sites -wafer” scripting command to extract 3D-based measurements in when using the 3DSlice pw_condition setting.

Check Enclosure With Image Cost Functions

You can check enclosure with image cost functions in Calibre nmOPC.

Figure 3-20. Check Enclosure With Image Cost Functions

```
...
denseopc_options options1 {
    version 1
    layer CA contact clear
    background clear
    ...
    wafer_enclose CA by -0.005
    wafer_enclose CA by -0.005
}
```

Checks if contacts are enclosed by the contour

In certain cases, such in the case of contacts or vias, one polygon must be “enclosed” within another polygon or inside of a contour. Contacts use their own contact layer (which performs an automatic corner rounding of each contact). If the contour generated by Calibre nmOPC or a polygon on another layer does not completely enclose the contact or via, it will fail, as illustrated in [Figure 3-21](#). To ensure that the contact or via is completely enclosed, either within a polygon on another layer or enclosed within an image contour, two cost functions are available.

Figure 3-21. Wafer Enclosure Example

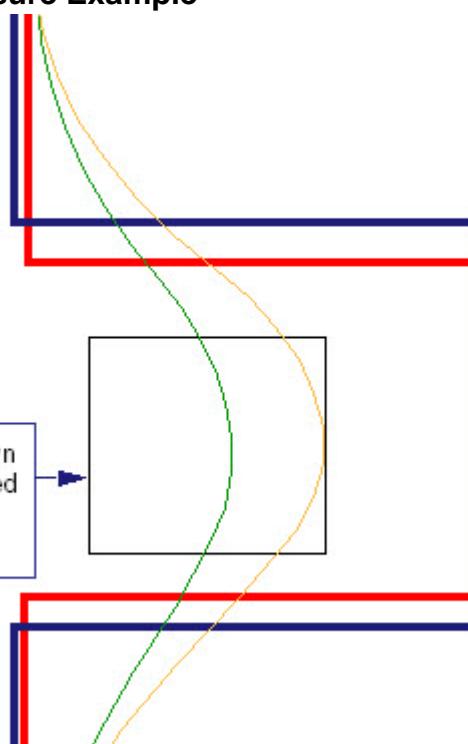
This is a printimage representing two different OPC runs

Red – No contact enclosure rule

Blue – Correction based on the following example:

```
pw_condition dose 1.1  
wafer_enclose layer1 by 0.01
```

This contact, on its own layer, must be enclosed by a polygon on a second layer.



A cost function, also known as an optimization, is a function that attempts to maximize or minimize certain effects by choosing the values of different variables (in addition to EPE) to garner the best possible results.

There are two image cost function keywords, [wafer_enclose](#) and [wafer_enclosedby](#). Each cost function can be associated with a process window condition.

- [wafer_enclose](#): This command allows you to create layer-based cost functions to check if your polygons on a specified layer are properly enclosed within the image contour. Use this command where enclosure of another layer (such as M1 enclosure of contacts and vias) is critical.
- [wafer_enclosedby](#): This command allows you to create layer-based cost functions to check if your image contour is properly enclosed within a polygon on a specified layer.

These keywords ensure that your printed image encloses or is enclosed by another layer. One of the primary advantages to using layer constraints is that contacts are given better visibility to OPC.

Refer to “[Enclosure of Contacts and Vias](#)” on page 627 for recommended practice information on contact and via enclosure.

Define No OPC Regions

You can define “No OPC” regions in Calibre nmOPC.

Figure 3-22. Define No OPC Regions

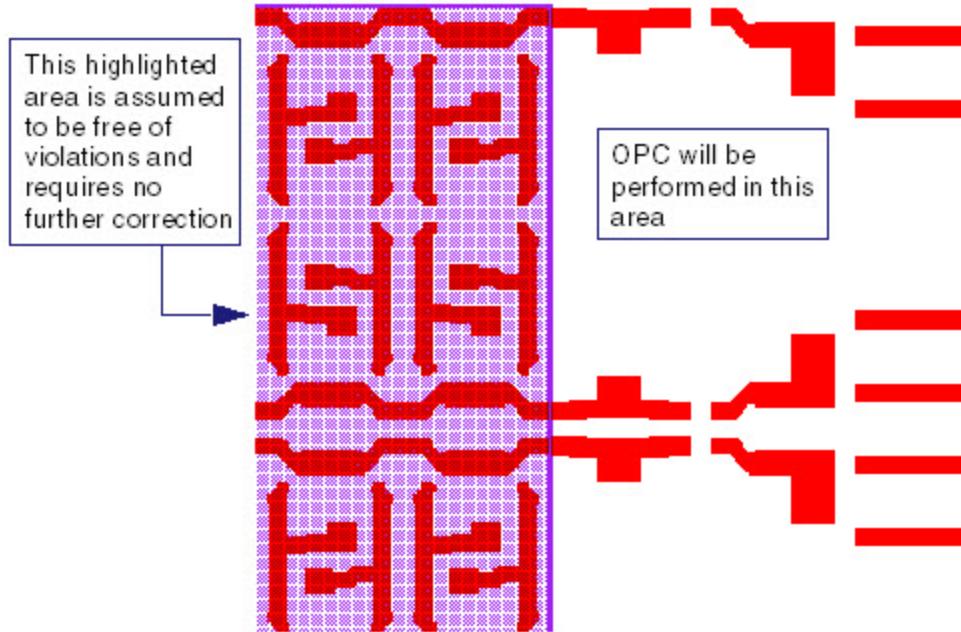
```
...
denseopc_options options1 {
    version 1
    background clear
    layer poly_main opc atten 0.06
    layer poly_assist visible atten 0.06
    ...
}

no_opc_region markerlayer
...
```

Specifies that target area identified by the marker layer not be modified by OPC

A “No OPC” region is an area designated on the layout where OPC 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).

Figure 3-23. No OPC Region



Using the `no_opc_region` command, you create a “do not OPC” region as a polygon layer that can be passed into nmOPC. Any fragments that touch this region will not be moved by OPC. Shapes inside the layer are also visible when fragments outside are being corrected.

A marker layer can be used to define areas that should not get OPC. For example:

```
no_opc_region marker_layer
```

Any fragment not outside of the marker layer will not be moved, which includes the fragment completely or partly inside the marker. Shapes inside the layer are visible optically, and marker layers must be type “hidden” in the layer statement.

Note

 Calibre nmOPC does not automatically resolve any mrc_rule conflicts that may arise when a “No OPC” region is defined.

Tip

 The marker layer is used to mark regions, and is not intended to be used to mark individual edges. To mark individual edges, use the Calibre nmOPC custom scripting capability to tag individual edges.

Smooth Biased Target Layers

You can smooth biasing on target layers in Calibre nmOPC in the setup file.

Figure 3-24. Smooth Biasing on the Target Layer

```
denseopc_options options {
    version 1
    algorithm 2

    background atten 0.06
    layer poly opc clear
    layer smoothed hidden clear

    max_iterations 12
    scale 1.2
    image optical f0 aerial 0.23

    # Copies the opc layer and creates a new target layer
    retarget_layer smoothed emulate
}

# The target layer is "smoothed" to remove small overcorrections
# in biasing.
setlayer smoothed = curve poly order 6 cpdist 0.036 \
    midpt 1 pts_per_cp 10 maxdist 0.06 scale1 2.0 scale2 0.5 \
    endpt_offset 0.01 midpt_offset 0

setlayer cpts = curve poly order 6 cpdist 0.036 \
    midpt 1 pts_per_cp 10 maxdist 0.06 scale1 2.0 scale2 0.5 \
    endpt_offset 0.01 midpt_offset 0 output_cpts 0.001
```

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 OPC.

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 because 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 the original layer (a pre-biased layer) before biasing it as an OPC layer, using the layer with jogs as a 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](#), [setlayer curve_target](#), or [target_curve](#) commands. The smoothed layers are passed to dense OPC as retarget layers.

The setlayer 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.

The setlayer curve_target command has simpler controls than setlayer curve, and can produce comparable results. The target_curve command is similar to setlayer curve_target, but allows tag-based overrides of global settings.

A general process for smoothing layers can be described as:

1. Get reasonably good OPC results without using smooth target.
2. Use post OPC simulation contour to estimate the amount of residual corner rounding.
3. Adjust curve parameters to match existing contour.
4. When done, compare OPC results with and without smoothed input.
 - a. Results should show improved OPC.
 - b. Some fragment length adjustment may help.

Figure 3-25. Retargeting the Input Layer

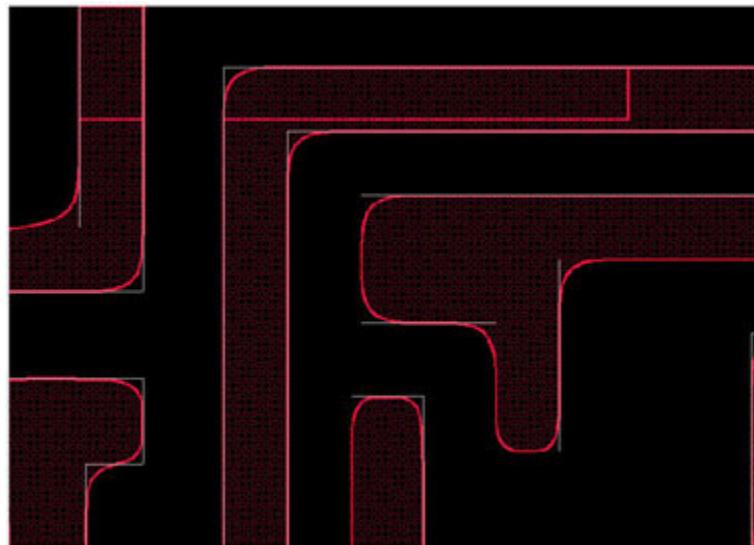


Figure 3-26. Smoothed Target Layer



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.
- Corner fragments are within $5 * \text{frgu}$ of a corner.
- If the recipe specifies a large value for scale and manual fragmentation, some secondary fragments might use max or min.

Adjust for Mask Corner Rounding

You can make adjustments for corner rounding during the OPC process in the setup file.

Figure 3-27. Adjust for Mask Corner Rounding

```
...
denseopc_options options3 {
    version 1
    background clear
    layer poly_main opc atten 0.06
    ...
    mask_chip_corner 0.01
}
...

```

“Chips” corners to account for rounding that occurs during lithographic exposure

Figure 3-28. Polygon With Corner Rounding



Corner rounding on photomasks can affect the image. The `mask_chip_corner` keyword can be used to approximate mask corner rounding. The corner rounding (also known as “chipping”) represents a 2D (X-Y plane) model of the effect of the mask writer on final mask corners on a real mask.

```
mask_chip_corner cx [cc]
```

If a corner value is supplied for `cx`, `mask_chip_corner` operates on both convex and concave corners. If `cc` is supplied, `cx` is for convex and `cc` for concave corners.

A triangular region with size defined by the inputs will be subtracted from convex corners. For concave corners, a triangular region will be added instead.

Simulate Layer-Based Images Using Calibre OPCverify

Calibre OPCverify has the ability to generate images based on layers created using the `setlayer` command. For Calibre nmOPC, this means that you can combine Calibre nmOPC correction and Calibre OPCverify verification in a single setup file.

Figure 3-29. Simulate Layer-Based Images Using Calibre OPCverify

```
modelpath ./models/  
optical_model_load f0 poly.opt  
resist_model_load cml poly.mod  
background clear  
layer POLY visible atten 0.06  
layer ASSIST visible atten 0.06  
  
denseopc_options opts {  
version 1  
background clear  
layer POLY opc atten 0.06  
layer ASSIST visible atten 0.06  
  
image optical f0 resist cml  
max_iterations 8  
}  
setlayer POLY_OPCT = denseopc POLY ASSIST MAP POLY OPTIONS opts  
  
# This is the post opc contour.  
  
setlayer postimage = image optical f0 background clear layer POLY_OPCT  
attenuation 0.06 layer ASSIST attenuation 0.06 resist_model cml  
  
# Any OPCVerify commands can now be run on the output image.  
  
...
```

Instructs Calibre OPCVerify to generate a layer-based simulation

Using the Calibre OPCVerify `background` and `layer` commands, you can create an image based on layers that are not listed in the layer table.

Using this output it is possible to run OPC, find errors, then pass the error markers into another opc run as a re-opc layer and perform repairs, all in a single setup file. However, this requires that post-OPC contours be generated and the simulation part of the setup file cannot have layer statements for layers not passed. Any Calibre OPCVerify check can be performed on the resulting contour.

Create Custom Scripts for Calibre nmOPC

You can augment or even replace Calibre nmOPC fragmentation schemes by creating a custom Tcl script. Calibre nmOPC supports a Tcl-based scripting language that provides a customizable environment for Calibre nmOPC with a Tcl interface.

Figure 3-30. Create Custom Tcl Scripts for Calibre nmOPC

```

background clear
layer poly hidden atten 0.06
denseopc_options dopc {
    version 1
    background clear
    layer poly opc atten 0.06
    image optical opticsfile resist resistfile
    ...
#begin custom tagging script
    NEWTAG all poly -out all
    NEWTAG line_end all -out line_end
    NEWTAG neighbor both line_end len >= [expr 4*$frgu] \
        corner convex -out lea
    ...
    OUTPUT_SHAPE fragment DEBUG line_end 0.05 0.004
#end custom tagging script

```

This custom script creates tag sets containing line ends

The Tcl-based commands enable operations such as:

- Tagging fragments and creating tag sets
- Fragmentation and fragment-based operations
- Measurement controls (EPE and CD)
- Dense simulation controls
- Target modification
- OPC controls
- Unit conversions
- Additional EPE sampling points

These commands are described in the “[Calibre nmOPC Custom Tcl Scripting Reference](#)” chapter.

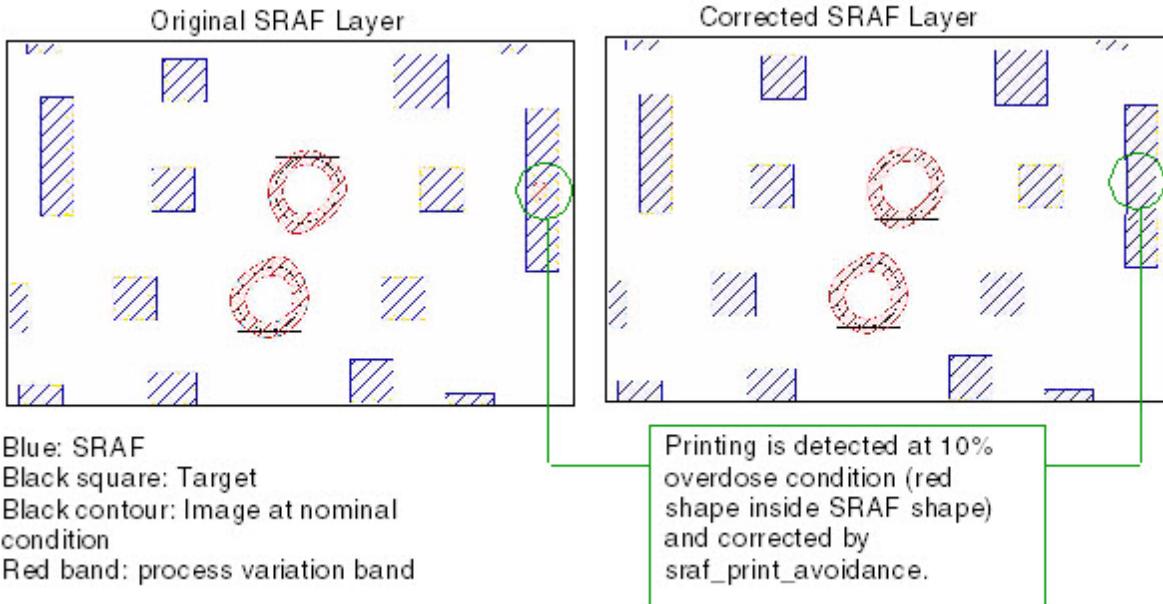
SRAF Operations in Calibre nmOPC

Sub-Resolution Assist Features (SRAFs) are shapes (such as scattering bars) introduced to the mask specifically to enhance the lithographic resolution of the layout on the wafer. As SRAFs do not represent actual design polygons, they are not meant to be printed.

The [sraf_print_avoidance](#) command is designed to detect and correct for any printing occurring around sub-resolution assist features (SRAFs). This command is executed iteratively using the [OPC_ITERATION](#) command. For each iteration, [sraf_print_avoidance](#) checks sites, compares to the specified threshold, and moves the SRAF fragments to prevent SRAF printing. This does not guarantee that incoming SRAFs that print will not print upon completion, but it will reduce

the probability of any printing upon completion and will not result in MRC violations.
Figure 3-31 illustrates a correction example.

Figure 3-31. SRAF Correction Example



The `sraf_print_avoidance` command is fragment-based and requires several key definitions in the setup file before it can be executed. These include:

- The SRAF layer must be defined as an SRAF type layer in the `denseopc_option` section. This type marks a layer as correctable but removes unnecessary EPE calculations.
- The SRAFs must have fragmentation defined using `fragment_layer` statements. If no fragmentation is defined, each SRAF edge is treated as a fragment.
- Evaluation sites must be placed on each fragment in order to define locations where SRAF intensity will be measured and compared against a user-defined threshold. This is done using the `sraf_sites_create` Tcl scripting command.

The following shows an example Calibre nmOPC setup file using `sraf_print_avoidance`.

Figure 3-32. sraf_print_avoidance Example

```

# ----- Denseopc options -----
background 0.0229 -0.0217
layer target opc      0.9771 0.0217
layer assist sraf     0.9771 0.0217
                                         Define layer "assist" as an
                                         sraf layer.

# ----- Image -----
image optical f0 dose 1.0 resist res
pw_condition nominal mask_size 0 optical opticsfile dose 1.000 pvband
pw_condition pw1 mask_size 0 optical opticsfile dose 1.000 pvband spa
...
                                         Use "pw_condition pvband spa" to define
                                         a specific process window at which
                                         SRAF printing occurs.

# ----- Assist Fragmentation -----
fragment_layer assist {
    fragment_min min_frag_val
    fragment_max max_frag_val
}
                                         Fragmentation for the assist layer is very
                                         primitive. The only rule to obey is not to
                                         create fragments smaller than MRC
                                         values while keeping fragment_max small
                                         enough for a thorough print check.

# ----- MRC Rules and Create Sites-----
...
mrc_rule area sraf_layer {
use area1
square use area2
}
...
                                         Use the mrc_rule area checks
                                         to define rules to make sure the
                                         polygons on the SRAF layer are
                                         small enough to avoid printing.

...
sraf_site_create
...
                                         Use the sraf_sites_create
                                         command to place sites on the
                                         SRAF layer polygons.

# ----- sraf_print_avoidance Function -----
...
#OPC iteration
OPC_ITERATION 5
...
For {set i 0} {$i < 5} {incr i} {
    OPC_ITERATION 1
    sraf_print_avoidance -layer assist \
        -pw pw1 0.102 \
        -step 0.002 \
        -allow_cut \
        -minimize jogs \
        -min_area 0.00045 \
        -min_square_area 0.000625
}
OPC_ITERATION 3
...
                                         Set up correction iterations to
                                         reduce the probability of printing
                                         SRAFs. The number of iterations
                                         of sraf_print_avoidance is
                                         defined as an upper value of the
                                         range of the loop test
                                         expression.

```

MRC rules will be obeyed by [sraf_print_avoidance](#) if specified for the SRAF layer. If the corrected shape is in violation with an internal MRC rule, violating portions will be cut or the entire shape will be deleted if necessary.

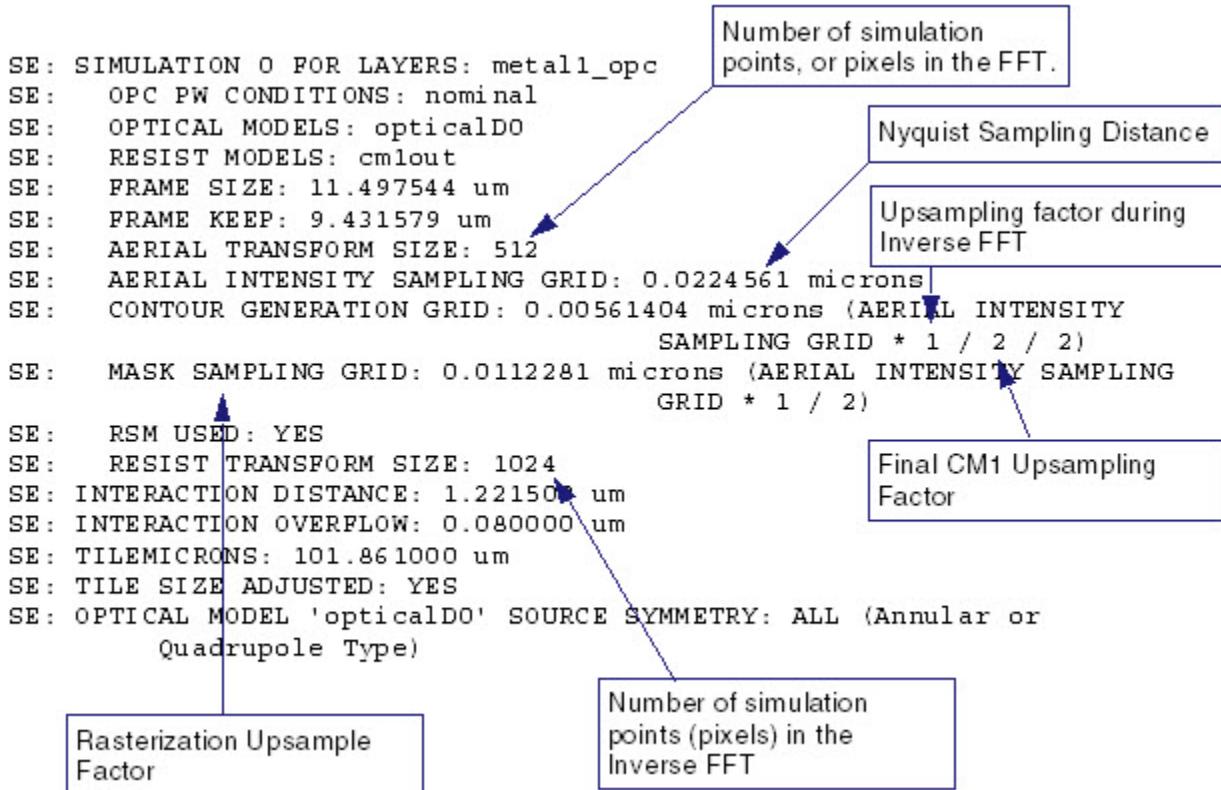
There are a number of best practices and recommendations available to help optimize this process. Refer to “[SRAF Print Avoidance Best Practices](#)” on page 661 for details.

Understanding the Transcript

At the end of each Dense OPC run, a transcript is produced. This transcript contains information that can be useful in analyzing the overall performance of the OPC run.

At the beginning of the transcript, information on the simulation run appears as shown in the figure.

Figure 3-33. Simulation Transcript Segment



Other notable outputs in the transcript include:

- AERIAL INTENSITY SAMPLING GRID is the Nyquist value. For example:
AERIAL INTENSITY SAMPLING GRID: 0.0224561 microns
- CONTOUR GENERATION GRID is equal to AERIAL INTENSITY SAMPLING GRID/4 (which is 1/2/2), unless you have `simulation_consistency` 1, then it is 3/8.
- MASK SAMPLING GRID is usually equal to 1/2 AERIAL INTENSITY SAMPLING GRID. If the setup file includes “`mask_sample_grid uniform`” then it is 1/4.
- INTERACTION OVERFLOW states by how much the extent of the outputs can exceed the extent of the inputs. For instance, if you have a “`setlayer size 0.02`”, the INTERACTION OVERFLOW 0.02; if you have a “`setlayer size -0.02`”, INTERACTION OVERFLOW would be 0.

For example:

```
INTERACTION DISTANCE = 5.009250
INTERACTION OVERFLOW = 0.200500
```

- FRAME SIZE is the size of frame that is valid for the given frame size. FRAME KEEP is the extra that is needed for the correct simulation results given the optical and resist interaction distances. For example:

```
FRAME SIZE 9.416092 um
FRAME KEEP 7.043678 um
```

The transcript contains information detailing the cumulative time for different portions of the Dense OPC run, and an overall timing summary them at the end of the run. This tool can be useful for evaluating the performance of your OPC run as you can see where the runtime is going in a hierarchical run, determine scalability and breakdown of time for various steps of the LITHO DENSEOPC operation.

The following is an example of that portion of the transcript:

DII:	cpu(s)	real(s)	count	avgreal	maxreal	tile	cell
DII: DETAILED: Performance for Dense OPC Operations							
DII:-----							
DII: operation							
DII:-----							
DII: FRAGMENT_AND_OP	8.81	9.37	1	9.367	9.367	0	---
DII: SIMULATE_AND_MEASURE_CODE	8.47	8.97	16	0.560	0.626	0	---
DII: SIMULATE_CODE	6.85	7.38	16	0.461	0.512	0	---
DII: MEASURE_CODE	1.52	1.50	16	0.093	0.103	0	---
DII: MOVEMENT_CODE	0.00	0.02	17	0.001	0.001	0	---
DII: DENSE_INITIALIZE_MASK	0.02	0.02	1	0.022	0.022	0	---
DII: DENSE_RUN_ITERATIONS	8.79	9.34	1	9.341	9.341	0	---
DII: DENSE_FILL_OUTPUT_AND_PROMOTION	0.00	0.00	1	0.003	0.003	0	---
DII: ALGO_MEASUREMENT_CODE	1.01	0.99	16	0.062	0.067	0	---
DII: MASK_CONSTR	0.00	0.00	18	0.000	0.000	0	---
DII: MASK_DESTR	0.00	0.00	18	0.000	0.000	0	---
DII: MASK_MOVE	0.00	0.00	17	0.000	0.000	0	---
DII: MASK_FIXMRC	0.00	0.00	17	0.000	0.000	0	---
DII: FRAGMENTATION	0.01	0.01	1	0.012	0.012	0	---
DII: FILTER_SETUP	0.00	0.01	16	0.001	0.001	0	---
DII:-----							

The timer names correspond roughly to the Calibre nmOPC script command names. In some cases, groups of commands are summarized together. For example, TAG_ELEMENT_OP also includes “fromlist” and “tolist”. All script command times are disjoint from each other, and should sum up to equal the total time (roughly). There are certain special timers, however, whose times are not disjoint from each other or from the script commands. These include:

- FRAGMENT_AND_OP

Indicates the total cumulative time for Dense OPC, excluding all database access, tiling and runtime lint. To compute the amount of time spent on database access and tiling, subtract the FRAGMENT_AND_OP time from total time for Dense OPC (TIME FOR ALL CELLS in hierarchical mode, or time for operation in flat mode).

- **SIMULATE_AND_MEASURE_CODE**
Indicates the time for “max_iterations” and EPE/cost calculation, assuming no script. If a script is present, this is the time for all OPC_ITERATION commands, but not for the DENSE_SIMULATE, DENSE_EPE, or DENSE_CD commands.
- **SIMULATE_CODE**
Specifies the time for max_iterations simulation only, not counting time for EPE/cost calculation.
- **MOVEMENT_CODE**
Specifies the time for max_iterations movement portion, or if script is present, the time for FRAGMENT_MOVE commands and OPC_ITERATION <n> movement time in the script.

For other information on the transcript, refer to the following resources:

- EUV slit information: See “[EUV Information in the Transcript](#)” on page 690.
- Overview of the contents of the transcript file: See “[Session Transcripts](#)” in the *Calibre Verification User's Manual*.
- Overview of the OPC contents of the transcript file: See “[Transcripts for Post-Tapeout Operations](#)” in the *Calibre Post-Tapeout Flow User's Manual*.
- Operation statistics (for example, FGC, HGC, LVHEAP): See “[Executive Process](#)” in the *Calibre Verification User's Manual*.
- OPC Errors: See “[OPC Operation Errors](#)” in the *Standard Verification Rule Format (SVRF) Manual*.
- Litho Errors: See “[Litho Operation Errors](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

Chapter 4

Calibre nmOPC Reference Syntax

Calibre nmOPC is configured through a series of batch script commands in a setup file.

Commands Grouped by Function	94
Calibre nmOPC SVRF Commands	97
LITHO DENSEOPC	98
LITHO CELLARRAY DENSEOPC	100
LITHO CL DENSEOPC	102
LITHO EUV DENSEOPC	105
LITHO ML DENSEOPC	107
RET FLARE_CONVOLVE	109
RET OPTIMIZE_EUV_HDB	113
LITHO DENSEOPC Setup File Commands.....	115
Calibre OPCverify Setup File Commands	117
clone_transformed_cells	119
ddm_model_load	121
denseopc_options	122
etch_imagegrid	124
etch_model_load	125
flare_longrange	126
flare_map_shift	129
flare_model_load	130
global_lithomodel_path	132
layer	133
lintmodel	135
ml_featurevector_load	139
ml_model_load	140
modelpath	141
optical_model_load	142
output_text_filepath	143
png	144
resist_model_load	145
setlayer curve	146
setlayer curve_target	152
setlayer denseopc	160
setlayer layer_simplify	162
setlayer tilegen	164
setlayer veb_retarget	167
simulation_consistency	171
svrf_var_import	172

veb_simulate	173
denseopc_options Keywords	174
active_layer	179
adjust_moved_target_sites	182
adjust_target_control_sites	183
algorithm	184
allow_single_site_correction	186
background (for denseopc_options block)	187
check_movement	190
chiplet_placement_flare_tolerance	191
corner_control	192
displacement_limit_mode	193
curvilinear_opc	194
effective_epe_regulation	195
enhance_corner_to_corner_site_pairing	196
epe_spacing	197
etch_model	198
feedback	199
feedback_balance (Deprecated in 2016.2)	200
feedback_mode	201
flare_model	203
fragment_coincident	204
fragment_consistency_mode	206
fragment_corner	207
fragment_flaglayer	217
fragment_grid	218
fragment_inter	219
fragment_interlayers	234
fragment_island	236
fragment_layer	242
fragment_layer_order	248
fragment_marker	252
fragment_max	253
fragment_min	257
fragment_minjog	259
fragment_nearly_coincident	260
fragment_nop	262
fragment_optimize	263
fragment_preserve_length	265
fragment_ripple	266
fragment_visible	271
freeze_skew_edges	272
grids_for_angle_45_snap	273
image	274
jog_freeze	279

jog_ignore_metric	280
lapi_exec_tcl	282
layer (for denseopc_options Block)	285
line_end_fragment	292
lint	293
mask_chip_corner	297
max_iterations	300
max_iter_movement	301
max_opc_move	302
min_dog_ear_length	303
mpopc	304
mrc_consistency	309
mrc_mode	310
mrc_rule	311
mrc_rule_area	318
mrc_rule_priority	319
no_opc_region	320
nominal_epe_limit	321
one_time_site_pairing	322
opc_grid_multiplier	323
pvbnd_tolerance	324
pw_condition	325
pwopc_mode (Deprecated in 2016.2)	332
read_layer_properties	333
retarget_layer	334
retargeting_options	338
scale	339
spa_promotion	340
sraf_promotion_distance	341
step_size	342
topo_model	343
topo_model_load	344
veb_corner_site_placement	345
veb_evalpts_per_fragment	347
veb_pseudo_nyquist	349
version	350
wafer_enclose	351
wafer_enclosedby	354
wafer_exclude	357
Calibre nmOPC File Formats	358
Lithomodel (Litho Model Format)	359

Commands Grouped by Function

The following sections collect some special purpose commands by functionality. The list is not exhaustive.

EUV Commands

EUV commands are used with EUV optical models, which include additional effects such as long-range flare and multiple slits. These commands should not be used with DUV models.

The list does not include keywords used in creating EUV models.

There are several different EUV flows, which are discussed in “[Calibre OPC EUV Flows](#)” on page 681.

Table 4-1. EUV-Specific Commands

Command	Description
SVRF Commands	
EUV_MODEL_REDUCTION	Toggles model reduction mode for EUV Dense OPC. May be specified at most twice.
LITHO EUV DENSEOPC	Derives output target layers for SVRF, specific to the EUV flow.
RET FLARE_CONVOLVE	Generates flare maps from a flare model.
RET OPTIMIZE_EUV_HDB	Finds regions with similar shadow biases and flares.
Setup File Commands	
euv_field_center ¹	Specifies the center of an EUV field.
euv_slit_x_center ¹	Sets a placement x-coordinate for EUV through-slit models.
flare_longrange	Associates a pre-computed long range flare convolution with a flare model contained inside a litho model.
flare_map_shift	Specifies an amount to shift the flare map.
flare_model_load	Loads the flare model file into Calibre nmOPC and assigns it the given label for later reference.
global_lithomodel_path	Specifies the filepath to a global litho model's top Lithomodel file.
output_text_filepath	Specifies a filename for the encrypted SVRF file generated by RET OPTIMIZE_EUV_HDB.
png	Specifies the path to a PNG file created by RET FLARE_CONVOLVE.

Table 4-1. EUV-Specific Commands (cont.)

Command	Description
denseopc_options Keywords	
chiplet_placement_flare_tolerance	Specifies a threshold for the difference in flare values that prevents re-using OPC results for different placements of a chiplet (large subassembly or block).
flare_model	References the flare model.

1. Documented in the *Calibre OPCverify User's and Reference Manual*

Machine Learning Commands

Calibre® mlOPC commands are used for capturing data to train a machine learning model and for then using the trained model in place of most OPC iterations to improve performance. For more information on Calibre mlOPC, see “[Calibre mlOPC](#)” on page 691.

Table 4-2. Machine Learning Commands

Command	Description
SVRF Commands	
LITHO ML DENSEOPC	Derives output target layers for SVRF using machine learning models.
Setup File Commands	
ml_featurevector_load	Loads the model metadata.
ml_model_load	Loads the machine learning model.
Tcl Scripting Commands	
ML_OP	Moves or retargets fragments using machine learning models.
ML_VECTOR_CAPTURE_END	Marks the end of the data capture segment of the setup file used for machine learning.
ML_VECTOR_CAPTURE_INIT	Marks the beginning of the data capture segment of the setup file used for machine learning.

Multi-Patterning Commands

Calibre mpOPC is a utility used for double and multi-patterning mask correction. The primary interface, the mpopc command, is one of the denseopc_options keywords, and can be added to a standard Calibre nmOPC setup file. The NEWTAG sadp command is useful for debugging multi-patterned fragmentation.

Table 4-3. DP- and MP-Specific Commands

Command	Description
denseopc_options Keywords	
mpopc	Adds special measurement sites to account for multi-patterning issues.
Tcl Scripting Commands	
NEWTAG sadp	Classifies fragments according to the relationship between associated cut masks and target layers.

Variable Etch Biasing (VEB) Commands

Several setup file commands have been created specifically for VEB. For instructions on how to use them, see “[The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC](#)” on page 671.

Table 4-4. VEB-Specific Keywords

Command	Description
Setup File Commands	
etch_imagegrid	Defines etch grid size.
etch_model_load	Loads a specified etch model.
setlayer veb_retarget	Creates and derives layers for VEB retargeting.
veb_simulate	Performs an etch bias simulation on an input contour layer.
denseopc_options Keywords	
etch_model	Loads etch model to be used by setlayer veb_retarget.
veb_corner_site_placement	Specifies where along a corner fragment to place the VEB evaluation point.
veb_evalpts_per_fragment	Specifies how many VEB evaluation points to place on each non-corner fragment.
veb_pseudo_nyquist	Defines a pseudo-Nyquist value for automatic fragmentation.

Calibre nmOPC SVRF Commands

Calibre nmOPC uses a set of SVRF commands for its operations.

The rule file may use many other SVRF commands in addition to the ones listed here. The rule file syntax and other SVRF commands are described in the *Standard Verification Rule Format (SVRF) Manual*.

Note

-  These keywords are separate from the “denseopc_options Keywords” on page 174. Specifying these keywords inside a denseopc_options block will result in an error.
-

Table 4-5. Calibre nmOPC SVRF Commands

Command	Description
LITHO DENSEOPC	Derives output target layers for SVRF.
LITHO CELLARRAY DENSEOPC	Derives output target layers for SVRF, specific to the CellArray OPC product.
LITHO CL DENSEOPC	Derives output target layers for SVRF for curvilinear input.
LITHO EUV DENSEOPC	Derives output target layers for SVRF, specific to the EUV flow.
LITHO ML DENSEOPC	Derives output target layers for SVRF using machine learning models.
RET FLARE_CONVOLVE	Generates flare maps from a flare model.
RET OPTIMIZE_EUV_HDB	Finds regions with similar shadow biases and flares.

LITHO DENSEOPC

Calibre nmOPC SVRF Commands

Derives output target layers for SVRF.

Usage

```
output_layer_name = LITHO DENSEOPC layer [layer...]  
FILE {filename | name | '['  
      inline_file  
  ']' }  
MAP 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.

- **layer**

A required original or derived polygon layer upon which to perform dense OPC operations. You can specify **layer** any number of times in one statement.

- **FILE {filename | name | '['**

inline_file

']'}

A required keyword, followed by one of the following:

A **filename** indicating the path to the setup file. The **filename** parameter can contain environment variables.

The **name** parameter of a LITHO FILE specification statement. The setup file is specified in the Litho File statement.

An **inline_file** between brackets ([]). Characters within the brackets and on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAP name**

A required keyword, followed by the name of a layer from the technology setup file. This layer is then mapped to the layer in the rule file for output.

Description

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

Examples

The following example performs OPC on the shapes in layer M1 using the MY_DENSE setup file options. In the MY_DENSE file, a setlayer denseopc command for the input layer (M1) is included in the MY_DENSE file along with many other settings and options (not shown). The output is written to the M1_OPc layer, which is then returned to LITHO DENSEOPC and mapped to the M1_bias2 layer. This layer is then made available to other Calibre operations using the layer assignment (*derived_layer* = ...).

```
M1_bias2 = LITHO DENSEOPC M1 FILE MY_DENSE MAP M1_bias2  
  
LITHO FILE MY_DENSE [  
    ...  
    setlayer M1_OPc = denseopc M1 sraf MAP m1 OPTIONS OPTS  
    ...  
]
```

LITHO CELLARRAY DENSEOPC

Calibre nmOPC SVRF Commands

Derives output target layers for SVRF, specific to the CellArray OPC product.

Usage

```
output_layer_name = LITHO CELLARRAY [EUV] DENSEOPC layer [layer...]
  FILE {filename | name | '['
    inline_file
  '']}
  MAP 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.

- EUV

An optional keyword that should be specified if EUV models are used. This keyword requires a Calibre EUV license.

- ***layer***

A required original or derived polygon layer upon which to perform dense OPC operations. You can specify ***layer*** any number of times in one statement.

- **FILE {filename | name | '['**

inline_file

'']

A required keyword, followed by one of the following:

A ***filename*** indicating the path to the setup file. The ***filename*** parameter can contain environment variables.

The ***name*** parameter of a LITHO FILE specification statement. The setup file is specified in the Litho File statement.

An ***inline_file*** between brackets ([]). Characters within the brackets and on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAP *name***

A required keyword, followed by the name of a layer from the technology setup file. This layer is then mapped to the layer in the rule file for output.

Description

The LITHO CELLARRAY DENSEOPC command unlocks Calibre® CellArray OPC features. It requires a separate license, as described in “[Calibre Cell Array OPC](#)” in the *Calibre Administrator’s Guide*. The litho setup file must use litho models.

Use LITHO CELLARRAY DENSEOPC to make use of [CLASSIFY_FRAGMENTS](#) -cleanup_tol, which causes all similar patterns to receive identical correction, within certain tolerances. The CellArray OPC features are designed specifically for highly repetitive layouts such as DRAMs and not recommended for use outside memory areas.

Limitations

Only the following setlayer commands can be used to construct the input layers within the setup file:

- setlayer and
- setlayer curve
- setlayer curve_target
- setlayer or

You can also construct the input layers before calling LITHO CELLARRAY OPC. The restriction only applies to layers derived within the same setup file that includes CLASSIFY_FRAGMENTS ...-cleanup_tol.

Related Topics

[LITHO DENSEOPC](#)

[LITHO EUV DENSEOPC](#)

[setlayer and \[Calibre OPCverify User’s and Reference Manual\]](#)

[setlayer curve](#)

[setlayer curve_target](#)

[setlayer or \[Calibre OPCverify User’s and Reference Manual\]](#)

LITHO CL DENSEOPC

Calibre nmOPC SVRF Commands

Derives output target layers for SVRF for curvilinear input.

Usage

```
output_layer_name = LITHO CL [EUV] DENSEOPC layer [layer...]
  FILE {filename | name | '['
    inline_file
  '']}
  MAP name
```

Arguments

- ***output_layer_name***

A required argument 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.

- EUV

An optional keyword that should be specified if EUV models are used. This keyword requires a Calibre EUV license.

- ***layer***

A required original or derived layer upon which to perform dense OPC operations. You can specify ***layer*** any number of times in one statement.

Traditional dense OPC requires polygon input with minimal skew edges. The CL OPC operations accept curved edges like those typically seen in photonic designs.

- FILE {filename | name | '['
 inline_file
 '']}

A required keyword, followed by one of the following:

A ***filename*** indicating the path to the setup file. The ***filename*** parameter can contain environment variables.

The ***name*** parameter of a LITHO FILE specification statement. The setup file is specified in the Litho File statement.

An ***inline_file*** between brackets ([]). Characters within the brackets and on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAP *name***

A required keyword, followed by the name of a layer from the technology setup file. This layer is then mapped to the layer in the rule file for output.

Description

The LITHO CL DENSEOPC command provides support for curvilinear layouts. It is a separately licensed product, Calibre® nmCLOPC. This command is used in conjunction with the [setlayer denseopc](#) command to derive output target layers for SVRF.

The litho setup file for CL runs should include [setlayer layer_simplify](#) to generate an approximation of the curved shape to use for OPC corrections. In the denseopc_options block, [curvilinear_opc](#) activates curvilinear correction and sets the MRC values.

The litho setup file may also contain typical dense OPC commands to fragment long edges, and simple mrc_rule statements to keep shapes from extreme adjustments. OPC correction is still performed with OPC_ITERATION or similar.

Examples

The following example demonstrates the layer setup for curvilinear OPC. It is not a complete rule file.

The SVRF file must contain LITHO CL DENSEOPC. Its input is the curvilinear layout, and the output is the post-OPC mask.

```
opc_target = LITHO CL DENSEOPC curved_layer FILE cl.setup MAP opc_target
```

The litho setup file *cl.setup* includes all the required keywords such as modelpath and layer, and also setlayer denseopc and setlayer layer_simplify. The setlayer layer_simplify command creates a derived layer representing the curve with line segments that can undergo OPC.

```
setlayer simplified = layer_simplify curved_layer
setlayer opc_target = denseopc simplified curved_layer OPTIONS clopc.in \
MAP simplified
```

In the denseopc_options block “clop.in” layers are mapped by sequence. The first layer, simplified, is the OPC layer. The OPC layer gets modified in order to make the simulated image better match curved_layer. The retarget_layer command identifies curved_layer as the ideal image.

```
denseopc_options clopc.in {
    version 1
    layer simplified    opc mask_layer 0
    layer curved_layer hidden
    retarget_layer curved_layer spline
```

The curvilinear_opc command defines how sharp or rounded the adjusted mask (opc_target) will be. It also can provide minimum width for shapes and spacing between shapes. This is recommended because MRC rules only act on Manhattan polygons.

```
curvilinear_opc smooth_radius 0.03 min_width 0.02 min_space 0.02
```

The denseopc_options block also contains required settings such as image, fragment_min, and fragment_max and a command (max_iterations or OPC_ITERATION) to perform OPC. If the curved_layer includes shapes that may create long edges (for example, a waveguide), the setup should also include custom fragmentation with fragment_layer and MRC rules.

Related Topics

[Layer Types](#)

[retarget_layer](#)

LITHO EUV DENSEOPC

Calibre nmOPC SVRF Commands

Derives output target layers for SVRF, specific to the EUV flow.

Usage

```
output_layer_name = LITHO EUV DENSEOPC layer [layer...]  
FILE {filename | name | '['  
      inline_file  
      ']'} MAP 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.

- **layer**

A required original or derived polygon layer upon which to perform dense OPC operations. You can specify **layer** any number of times in one statement.

- **FILE {filename | name | '['
 inline_file
 ']}'**

Required keyword, followed by one of the following:

A **filename** indicating the path to the setup file. The **filename** parameter can contain environment variables.

The **name** parameter of a LITHO FILE specification statement. The setup file is specified in the Litho File statement.

An **inline_file** between brackets ([]). Characters within the brackets and on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAP name**

A required keyword, followed by the name of a layer from the technology setup file. This layer is then mapped to the layer in the rule file for output.

Description

The EUV DENSEOPC command is a tool argument to the Calibre LITHO batch operation specifically for the Extreme Ultraviolet Lithography (EUV) correction. This command is used in conjunction with the setlayer denseopc command to derive output target layers for SVRF. When used with “processing_mode flat”, you must manually specify either ULTRAFlex or TURBOflex mode. This command is otherwise identical to the LITHO DENSEOPC command.

Related Topics

[setlayer denseopc](#)

[LITHO DENSEOPC](#)

[processing_mode \[Calibre Post-Tapeout Flow User's Manual\]](#)

LITHO ML DENSEOPC

Calibre nmOPC SVRF Commands

Derives output target layers for SVRF using machine learning models.

Usage

```
output_layer_name = LITHO ML [EUV] DENSEOPC layer [layer...]  
FILE {filename | name | '['  
      inline_file  
      ']'}  
MAP 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.

- EUV

An optional keyword that should be specified if EUV models are used. This keyword requires a Calibre EUV license.

- ***layer***

A required original or derived polygon layer upon which to perform dense OPC operations. You can specify ***layer*** any number of times in one statement.

- **FILE {filename | name | '['**

inline_file

']'

A required keyword, followed by one of the following:

A ***filename*** indicating the path to the setup file. The ***filename*** parameter can contain environment variables.

The ***name*** parameter of a LITHO FILE specification statement. The setup file is specified in the Litho File statement.

An ***inline_file*** between brackets ([]). Characters within the brackets and on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAP *name***

A required keyword, followed by the name of a layer from the technology setup file. This layer is then mapped to the layer in the rule file for output.

Description

The LITHO ML DENSEOPC command allows machine-learning commands to be included in the Calibre nmOPC setup file. It also requires a Calibre mlOPC license, as described in “[Calibre mlOPC](#)” in the *Calibre Administrator’s Guide*.

Note

 Machine-learning setup files must use litho models. The older syntax is not supported.

Calibre mlOPC setup files have distinct commands for the two phases that are run with Calibre:

- **Data Acquisition** — During the first phase, the run captures final corrections from known-good runs. These runs should start with layouts and rule files that are known to have produced good final masks. Add the commands “[ML_VECTOR_CAPTURE_INIT](#)” and “[ML_VECTOR_CAPTURE_END](#)” before and after the movement section. Fragmentation should occur before [ML_VECTOR_CAPTURE_INIT](#). (The movement section typically includes [OPC_ITERATION](#) or [FRAGMENT_MOVE](#).)
- **Deployment** — The second phase makes use of the trained machine learning model. The model should be used only for the same layer and process it was trained upon. The model is loaded by “[ml_model_load](#)” and used by “[ML_OPCT](#)” to do the initial fragment movements.

Model training does not use SVRF.

Calibre mlOPC is not compatible with Calibre LPE.

Related Topics

[Calibre mlOPC](#)

RET FLARE_CONVOLVE

Calibre nmOPC SVRF Commands

Generates flare maps from a flare model.

Usage

```
dummy_name = RET FLARE_CONVOLVE input_layer [marker_layer]  
[INSIDE OF EXTENT | INSIDE OF x1 y1 x2 y2 | INSIDE OF LAYER layer3]  
FILE '['  
    { flare_model filename / litho_model directory }  
    pngout pngfile  
    [window wxy]  
    {truncate | wrap [field_overlap um [um] [field_type clear | dark]]}  
    [input_mask format_specific_options]  
    [dummy_fill {value | auto}]  
'']
```

Arguments

- ***dummy_name***
A required dummy name required to generate the flare map.
- ***input_layer***
A required name of an original or derived polygon layer. This is the layer containing the data to which Calibre applies the convolution functionality.
- ***marker_layer***
The name of an original or derived polygon layer. This layer should only be specified with the *dummy_fill* argument.
- INSIDE OF EXTENT | INSIDE OF *x1 y1 x2 y2* | INSIDE OF LAYER *layer3*
An optional keyword set that defines a rectangular boundary within which the tool creates the density grid. The choices are:
 - INSIDE OF EXTENT — Specifies that the rectangular boundary is the extent of the *input_layer*. This is the default behavior if you do not specify one of the INSIDE OF options.
 - INSIDE OF *x1 y1 x2 y2* — Specifies the lower-left and upper-right corners of the rectangular boundary in user units. You must write the coordinates relative to the origin of the Calibre database, and enclose any negative coordinates in parentheses (). For example:

INSIDE OF (-123) 123 (-12) 456
 - INSIDE OF LAYER *layer3* — Specifies that the rectangular boundary is the extent of *layer3*. The layer specified by *layer3* cannot be the same as *input_layer*.

- **flare_model *filename***

Specifies the flare model file to be input for convolution. Cannot be specified with **litho_model**.

- **litho_model *directory***

Specifies a litho model file containing flare model information. Cannot be specified with **flare_model**.

- **pngout *pngfile***

Specifies the PNG-format output file (the flare map).

- **window *wxy***

An optional keyword that specifies the window size used to compute density in microns. The step argument for density computation is also set to the specified window size (**wxy**). The **wxy** parameter is a single number that defines the size of the square window for density convolution. The default is the same as the **DENSITY** SVRF command. If the window is larger than the boundary in the x- or y-direction, it is truncated to the boundary size just as in **DENSITY** command.

- **truncate | wrap**

A keyword set that specifies how the tool calculates the density of a window that overlaps the boundary box of the **input_layer**.

truncate — An optional keyword that instructs the tool to compute the density of the overlapping window by truncating the size of the window so that it does not extend beyond the boundary box.

wrap — An optional keyword that instructs the tool to compute the density of the overlapping window by duplicating and adding the boundary box, and its data, to the right-hand or top side of the main boundary box. The density measurement is then taken in the window that intersects the duplicated boundaries.

- **field_overlap *um* [*um*] [field_type *clear* | *dark*]**

An optional argument for **wrap** that specifies how much overlap to assume in the neighboring fields. The distance *um* should be in microns measured at the wafer.

field_overlap *um* — This form calculates the overlap from all eight neighboring fields (top, left, right, bottom, ne, se, nw, and sw) to the center is the same.

field_overlap *um um* — This form applies the first floating-point value to the left and right neighbors, and the second value to the top and bottom neighbors. The corners (ne, nw, se, and sw) combine the overlaps of their respective X and Y values.

A warning is added to the log file if overlapping polygons are detected during the part of the operation when eight neighboring fields are created.

field_type {*clear* | *dark*} — An optional argument for **field_overlap** that specifies the type of the final layout, clear or dark. (The default is dark.) If it is set incorrectly, the **field_overlap** results can be inaccurate.

The **field_overlap** argument cannot be specified with **input_mask**.

- *input_mask format_specific_options*

An optional argument for direct input of VSBJOB or MEBESJOB jobdecks or MEBES pattern files for density calculations. When this parameter is specified, RET FLARE_CONVOLVE uses the geometrical data from the specified mask instead of the input hierarchical layer. All data in the mask is merged. If the extent is specified with INSIDE OF, it must conform to the coordinate system of the jobdeck. The precisions specified through SVRF overrides the mask units.

VSBJOB: *jobdeck_path mask_name* — Specify a directory path and mask name.

MEBES: *pattern_path* — Specify a directory path to the MEBES pattern.

MEBESJOB: *jobdeck_path [-levels ALL | list]* — Specify a jobdeck file path and the levels to consider. By default, all levels are included in the computation. If using a list, separate levels by commas. Ranges are allowed but not spaces; for example, “-levels 501-504,508,512”.

Because jobdecks can have chips with different units, a common unit is computed using all the chip units and the placement information from the jobdeck. This common unit is referred to as the jobdeck unit.

- *dummy_fill {value | auto}*

An optional keyword that assigns the area covered by *marker_layer* a pre-convolution density. This can be useful when the layout has large “cutouts” with zero density, as is common with kerf regions.

A *marker_layer* is required when specifying *dummy_fill*. The layer may contain multiple polygons.

The *dummy_fill* keyword must include one of the following arguments that specifies the density to assign:

value — A number from 0 to 1, inclusive.

auto — A keyword that uses the average density of *input_layer* outside of *marker_layer*.

Note that the *dummy_fill* value is *added* to any density that would normally be computed during convolution for the region covered by the marker layer.

Description

This EUV operation creates a PNG file that shows density values after applying convolution algorithms. The algorithms evaluate the density value of a window by taking into consideration the layout densities in the surrounding regions.

Used by extreme ultraviolet (EUV) OPC, RET FLARE_CONVOLVE generates flare maps based on the flare model. For more information, refer to “[Models in EUV Lithography](#)” on page 50.

RET FLARE_CONVOLVE does not produce layers.

Examples

```
LAYER m1 4

Fdummy = RET FLARE_CONVOLVE m1 FILE /*  
    flare_model /Model/flaremodel  
    pngout flare_map_ret.png  
    WINDOW 0.4  
    WRAP  
*/]

Fdummy { copy Fdummy} drc check map Fdummy 100
```

RET OPTIMIZE_EUV_HDB

Calibre nmOPC SVRF Commands

Finds regions with similar shadow biases and flares.

Usage

dummy_layer = RET OPTIMIZE_EUV_HDB layer MAP dummy_layer FILE name

Arguments

- ***dummy_layer***

A required name for the derived layer. This layer is not actually used in the later steps, but is required for RET SVRF commands.

- ***layer***

A required argument specifying the name of an original or derived polygon layer. The layer is analyzed in conjunction with the long range flare and shadow bias to determine regions with similar pattern and exposure.

- ***MAP dummy_layer***

A required argument that normally is used to pass back a layer created within the litho setup file. As the RET OPTIMIZE_EUV_HDB setup commands do not generate a new layer, specify a dummy layer name here to satisfy the parser.

- ***FILE name***

A required argument specifying a name that matches that of a LITHO FILE statement in the SVRF rule file. The LITHO FILE for RET OPTIMIZE_EUV_HDB can contain only the following keywords:

- [euv_slit_x_center](#) (*Calibre OPCverify User's and Reference Manual*)
- [flare_map_shift](#)
- [global_lithomodel_path](#)
- [output_text_filepath](#)
- [png](#)

Description

This optional SVRF command analyzes its input to find polygonal regions with similar shadow biases and flare ranges within a narrow tolerance. It generates an encrypted SVRF file, *optimize_euv_hdb.out*. (You can give the file another name with *output_text_filepath*.)

The *optimize_euv_hdb.out* is used for HDB construction in the hierarchical EUV OPC flow. It contains the SVRF variable EUV_RECOMMEND_FLAT and additional SVRF commands that are added to an SVRF recipe that calls LITHO EUV DENSEOPC to properly handle the design.

Note

- The *optimize_euv_hdb.out* file is specific to a particular combination of input layer, PNG flare map, and litho model. If the layer or EUV process models change, a new *optimize_euv_hdb.out* file must be created.
-

Examples

The following example shows RET OPTIMIZE_EUV_HDB within the context of an SVRF section. Notice that the dummy output layer is written in a DRC CHECK MAP statement to keep the Calibre optimizer from eliminating the operation.

```
LAYER m1 16

LITHO FILE optimize /*  
    png models/euv_model/flare.png  
    global_lithomodel_path models/euv_model/Lithomodel  
    euv_slit_x_center 0  
*/]

DUMMY_OUT = RET OPTIMIZE_EUV_HDB m1 MAP DUMMY_OUT FILE optimize
DUMMY { COPY DUMMY_OUT } DRC CHECK MAP DUMMY 1
```

LITHO DENSEOPC Setup File Commands

The *inline_file* argument to LITHO DENSEOPC specifies an inline LITHO FILE parameter block containing a *lowercase only* command initialization block consisting of the following keywords, specified one per line.

The following figure illustrates the location of the LITHO DENSEOPC commands within a setup file.

Figure 4-1. Setup File Location for LITHO DENSEOPC Commands

```
ml_out = LITHO DENSEOPC met1 sraf
FILE ml_opc MAP ml_opc
LITHO FILE ml_opc /*

modelpath ./models
optical_model_load opt ml.opt
resist_model_load res ml.cml

layer ml hidden clear
layer sraf hidden clear

denseopc_options ml_opt {

    version 1
    background poly opc atten 0.06
    layer ml opc clear
    layer sraf visible clear
    image optical opt resist res

    NEWTAG all POLY -out A
    NEWTAG edge A len < 0.3 corner1 convex corner2 convex
        -out line_end
    NEWTAG neighbor both line_end len < 0.4 corner convex \
        -out line_end_adj
    ...
    OPC_ITERATION 8
    NEWTAG all POLY -out A

}

setlayer ml_opc = denseopc ml sraf MAP ml OPTIONS ml_opt
```

LITHO DENSEOPC Setup File
Commands

Note

- These keywords are separate from the “[denseopc_options Keywords](#)” on page 174.
Specifying these keywords inside a denseopc_options block will result in an error.

Table 4-6. LITHO DENSEOPC Setup File Commands

Command	Description
Calibre OPCVerify Setup File Commands	You can also use verification commands (Calibre OPCverify batch commands) inside the same setup file. Note that several Calibre OPCverify commands and Calibre nmOPC commands are, in fact, identical and shared (such as background).
clone_transformed_cells	Creates clones of cells with nonzero transforms so that simulation with asymmetric optical models will be correct.
ddm_model_load	Loads the optional Domain Decomposition Method model.
denseopc_options	Creates a uniquely named sub-command block that contains OPC instructions.
etch_imagegrid	Defines etch grid size.
etch_model_load	Loads a specified etch model.
flare_longrange	Associates a pre-computed long range flare convolution with a flare model contained inside a litho model.
flare_map_shift	Specifies an amount to shift the flare map.
flare_model_load	Loads the flare model file into Calibre nmOPC and assigns it the given label for later reference.
global_lithomodel_path	Specifies the filepath to a global litho model's top <i>Lithomodel</i> file.
layer	Defines layer characteristics such as name, type, transmission, and mask.
lintmodel	Loads a file with custom lint checks.
ml_featurevector_load	Loads the model metadata.
ml_model_load	Loads the machine learning model.
modelpath	Specifies the default search location for model files.
optical_model_load	Specifies an optical model.
output_text_filepath	Specifies a filename for the encrypted SVRF file generated by RET OPTIMIZE_EUV_HDB.
png	Specifies the path to a PNG file created by RET FLARE_CONVOLVE.
resist_model_load	Specifies a resist model.
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.

Table 4-6. LITHO DENSEOPC Setup File Commands (cont.)

Command	Description
<code>setlayer denseopc</code>	Creates and derives DENSEOPC target layers.
<code>setlayer layer_simplify</code>	Creates a curve-like layer that can receive correction from curvilinear_opc.
<code>setlayer tilegen</code>	Creates a layer to display tile and cell boundaries.
<code>setlayer veb_retarget</code>	Creates and derives layers for VEB retargeting.
<code>simulation_consistency</code>	Enables the simulation engine to perform actions to improve consistency.
<code>svrf_var_import</code>	Imports one or more variables previously defined in the SVRF rule file into the LITHO DENSEOPC block.
<code>veb_simulate</code>	Performs an etch bias simulation on an input contour layer.

Calibre OPCverify Setup File Commands

You can also use verification commands (Calibre OPCverify batch commands) inside the same setup file. Note that several Calibre OPCverify commands and Calibre nmOPC commands are, in fact, identical and shared (such as background).

Refer to the *Calibre OPCverify User's and Reference Manual* for complete descriptions of the commands listed in [Table 4-7](#).

Note

-  These keywords are separate from the “[denseopc_options Keywords](#)” on page 174.
 - Specifying these keywords inside a denseopc_options block will result in an error.
-

Table 4-7. Calibre OPCverify Setup File Commands

Command	Description
<code>background</code>	Defines the background transmission values of the mask for each exposure.
<code>euv_field_center</code> <code>euv_slit_x_center</code>	Defines the center slit and the center of the field for use with EUV optical models.
<code>imagegrid</code>	Defines the value of the final contour image grid in microns. The smaller the imagegrid value is, the longer simulation takes, and more vertices will be produced in the simulation contours.
<code>mask_sample_grid</code>	Specifies the mask sampling grid.

Table 4-7. Calibre OPCVerify Setup File Commands (cont.)

Command	Description
processing_mode	Specifies the processing mode used by the litho operation.
progress_meter	Issues completion percentage reports as each tile is processed in OPC.
rasterizer_upsample_factor	Sets the rasterizer upsample factor for the simulation mask sample grid relative to the optical (Nyquist) grid.
setlayer	Derives a layer. Many options can be specified. Refer to “ Setlayer Operations Reference ” in the <i>Calibre OPCVerify User’s and Reference Manual</i> for details.
tilemicrons	Specifies the size of a tile.

clone_transformed_cells

LITHO DENSEOPC Setup File Commands

Creates clones of cells with nonzero transforms so that simulation with asymmetric optical models will be correct.

Usage

`clone_transformed_cells {decide | override}`

Arguments

- **decide**

The optical model is checked to decide if cell cloning is activated. More specifically, this option causes the Calibre engine to check if any asymmetric sources are present in the optical model. If so, the Calibre database constructor is instructed to clone cells which have rotated or mirrored placements, so that hierarchical simulation of the rotated structures will be correct.

This is the default.

- ***override***

Always use the specified behavior, regardless of the optical model. One and only one of the following keywords must be specified:

no — Do not clone any cells.

rotations — Clone rotated cells. This is equivalent to [LAYOUT CLONE ROTATED PLACEMENTS YES](#) and can multiply cell count by two.

x_axis_reflection or **y_axis_reflection** — Clone rotated cell placements *and* cells that are reflected across the X- or Y-axis.

yes — Clone all rotated or reflected cells, as would occur for asymmetric source shapes. This is equivalent to [LAYOUT CLONE TRANSFORMED PLACEMENTS YES](#) and can multiply cell count by eight.

Description

When using an asymmetrical source, cell orientation can have a significant impact on results. You can control how the simulations handle cell orientation when calculating the aerial image for a hierarchical design. The application can:

- Calculate the aerial image for the original cell, then adjust the orientation of the image to match each cell.
- Calculate a separate aerial image for each orientation in the cell. To do this, the application creates one “clone” of the cell for each orientation represented in the design and transforms the clone as necessary to match the design, and calculates a separate aerial image for each clone.

See “[clone_transformed_cells](#)” in the *Calibre OPCVerify User’s and Reference Manual* for a detailed discussion of the decision algorithm and cloning-related information in the transcript.

Examples

This example enables optical model checking for cell cloning:

```
clone_transformed_cells decide
```

ddm_model_load

LITHO DENSEOPC Setup File Commands

Loads the optional Domain Decomposition Method model.

Usage

`ddm_model_load ddm_model_name {filename | '{' inline_model '}'}`

Arguments

- *ddm_model_name*

A required argument specifying the name that you will use to refer to this DDM model in the Calibre OPCVerify setlayer [image](#) commands.

- *filename*

A required argument (you must specify either *filename* or *inline_model*) specifying a filename to load containing the DDM model. If you do not specify a filename, an inline optical model must be supplied instead.

- '{' *inline_model* '}'

A required argument (you must specify either *filename* or *inline_model*) defining a DDM model. The entire specification must be enclosed in braces.

Description

This is an optional command that loads the Domain Decomposition Method (DDM) model contained in the file *filename*, or is defined by specifying model text inline within braces ({}). The DDM model is assigned the user-defined *ddm_model_name*, which you may later reference in the [image](#) command.

Note

 For more information on DDM models, see the [Calibre WORKbench User's and Reference Manual](#).

denseopc_options

LITHO DENSEOPC Setup File Commands

Inline sub-command block

Creates a uniquely named sub-command block that contains OPC instructions.

Usage

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

Arguments

- *opt_name*

A required option that specifies the unique name of a sub-command block of LITHO DENSEOPC setup file commands defined inside a LITHO FILE parameter block. This unique name will be passed to a setlayer [denseopc](#) command for processing. There is a 15 character limit on the name.

- *options_list*

The sub-command block of LITHO DENSEOPC setup file commands defined inside the **denseopc_options** parameter block.

Refer to the section “[denseopc_options Keywords](#)” for a complete list of keywords supported for this sub-command block.

Description

Creates a uniquely-named sub-command block inside a LITHO FILE statement to set nmOPC options. This uniquely-named block is then passed to a setlayer **denseopc** command to be processed.

The text within the **denseopc_options** command is run through the Tcl subst command once to evaluate all variables before run time.

The **denseopc_options** command is required when using a setlayer **denseopc** command.

Examples

The following excerpt from an SVRF file illustrates a setlayer **denseopc** command referring to a **denseopc_options** block, *opts*. Both the setlayer **denseopc** command and the **denseopc_options** block are within a LITHO FILE statement.

```
VARIABLE minInternal 0.06  
POLY_OPCT = LITHO DENSEOPC POLY SBAR FILE dopc_setup MAP L1  
  
LITHO FILE dopc_setup /*  
  ...
```

```
set MINEXT 0.010
set the_scale 1.1
svrf_var_import minInternal

denseopc_options opts {
    version 1
    ...
    mrc_rule external POLY { use $MINEXT}
    mrc_rule internal POLY { use $minInternal}
    scale $the_scale
}

setlayer L1 = denseopc POLY SBAR MAP POLY OPTIONS opts
*/]
```

Related Topics

[denseopc_options Keywords](#)

etech_imagegrid

Variable Etch Biasing (VEB) Commands

Defines etch grid size.

Usage

etech_imagegrid *etch_grid_microns*

Arguments

- ***etch_grid_microns***

A required argument that specifies the etch grid size.

Description

Defines the grid size to use for etch simulations only.

- If the Calibre OPCverify command **imagegrid** is explicitly specified, **etech_imagegrid** defaults to the value specified for **imagegrid**.
- If **etech_imagegrid** is specified, it overrides the supplied value for **imagegrid** for etch simulation (see **veb_simulate**) only.
- If neither **etech_imagegrid** nor **imagegrid** are specified, the grid value used for etch simulation is derived from the etch model's parameters.

Tip

 Smaller values of **etech_imagegrid** will slow down performance.

etch_model_load

Variable Etch Biasing (VEB) Commands

Loads a specified 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](#) command.

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 braces ({}) are required and must surround the inline model definition.

Description

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

Note

 Currently, there is a noticeable runtime performance penalty when using VEB models (especially for models using the direct visible kernel during image generation). VEB models should only be used for small critical areas of the layout. The performance penalty is small when used in the pre-OPC model-based retargeting flow, but becomes large when used with Calibre nmOPC or Calibre OPCverify. For this reason, use of visible kernels is specifically not recommended for full-chip use other than with the pre-OPC model-based retargeting flow.

Examples

```
etch_model_load veb1 etch_veb1.mod
```

flare_longrange

LITHO DENSEOPC Setup File Commands

Associates a pre-computed long range flare convolution with a flare model contained inside a litho model.

Usage

```
flare_longrange litho_model_name
  {none | {constant value} | {png [no_precheck] pngfile [png_options]}}
    [name name_string]
```

Arguments

- ***litho_model_name***

A required argument that specifies the name of file in a litho model that contains a flare model with long range flare. The model name must either be found in the modelpath or be a fully qualified path.

You must specify only one of the next three arguments to indicate the long-range convolution.

- **none**

No pre-computed long range convolution will be used when doing simulations using this model. This is the default behavior if the litho model contains a flare model, and **flare_longrange** is not specified.

- **constant *value***

Use a constant percentage of the clearfield transmission for long range flare. For example, a value of 0.06 adds 6% flare. In general the total intensity including constant flare is given by

$kf * I_{clear} * value$

where kf is the coefficient from the flare model, I_{clear} is the clearfield value taking into account both optics and DDM mask transmission, and **value** is the value supplied with this argument.

- **png [no_precheck] *pngfile* [*png_options*]**

A required argument set that specifies to use the long-range convolution pre-computed from the fractal kernel of the loaded model and the current layout. The convolution is stored in ***pngfile***, which may be in the current working directory, an explicit path or in the modelpath. The file is generated using RET FLARE_CONVOLVE and must be in PNG format.

The following additional options can be specified:

- **no_precheck**

An option that must be placed between **png** and ***pngfile*** when used.

If the **no_precheck** option is specified, the PNG file is not checked until the operation that uses it runs. This can be useful when the same SVRF file generates the

flare map and runs OPC. By default, the PNG file is verified when parsing the SVRF file.

- constant {average | minimum | maximum}

An option to the **png *pngfile*** argument; cannot be specified with delta *value*.

If the constant option is specified, it replaces flare values from the PNG file with a constant flare approximation whose value is computed directly from the flare map based on the setting:

average — Computes the average value in the flare map.

minimum — Returns the smallest value anywhere in the flare map.

maximum — Returns the large value anywhere in the flare map.

Note

 When the constant argument is used, flare is added to the entire simulation and does not consider the flare map's extent.

The values which are used by the “constant” option are seen by using the Calibre Workbench **getflare** command: “getflare range -pngin *png_file*”.

- delta *value*[%]

An option to the **png *pngfile*** argument; cannot be specified with constant.

If the delta option is specified, the values in the flare map are adjusted. If the specified *value* causes the flare value to fall locally below zero, it is truncated to zero at that point.

“delta *value*%” — The adjustment is by a percentage, up or down. The *value* must be greater than -100.

“delta *value*” (no %) — A constant offset, interpreted as a fraction of the clear field transmission, is added. The *value* must be greater than or equal to zero. It is calculated as in **constant *value***.

- flare_map_shift [-field] *x_um* *y_um*

An option to the **png *pngfile*** argument.

If the flare_map_shift option is specified, the flare map is shifted by the amounts specified for *x_um* and *y_um* in microns before being used in the run. Set this to the negative of any coordinate shifts applied to the layout before creating the flare map.

When -field is specified, the adjustment is relative to the EUV field center, specified by **euv_field_center**. By default, the adjustment is relative to the global coordinates stored in the flare map.

The default value for flare_map_shift is 0 0.

Note

 The flare_map_shift change is *only* used for flare maps. Other EUV field offsets (such as for through-slit models) must be specified separately.

- name *name_string*

An optional argument that defines a named instance of a flare_longrange command for use with setlayer image commands.

Named instances allow you to have multiple flare_longrange configurations in the same file, and are used by specifying “flare *name_string*” in the setlayer image command.

At least one non-named (anonymous) specification must be coded, because the nameless instance is used as the default flare_longrange settings.

Description

This command is only used if long range information is a part of the flare model contained within a litho model.

As of 2017.2, this command is optional. If not specified, the default is **flare_longrange litho_model_name none**.

Tip

 Variables can be used to substitute arguments to this command. For example, the following command runs flare_longrange on a contact layer with the arguments in the named variable \$FLARE_ARG:

```
eval flare_longrange contact1 $FLARE_ARG
```

Examples

```
flare_longrange new png new/flare.png
```

Related Topics

[flare_model_load](#)

[RET FLARE_CONVOLVE](#)

[Lithomodel \(Litho Model Format\)](#)

[Flare Model File Format \[Calibre WORKbench User's and Reference Manual\]](#)

[setlayer image \[Calibre OPCverify User's and Reference Manual\]](#)

flare_map_shift

LITHO DENSEOPC Setup File Commands

Specifies an amount to shift the flare map.

Usage

flare_map_shift x y

Arguments

- **x y**

A required pair of floating point numbers specifying the amount in microns to be added to the lower left corner of the flare map given in the png command. The default value is 0 0.

If the layout was shifted before creating the flare map, the values supplied for x and y should be the negative of that shift. Typically, flare_map_shift matches euv_slit_x_center but this is not required.

Description

This optional litho setup file command is used in conjunction with RET OPTIMIZE_EUV_HDB. The reason to shift flare maps is that the final GDS coordinates of the layout may not match precomputations, particularly if initial OPC was done on the chiplets before placement.

Each chiplet has its own coordinate system. When the chiplets have precomputed flare maps, it is simpler to shift the flare map than to shift the post-OPC chiplet.

Examples

`flare_map_shift -9.2 -1.8`

flare_model_load

LITHO DENSEOPC Setup File Commands

Loads the flare model file into Calibre nmOPC and assigns it the given label for later reference.

Usage

```
flare_model_load name {filename | '{' inline_model_definition '}'}
    longrange {none | png pngfile}
```

Arguments

- **name**
A required argument specifying the name to use for the flare model inside the Calibre OPCVerify command file.
- **filename | '{' inline_model_definition '}'**
A required argument defining the flare model file. If **filename** is specified, it is loaded from the specified location. If an **inline_model_definition** is specified, it must be enclosed in braces.
- **longrange {none | png pngfile}**
A required argument that specifies a location that contains the result of a pre-computed convolution of a fractal kernel (long range part) used by the loaded flare model and the currently processed layout.
 - **none** — No pre-computed long range convolution will be used when doing simulations using this model.
 - **png pngfile** — Uses the specified png file representing the long range flare convolution results, generated beforehand with RET FLARE_CONVOLVE.

Description

Note

 This command requires you to include the EUV keyword in the [LITHO EUV DENSEOPC](#) statement. It also requires an additional license (caleuv) to function.

Reads a flare model from the specified **filename** or defines it using the supplied **inline_model_definition**, assigning it to the specified **name** for later use.

If you have long range model information for the flare model within a litho model, you must use the [flare_longrange](#) command instead.

Examples

```
flare_model_load FM fmodel longrange png flare_1.png
```

Related Topics

[flare_longrange](#)

[flare_model](#)

[RET FLARE_CONVOLVE](#)

global_lithomodel_path

LITHO DENSEOPC Setup File Commands

Specifies the filepath to a global litho model's top *Lithomodel* file.

Usage

global_lithomodel_path *path*

Arguments

- *path*

A required argument specifying the path to the *Lithomodel* file of a global litho model. Paths can be absolute or relative to the run directory. They must terminate in “Lithomodel”.

Description

This command is required for the setup file for RET OPTIMIZE_EUV_HDB and invalid in all other setup files.

If the specified file is not the top *Lithomodel* of a global litho model, the run exits with the error

```
Failed to find at least one line of global lithomodel file with correct
format (litho_model sl* x <slit_region_center_in_microns>).
```

This message refers to the lines in a global litho model's file that identify subdirectories. For more information, see “[EUV Through-Slit Litho Model Extension](#)” in the *Calibre OPCverify User's and Reference Manual*.

Examples

```
global_lithomodel_path /models/EUV3/Lithomodel
```

layer

LITHO DENSEOPC Setup File Commands

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

Usage

```
layer layer_name [layer_type] [transmission] [mask_num dose]
```

Arguments

- *layer_name*

A required argument specifying the name of the layer. Calibre nmOPC uses this name in the rest of the setup file to refer to this layer. Typically, for Calibre nmOPC operations, you only need to specify the layer name. If a background command is defined (such as implementing Calibre OPCverify operations in the same setup file), then you must specify the layer type, transmission, and so on.

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

- *layer_type*

An optional parameter that specifies how the layer is used by Calibre nmOPC. The layer types are hidden or visible. Refer to “[layer](#)” in the *Calibre OPCverify User’s and Reference Manual* for further information on the layer types.

- *transmission*

An optional argument for visible layers that defines the mask layer’s optical transmission type. This is typically ignored by Calibre nmOPC, but used in cases where the input layer is used for a non-correction operation (such as Calibre OPCverify).

The layer’s optical transmission type must be the complement of the background, that is, you cannot specify clear features if you specify a clear background. Refer to the Calibre OPCverify command for more information about physical layer transmission in relation to background.

- *mask_num dose*

An optional parameter pair used with phase-shifting masks to allow the simulators to support multiple masks. This is only used for Calibre OPCverify operations.

- *mask_num* — The number of the mask to which the layer belongs.
- *dose* — The layer’s light energy dose, expressed as a percentage of the full energy dose for the mask.

Description

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

Note

 The order of layer statements in the setup file is the order the designated layers are used in LITHO DENSEOPC SVRF calls. This has the advantage of giving you reuse capabilities with different legacy SVRF files.

Also, when using the layer command with a litho model, do not map opc layers to mask layers.

Examples

Declares a layer called SHAPES:

```
layer SHAPES
```

Related Topics

[background \[Calibre OPCverify User's and Reference Manual\]](#)

[layer \[Calibre OPCverify User's and Reference Manual\]](#)

[layer \(for denseopc_options Block\)](#)

lintmodel

LITHO DENSEOPC Setup File Commands

Loads a file with custom lint checks.

Usage

lintmodel *lint_checks_file_name* *setup_name*

Arguments

- ***lint_checks_file_name***

A required keyword that specifies the name of the custom lint check file. The file defines checks to run with the [lint](#) command. This file's location should be added to your [modelpath](#). See “[Lint Check File Format](#)” on page 135 for the file syntax.

The pound sign (#) is the comment character.

- ***setup_name***

A required keyword that references the name of the setup file block inside the lint checking file.

Description

The **lintmodel** keyword allows you to add custom [lint](#) checks to verify your OPC recipes by creating a command block known as a lint checking setup file. The lint checking file can then be loaded into a Calibre nmOPC setup file.

This capability allows you to add your own specifications that can be tested using the [lint](#) command, including updated best practices and other changing guidelines. You can also overwrite any existing lint configuration file by pointing to an alternate configuration file that matches your mask layer and technology node.

Lint Check File Format

Lint check files are used only by the **lintmodel** command. The file should be in one of the directories specified by [modelpath](#).

Format

A text file containing one or more setup blocks with following format:

```
setup setup_name
[include setup_name_2]
[[CRITICAL] parameter_name [constraint] ]...
```

Text following a # is ignored.

For example, this defines two setup blocks:

```
#Overrides only the default fragment_max check
setup frag_check
fragment_max >0.050 <=0.2

#Overrides fragment_max and tilemicrons checks
setup both_checks
include frag_check
tilemicrons = 25
```

Parameters

- **setup *setup_name***

A required keyword that specifies the name of a block of checks. This name is used both by the lintmodel ***setup_name*** argument and the lint check's include keyword.

- **include *setup_name_2***

An optional keyword that includes the contents of another block of checks when ***setup_name*** is run. Both blocks must be in the same lint check file.

- **[CRITICAL] *parameter_name* [*constraint*]**

An optional line that specifies a check. A setup block can include any number of checks. The *parameter_name* can be any of the denseopc_options keywords, but keywords not listed in the “Custom Check Name” column of [Table 4-8](#) must be followed by a constraint.

- **CRITICAL** — An optional keyword that causes the run to exit with an error if the check fails. If “CRITICAL” is not used, failed checks generate a warning.
- ***constraint*** — An optional argument that overrides the default check for the specified *parameter_name*. Constraints can take one of three forms:
 - ***eq_neq_in value*** — Creates an equality or non-equality lint check. Use = or != for *eq_neq_in*. For example:

```
tilemicrons = 25
```

- ***cmp_value1 cmp_value2*** — Checks if a value falls within a specified range. Operators are <, <=, >, >=. For example:

```
fragment_max >0.050 <= 0.20
```

- ***in value1 | value2 [| value3]...*** — Checks for specific values. Values should not include the characters != <> | \\" unless enclosed in quotation marks. For example:

```
algorithm in 1 | 2
```

If there is no expression specified after the *parameter_name*, the default check is used.

Table 4-8. Custom Lint Checks

Custom Check Name	Default Warning
tilemicrons	“tilemicrons” should be set to at most 35 microns for 32nm technology, and at most 50 microns for 45nm technology.
image_dose pw_image_dose	“image”, “pw_condition”: At least one dose is set to less than 0.95 or more than 1.05.
max_opc_move	“max_opc_move”: The default maximum inside/outside edge movement is 0.08. You should decrease it, if possible, to get a speedup.
epe_spacing	“epe_spacing”: epe_spacing is too large. It should be smaller than half the minimum fragment length.
step_size	“step_size”: step size is more than 0.5 nm. Decrease it to get more consistent opc output.
feedback	“feedback”: The feedback is likely too high or too low. The suggested range is -0.8 to -0.05.
small_fragment	Some fragments can be smaller than 2*nyquist, according to fragmentation parameters. This might be suboptimal.
fragment_max	“fragment_max”: Maximum fragment length is set to <value>. It should be set to no more than <value> to avoid an increase in the interaction distance.
jog_ignore_metric	“jog_ignore_metric”: recommended setting is in the range nyquist/4 to nyquist/2.
jog_freeze	“jog_freeze”: recommended setting is in the range nyquist/4 to nyquist/2.
mrc_constraint	At least one MRC_RULE constraint for layer \"%s\" is greater than the minimum fragment length %f. This can cause unpredictable OPC corrections.

Examples

If the lint check file is named *models/denseOPC/models/lint.chk*, Calibre nmOPC would load it using these lines:

```
modelpath ./models/denseOPC/models
lintmodel lint.chk metal_lint.setup
```

The *lintmodel* line calls the setup block “metal_lint.setup”. If the content of *lint.chk* is as shown below, the *metal_lint.setup* checks check tilemicrons, fragment_min, fragment_max, and algorithm. It does not include max_opc_move.

```
setup "m1_lint_setup"
    include "metal_lint.setup"
    max_opc_move >0.050 <0.10

setup metal_lint.setup
    include lint.setup
    CRITICAL fragment_min >=0.020 <=0.040
    fragment_max >=0.051 <=0.2
    algorithm in 1 | 2

setup lint.setup
    tilemicrons =25
```

ml_featurevector_load

LITHO DENSEOPC Setup File Commands

ML OPC only

Loads the model metadata.

Usage

ml_featurevector_load *name file*

Arguments

- ***name***
A required argument specifying the name to use when referring to this model.
- ***file***
A required argument specifying the name of the file that contains metadata about the model.

Description

The optional **ml_featurevector_load** command loads metadata for a neural net model. To associate the metadata with a particular model, the same name must be specified for **ml_featurevector_load** and the [ml_model_load](#) command, but association is not required.

Each metadata file can be associated with at most one model. A single setup file may load multiple models and metadata files.

If you receive the following error, either the directory containing the file is not in the **modelpath** or the **ml_featurevector_load** command is not in the setup file:

```
Unsupported nlitho modelType
Could not load nlitho model properly!
```

Examples

The following example loads a model and associates the metadata with it by naming both “nn1.” The code also creates a second name, “nn2,” for the same metadata file.

```
LITHO FILE ml_opc.in [ /*
    processing_mode flat
    modelpath models

    ml_model_load      nn1 mxopc_v54/saved_model/saved_model.pb
    ml_featurevector_load nn1 mxopc.mod

    ml_featurevector_load nn2 mxopc.mod
    ...
*/ ]
```

ml_model_load

LITHO DENSEOPC Setup File Commands

ML OPC only

Loads the machine learning model.

Usage

ml_model_load *name* *filepath*

Arguments

- ***name***
A required argument specifying the name to use when referring to this model.
- ***filepath***
A required argument specifying the filename or relative path of the machine learning model.
The model can be in the current directory or the modelpath.
If no filename is given, `ml_model_load` searches for a file named `frozen_model.pb`. At a minimum, the command must specify the directory.

Description

The `ml_model_load` command loads the machine learning model. A machine learning model contains a version of the trained neural network in `.pb` format. The default name is `phv0.pb`.

Metadata for the model is stored separately and is loaded by the [ml_featurevector_load](#) command.

This command can appear more than once in the setup file, but each instance must use a different name.

Examples

This example loads the model `phv0.pb`. The path is relative to the models directory. Other commands in the setup file (for example, `image`) can refer to it by “`nn1`”.

```
modelpath models
ml_model_load      nn1 mxopc_v54/saved_model/phv0.pb
```

modelpath

LITHO DENSEOPC Setup File Commands

Specifies the 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
```

optical_model_load

LITHO DENSEOPC Setup File Commands

Specifies an optical model.

Usage

optical_model_load *optical_model_name* {*filename* | '{' *inline_model* '}'}

Arguments

- ***optical_model_name***

A required argument specifying the name that you will use to refer to this optical model in all simulation commands.

- ***filename***

A required argument (you must specify either ***filename*** or ***inline_model***) specifying a filename to load containing the optical model. If you do not specify a filename, an inline optical model must be supplied instead.

- **'{ ' *inline_model* '}'**

A required argument (you must specify either ***filename*** or ***inline_model***) defining an optical model. The entire optical model specification must be enclosed in braces.

Description

This is a required setup file parameter. At least one **optical_model_load** command must always be present. Multiple optical models may be loaded and used.

This command loads an optical model found in ***filename***, or defined by specifying model text inline inside braces (**{}**). The optical model is given a user-defined ***optical_model_name*** which you use with simulation operations.

An alternate method of loading models through a litho model is available. Refer to “[Load Model Information With a Litho Model](#)” on page 48 for more information.

Examples

```
optical_model_load opticsfile poly65
```

output_text_filepath

LITHO DENSEOPC Setup File Commands

EUV Command

Specifies a filename for the encrypted SVRF file generated by RET OPTIMIZE_EUV_HDB.

Usage

output_text_filepath *path*

Arguments

- *path*

A required argument specifying the filename for the encrypted SVRF file that is generated by RET OPTIMIZE_EUV_HDB. If no directories are specified, the file is created in the directory from which the run is invoked.

The value of ***path*** is treated as a string. Environment variables are not supported.

If you specify a directory that does not exist, the run exits with the error

RET OPTIMIZE_EUV_HDB: expected a valid path for 'output_text_filepath'.

Description

This optional command allows you to specify a location and name for the file generated by RET OPTIMIZE_EUV_HDB. If the command is not specified, the file is written to the directory in which the run was started and is named *optimize_euv_hdb.out*.

Examples

`output_text_filepath output/optEUV.encrypted`

Related Topics

[RET OPTIMIZE_EUV_HDB](#)

png

LITHO DENSEOPC Setup File Commands

EUV Commands

Specifies the path to a PNG file created by RET FLARE_CONVOLVE.

Usage

png *path*

Arguments

- *path*

A required argument specifying the absolute or relative path to a PNG file containing flare information. Typically, this file is part of the litho model but that is not required.

Examples

```
png Models/new/flare_map_ret.png
```

Related Topics

[RET FLARE_CONVOLVE](#)

resist_model_load

LITHO DENSEOPC Setup File Commands

Specifies a resist model.

Usage

```
resist_model_load resist_model_name {filename | '{' inline_model '}'}
```

Arguments

- ***resist_model_name***

A required argument specifying the name that you will use to refer to this resist model in simulation commands.

- ***filename***

A required argument (you must specify either ***filename*** or ***inline_model***) specifying a filename to load containing the resist model. If you do not specify a filename, an inline resist model must be supplied instead.

- '{' *inline_model* '}'

A required argument (you must specify either ***filename*** or ***inline_model***) defining a resist model. The entire specification must be enclosed in braces.

Description

Loads the resist model contained in the file ***filename***, or defined by specifying model text inline inside braces (**{}**). The resist model is assigned the user-defined ***resist_model_name***, which you reference in the image command.

An alternate method of loading models through a litho model is available. Refer to “[Load Model Information With a Litho Model](#)” on page 48 for more information.

Examples

```
resist_model_load resistfile poly65.mod
```

setlayer curve

LITHO DENSEOPC Setup File Commands

Subset of the setlayer command

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 OPC.

- 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`. If `jog_tol` and `jog_adj` are not set, all line segments are considered edges.
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`, `jog_extra`, and `new_cp`. 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 OPC, 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 OPC results without using the smoothed target.
2. Use post-OPC simulation contour to estimate the amount of residual corner rounding.
3. Adjust curve parameters to match the existing contour.
4. When done, compare OPC results with and without smoothed input. The results should show improved OPC, though some fragment length adjustment may help.

Recommended settings for the `setlayer curve` command can be found in the “[Calibre nmOPC Best Practices](#)” chapter.

Warnings

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
```

Examples

Example 1

The following example creates a smoothed target layer:

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

Example 2

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

```
setlayer smooth = curve target order 6 cpdist 0.070 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
```

Example 3

This new_cp argument inserts a control point along an edge that is longer than 250 nm with a convex corner on one side formed with an edge less than 15 nm (length1), and a concave corner on the other side formed with an edge also less than 15 nm (length2). The new control point is placed 10 nm from the convex corner.

```
new_cp convex concave length > 0.250 length1 < 0.015 length2 < 0.015 \
offset 0.010
```

Related Topics

[Optimize Results From setlayer curve](#)

setlayer curve_target

LITHO DENSEOPC Setup File Commands

Subset of setlayer command

Creates a target that mimics the final image.

Usage

```
setlayer output_layer_name = curve_target input_layer_name [criticalDistance value]  
[cornerRadius value] [cornerRadiusConcave value] [cornerRadiusConvex value]  
[stepRadiusConcave value stepRadiusConvex value] [minStepTanDist value]  
[jogRampConcave value jogRampConvex value] [colinearAngleTolerance degrees]  
[lineEndRatio value] [linearRatio {true | false}] [spaceEndRatio value]  
[lineEndWidth value] [spaceEndWidth value] [maxJog value]  
[roundCorner {true | false} [roundedCornerRadius value]]  
[preferMid {true | false} [midSpanExt value] [maxMidSpan value]] [enclose layer by value]  
[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 value***

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 value***

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** *value*

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.

- **stepRadiusConcave** *value* **stepRadiusConvex** *value*

An optional pair of arguments that allow setting a smaller value than cornerRadius for S-steps. (An S-step is a concave corner and a convex corner separated by a single fragment.) You cannot use stepRadiusConcave and stepRadiusConvex separately.

The stepRadius arguments are ignored for overlapped S-steps when roundCorner true is set.

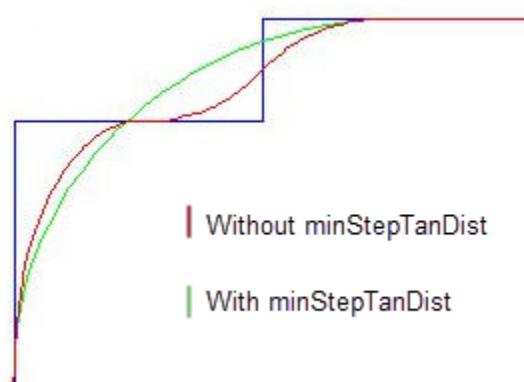
The setting for both arguments must be in the range defined by the minimum and maximum of cornerRadius, cornerRadiusConcave, and cornerRadiusConvex. If they are set to a value outside the range, the run issues a warning and uses the nearest end of the range.

- **minStepTanDist** *value*

An optional argument that sets a minimum tangent distance for both legs of a step shape. If the tangent distance on either side is not met, the step is smoothed. In effect, it defines the minimum curvature.

The default value is 0, no required minimum distance.

Figure 4-2. Effect of minStepTanDist



- **jogRampConcave** *value* **jogRampConvex** *value*

An optional pair of arguments that allow setting different values for the distance between the tangent point and the jog corner for concave and convex jog corners. (Whether an edge segment is a fragment or a jog is controlled by maxJog.) You cannot use jogRampConcave and jogRampConvex separately.

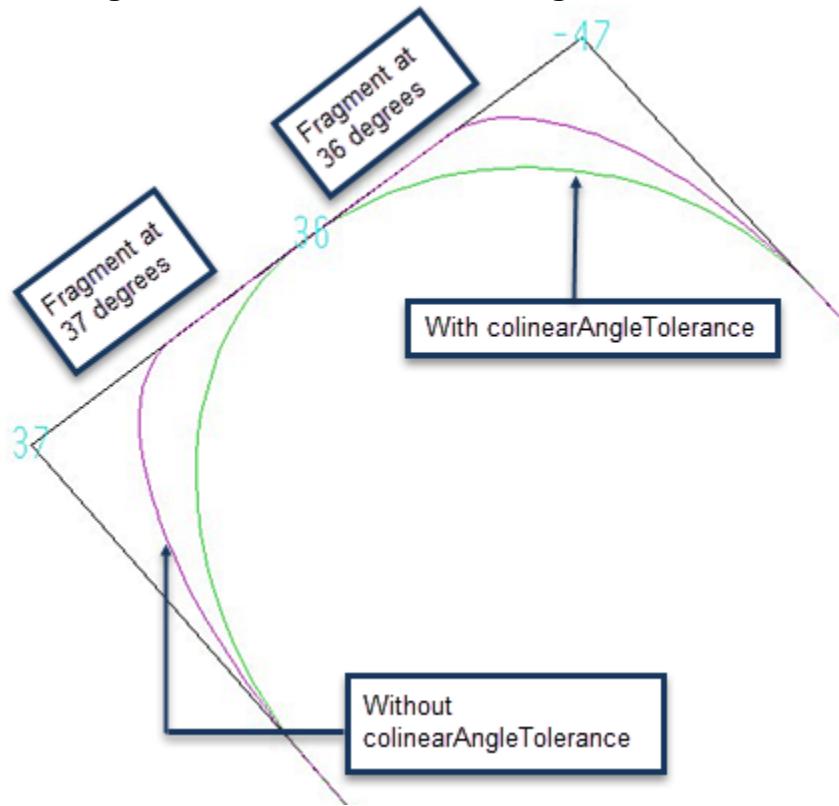
The setting for both arguments must be in the range defined by the minimum and maximum of cornerRadius, cornerRadiusConcave, and cornerRadiusConvex. If they are set to a value outside the range, the run issues a warning and uses the nearest end of the range.

- **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 4-3. Effect of colinearAngleTolerance 1



- **lineEndRatio *value***

Allows for the specification of an elliptical curve at line ends. The curve is tangent to the side at a distance *value* from the corner (where *value* 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.

Because of the layout resolution, the last segment of the elliptical curve may be collinear with the side, as in the right curve in [Figure 4-4](#). The end of the ruler (shown by the box with 0.0520) appears to extend past the curve and into the flat portion. The internal calculations

are done using floating point numbers and then rounded to integers to give coordinates in dbus.

- linearRatio {true | false}

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

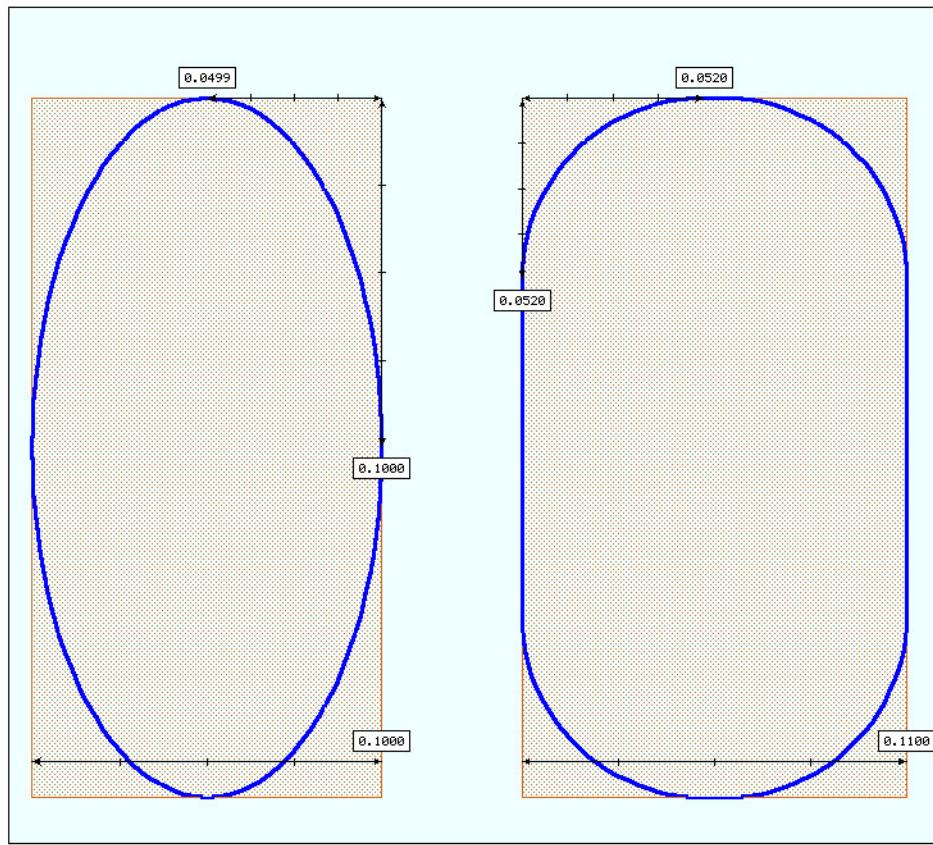
- For the line ends smaller than lineEndWidth (or if not defined, smaller than $2 * \text{cornerRadius}$):
 - On the line end (LE) fragment, the tangent point of the curve is set at the middle of the LE fragment.
 - On the LE adjacent fragment, the tangent point of the curve is set at a distance specified 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 4-4 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 4-4. Curvature Example



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

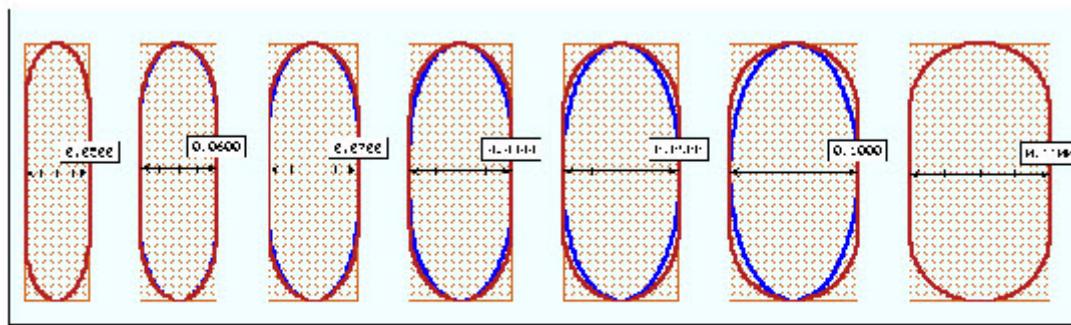
- No adjustment occurs for line ends smaller than cornerRadius.
- For the line ends larger than cornerRadius but smaller than $2 * \text{cornerRadius}$, the tangent point of the curve is set at a distance specified by that particular (LE CD) * lineEndRatio * multiplier.
- The multiplier is an additional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

Figure 4-5 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 4-5. 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 $(\text{LE CD}) * \text{lineEndRatio} * \text{multiplier}$.
 - 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.

- **spaceEndRatio *value***

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.

- **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 $2 * \text{cornerRadiusConcave}$.

- **maxJog *value***

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 *value* are user units. The default value is approximately $0.292893 * \text{radius}$, where *radius* is the minimum of *cornerRadiusConcave*, *cornerRadiusConvex*, and *cornerRadius*.

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

An optional parameter that causes Calibre nmOPC 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 is may be a more appropriate fit than curves that are typically used. The default value for this parameter is false.

- **midSpanExt *value***

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 (*value* 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 *value***

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).

- **enclose *layer* by *value***

An optional flag that allows the target to be drawn around contacts and vias, where *value* specifies the amount of enclosure or internal distance requested for OPC. This is similar to contact enclosure as specified by the [wafer_enclose](#) command, but the target itself will enclose the contact instead. This can be useful if the target is being used outside of Calibre nmOPC.

- taglayer *layer overrides*

An optional keyword set that specifies a marker layer (*layer*) and setting overrides for fragments that interact with *layer*. The following arguments can appear as *overrides*:

cornerRadiusConcave	cornerRadiusConvex	stepRadiusConcave
stepRadiusConvex	colinearAngleTolerance	lineEndRatio
linearRatio	spaceEndRatio	lineEndWidth
spaceEndWidth	roundCorner	roundedCornerRadius
preferMid	midSpanExt	maxMidSpan

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 OPC results.

Examples

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

Related Topics

[target_curve](#)

setlayer denseopc

LITHO DENSEOPC Setup File Commands

Subset of setlayer command

Creates and derives DENSEOPC target layers.

Usage

```
setlayer output_layer_name = denseopc input_layer [input_layer...] MAP name OPTIONS  
          opt_name [CHIPLET_PLACEMENT index] [property '{' 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***

A list specifying the name of the input layer(s). At least one layer must be specified. Each of the layers specified in this list must be a valid Calibre layer containing polygon data.

- **MAP *name***

An 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 *opt_name***

A required option that specifies the name of a [denseopc_options](#) sub-command block of LITHO DENSEOPC setup file commands defined inside a LITHO FILE parameter block.

Refer to the section “[denseopc_options Keywords](#)” for a complete list of keywords supported for a [denseopc_options](#) sub-command block.

- **CHIPLET_PLACEMENT *index***

An optional keyword and value used with the EUV chiplet reuse flow. See “[Comparison of EUV Flows](#)” on page 682.

- **property '{' *name* '}'**

An optional keyword and value used to pass numeric values attached to a layer by an appropriately constructed [OUTPUT_SHAPE fragment](#) statement. If there are multiple properties, each name must be on a separate line, as described in “[Property Block](#)” in 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 DENSEOPC 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 DENSEOPC.

The denseopc_options command is required in order to call a setlayer denseopc command.

Examples

Example 1: Basic Syntax

The following example runs Calibre nmOPC using two layers, M1 and sraf, with the settings in the OPTS denseopc_options block. The derived layer M1_opc receives the M1 results; the layer M1 itself is unchanged at this scope even after OPC.

```
setlayer M1_opc = denseopc M1 sraf MAP M1 OPTIONS OPTS
```

Example 2: Multiple Properties

In the following example, layer M1_width_space contains the properties x, width, and space. To write the properties to the Calibre nmOPC output, the setlayer denseopc command would be formatted as follows:

```
setlayer output_layer = denseopc M1 sraf MAP M1_width_space OPTIONS OPTS \
    property { x
        width
        space
    }
```

(The entry for “x” could also be on a line of its own.) Property blocks, unlike the rest of the command, should not use a line continuation character (\).

To write output_layer properties to an OASIS file, use a rule check and be sure to include the PROPERTIES keyword:

```
output_layer {COPY output_layer} DRC CHECK MAP output_layer 300 PROPERTIES
```

To write the values to an RDB, use a DFM RDB as follows:

```
dfm_rdb { DFM RDB output_layer "rdb_name.dfmdb" }
```

Related Topics

[denseopc_options](#)

setlayer layer_simplify

LITHO DENSEOPC Setup File Commands

Subset of setlayer command

Creates a curve-like layer that can receive correction from curvilinear_opc.

Usage

```
setlayer output_layer_name = layer_simplify input_curve_layer [max_deviation value]  
[min_segment_length value]
```

Arguments

- *output_layer_name*

A required argument 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 is a valid Calibre layer containing polygon data.

- *input_curve_layer*

A required argument specifying the name of the input layer. This layer is typically an original layer from the layout and contains curvilinear shapes. Only one input layer can be specified.

- *max_deviation value*

An optional argument specifying the maximum allowed difference between the shape on the input layer and the simplified geometry on the output layer. The value is in microns and must be greater than 0.

The default is 0.005 microns.

- *min_segment_length value*

An optional argument specifying the shortest possible segment in microns. Segments are similar to traditional OPC's fragments. The algorithm attempts to intersperse minimum length segments between any pair of longer segments.

The default is 0.02 microns.

Description

The optional setlayer layer_simplify operation performs orthogonal geometrical simplification on the input layer. It represents curved shapes as a series of segments. The segments differ from the traditional results of fragmentation in that other features do not generate fragmentation points, nor is there a maximum segment length.

Examples

This creates a layer that can receive OPC corrections from curvilinear_opc:

```
setlayer for_opc = layer_simplify photonic_device_layer
```

Related Topics

[curvilinear_opc](#)

setlayer tilegen

LITHO DENSEOPC Setup File Commands

Subset of setlayer command

Creates a layer to display tile and cell boundaries.

Usage

```
setlayer output_layer_name = tilegen value
  [property '{'
    location
    remote
    lvheap_max
    lvheap_current
    rss_max
  '}]
```

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 used in subsequent processing for the operation to occur.

- ***value***

A required numeric argument that specifies the style of output. The value specified must be between -0.1 and 0.1 microns. The tile is sized by value and written to ***output_layer_name*** as polygons.

- -0.1 to < 0 microns — The tile is sized by value and written to ***out_layer*** as polygons.
- 0 to 0.1 microns — The boundaries between valid tile regions are sized, and the boundaries are written to ***output_layer_name***.

- **property**

An optional argument that enables tile information to be output as properties. The list of properties must be enclosed in braces ({}).

Note

 When using a property block, the ***value*** must be negative.

The properties include the following:

- **location** — Outputs the geographical location of the tile as a property. The information may include the cell name, tile index, or coordinates.
- **remote** — Specifies the host name of the machine that processes the tile.

- lvheap_current — Specifies (in megabytes) the LVHEAP used (not currently allocated, which may be more) at the time of executing the command.
- lvheap_max — Specifies (in megabytes) the LVHEAP peak at the time of executing this command.
- rss_max — Specifies (in megabytes) the Resident Set Size (RSS) peak at the time of executing this command.

This argument is used primarily for debugging purposes; the RDB output can potentially be very large for a full-chip run.

Description

When performing optical process correction, Calibre nmOPC processes the data in chunks, called tiles. This command creates a supplementary output layer that displays the tile and cell boundaries used when processing your layout.

The output layer contains polygons representing every other tile. Thus, when you display the output layer using a fill pattern, you see a checkerboard pattern.

Using the “properties” keyword, you can also output tile information as properties to track a remote’s name and memory for each tile. Note that this is used primarily for debugging purposes.

Examples

The following example displays tile boundaries as a grid:

```
setlayer t1 = tilegen 0
```

The following example displays the tiles with 0.1 microns between edges:

```
setlayer t1 = tilegen -0.05
```

The following is an example of using the property argument to aid debugging:

```
litho file my_opc_setup [
    processing_mode flat
    tilemicrons 30 exact
    ...
    setlayer opc_out = denseopc ...
    setlayer tiles = tilegen -0.010 property {
        location
        remote
        lvheap_current
        lvheap_max
        rss_max
    }
]
tile = LITHO DENSEOPC ... MAP tiles FILE my_opc_setup
tile_oas { COPY tile } DRC CHECK MAP tile 20
tile_rdb { DFM RDB tile tiles.rdb ALL CELLS CHECKNAME "tile_info" }
```

The properties output could like the following:

```
...
p 1 4
cell gh1347
tile 10
remote b1
lvheap_current 3.38393
lvheap_max 101.775
rss_max 196
-8272 -8272
-400 -8272
-400 -400
-8272 -400
p 2 4
...
```

setlayer veb_retarget

Variable Etch Biasing (VEB) Commands

Subset of the setlayer command

Creates and derives layers for VEB retargeting.

Usage

```
setlayer output_layer_name = veb_retarget input_etch_layer [optional_layer] MAP name  
OPTIONS opt_name
```

Arguments

- ***output_layer_name***

Specifies an output layer name for the results of the operation. Represents the OPC pre-etch (or resist) target. This named layer must be used as the MAP argument when you program the SVRF rule file.

- ***input_etch_layer***

Specifies input layer to be retargeted. Represents the final post-etch OPC target.

- ***optional_layer***

Specifies an optional visible layer used for the OPC calculations.

- **MAP *name***

Specifies a layer ***name*** that contains the output to be returned.

- **OPTIONS *opt_name***

Specifies the name of an options sub-command block specified by denseopc_options. The ***opt_name*** block supports a subset of Calibre nmOPC options and some options unique to veb_retarget, as shown in the following table.

Table 4-9. Keywords Accepted in veb_retarget OPTIONS Block

Keyword	Works in setlayer denseopc options	Works in setlayer veb_retarget options	Comments
background (for denseopc_options block)	yes	yes	Required, but the information is not used for VEB bias calculation.
check_movement	yes	yes	Same as the Calibre nmOPC control.
corner_control	yes	yes	Same as the Calibre nmOPC control.
etch_model	no	yes	Required.

Table 4-9. Keywords Accepted in veb_retarget OPTIONS Block (cont.)

Keyword	Works in setlayer denseopc options	Works in setlayer veb_retarget options	Comments
feedback	yes	yes	Works the same way as Calibre nmOPC, except that etch corrections are much more linear than optical or resist corrections. The feedback factor used for VEB retargeting is typically larger than the feedback used for OPC.
fragment_corner	yes	yes	Same as the Calibre nmOPC control.
fragment_inter	yes	yes	Same as the Calibre nmOPC control.
fragment_island	yes	yes	Same as the Calibre nmOPC control.
fragment_layer	yes	yes	Same as the Calibre nmOPC control.
fragment_max	yes	yes	Same as the Calibre nmOPC control.
fragment_min	yes	yes	Same as the Calibre nmOPC control.
fragment_preserve_length	yes	yes	Same as the Calibre nmOPC control.

Table 4-9. Keywords Accepted in veb_retarget OPTIONS Block (cont.)

Keyword	Works in setlayer denseopc options	Works in setlayer veb_retarget options	Comments
image	yes	yes	The image command is never executed and is only used to compute an optical radius (equal to the Nyquist value) to set defaults for fragmentation parameters. DDM models are also ignored and the pattern keyword is not supported. If you use a Litho model, do not use the image command for corner-chopping along with “mask_layer CC/CX” in the Litho model. Instead, use the mask_chip_corner setup file command for veb_retarget runs only.
layer	yes	yes	Required. Only the layer types opc, visible, and island are supported. VEB etch bias calculations accounts for both opc and visible layers, but corrections are only applied to opc layers.
mask_chip_corner	yes	yes	Same as the Calibre nmOPC control.
max_iterations	yes	yes	Same as the Calibre nmOPC control.
max_iter_movement	yes	yes	Same as the Calibre nmOPC control.
max_opc_move	yes	yes	Same as the Calibre nmOPC control.
mrc_rule	yes	yes	Same as the Calibre nmOPC control.
no_opc_region	yes	yes	Same as the Calibre nmOPC control.

Table 4-9. Keywords Accepted in veb_retarget OPTIONS Block (cont.)

Keyword	Works in setlayer denseopc options	Works in setlayer veb_retarget options	Comments
opc_grid_multiplier	yes	yes	Same as the Calibre nmOPC control.
scale	yes	yes	Optional.
step_size	yes	yes	Same as the Calibre nmOPC control.
veb_pseudo_nyquist	no	yes	Used only for VEB to set up automatic fragmentation.
version	yes	yes	None.

In addition, all Tcl customization scripting commands documented in the “[Calibre nmOPC Custom Tcl Scripting Reference](#)” chapter are supported in the VEB setup file.

Description

The veb_retarget command takes an input layer representing the OPC target after etching. It is biased by the value of VEB, multiplied by the feedback value specified in the denseopc_options block. The results are output into the layer name specified by the MAP keyword.

Note

-  When using a litho model with setlayer veb_retarget, the litho model may not use any geometry parameters for the mask_layer parameter (CC, CX, XSHIFT, YSHIFT, BIAS, XBIAS, YBIAS).
-

Examples

```
setlayer ret_output = veb_retarget RETARGET_ME MAP RETARGET_ME \
OPTIONS OPTS
```

Related Topics

[Load Model Information With a Litho Model](#)

[Practices to Improve VEB Results](#)

simulation_consistency

LITHO DENSEOPC Setup File Commands

Enables the simulation engine to perform actions to improve consistency.

Usage

```
simulation_consistency {0 | 1 | 2 | 3 | euv}
```

Arguments

- **0 | 1 | 2 | 3 | euv**

A required argument specifying the type of adjustment to use.

0 — No adjustments made. This is the default.

1 — Adjusts certain settings to improve output consistency.

2 — Adjusts settings to simulate more quickly than when set to 1, but still provide better output consistency than 0. This setting is recommended for DUV processes.

3 — Adjusts settings for improved output consistency and accuracy compared to 2, but runs slightly slower.

euv — Adjust settings for better output consistency with EUV processes.

Description

The simulation_consistency option improves OPC output consistency by adjusting several simulation engine settings.

When simulation_consistency is set to any value other than 0, the setup file may not contain final_upsample, imagegrid, or rasterizer_upsample_factor settings.

Examples

```
simulation_consistency 2
```

Related Topics

[final_upsample \[Calibre OPCverify User's and Reference Manual\]](#)

[imagegrid \[Calibre OPCverify User's and Reference Manual\]](#)

[rasterizer_upsample_factor \[Calibre OPCverify User's and Reference Manual\]](#)

svrf_var_import

LITHO DENSEOPC Setup File Commands

Imports one or more variables previously defined in the SVRF rule file into the LITHO DENSEOPC block.

Usage

svrf_var_import var1 ... varN

Arguments

- **var1...varN**

Specifies SVRF variables to be imported into the setup file.

Description

Imports each of the listed variables from SVRF (defined using the VARIABLE command), which creates Tcl variables of the same name in the global scope of the setup command block. If a listed variable name is not defined in SVRF, it will result in an error. If a listed variable name already exists in the setup, this will override the variable value.

Examples

In this example, variable substitution from standard Tcl variables and from SVRF is performed.

```
VARIABLE minInternal 0.06
POLY_OPc = LITHO DENSEOPC POLY SBAR FILE dopc_setup MAP L1
LITHO FILE dopc_setup /* ...
...
set MINEXT 0.010
set the_scale 1.1
svrf_var_import minInternal
denseopc_options opts {
version 1
...
mrc_rule external POLY { use $MINEXT}
mrc_rule internal POLY { use $minInternal}
scale $the_scale
}
setlayer L1 = denseopc POLY SBAR MAP POLY OPTIONS opts
*/]
```

veb_simulate

Variable Etch Biasing (VEB) Commands

Performs an etch bias simulation on an input contour layer.

Usage

veb_simulate *layer* **etch_model** *veb_model*

Arguments

- *layer*

A required argument. This layer should contain the resist contour before etch biasing.

- *veb_model*

A required argument specifying the name of the VEB etch model to use for the simulation.

Description

This command performs an etch bias simulation that is used for model-based retargeting verification and VEB model verification purposes. The etch model being used should have been previously loaded with an [etch_model_load](#) command.

Examples

```
setlayer etch_contour = veb_simulate resist_contour etch_model etch_mod
```

denseopc_options Keywords

The OPTION *opt_name* argument in the setlayer denseopc command 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.

The following figure illustrates the location of the denseopc_options block inside a setup file.

Figure 4-6. Setup File Location for denseopc_options Commands

```
m1_out = LITHO DENSEOPC met1 sraf
FILE m1_opc MAP m1_opc
LITHO FILE m1_opc /*

modelpath ./models
optical_model_load opt m1.opt
resist_model_load res m1.cml

layer m1 hidden clear
layer sraf hidden clear

denseopc_options m1_opt {
    version 1
    background poly opc atten 0.06
    layer m1 opc clear
    layer sraf visible clear
    image optical opt resist res
}

NEWTAG all POLY -out A
NEWTAG edge A len < 0.3 corner1 convex corner2 convex
    -out line_end
NEWTAG neighbor both line_end len < 0.4 corner convex \
    -out line_end_adj
...
OPC_ITERATION 8
NEWTAG all POLY -out A

}

setlayer m1_opc = denseopc m1 sraf MAP m1 OPTIONS m1_opt
```

Refer to the [setlayer denseopc](#) and [denseopc_options](#) commands for further descriptions.

Note

 Do not use [LITHO DENSEOPC Setup File Commands](#) in a denseopc_options block. They will generate an error.

Table 4-10. denseopc_options Summary

Keyword	Description
active_layer	Creates intersecting hidden layers used to define gates.
adjust_moved_target_sites	Adjusts dense sites after retargeting with target_curve.
adjust_target_control_sites	Determines how dense sites are adjusted for target control.
algorithm	Bypasses contour creation to improve output consistency.
allow_single_site_correction	Allows jog fragments to be corrected.
background (for denseopc_options block)	Defines background imaging characteristics.
check_movement	Enables/disables OPC edge movement constraints.
chiplet_placement_flare_tolerance	Specifies a threshold for the difference in flare values that prevents re-using OPC results for different placements of a chiplet (large subassembly or block).
corner_control	Sets the length of the first fragment next to corners.
displacement_limit_mode	Specifies whether to use the most recent or the most restrictive DISPLACEMENT setting.
curvilinear_opc	Activates curvilinear OPC. Use with LITHO CL DENSEOPC.
effective_epe_regulation	Regulates effective EPE calculations per OPC iteration.
enhance_corner_to_corner_site_pairing	Enables an enhanced site placement algorithm for improved site pairing.
epe_spacing	Sets spacing between EPE or image samples along fragments.
etch_model	Loads etch model to be used by setlayer veb_retargt.
feedback	Automatic convergence controller command. Sets feedback values for all fragments, unless overridden by FEEDBACK.
feedback_balance (Deprecated in 2016.2)	Optimizes feedback to improve convergence.
feedback_mode	Sets precedence for conflicting definitions of FEEDBACK for fragments that are in more than one tag set.
flare_model	References the flare model.
fragment_coincident	Copies fragmentation from one layer onto another where edges are coincident.
fragment_consistency_mode	Controls how fragments generated by fragment_max are promoted around tile boundaries.
fragment_corner	Fragmentation control: Used for fragmentation at corners.

Table 4-10. denseopc_options Summary (cont.)

Keyword	Description
<code>fragment_flaglayer</code>	Fragmentation control: Controls the fragmentation operation for a specified layer.
<code>fragment_grid</code>	Snaps variable-length fragmentation to the user-specified grid.
<code>fragment_inter</code>	Fragmentation control: Used to define parameters for interfeature fragmentation.
<code>fragment_interlayers</code>	Fragmentation control: Specifies layers for interfeature fragmentation.
<code>fragment_island</code>	Fragmentation control: Used to define parameters for island-layer fragmentation.
<code>fragment_layer</code>	Fragmentation control sub-command block: Used to reset global fragmentation settings for special-case fragmentation.
<code>fragment_layer_order</code>	Changes the default processing order of fragmentation schemes in a <code>fragment_layer</code> command block.
<code>fragment_marker</code>	Adds fragmentation vertices based on proximity to markers on a marker layer.
<code>fragment_max</code>	Fragmentation control: Used to set maximum fragment length constraint.
<code>fragment_min</code>	Fragmentation control: Used to set minimum fragment length constraint.
<code>fragment_minjog</code>	Fragmentation control: Controls jog definition.
<code>fragment_nearly_coincident</code>	Copies fragmentation from one layer to another where the layers' edges are within a specified distance.
<code>fragment_nop</code>	Controls concurrency for fragmentation inside a <code>fragment_layer</code> block with <code>fragment_layer_order</code> full.
<code>fragment_optimize</code>	Performs fragment optimization based on specified corner and inflection parameters.
<code>fragment_preserve_length</code>	Fragmentation control: Specifies the longest fragment that can avoid implicit fragmentation.
<code>fragment_ripple</code>	Repeats fragments from vertices created by corners, island fragmentation, or interfeature fragmentation within a <code>fragment_layer</code> block.
<code>fragment_visible</code>	Defines layers for visible layer fragmentation.
<code>freeze_skew_edges</code>	Freezes skew edges and neighboring fragments.
<code>grids_for_angle_45_snap</code>	Used to specify how 45-degree angles are moved during OPC.

Table 4-10. denseopc_options Summary (cont.)

Keyword	Description
image	Used to create an ideal image condition.
jog_freeze	Prevents a fragment from moving during OPC if it is a jog edge.
jog_ignore_metric	Changes fragment corner classification when shared with a short jog.
lapi_exec_tcl	Enables tagging commands not accepted in the tag scripting section (formerly the <script>...<endscript> method).
layer (for denseopc_options Block)	Defines layer characteristics such as name, type, transmission, and mask.
line_end_fragment	Enables or disables line end fragmentation.
lint	Checks a setup file or input geometries for suboptimal settings or other performance issues.
mask_chip_corner	Used to simulate corner reduction during lithography process.
max_iterations	Limits number of OPC iterations.
max_iter_movement	Limits edge movement distance for an OPC iteration.
max_opc_move	Specifies maximum total distance all fragments are allowed to be moved by the entire OPC run.
min_dog_ear_length	Sets a minimum fragment length for diagonal fragments with two 135-degree corners.
mpopc	Adds special measurement sites to account for multi-patterning issues.
mrc_consistency	Loosens MRC settings in particular cases.
mrc_mode	Specifies a new fragmentation movement algorithm.
mrc_rule	Used to create external and internal spacing constraints.
mrc_rule_area	Used to check SRAF layer spacing constraints for sraf_print_avoidance.
mrc_rule_priority	Specifies how MRC rules are applied if multiple rules apply to an edge pair.
no_opc_region	Creates a “non-opc” layer that designates areas where edges will not be moved by OPC (areas that intersect the “non-opc” layer).
nominal_epe_limit	Allows nominal EPE to converge to a value within a specified range.
one_time_site_pairing	Performs site paring only in the first iteration during the execution of an OPC_ITERATION command.

Table 4-10. denseopc_options Summary (cont.)

Keyword	Description
opc_grid_multiplier	Refines the correction grid by a specified factor.
pvband_tolerance	Prevents further changes to a PV band if it is within a specified tolerance.
pw_condition	Creates alternate image conditions for dose, focus, resist, and aerial threshold, as opposed to a nominal image.
pwopc_mode (Deprecated in 2016.2)	Defines how final EPE is calculated given nominal and process window condition EPEs.
read_layer_properties	Makes numeric DFM properties accessible in Calibre nmOPC recipes.
retarget_layer	Defines a new target layer, leaving the fragmentation on the opc layers.
retargeting_options	Specifies values for matrix retargeting parameters.
scale	Used to resize fragmentation lengths according to a scale multiplier.
spa_promotion	Controls SRAF promotion during an OPC run.
sraf_promotion_distance	Set the distance from the main feature in which to check for SRAFs that may also need to be promoted.
step_size	Sets the minimum edge movement distance for OPC.
topo_model	Applies previously-loaded topography models to OPC run.
topo_model_load	Specifies the name of the topography model.
veb_corner_site_placement	Specifies where along a corner fragment to place the VEB evaluation point.
veb_evalpts_per_fragment	Specifies how many VEB evaluation points to place on each non-corner fragment.
veb_pseudo_nyquist	Defines a pseudo-Nyquist value for automatic fragmentation.
version	Specifies the Calibre nmOPC version number.
wafer_enclose	Creates a layer-based cost function to check if a shape is enclosed within an image contour.
wafer_enclosedby	Creates a layer-based cost function to check if a shape is enclosed within a polygon on another layer.
wafer_exclude	Used in conjunction with PV Band PWOPC, prevents the image from getting too close to shapes on a specified layer.

active_layer

denseopc_options Keywords

Creates intersecting hidden layers used to define gates.

Usage

```
active_layer layer [weight w] [tolerance tol_value]
```

Arguments

- *layer*

Identifies the active layer.

- weight *w*

An optional parameter that specifies the priority (the “weight”) to give to the cost function. The default weight is 1000. The weight is calculated against the cumulated weights of all the other cost functions.

- tolerance *tol_value*

The optional tolerance parameter specifies the minimum tolerance a gate “flare” will be given. If the “flare” is less than the given tolerance, the appropriate fragments will not attempt to correct for the flare (a flare is equal to the difference between the EPE measured at the point nearest to active and the closest local minimum EPE).

Instead, the fragments will move according to the other cost functions placed on them. The default tolerance is 0.003 nm.

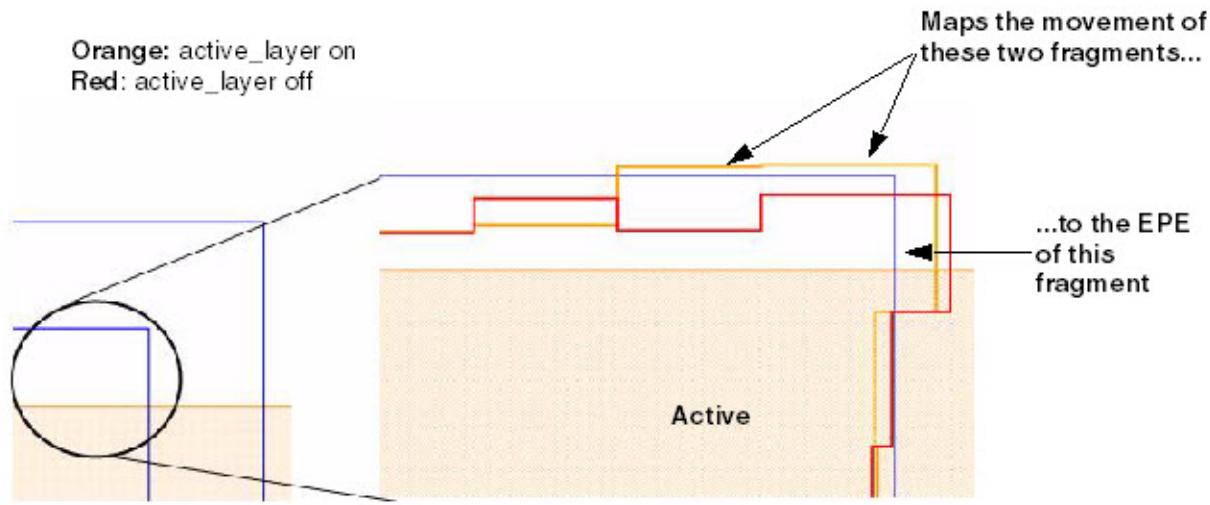
Description

The active_layer command is used to define a hidden layer that intersects the target to define gates. This allows you to pass the POLY and ACTIVE layers into OPC and identify all gates to OPC. The following process occurs when active_layer is enabled:

1. All fragments that touch the active layer are identified.
2. The top and bottom fragments of the gate are identified.
3. If a concave corner exists within $3 * \text{Nyquist}$, then identify the two fragments from the concave corner perpendicular to the gate as being mapped to control the gate flare instead of their own EPE.
4. Gate sizing which may result in “tiny jogs” (less than $0.5 * \text{Nyquist}$) are ignored when searching for those concave corners near the flaring gates.
5. The effective EPE function for those mapped fragments is determined by equal weighting of all wafer cost functions and the EPE at the most flared portion of the side of the gate controlled by those mapped fragments. The EPE for the mapped concave corner fragments themselves are completely ignored (equivalent of 0 weight). The feedback for those two fragments is set to -0.5 overriding any other feedback.

Figure 4-7 shows a visual example of how active_layer works.

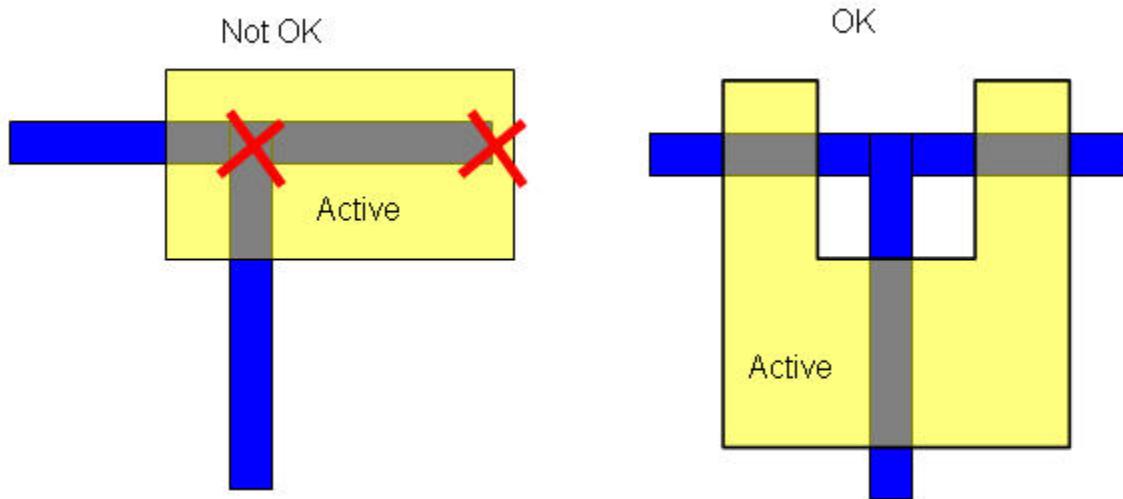
Figure 4-7. Gate CD Prioritization



Note

- When objects intersect between the “active” layer and opc layer, objects on the “active” layer must never enclose any corner on the opc layer, as illustrated in Figure 4-8.

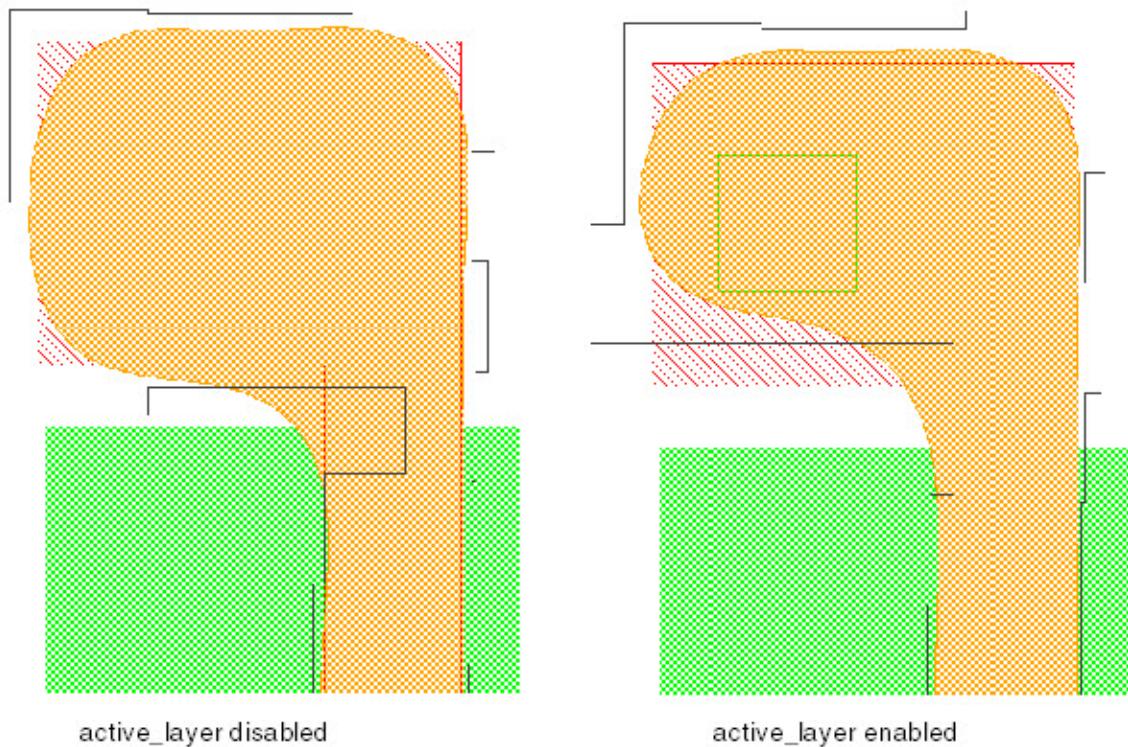
Figure 4-8. Active Layer Enclosure



Examples

The following example shows how the active_layer command can be used to reduce or eliminate a “gate-flare” effect.

Figure 4-9. Gate Flare Reduction Using active_layer



On the left, the active layer command is not used and “gate flare” of the CD occurs at the top of the gate. With active_layer turned on, the gate flare is almost completely eliminated within the active region.

adjust_moved_target_sites

[denseopc_options](#) Keywords

Adjusts dense sites after retargeting with target_curve.

Usage

adjust_moved_target_sites {on | off}

Arguments

- **on | off**

A required argument that specifies whether sites are moved.

on — When “[target_curve -moved](#)” is specified, the sites on opc fragments are moved.
This is the default behavior.

off — The control sites on opc fragments are not adjusted.

Description

“target_curve -moved” generates a new target based on the opc layer’s current fragment positions, presumed to be post-correction. By default, the sites are adjusted as necessary when a new target is created. There may be some cases in which this is not desirable, however.

adjust_target_control_sites

[denseopc_options](#) Keywords

Determines how dense sites are adjusted for target control.

Usage

adjust_target_control_sites {on | off}

Arguments

- **on**

Adjusts dense sites for involved fragments. This is the default setting.

- **off**

Specifies that dense sites are not to be adjusted.

Description

Target control is typically a two-step process. The first step is to define a target function and associates it with a slot number (which supports a range of target shift types) from constant values to quadratic curves. The second step is to apply the target function to a tag set of fragments. The limitation is that max number of slots available is 255.

Using the [TARGET_CONTROL](#) Tcl scripting command with the -constant option allows you to specify target shifts as a constant numeric value in addition to the two-step process. This command controls how dense sites will be adjusted for fragments controlled with “[TARGET_CONTROL tag -constant val](#)” for algorithm 2 or PV Band PWOPC mode.

algorithm

denseopc_options Keywords

Bypasses contour creation to improve output consistency.

Usage

algorithm {0 | 1 | 2}

Arguments

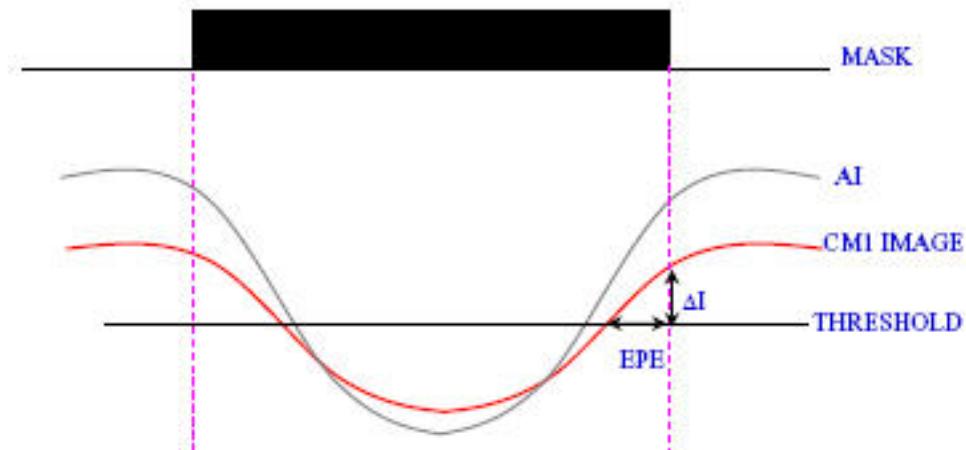
- **0** (This option is deprecated as of the 2016.2 release)

Minimizes the EPE (attempts to drive EPE to zero at mask edge). This option uses contours to determine EPEs.

- **1** (This option is deprecated as of the 2016.2 release)

Minimizes the delta of intensity, ΔI (the difference between the CM1 Intensity map and Threshold intensity). This option uses intensity sampling points to determine EPEs. See [Figure 4-10](#) for an illustration.

Figure 4-10. Algorithm 1 Effects



- **2**

This option uses sites placed using a method similar to “algorithm 0” to determine EPEs. Sites are automatically created for algorithm 2 for opc layers. The 2 option is used for PV Band PWOPC exclusively. This is the default setting.

Description

The algorithm command is used to bypass contour creation.

Calibre nmOPC uses a sub-pixel interpolation engine to interpolate from the image grid down to the database unit. This interpolation can be used with the algorithm command to increase consistency and speed in Calibre nmOPC. The available settings are as follows:

- algorithm 0 (deprecated in 2016.2): In this case, OPC is performed by minimizing a cost function, and this cost function is the Edge Placement Error (EPE), which is the displacement between the CM1 contour and the target. This requires computing the contour and measuring the EPE. Generating the contour requires interpolation from a large simulation grid down to the database unit. This “contouring” process can generate simulation output inconsistencies.
- algorithm 1 (deprecated in 2016.2): In this case, instead of plotting the contour and measuring the EPE as the cost function, DI (the delta between the CM1 Intensity map and Threshold intensity) is used. The intensity difference is minimized rather than the EPE. Slight differences in OPC output may result from using algorithm 1.
- algorithm 2 (default): This setting uses sites placement similar to algorithm 0 to determine EPE. Sites are automatically created for algorithm 2 for opc layers.

When determining where sites are placed, use [SITES_DUMP](#). Because algorithm 2 does not use measurement locations, the [OUTPUT_MEASUREMENT_LOCATIONS](#) method returns an empty layer.

Note

 Although the EPE value used to determine the correction step is approximate (this is not the actual EPE value), this does not mean that algorithm 1 is less accurate than algorithm 0, as convergence is based on the delta of Intensity which is accurate. EPE is only used for the correction step and not convergence.

Algorithm 1 cannot be used with the following commands:

- [DENSE_EPE](#)
- [DENSE_CD](#)
- [NEWTAG cd](#)
- [NEWTAG epe](#) (There is an exception in this case: NEWTAG epe can be used with algorithm 1 as long as the mode is effective_epe. This is the mode for NEWTAG epe that does not need a contour. All that is required is an [OPC_ITERATION](#).)
- [OPC_ITERATION](#) (algorithm will not work with the -registers option)
- [DENSE_SIMULATE](#)

Related Topics

[OUTPUT_MEASUREMENT_LOCATIONS](#)

allow_single_site_correction

[denseopc_options](#) Keywords

Allows jog fragments to be corrected.

Usage

allow_single_site_correction {off | on}

Arguments

- **off | on**

A required argument that controls whether jog fragments are corrected.

Description

This optional command overrides the default treatment of fragments with single sites. By default, a fragment with a single site, one convex corner, and one concave corner does not receive any correction. Typically these fragments are jogs and should not be corrected. Set this optional command to “on” if you have created fragments with single sites on purpose and want to move them.

Examples

Example 1

This allows fragments with single sites and one convex and one concave corner to move:

```
allow_single_site_correction on
```

background (for denseopc_options block)

denseopc_options Keywords

Defines background imaging characteristics.

Usage

```
background {[pattern num]{dark | clear | {attenuatedfactor} | {real imag} }}{...}
```

Arguments

- pattern *num*

For multi-patterning, this optional argument specifies different backgrounds for each patterning step. If the pattern keyword is not used, the same backgrounds are used for all patterns. All the patterns must be specified as part of a single background command, and must be specified in the following order: 0,1,2, and so on. The number of patterns specified must be the same as the number defined by the layer statements:

```
background pattern 0 dark pattern 1 atten 0.06
```

Note

 For non-Litho model implementations, pattern definitions must be the same between the [background \(for denseopc_options block\)](#), [image](#), and [pw_condition](#) statements.

- **dark**

A literal argument designating dark field imaging. The transmission added to the mask is {layertrans + 0} for this layer.

- **clear**

A literal argument designating clear field imaging. The transmission added to the mask is {layertrans + 1} for this layer.

- **attenuated*factor***

An attenuated phase-shifting background is specified using either the **attenuated** command or its abbreviation, **atten**, with the attenuation *factor*. This *factor* is specified as the normalized intensity that should transfer to the wafer for a clear exposure.

factor — a **required** numerical argument to the attenuated option. It is the attenuation factor for attenuated backgrounds. The unit is a percentage represented as a number between 0 and 1 (for example, for an 8% attenuated phase shift mask, the *factor* should be entered as 0.08). The maximum allowed value is 0.36 (36%).

When Calibre LITHO tools encounter **attenuated*factor***, they translate the *factor* into a real imaginary pair such that the attenuated background has a transmission value

$$\text{RE(bg)} = -\sqrt{\text{factor}}$$

$$\text{IM(bg)} = 0$$

Do not use an attenuated background with a phase180 layer, as this will cause an error.

Transmission added to mask will be $\{1.\textit{factor} - \text{bg}\}$ for this layer.

- ***real imag***

An optional pair of real numbers specifying the layer transmission value and background value, respectively. Note that the attenuated syntax for the transmission is recommended over the ***real imag*** pair syntax. The ***real imag*** syntax supports any arbitrary phase and transmission combination.

If you specify background with the ***real imag*** pair for a specified layer transmission value, the resulting transmission value for the output layer with a ***real imag*** pair is resolved relative to the background pair. The layer transmission value is resolved according to the following equation:

$$\text{background (real imag)} + \text{layer (real imag)} = \text{transmission_value}$$

For example:

```
background 1 0
name      opt          type
chrome    visible     0 0
```

The background is specified with a (1, 0) pair, while the chrome layer is defined with a transmission value of (0,0). With clear background (1,0) and a layer specified with (0,0), the resulting transmission layer is (1,0) + (0,0) = clear (1,0).

```
background 1 0
name      opt          type
p180     visible     -1 0
```

In this case, the layer p180 is defined as (-1 0). With clear background (1,0) and a layer specified with (-1,0), the resulting transmission layer is (1,0) + (-1,0) = dark (0,0).

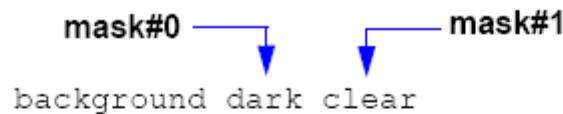
Description

A configuration command describing the imaging background. Note that this command cannot be used if you are using a litho model file. (See “[Load Model Information With a Litho Model](#)” on page 48.)

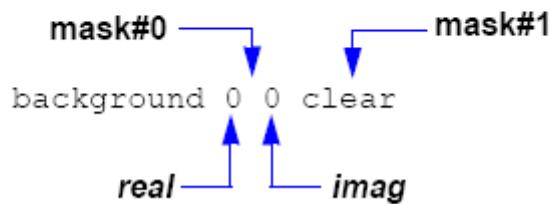
You must specify one background type per mask exposure. When multiple masks are used, the arguments must be supplied in order, with the first argument referring to Mask 0, the second referring to Mask 1, and so on.

[Figure 4-11](#) shows how to specify multiple-mask exposures using different arguments.

Figure 4-11. Multiple-Mask Exposures



Or



The first background applies to the first mask exposure, “mask#0”; the second background applies to the second exposure, “mask#1”.

Note

 The layer transmission value is tied to the background value. For example, a clear field mask does not support any layers that are defined as clear.

Examples

This example shows the combination of layer and background settings needed to define an attenuated phase shift mask. The background is attenuated and the layer type is clear because light is not obstructed when passing through the polygons in the VIA layer.

```
background attenuated 0.08
layer VIA opc clear
```

check_movement

[denseopc_options Keywords](#)

Enables/disables OPC edge movement constraints.

Usage

check_movement {on | off}

Arguments

- **on | off**

If check_movement is set to on, OPC 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 OPC mask constraints. Turning this argument off will result in mask constraints not being checked. This command is particularly useful for debugging constraints.

chiplet_placement_flare_tolerance

[denseopc_options](#) Keywords

EUV Command

Specifies a threshold for the difference in flare values that prevents re-using OPC results for different placements of a chiplet (large subassembly or block).

Usage

chiplet_placement_flare_tolerance *fraction*

Arguments

- *fraction*

A required argument specified as a number between 0.0 and 1.0. The default is 0.0.

Description

This optional command can improve performance by reusing OPC results for different placements of identical blocks that differ only slightly in the flare map intensity. By default, OPC results in EUV processes are recalculated for each pixel. Larger tolerance values result in more reuse.

This command is ignored unless [setlayer denseopc](#) includes CHIPLET_PLACEMENT.

The difference is calculated by comparing all corresponding pixels in previously calculated placements.

Examples

The following setting only allows the OPC of a chiplet to be reused in tiles where the flare maps of the two placements have a maximum pixel-wise difference of strictly less than 0.1%:

```
chiplet_placement_flare_tolerance 0.001
```

corner_control

[denseopc_options](#) Keywords

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. This command is used to control “oscillations”, or “ringing” which propagates from corners.

Fragment lengths are based on the Nyquist value. There is an optional LITHO DENSEOPC command **scale** that can be 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
```

displacement_limit_mode

denseopc_options Keywords

Specifies whether to use the most recent or the most restrictive DISPLACEMENT setting.

Usage

displacement_limit_mode {LAST | STRICT}

Arguments

- **LAST**

A required keyword indicating that the most recent DISPLACEMENT limits are used. This is the default behavior.

- **STRICT**

A required keyword indicating that the tightest DISPLACEMENT limits are used.

Description

This optional command specifies how to resolve conflicting DISPLACEMENT limit settings. The arguments “LAST” and “STRICT” are case-insensitive.

Examples

In this example, the same fragment is in tag sets tagA and tagB, which receive different inner limits.

```
DISPLACEMENT tagA -limit 0 0.02
DISPLACEMENT tagB -limit -0.01 0.01
```

By default, the fragment’s movement is limited to 0.01 into the polygon, and 0.01 out from the polygon. If “displacement_limit_mode strict” is set, the limits instead are 0 into the polygon and 0.01 out from the polygon.

curvilinear_opc

[denseopc_options](#) Keywords

Activates curvilinear OPC. Use with LITHO CL DENSEOPC.

Usage

```
curvilinear_opc smooth_radius value [min_width value] [min_space value]
```

Arguments

- **smooth_radius *value***

A required argument specifying the desired smoothing to apply to OPC results.

Conceptually, the OPC-generated contours must allow a circle with the radius given by *value* to fit along the curve. OPC corrections are adjusted to prevent “fingers” too small for the smooth radius circle to fit inside the predicted image. The value is in microns.

- **min_width *value***

An optional argument specifying the minimum width for curved shapes for MRC.

Measurements are taken perpendicular to the tangent of the curve. The value is in microns.

If this argument is not specified, no check for minimum width is performed for curved shapes.

- **min_space *value***

An optional argument specifying the minimum space between curved shapes for MRC.

Measurements are taken perpendicular to the tangent of the curve. The value is in microns.

If this argument is not specified, no check for minimum space is performed for curved shapes.

Description

The curvilinear_opc keyword activates curvilinear OPC and also sets basic MRC constraints for curved shapes. (MRC_RULE and mrc_rule only check Manhattan polygons.) The OPC layer should be produced by [setlayer layer_simplify](#) or a deangling operation, with the original drawn layer used as a retarget layer.

The smooth_radius argument controls the aggressiveness of the smoothing. A smaller radius value allows sharper features in the image. (It does not affect the OPC layer directly like min_width and min_space do.)

Examples

This sets a smooth radius of 50 nanometers, but does not prevent mask layer shapes from merging or splitting:

```
curvilinear_opc smooth_radius 0.05
```

effective_epe_regulation

[denseopc_options](#) Keywords

Regulates effective EPE calculations per OPC iteration.

Usage

effective_epe_regulation {on | off}

Arguments

- **on | off**

If enabled, the effective EPE is adjusted such that the next correction is kept to a distance value that is close to the previous correction. If set to off, the effective EPE is calculated without regulation. The default is off.

Description

The **effective_epe_regulation** keyword allows you to prevent overcorrections or oscillations in fragment placement by regulating effective EPE calculations per OPC iteration.

For example, suppose you run PWOPC and in iteration 5, the effective EPE is calculated at +5 nm. In iteration 6, the effective EPE is unexpectedly calculated as -5 nm, which may cause oscillations in the fragment placement. If **effective_epe_regulation** is enabled, then the effective EPE is regulated to a number that is closer to the +5 nm value from iteration 5.

This command only works for PV Band PWOPC mode and is ignored otherwise.

enhance_corner_to_corner_site_pairing

[denseopc_options](#) Keywords

Enables an enhanced site placement algorithm for improved site pairing.

Usage

```
enhance_corner_to_corner_site_pairing {on | off}
```

Arguments

- **on | off**

When set to on, site pairing for the following corner-to-corner locations are enhanced by using shape recognition for better site placement. The default is off.

Description

This option only works for PV Band PWOPC mode and is ignored otherwise.

Two sets of corner edges must be in opposite directions and the pairing edges should project onto each other. The corner pairs can be external corner-to-corner or internal corner-to-corner (as illustrated in [Figure 4-12](#) and [Figure 4-13](#), respectively).

Figure 4-12. External Corner-to-Corner

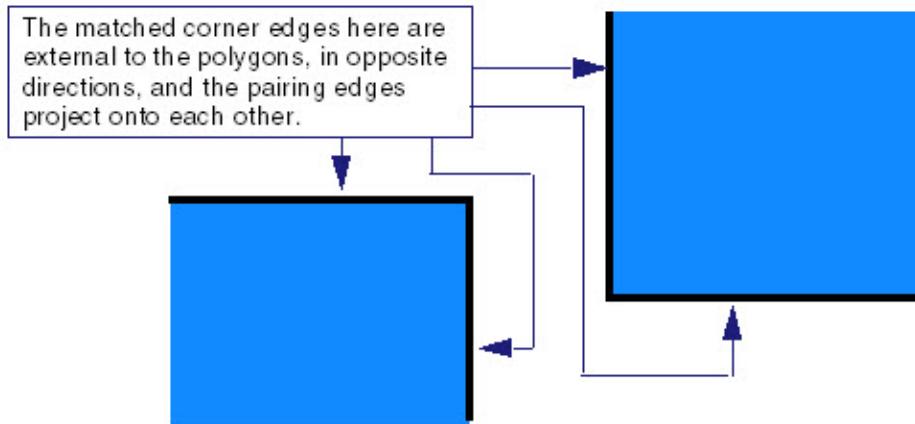
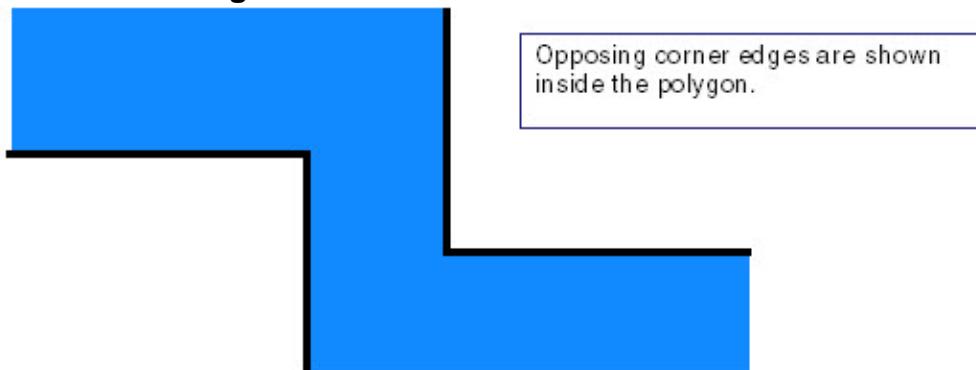


Figure 4-13. Internal Corner-to-Corner



epe_spacing

[denseopc_options](#) Keywords

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 0.01 (in microns).

Description

This command sets a global value for the spacing between EPE or image samples along fragments. This is only used for manual site creation for all fragments. Local EPE spacing is controlled using the [SITES_CREATE](#) -epe_spacing Tcl command. You must first regenerate all sites using [SITES_DELETE](#) and [SITES_CREATE](#) in order for the EPE spacing changes to take effect.

Examples

```
epe_spacing 0.03
```

etch_model

[Variable Etch Biasing \(VEB\) Commands](#)

[denseopc_options Keywords](#)

Loads etch model to be used by setlayer veb_retarget.

Usage

etch_model *veb_model_name*

Arguments

- ***veb_model_name***

A required argument specifying a name to refer to the etch model in the [image](#) setlayer command.

Description

Loads the specified VEB model to be used by [setlayer veb_rettarget](#).

Examples

```
etch_model veb1
```

feedback

denseopc_options Keywords

Automatic convergence controller command. Sets feedback values for all fragments, unless overridden by FEEDBACK.

Usage

feedback *feedback_val* ...

Arguments

- *feedback_val* ...

Specifies a user-defined feedback value. The default is -0.4, but can range from -2 to 0, inclusive. Values below -1 are not recommended for OPC applications.

Description

An optional command that defines feedback values globally to all fragments to be used per iteration. If more iterations are performed than the supplied values, the remaining iterations use the last *feedback_val*.

Edge movement is set by (EPE * feedback). If there is no valid EPE, edge movement is set to a default of Nyquist/3.

You can explicitly increase or decrease the feedback to adjust edge movement. Supplying *n* values sets feedback for the first *n* iterations.

Note

 As a general rule, the first two and last two iterations should be set to no more than -0.4.

You can use higher feedback values (up to -1) for the middle iterations. Failure to adhere to these guidelines can produce inconsistent results.

To adjust the feedback values for individual fragments or tagged sets of fragments, use the **FEEDBACK** Tcl scripting command.

If feedback or FEEDBACK are not explicitly defined, then each OPC iteration uses a default value of -0.4.

Examples

```
feedback -0.4 -0.4 -0.8 -0.3 -0.3
```

feedback_balance (Deprecated in 2016.2)

[denseopc_options](#) Keywords

Optimizes feedback to improve convergence.

Note

 This command is deprecated as of the 2016.2 release and will no longer be documented in future releases.

Usage

feedback_balance *value*

Arguments

- ***value***

A number between 0 and 0.99 that controls how much the pseudo-EPE depends on the actual image slope, versus the pre-calculated maximum image slope. The default setting is 0.5.

Description

This command is used to optimize feedback (speed versus accuracy) to achieve convergence. This command only affects OPC when [algorithm](#) 2 is set. When feedback_balance is 0, the OPC convergence is slower but will avoid overshoot. When feedback_balance is set to 0.99, the OPC convergence occurs faster but it can also introduce potential instabilities.

Examples

```
feedback_balance 0.3
```

feedback_mode

denseopc_options Keywords

Sets precedence for conflicting definitions of FEEDBACK for fragments that are in more than one tag set.

Usage

feedback_mode {0 | 1}

Arguments

- **0 | 1**

A required argument.

0 — The default. The value used in cases of conflict depends on several situational factors. This is the behavior seen in versions prior to Calibre 2020.1, and remains the default for backwards consistency.

1 — A recommended mode for better consistency. The values that are set last relative to the movement command are used.

Description

An optional command that can be specified at most once per setup file. It addresses how to handle the adjustment of a fragment that belongs to multiple tag sets, each of which has distinct FEEDBACK values.

Consider a fragment that is tagged as “vertical” and “line_end”. In the default case, given

```
FEEDBACK -keep vertical 0.01 0.01 0.01
FEEDBACK -keep line_end 0.1 0.04 0.02
```

the fragment would use 0.01 for its adjustment. Because the vertical tag set’s values are all the same, however, the commands might be written as

```
FEEDBACK -keep vertical 0.01
FEEDBACK -keep line_end 0.1 0.04 0.02
```

In this case, the fragment that is tagged as both vertical and line_end would use the line_end settings.

By setting “feedback_mode 1”, both cases would use the line_end values because they were most recently set. This also means that if the order were swapped, as shown following, the tag set would use the vertical settings whether the 0.01 were written once or all three times. That is,

```
FEEDBACK -keep line_end 0.1 0.04 0.02
FEEDBACK -keep vertical 0.01
```

gives the same results as

```
FEEDBACK -keep line_end 0.1 0.04 0.02  
FEEDBACK -keep vertical 0.01 0.01 0.01
```

flare_model

[denseopc_options](#) Keywords

References the flare model.

Usage

flare_model *model_name*

Arguments

- ***model_name***

Refers to the model name loaded by [flare_model_load](#).

Description

This command specifies the name of the flare model to be used for correction. This command is used in conjunction with the [flare_model_load](#) command.

Note

 This command cannot be used with [Lithomodel \(Litho Model Format\)](#).

Examples

```
feedback -0.5
max_iterations 4
flare_model FM
```

fragment_coincident

[denseopc_options](#) Keywords

Copies fragmentation from one layer onto another where edges are coincident.

Usage

fragment_coincident *layer* **-overlay** {**0** | **1**}

Arguments

- *layer*

The layer name with which edges must be coincident in order to copy fragmentation points.
The *layer* 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 *layer* are overlaid onto the current layer. The fragmentation points are copied from the coincident edges to the current layer while 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 command 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.

Examples

The following example shows fragmentation specified for edges coincident with layer L1. (Both examples use the same setup file.)

```
fragment_layer L2 {  
    fragment_coincident L1  
}  
  
fragment_layer L3 {  
    fragment_coincident L1  
}
```

The following figure shows the results. (The highlighted dots represent interfeature vertices created on the coincident edges.)

Figure 4-14. Changing the Coincident Layer to Affect Fragmentation



fragment_consistency_mode

[denseopc_options](#) Keywords

Controls how fragments generated by fragment_max are promoted around tile boundaries.

Usage

fragment_consistency_mode {0 | 1} [-dist microns]

Arguments

- **0 | 1**

A required argument that turns the method on or off.

0 — Off. This is the default for LITHO DENSEOPC runs.

1 — On. This is the default for LITHO NMBIAS runs.

- **-dist microns**

An optional argument that defines how far from the tile boundary to consider. The value must be smaller than the interaction distance.

The default value is the smaller of 2*[fragment_max](#) and 0.25*[interaction distance](#).

Description

This optional command may be useful for setups that rely heavily on fragment_max fragmentation and have issues with consistency near tile boundaries. Fragmentation consistency is achieved by first promoting fragments within -dist of the tile boundary if they were created by fragment_max, and then updating fragmentation points in the parent tile or cell to align with the promoted fragments. The improved fragmentation consistency leads to improved tagging consistency.

In most situations, there is no benefit to overriding the default settings.

Examples

This shows the default setting for Calibre nmOPC:

```
fragment_consistency_mode 0
```

fragment_corner

denseopc_options Keywords

Fragmentation control: Used for fragmentation at corners.

Usage

```
fragment_corner edge_spec [fragment] rule [corner2 [fragment] rule]  
[{breakinhalf | alwaysbreakinhalf} [enforce]] [allowzero] [source N]
```

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 | any}
```

```
corner_subtype = {end_adjacent | non_end_adjacent | angle_90 | angle_non90}
```

```
rule = [split n [force]] |  
[len1...lenN [priority] [rem r]] [applyratio] [{allowtrim | alwaysallowtrim} 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 | alwaysbreakinhalf} [enforce]

An optional keyword that splits the middle fragment of edges in half when ripples (**len1...lenN** or “repeat **length**”) are specified.

breakinhalf — When the ripples cannot all be placed due to constraints, this option breaks the middle fragment in half provided both halves are greater than **fragment_min**.

`alwaysbreakinhalf` — No matter how many ripples are placed, this option breaks the middle fragment in half provided both halves are greater than `fragment_min`.

By default, the halves of the split fragment must be greater than `fragment_min`; `breakinhalf` and `alwaysbreakinhalf` do not enforce the rem value of the rule. 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](#)” on page 214.

The `enforce` option may only be specified immediately following `breakinhalf` or `alwaysbreakinhalf`.

- `allowzero`

An optional keyword that allows fragments to exactly split an edge. By default, when a fragmentation scheme exactly matches the edge length, the last pair of fragments is not inserted because the zero-length “remainder” is less than the minimum length.

- `source N`

An optional argument that specifies an identifier for the rule. `N` must be an integer from 1 to 255, inclusive, and does not need to be unique to the `fragment_corner` statement, although this is recommended.

Fragments created by a rule with “source 1” can be added to a tag set later using “`TAG_FRAG_SRC layer 1 -out tag_name`”. When these tag sets are output to layers, you can compare to the correction layer to see why a particular edge was fragmented in the manner it was.

***edge_spec* Arguments**

- `corner_type`

`corner_type_next`

A required argument indicating whether a corner is concave, convex, or either (any). 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 215.)

The `concave_next` and `convex_next` keywords can only be specified after a `corner_type` [`corner_subtype`]. “`any_next`” is not a valid keyword.

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 4-16](#) on page 215. The constraint takes the form of a DRC constraint and the value is specified in microns.

When only one corner is specified, length2 is ignored. Use “any” for the second *corner_type* when using length2 if you do not care whether the second corner is convex or concave.

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 would violate [fragment_min](#), the command generates an error. The above example would likely generate an error because lengths down to 0 would be split into four fragments. You can prevent this by providing a lower bound, as in this example:

```
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 works by taking an edge and attempting 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. From that point onward, as ripples are created, intrafeature fragmentation will continue working inward toward the center of the edge creating ripples. It will keep doing this until no ripples can be inserted without violating rem/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 + ... A_fragment_nth + rem_A <= 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 OPC 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 would produce a fragment with length smaller than *r*, then it will not fragment the edge. If not specified, the default value used 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 | alwaysallowtrim} *amount*

An optional argument specifying an amount of length reduction that can be done to the corner fragment (allowtrim) or last fragment before the center (alwaysallowtrim) in order to make room for a middle fragment of minimum length.

Only one of “allowtrim” or “alwaysallowtrim” can be specified per command.

- 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*.

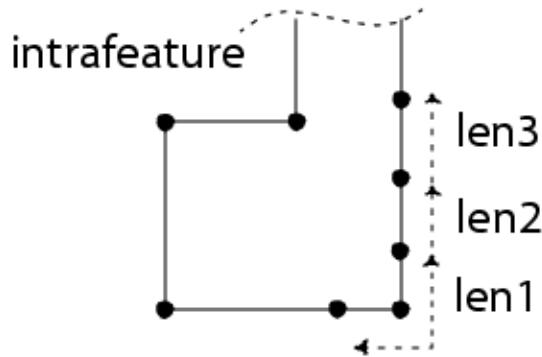
If force is specified and allowtrim is present, an attempt is made to trim the ends in order to allow the repeated fragments to have exactly the specified length. If possible, the end fragments are trimmed in an attempt to avoid creating fragments longer than this length.

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 OPC 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 4-15. 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 214.

Examples

Example 1: Declaration Order

The declaration order has an impact on the results. Each intrafeature command that is executed 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 one convex and one 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 two 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:

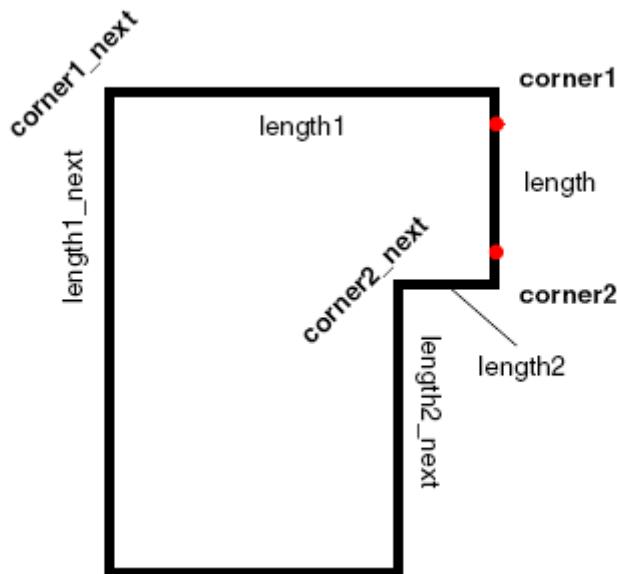
```
#This works better:
fragment_corner concave concave length >0.8 <=0.9 fragment 0.15 0.15
fragment_corner concave           length >0.9      fragment 0.15 0.15
```

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 4-16. 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
```

Example 7: Length2 With an Indeterminate Corner

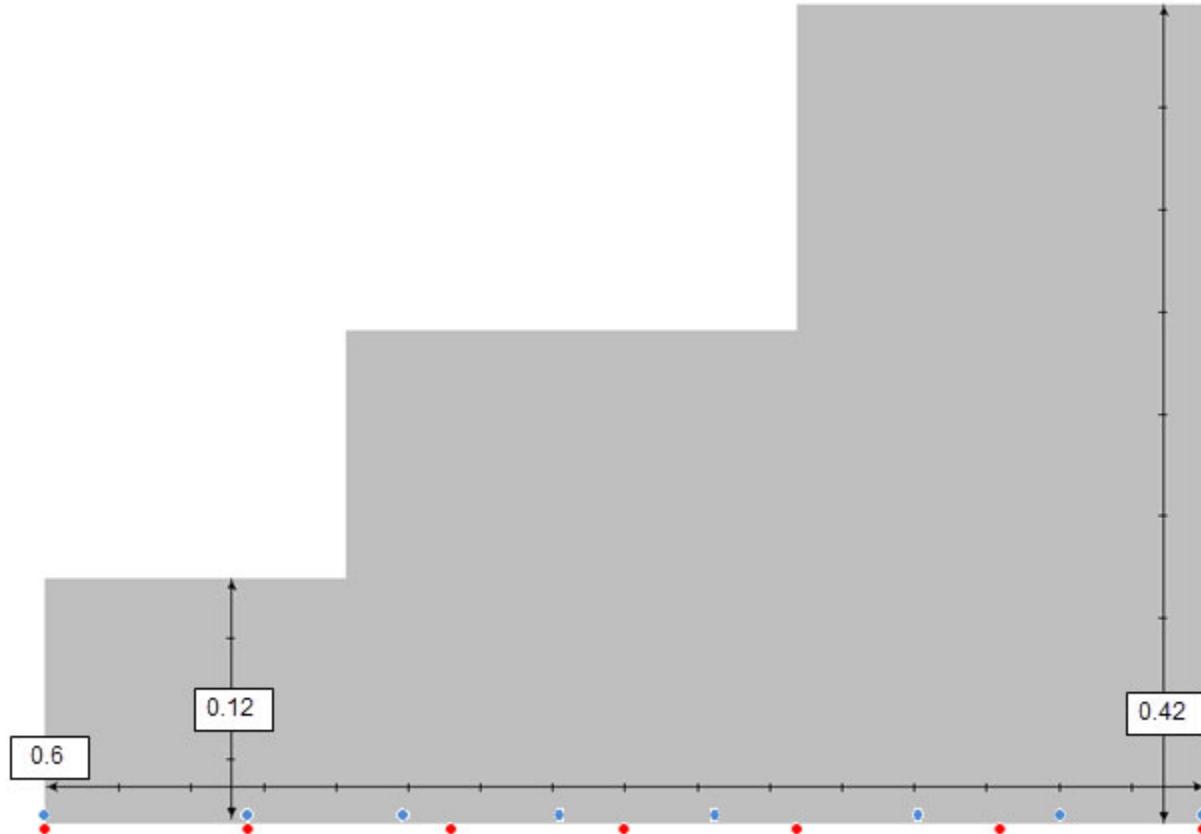
The corner type “any” allows for more control of fragmentation. Consider the following case, a stepped polygon with the measurements shown in [Figure 4-17](#) on page 216 and the following fragmentation rules:

```
fragment_min 0.025
fragment_max 0.1
```

```
fragment_corner convex convex_next any length > 0.500 <= 0.600 \
length1 >= 0.400 <= 0.450 \
length2 >= 0.2 < 0.25 fragment 0.070 0.070
```

Without the “any” keyword for the second corner, the length2 argument is ignored so the rule is applied to the long edge (blue dots). With “any,” the edge beyond the next corner (0.12 um) is checked. It is too short for the rule to apply, so the long edge is fragmented by fragment_max instead (red dots).

Figure 4-17. Effect of any on length2



fragment_flaglayer

denseopc_options Keywords

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 set. Available flag names are described in the following table.

Table 4-11. 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
interfeature_visible	Do interfeature fragmentation on visible layers
intersection_island	Break layers crossed by this island layer

- {**enable** | **disable**}

Indicates whether the *flagName* feature should be enabled or disabled.

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_grid

[denseopc_options Keywords](#)

Snaps variable-length fragmentation to the user-specified grid.

Usage

fragment_grid size

Arguments

- **size**

A required argument specifying the size of the grid in microns. If **size** is not an integer multiple of the database grid, it is rounded to an integer multiple of it. For example, if the PRECISION is 8000 the database grid is using 0.00125 microns as the database unit (dbu). Valid **size** values are then 0.00125, 0.00250, 0.00375, and so on. If you specify 0.002, it will be rounded to 0.0025.

By default, 1 dbu is used.

Description

This optional keyword affects only variable-length fragments such as those created by the repeat, allowtrim, and applyratio keywords in fragment_corner and fragment_ripple. Fixed-length fragmentation, such as those created by the **len1...lenN** arguments to fragment_corner, are not changed.

Examples

```
fragment_grid 0.01
```

Related Topics

[fragment_corner](#)

[fragment_ripple](#)

fragment_inter

denseopc_options Keywords

Fragmentation control: Used to define parameters for interfeature fragmentation.

Usage

```
fragment_inter [-adjust] [-away] [-combined] [-conflictPriority type...]
[-cornerDist w] [-cornertype corner1 corner2]
[-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 | -centershift} z] [-skipCorners] [-split {0 | 1}]
[-srclength constraint] [-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.

- **-centershift z**

An optional argument specifying the distance that an interfeature fragmentation break should be shifted relative to the centerline of the originating polygon, in microns. Normally, a fragmentation break will occur on a fragment at a location that is exactly perpendicular to a corner. The centershift option shifts the fragmentation points to a distance *z* on either side of the polygon centerline.

The -centershift option requires -ripplestyle 2 or -conflictPriority, and cannot be specified with -shift. It is strongly recommended to use -srclength with -centershift to restrict the shift to narrower originating polygons.

- **-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 232.

- **-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 228.

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.

- **-cornertype *corner1 corner2***

An optional argument that restricts interfeature fragmentation to edges which have both corners matching the given types (convex or concave). The order is not important; both “-cornertype convex concave” and “-cornertype concave convex” match the same edges.

The -cornertype argument requires -ripplestyle 2 or -conflictPriority.

- -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).

The value *y* can be either a number, such as 0.1, or a constraint, such as $\geq 0.1 \leq 0.15$. In both cases, edges closer than 0.1 microns are not fragmented. The constraint form, however, specifies an upper bound so that edges farther than 0.15 microns from the corner are also not fragmented.

- -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

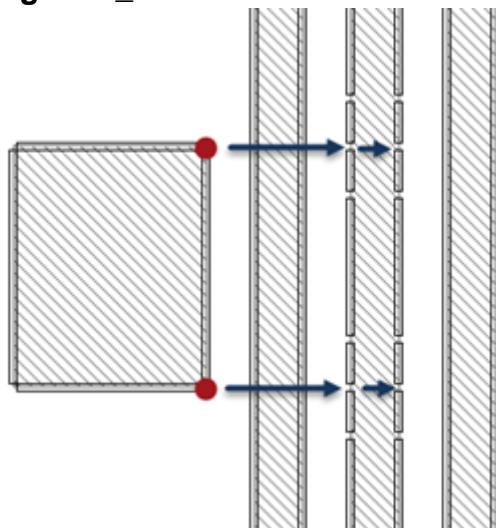
The shield option defaults to 1 for internalOnly, and no other value may be specified.

- -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 4-18. 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 $0.5 * (\lambda / NA) / \text{ripplenlen}$, 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 4-19](#).

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:

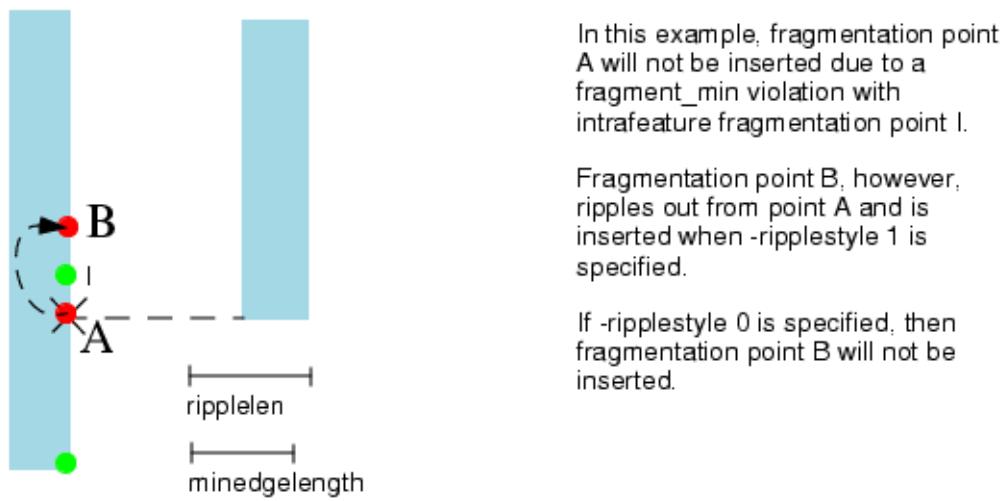
- Ripple points of the same priority are divided into sets such that there are no -rem conflicts between ripple points in different sets and there are -rem conflicts between neighboring ripple points inside each set.
- Each set is processed according to the size of the set:

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 4-19. Example -ripplestyle



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.

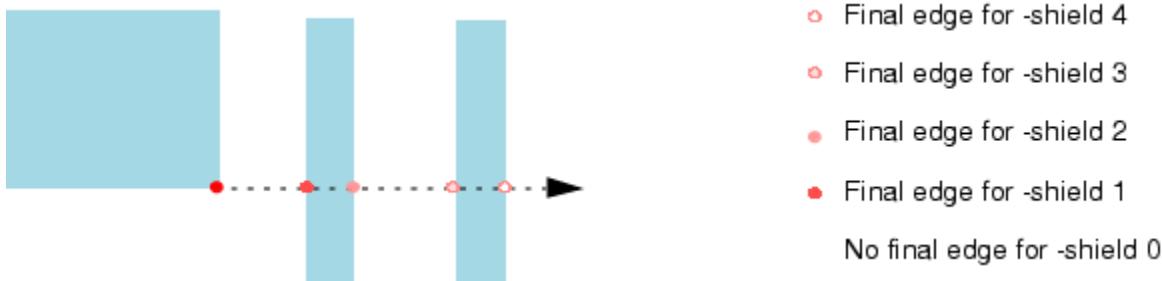
- Any preceding fragmentation caused by commands listed before fragment_inter.
- Interfeature (fragment_inter) fragmentation points have priority over ripples from all other interfeature fragmentation points. The priority of an interfeature point is, from highest to lowest,
 - The number of shielding 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 4-20](#) shows an example of shielding.

Figure 4-20. 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.

The -shift option cannot be specified with -centershift.

- -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 4-23](#) on page 231). Use this option to preserve corner fragmentation.

- `-split {0 | 1}`

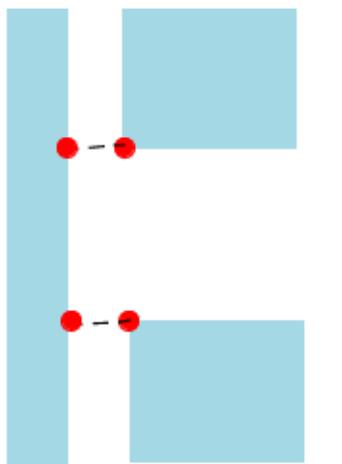
An optional argument controlling the splitting of fragments. The following values are possible:

- 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.

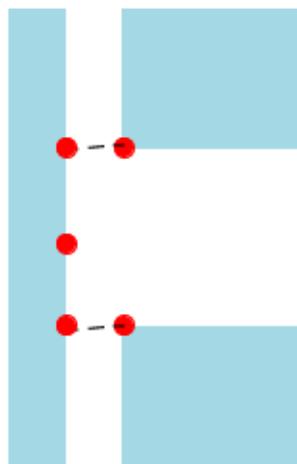
Note

 If the -ripplestyle is set to 1, -split will be set to 1, overriding the specified -split user setting in the setup file.

Figure 4-21. -split Example



(A) with -split 0



(B) with -split 1

- `-srclength constraint`

An optional argument that restricts the edges whose corners can initiate interfeature fragmentation. The constraint must include an upper bound to not exceed the interaction distance. Values are in microns.

Note

 Use -srclength in conjunction with -cornertype to create detailed rules based on the geometry of neighboring polygons.

The -srclength argument requires -ripplestyle 2 or -conflictPriority.

- -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 4-24](#) and [Figure 4-25](#) on page 232 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.

See also [-centershift z](#), which inserts a pair of fragmentation points at a specified distance from the center of the originating polygon.

Description

This keyword defines the fragmentation parameters that influence interfeature fragmentation.

Note

For EUV OPC flows, a [fragment_interlayers](#) command must also be specified in the fragment_layer block containing this command.

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:

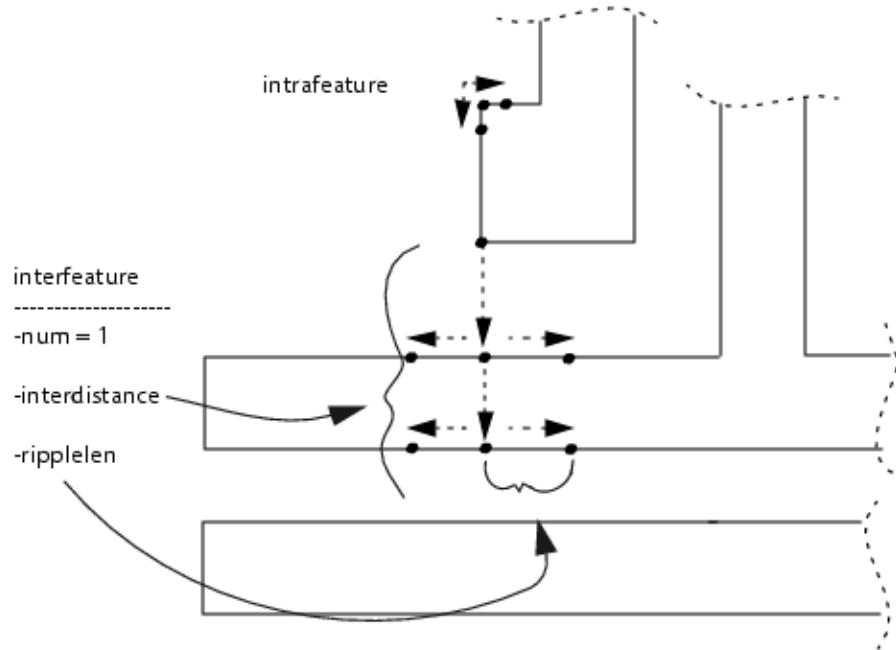
- The distance from the edge to the corner causing the fragmentation is less than or equal to the distance specified by -interdistance.
- None of the fragments that would result from fragmentation is shorter than the distance specified by -rem.

- 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 4-22](#) shows how the interfeature fragmentation takes place.

Figure 4-22. Using Shield With Interfeature Fragmentation



Fragmentation Point Priority

When placing fragmentation points using -cornerDist, priority is established using the following rules:

- [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 nmOPC 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:

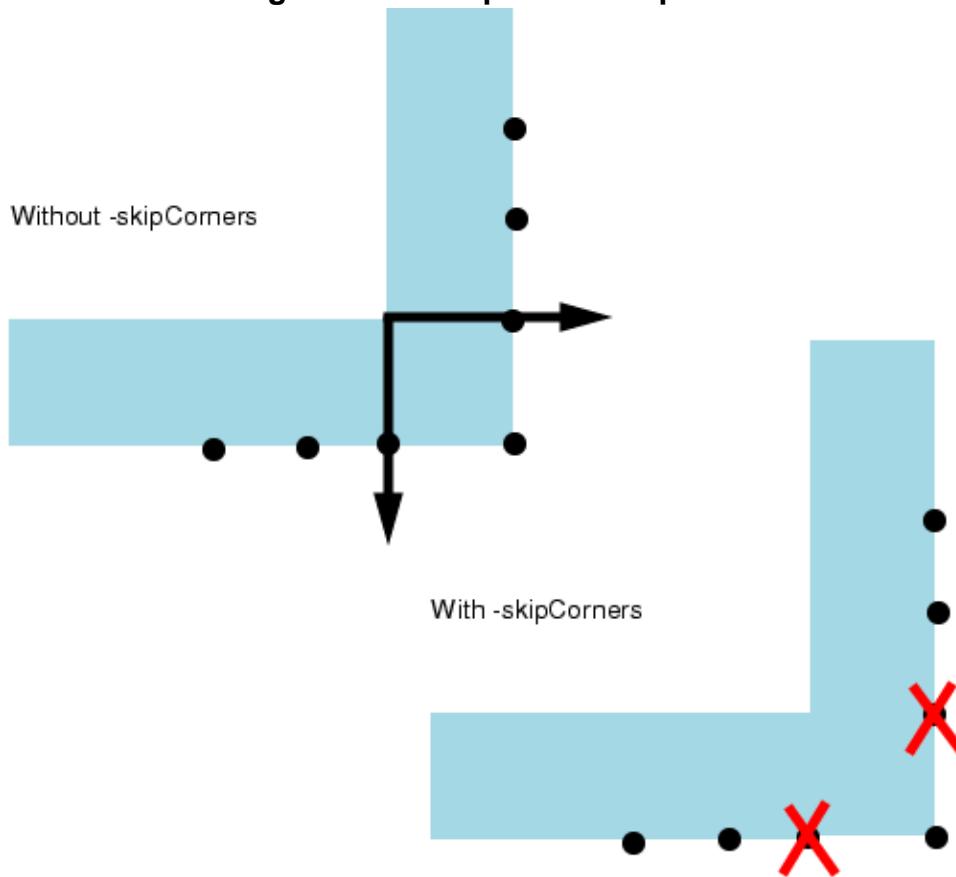
```
fragment_corner convex 0.1 0.05
fragment_inter -interdistance 0.13 -skipCorners
```

However, the -skipCorners option is enabled in this case to prevent from breaking the initial set of intrafeature frags.

Another way to achieve the same effect in this specific instance is by setting -cornerDist to the *len1* of fragment_corner:

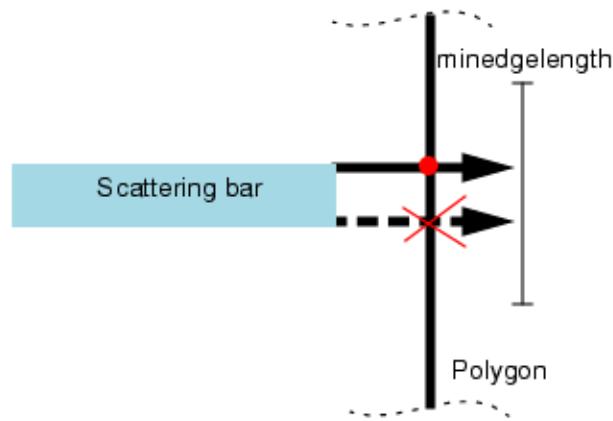
```
fragment_corner convex 0.1 0.05
interfeature -interdistance 0.13 -cornerDist 0.1
```

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 4-23. -skipCorners Option

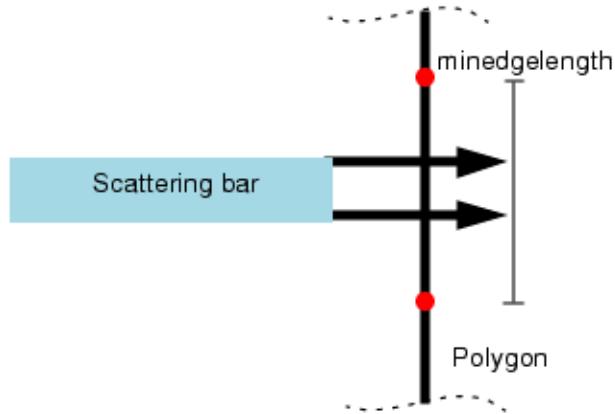
Example 2 - Scattering Bars

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 4-24.

Figure 4-24. 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 4-25](#). 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 4-25. 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 4-26](#) 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):

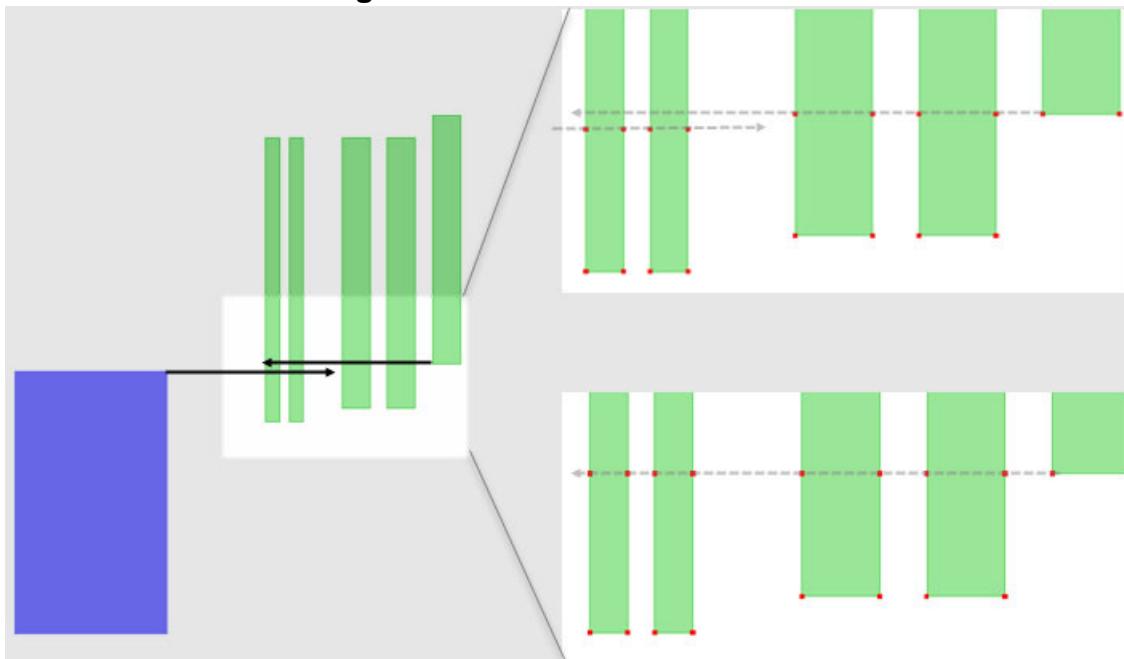
```
layer Green opc
layer Blue island

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 4-26. Effect of -combined



Related Topics

- [fragment_island](#)
- [fragment_ripple](#)
- [fragment_interlayers](#)
- [fragment_layer_order](#)

fragment_interlayers

[denseopc_options](#) Keywords

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. Unless [fragment_inter](#)...-combine is specified, 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.

This keyword can only be used inside of a [fragment_layer](#) block. Using it outside of a [fragment_layer](#) block generates an error.

- For [fragment_layer_order](#) standard or partial, this keyword may only be specified once per [fragment_layer](#) block. Specifying it more than once generates an error.
- For [fragment_layer_order](#) full, [fragment_interlayers](#) can be specified more than one time. It must be specified before any [fragment_inter](#) commands that it is to affect. It then affects all following [fragment_inter](#) commands until the next [fragment_interlayers](#) command. Note that for backward compatibility, if a [fragment_layer](#) block has a *single* [fragment_interlayers](#) command, it affects a preceding [fragment_inter](#) command.

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 only occurs when you 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
}
```

Related Topics

[fragment_layer_order](#)

fragment_island

[denseopc_options Keywords](#)

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.
- **-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](#).
- **-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 set to 0, fragmentation can result in [fragment_min](#) violations. The default value is 1.
- **-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.

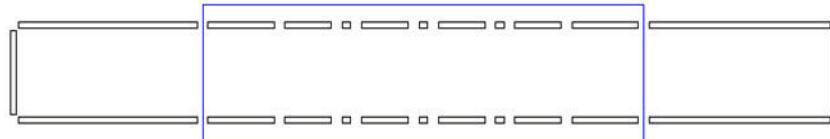
Either -rippleinside or -rippleboth may be specified, but not both.

- -dontcross | -allowcross

The -dontcross option prevents ripples from crossing an island layer, or crossing one another. In [Figure 4-27](#), 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 4-27. 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 4-28](#) shows the result of the same code using the -dontcross option.

Figure 4-28. 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.

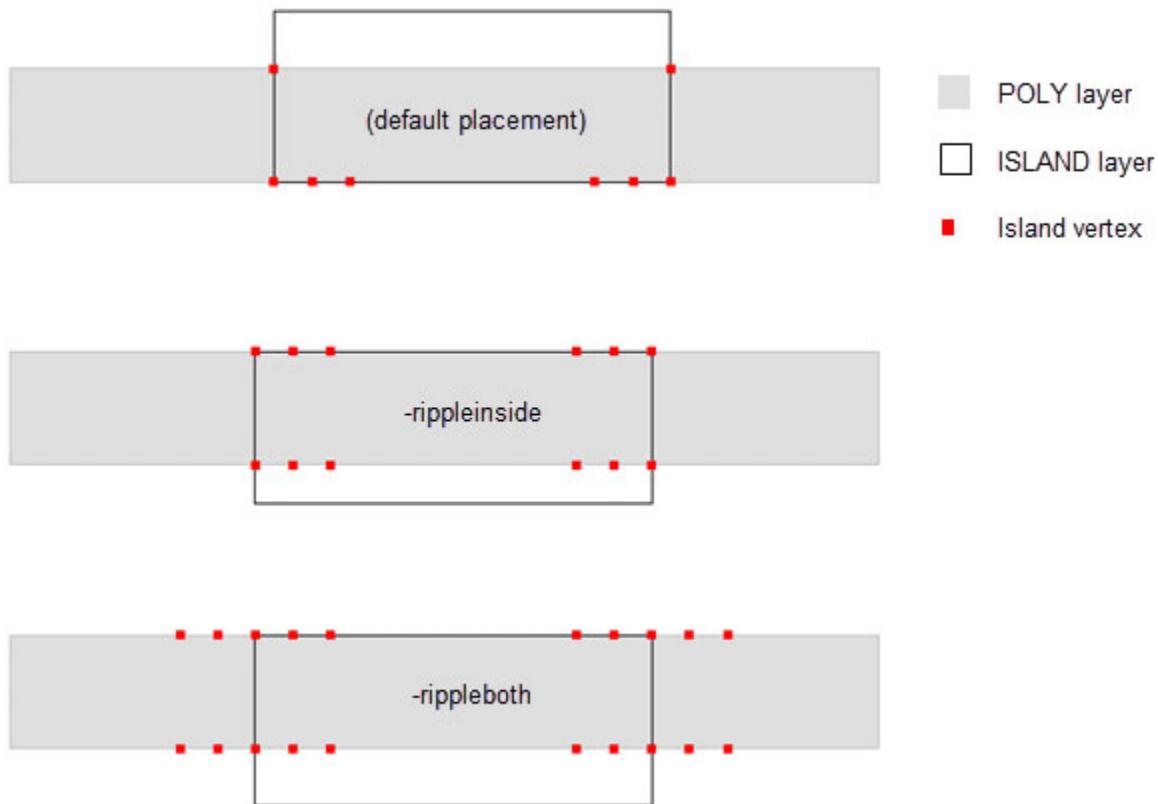
Additionally, the -ripples and -ripplelen parameters define the number of ripples and the distance between consecutive ripple points to be created at each intersection. Ripples are created toward the interior of the island edge only.

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.

Examples

The following example shows the various -ripple parameters. The ISLAND layers are three separate layers in the setup file: ISLA1, ISLA2, and ISLA3. Notice that although a vertex is generated where the island layer crosses the layer to be fragmented, only coincident edges receive ripples from that vertex by default.

Figure 4-29. Island Layer Fragmentation Example



```
modelpath .:models
optical_model_load OPTICAL_1 defaultopt
background dark

layer POLY hidden clear 0 1.0
layer ISLA1 hidden clear
layer ISLA2 hidden clear
layer ISLA3 hidden clear

denseopc_options dopc_opt {
    version 1
    background dark

    layer POLY opc clear mask 0
    layer ISLA1 island clear
    layer ISLA2 island clear
    layer ISLA3 island clear

    image optical OPTICAL_1 dose 1.0 aerial 0.3

    fragment_min 0.1
    fragment_max 0.4
    fragment_layer POLY {
        fragment_island ISLA1 -ripples 2 -ripplelen 0.2
        fragment_island ISLA2 -ripples 2 -ripplelen 0.2 -rippleinside
        fragment_island ISLA3 -ripples 2 -ripplelen 0.2 -rippleboth
    }
}
```

```
fragtypetag POLY fragtype_layer ifrags
OUTPUT_SHAPE fragment island_fragments ifrags -point1 0.0095 0.005
}

setlayer OPC_OUT = denseopc POLY ISLA1 ISLA2 ISLA3 OPTIONS dopc_opt \
    MAP POLY
setlayer PTS_OUT = denseopc POLY ISLA1 ISLA2 ISLA3 OPTIONS dopc_opt \
    MAP island_fragments
```

fragment_layer

[denseopc_options](#) Keywords

Fragmentation control sub-command block: Used to reset global fragmentation settings for special-case fragmentation.

Usage

Directional Fragmentation

```
fragment_layer layer [direction] '{  
    settings  
    ...  
}'
```

Marker Layer Fragmentation

```
fragment_layer layer {inside | outside | not_inside | not_outside} marker_layer  
[marker_layer]... [full_edge] '{  
    settings  
    ...  
}'
```

Arguments

- *layer*

The layer name to which the fragmentation parameters within this keyword apply.

- *direction*

An optional argument that causes fragmentation to be applied only to edges in the specified direction. Edges in any other direction are not fragmented. The *direction* can be one of the following keywords:

- horizontal
- vertical
- non90

The default behavior is to not restrict direction. 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. Except for the default case, it only applies to interfeature fragmentation, corner fragmentation, and maxedgelenlength fragmentation. Any of the layer-based commands, like coincident layer, apply to all edges irrespective of any direction specification.

Directional fragmentation and marker layer fragmentation should not be combined in the same denseopc_options block.

- **inside** *marker_layer* [*marker_layer*]...
- outside** *marker_layer* [*marker_layer*]...
- not_inside** *marker_layer* [*marker_layer*]...
- not_outside** *marker_layer* [*marker_layer*]...

For marker layer fragmentation, a required argument that causes fragmentation to be applied only to edges that are inside (outside) the polygons of the marker layers. To also include coincident edges, use **not_inside** or **not_outside**.

The marker layers must contain polygons indicating the area to be fragmented. At least one **marker_layer** must be specified.

Marker layers are merged to determine the region receiving fragmentation.

For **inside**, those portions of *layer* that lie strictly within the polygons on the marker layer are fragmented, and those regions that are outside are not fragmented. To fragment the coincident edges as well as the inside edges, use **not_outside**.

For **outside**, the regions of *layer* inside the marker polygons are not fragmented. To include the edges outside the marker layer and also the edges coincident with the marker layer, use **not_inside**.

If a marker layer polygon crosses a *layer* polygon, a fragmentation point is inserted on the *layer* edge at the crossing. A fragment break is inserted if it does not violate **fragment_min**.

Marker layer fragmentation requires that all **fragment_layer** statements in a **denseopc_options** block include an “outside” or “inside” clause. They cannot be used with an entire layer (no markers).

- **full_edge**

For marker layer fragmentation, an optional keyword that affects **fragment_corner** measurements within the **settings** block. It may only be specified with **inside** or **outside**.

By default (that is, when **full_edge** is not specified), only the portion of an edge that is part of the area being fragmented is considered. This means that when the marker layer intersects an edge, **fragment_corner** may not “see” some corners. Specifying **full_edge** causes the **fragment_corner** measurement to include the rest of the edge, though only the portion within the marker layer receives fragmentation. See “[Example 3](#)” on page 246.

- **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 4-12. Supported Fragmentation Keywords in fragment_layer Block

fragment_coincident	fragment_corner	fragment_island
fragment_inter ¹	fragment_interlayers	fragment_marker
fragment_max	fragment_min	fragment_minjog
fragment_nearly_coincident	fragment_nop	fragment_ripple
	fragment_visible	

1. Although fragment_inter can be specified any number of times, only the last specified fragment_inter command gets executed unless “fragment_layer_order full” is set.

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.

Whenever the litho setup file includes fragment_layer, automatic fragmentation is not calculated. Instead, the litho setup file must explicitly set both global fragment_min and fragment_max values and the ones inside the fragment_layer block. Calibre nmOPC will generate an error if these values are not explicitly defined.

If fragment_layer_order partial is set and you want default fragmentation applied after the settings in the fragment_layer block, you must define the fragmentation settings in the denseopc_options outside the fragment_layer blocks. For most fragmentation keywords, the default is off. The exceptions are island fragmentation (on when the setup file includes a layer of type “island”) and interfeature fragmentation (on when the litho setup file defines an optical model).

Examples

Example 1

This shows a fragment_layer block for the CORRECTION layer. It overrides the global fragment_island setting.

```
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 bold section indicates where a change occurred):

```
fragment_island DIFF -ripples 1 -ripplelen 0.1 -interdistance 0.45
```

The order in which the fragmentation is applied depends on [fragment_layer_order](#). See “Examples” in that topic.

Example 2

When using marker layer fragmentation, you must be careful to derive your marker layers such that no region is subject to more than one fragment_layer definition. For example, the combination following generates an error message “Can’t specify the same layer (“m1”) with and without a inside/outside marker.”

```
# This code will not run
fragment_layer m1 inside DIFF {
    ...
}
fragment_layer m1 {
    ...
}
```

Instead, the fragment_layer statements should be set up as an inside/outside pair:

```
fragment_layer m1 inside DIFF {
    ...
}
fragment_layer m1 outside DIFF {
    ...
}
```

Although the parser does check for the case noted above, it does not check for combinations of marker layer fragmentation that are otherwise problematic, such as overlapping marker layers.

For example, consider a full-chip POLY layer. Some regions are marked as LOGIC and some as SRAM. Others are not marked at all. The appropriate way to handle this is to derive a third layer (all_markers) and define fragmentation to occur outside of it, as in the following example.

```
modelpath /models/litho_models

layer POLY
layer LOGIC
layer SRAM

denseopc_options opts {
    version 1
    ...
    layer POLY    opc
    layer LOGIC   hidden
    layer SRAM   hidden
    layer ALL     hidden

    fragment_layer POLY inside LOGIC {
        ...
    }
```

```
fragment_layer POLY inside SRAM {  
    ...  
}  
  
fragment_layer POLY outside ALL {  
    ...  
}  
  
setlayer ALL = or LOGIC SRAM  
setlayer POLY.opc = denseopc POLY LOGIC SRAM ALL MAP POLY OPTIONS opts
```

Example 3

The `full_edge` keyword in conjunction with marker layers allows intrafeature fragmentation to detect a corner outside the region. This can be especially useful when a marker layer covers part of a 2D polygon.

Figure 4-30. Interfeature Fragmentation With `full_edge`

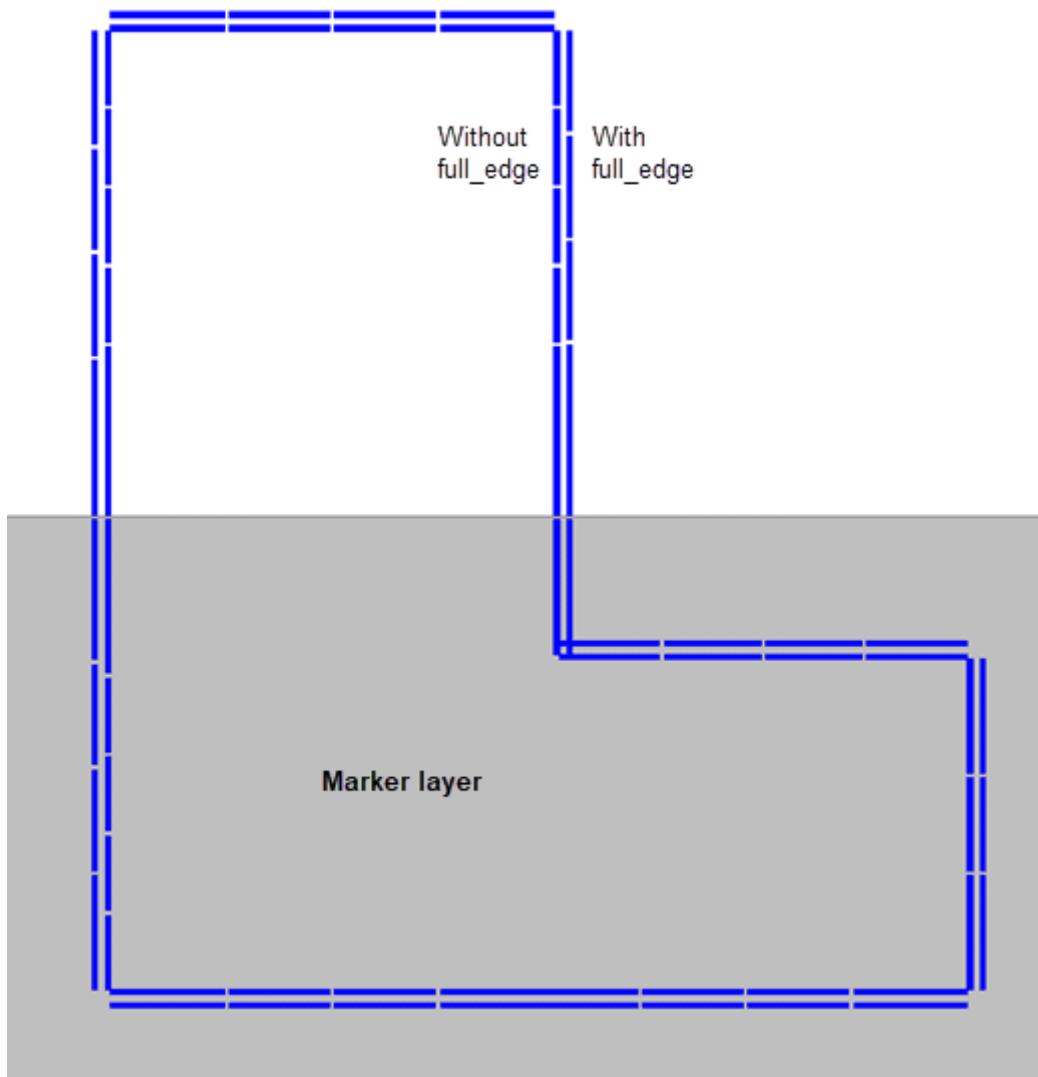


Figure 4-30 was produced with the following fragment_corner rules. (The same values were used both for inside and outside the marker layer, so the example code shows only one.) The inner set of fragments is the default behavior. The outer set of fragments are generated by adding “full_edge” and offsetting the placement for visibility.

```
fragment_layer opc_layer outside marker full_edge {  
    # rule 1  
    fragment_corner convex convex length >= 0.08 < 1 \  
        fragment 0.045 0.04 0.04 breakinhalf  
    # rule 2  
    fragment_corner convex concave length >= 0.08 < 1 \  
        fragment 0.040 0.04 0.04 breakinhalf  
    # rule 3  
    fragment_corner convex fragment 0.03 0.03 0.03 0.03  
    # rule 4  
    fragment_max 0.1  
}
```

The two edges that cross the marker layer have different fragmentation based on full_edge. Without full_edge, the long vertical edge is fragmented by rule 3, because only one convex corner is detected. The edge across from it is also subject to rule 3 for the same reason. With full_edge, the long vertical edge is fragmented by rule 1 and the edge across from it by rule 2 because the corners in the other region are “seen” by fragment_corner.

Related Topics

[fragment_layer_order](#)

fragment_layer_order

[denseopc_options](#) Keywords

Changes 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 uses the default order as listed in [Table 4-13](#). Some forms of fragmentation, such as **fragment_ripple**, are not included in the default order.

- **partial**

Specifies that fragmentation within a fragment_layer block is set by the order of the first appearance of command types, followed by any default fragmentation that was not listed. This is the default value.

For example, with partial ordering, if a fragment_layer block lists only **fragment_corner** (intrafeature fragmentation) and **fragment_ripple**, the order is **fragment_corner**, **fragment_ripple**, **fragment_island**, **fragment_inter**, and **fragment_max**. (The **denseopc_options** block must define **fragment_max**, and include an optical model and an island layer.)

- **full**

Specifies that the fragmentation within a fragment_layer block is performed in explicit order, and no additional fragmentation. This differs from partial in the following ways:

- If the fragment_layer block has multiple **fragment_corner** and **fragment_inter** specifications, intrafeature and interfeature fragmentation are performed repeatedly.

For example, specifying **fragment_corner**, **fragment_inter** and then another **fragment_corner** 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 island layer fragmentation is not specified, it is not performed.

Description

Note

 If a litho setup file contains a fragment_layer block or fragment_layer_order command, Calibre nmOPC uses [Custom Fragmentation](#). No automatic fragmentation settings are used. The litho setup file must explicitly define the fragmentation keywords, either in the denseopc_options block, the fragment_layer block, or both. (Island fragmentation occurs whenever a layer is defined as type “island”. Interfeature fragmentation uses an optical model, and does not run if no optical model is defined, as is common with LITHO NMBIAS.)

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”, which means the fragmentation order is executed by the order of the first appearance of the fragmentation command inside the fragment_layer_order block. However, if the fragmentation type is not explicitly stated in the fragment_layer block, Calibre will still execute according to the type’s default execution order.

[Table 4-13](#) shows the forms of fragmentation allowed within a fragment_layer command block along with the keywords that control them.

Table 4-13. Fragmentation Allowed in a fragment_layer Block

Default Order	Method	Keyword(s)
1	coincident	fragment_coincident
2	visible layer	fragment_visible
3	island layer	fragment_island
4	marker	fragment_marker
5	intrafeature	fragment_corner
6	interfeature	fragment_inter
7	maxedgeglength	fragment_max
Not default	All other methods	fragment_ripple, fragment_nop, etc.

The [fragment_min](#) keyword has no effect on the order of fragmentation if fragment_layer_order is set to partial and may be specified at any point in the fragment_layer block.

If you set fragment_layer_order to “full”, you are explicitly instructing Calibre nmOPC to execute the fragmentation commands in the order specified. If a form of fragmentation is not specified in the fragment_layer block, then that form of fragmentation will not be done. If you use this mode, you will be fully responsible for making your own custom fragmentation schemes since the full setting overrides ALL defaults.

The fragment_layer_order setting applies to *all* fragment_layer blocks. It is not possible to specify that one fragment_layer block be processed in user-defined order while another is

processed in default order. You must manually specify the operations in the correct order for each block.

Examples

Consider the following lines:

```
fragment_max 0.6
fragment_min 0.03

fragment_layer CORRECTION{
    fragment_max 0.4
    fragment_corner convex 0.06 0.07
    fragment_inter -interdistance 0.4 -ripples 2 -ripplelen 0.06
    fragment_corner concave 0.06 0.08
}
```

Because the keywords outside the `fragment_layer` block do not include a definition of `fragment_coincident`, `fragment_visible`, or `fragment_marker`, in none of the following cases do those fragmentation methods run, even though they are default.

fragment_layer_order standard

If `fragment_layer_order` is set to standard, the fragmentation occurs in the following order:

1. Visible and island fragmentation (`fragment_island`), if any layers of type visible or island are defined. The same values are used for CORRECTION and other layers.
2. Intrafeature fragmentation (`fragment_corner`). The CORRECTION layer uses different settings. Both `fragment_corner` rules are applied.
3. Interfeature fragmentation (`fragment_inter`). The CORRECTION layer uses different settings.
4. Maxedgelength fragmentation (`fragment_max`). The CORRECTION layer uses a setting of 0.4, while other layers use 0.6.

fragment_layer_order partial

If `fragment_layer_order` is set to partial, the fragmentation for the CORRECTION layer occurs in the following order:

1. Maxedgelength fragmentation.
2. Intrafeature fragmentation. Both `fragment_corner` rules are applied.
3. Interfeature fragmentation.
4. Island fragmentation. Notice that because island fragmentation is part of the default order, it still occurs though not explicitly listed.

fragment_layer_order full

If fragment_layer_order is set to full, the fragmentation for the CORRECTION layer occurs in the following order:

1. Maxedgelength fragmentation.
2. The fragment_corner convex rule.
3. Interfeature fragmentation.
4. The fragment_corner concave rule.

No island fragmentation is performed, as it is not included in the fragment_layer block.

Related Topics

[Select Correction Method](#)

[fragment_layer](#)

fragment_marker

[denseopc_options](#) Keywords

Adds fragmentation vertices based on proximity to markers on a marker layer.

Usage

fragment_marker *marker_layer* [-dist *x*]

Arguments

- ***marker_layer***

A required layer name identifying the layer in the layout containing marker shapes.

- -dist *x*

An optional argument specifying the maximum distance in microns at which marker fragmentation occurs. The default value is 0.010 microns.

Description

An optional keyword that defines fragmentation parameters for marker layer fragmentation. Marker layer fragmentation adds vertices to the target layer based on proximity to shapes on ***marker_layer***.

Marker layer fragmentation uses the center of the shapes on the marker layer as though they were source fragmentation corners in fragment_inter. In general, a good marker is a small square shape centered on the location where a fragment break should occur.

Examples

```
fragment_layer_order full

fragment_layer m3 {
    fragment_marker custom_frag_points
    fragment_min 0.030
}
```

Related Topics

[fragment_ripple](#)

[fragment_layer_order](#)

fragment_max

denseopc_options Keywords

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 Nyquist * 4.

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 is automatically reduced to tilemicrons/6.

- **-rem r**

An optional argument specifying the minimum allowed remainder. No edges shorter than r are produced. 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 fragment_max fragments. 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 (the range of correction effects), so longer fragment_max lengths can have a negative impact on runtime.

The fragment_max fragmentation algorithm attempts to break all edges equal to or longer than $2*\text{maxedge} + \text{fragment_min}$. These edges are split into two or more **maxedge** parts and one part greater than or equal to fragment_min. Fragmentation is symmetric with respect to the center of the edge. [Figure 4-31](#) shows the results of fragmenting an edge of length $2*\text{maxedge} + N$, where $N > \text{fragment_min}$.

No fragment smaller than fragment_min is generated by this operation. In the figure, 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 4-14](#) shows several ranges of values of

possible fragment lengths and shows how Calibre nmOPC splits the fragment using only `fragment_max` MAX. The `-rem` option changes MIN from `fragment_min` to r ; the `-minlen` option changes the conditions under which a fragment is not broken. (See “[Example 2](#)” on page 255.)

Figure 4-31. `fragment_max`

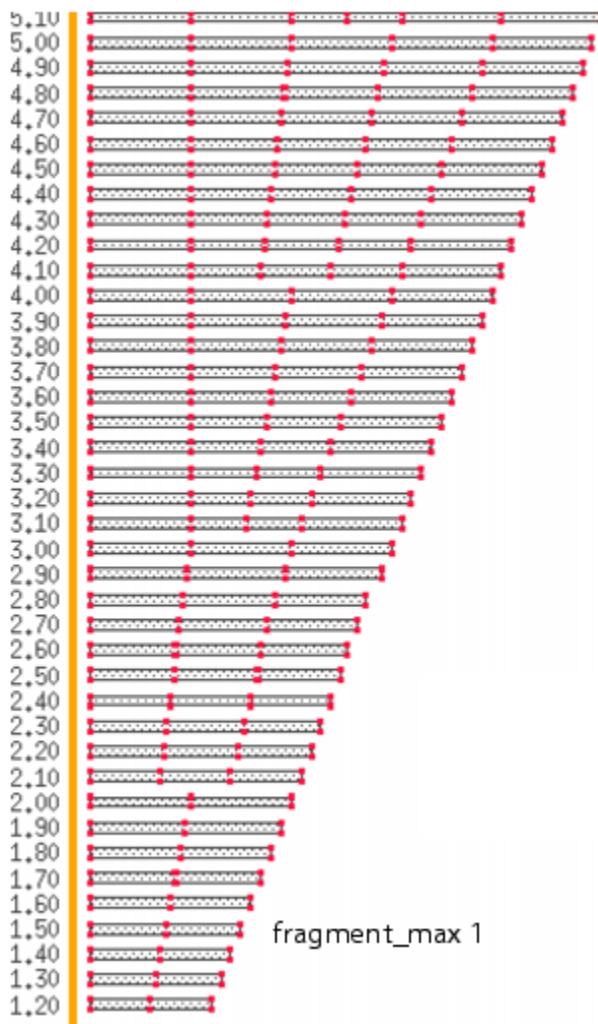


Table 4-14. Examples of Fragment Splitting

Conditions	Effect
$< 2\text{MIN}$ or $< \text{MAX}$	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).

Table 4-14. Examples of Fragment Splitting (cont.)

Conditions	Effect
> (3MAX)	Fragments of length fragment_max, with a centered remainder split into two or three equal fragments that are as long as possible with none longer than fragment_max, and none shorter than fragment_min. Note: if the original long fragment is <(3MAX) there will be no fragments of fragment_max.

Note

 By default, global fragmentation settings for fragment_max and fragment_min are calculated automatically by Calibre nmOPC. However, if the setup file uses custom fragmentation (fragment_layer), you must explicitly set fragment_max and fragment_min both inside and outside the fragment_layer block.

Examples

Example 1

This example shows setting fragment_max for custom fragmentation. The keyword must appear in the global area as well as the fragment_layer block.

```
fragment_max 0.6
fragment_min 0.03

fragment_layer CORRECTION{
    fragment_max 0.4
    fragment_corner convex 0.06 0.07
    fragment_inter -interdistance 0.4 -ripples 2 -ripplelen 0.06
    fragment_corner concave 0.06 0.08
}
```

Example 2

To protect shorter edges from being fragmented by fragment_max while still restricting overall fragment length, use the -minlen option.

This example shows applying the fragment_max of 0.08 only to edges at least 0.11 long. Edges that are 0.10 long are not fragmented even though they are larger than fragment_max.

```
fragment_layer TARGET {
    fragment_min 0.04
    fragment_corner
    fragment_inter off
    fragment_max 0.08 -minlen >= 0.11
}
```

Related Topics

[fragment_min](#)

fragment_layer

[tilemicrons \[Calibre OPCverify User's and Reference Manual\]](#)

fragment_min

[denseopc_options Keywords](#)

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 nmOPC. However, if the setup file uses custom fragmentation (**fragment_layer**), you must explicitly set **fragment_max** and **fragment_min** both inside and outside the **fragment_layer** block.

Examples

Consider the following fragmentation block:

```
fragment_min 0.011
fragment_max 0.4

fragment_layer M1 {
    fragment_min 0.015
    fragment_inter ...
}
```

The clear intention is for the common fragmentation to have a minimum fragment length of 11 nm but for layer M1 it should be 15 nm. Both scopes will use 11 nm, however, unless you add the enforce keyword:

```
fragment_min 0.011
fragment_max 0.4

fragment_layer M1 {
    fragment_min 0.015 enforce
    fragment_inter ...
}
```

The parser does flag the lack of enforce with the following lint warning:

```
WARN: The global fragment_min and fragment_layer fragment_min values do
      not match.
WARN: the fragment_layer fragment_min commands should be replaced with
      fragment_min <len> enforce to prevent inconsistent fragmentation.
```

Related Topics

[fragment_max](#)
[fragment_layer](#)

fragment_minjog

[denseopc_options Keywords](#)

Fragmentation control: Controls jog definition.

Usage

fragment_minjog *minjogsz*

Arguments

- ***minjogsz***

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 ***minjogsz*** 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

[denseopc_options](#) Keywords

Copies fragmentation from one layer to another where the layers' edges are within a specified distance.

Usage

```
fragment_nearly_coincident source_layer [-dist microns] [-cornerDist microns] [-nocorners]  
[-nonparallel]
```

Arguments

- ***source_layer***

A required argument specifying the layer that provides fragmentation.

- **-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.

- **-cornerDist *microns***

An optional argument indicating the minimum distance from a corner for a fragmentation point. The point is only copied to the current layer if it is at least *microns* from a corner. The default is 0 (no minimum distance).

- **-nocorners**

An optional argument that prevents corners on the source layer from creating fragmentation 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 keyword copies fragmentation from the source layer to the current layer when the layers' edges are within a specified distance and at no more than 45 degrees to each other. Contrast to `fragment_coincident`, which only copies fragmentation when the edges have no separation. Use `fragment_nearly_coincident` when 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.

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  
}
```

Related Topics

[fragment_coincident](#)

[fragment_layer](#)

[FRAGMENT_CONFORM](#)

fragment_nop

[denseopc_options Keywords](#)

Controls concurrency for fragmentation inside a fragment_layer block with fragment_layer_order full.

Usage

fragment_nop

Arguments

None.

Description

The optional fragment_nop command is used as separator between fragmentation commands when “fragment_layer_order full” is specified. It performs no operation, but does interrupt the concurrency of the fragment_corner command. It is ignored if fragment_layer_order is not set to full.

Examples

In the following code, the first two fragment_corner commands are run concurrently and then the second two fragment_corner commands. Finally, the fragment_max fragmentation occurs.

```
fragment_layer_order full

fragment_layer opc_target_frag{
    fragment_corner convex concave length > 0.24 <= 0.25 0.060 priority \
        corner2 0.050 0.050
    fragment_corner convex concave length > 0.25 <= 0.27 0.060 priority \
        corner2 0.050 0.050
    fragment_nop
    fragment_corner convex length > 0.09 <= 0.110 0.045
    fragment_corner convex convex length >= 0.123 <= 0.130 fragment 0.042
    fragment_max 0.125
```

Without the fragment_nop command, all four fragment_corner commands would be run concurrently.

Related Topics

[fragment_layer](#)

[fragment_layer_order](#)

fragment_optimize

denseopc_options Keywords

Performs fragment optimization based on specified corner and inflection parameters.

Usage

```
fragment_optimize [layer_name]
  [{inside | outside | not_inside | not_outside} region_layer]
  [corner_sharpness convex_corner_slope [concave_corner_slope]]
  [epe_threshold threshold] [reset_displacement]
```

Arguments

- *layer_name*
An optional argument specifying the name of a layer to have its fragments optimized. If this argument is not specified and there is only one layer of type opc in the design, that layer is used by default. If multiple OPC layers exist in the design, *layer_name* is required.
- {inside | outside | not_inside | not_outside} *region_layer*
An optional argument specifying a filter region. Only fragments whose whole edge meets the filter constraint are optimized.
- corner_sharpness *convex_corner_slope* [*concave_corner_slope*]
An optional argument specifying the intensity slope for a corner marker. Only the first argument is required; if the second argument is not specified, the value specified for the first argument is used for both. The default value is 0.5.
- epe_threshold *threshold*
An optional argument specifying a threshold in microns for deciding whether or not an edge's fragmentation needs optimization. If there is no site in between two corner makers whose EPE absolute value is greater than *threshold*, that edge's fragmentation is left unchanged. The default value is 0.0001 (0.1 nm)
- reset_displacement
An optional argument that resets the displacement of all fragments to 0 when optimization is complete.

Note



Calibre LPE runs must specify reset_displacement.

Description

This optional keyword optimizes model-based fragmentation on an OPC layer. It cannot appear before any fragment_min, fragment_max, and fragment_layer commands, because it is a post-fragmentation operation.

For the best results, use fragment_layer_order full.

Examples

Example 1 — fragment_optimize Explanation

The following command optimizes fragments as follows:

- Uses the layer “TARGET”
- Considers only convex corners with an intensity slope of at least 0.4 and concave corners with a slope of at least 0.5

```
fragment_optimize TARGET corner_sharpness 0.4 0.5
```

Example 2 — Litho Setup File Ordering

The following setup file lines show the ordering of fragment_optimize statements appearing after fragment_min and fragment_max statements. The file defines two layers of type opc:

```
denseopc_options dopc_opt {  
    layer m0 opc atten 0.06 pattern 0  
    layer m1 opc atten 0.06 pattern 1  
  
    ...  
    fragment_min 0.020  
    fragment_max 0.100  
  
    fragment_layer_order full  
    ...  
    fragment_layer m0 {  
        fragment_min 0.020  
        fragment_max 0.100  
    }  
  
    fragment_layer m1 {  
        fragment_min 0.020  
        fragment_max 0.100  
    }  
  
    ...  
    fragment_optimize m0  
    fragment_optimize m1  
    ...  
}
```

Related Topics

[fragment_layer_order](#)

fragment_preserve_length

[denseopc_options](#) Keywords

Fragmentation control: Specifies the longest fragment that can avoid implicit fragmentation.

Usage

fragment_preserve_length *microns*

Arguments

- ***microns***

A required argument specifying the length of the largest fragment that is left intact by implicit fragmentation. Values must be less than 10*fragment_min.

Description

This optional keyword controls implicit fragmentation around tile boundaries.

Calibre nmOPC automatically fragments edges that cross tile boundaries to ensure proper context and performance. It automatically determines the largest length based on fragment settings.

Some setup files tag long edges, and implicit fragmentation at tile boundaries creates unexpectedly short fragments. In these cases, use fragment_preserve_length to shield these edges from implicit fragmentation.

Examples

To preserve a 0.140 micron fragment:

```
fragment_preserve_length 0.141
```

Related Topics

[fragment_min](#)

fragment_ripple

[denseopc_options Keywords](#)

Repeats fragments from vertices created by corners, island fragmentation, or interfeature fragmentation within a fragment_layer block.

Usage

fragment_ripple edge_spec [fragment] **rule** [corner2 [fragment] **rule**] [**breakinhalf** [**enforce**]]

where:

edge_spec = {**type** [**subtype**]} [**type** [**subtype**]] [**length constraint**] [**length1 constraint**]
[**length2 constraint**] (See “edge_spec Arguments”)

rule = [**split n [force]**] |
[**len1...lenN [priority] [rem r]**] [**applyratio**] [**allowtrim amount**]
[**repeat length [max_length] [force]**] (See “rule Arguments”)

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 **types**, and the definitions are unambiguously different. (For example, concave convex or noncorner frag_corner noncorner frag_inter.)

If corner2 is specified, it must be followed by a fragmentation rule for the second endpoint.

In cases where a single-**type** rule and a two-**type** rule both apply to a corner, the more specific two-**type** 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** block’s **fragment_min** value.

The **enforce** option may only be specified immediately following **breakinhalf**.

edge_spec Arguments

- ***type***

A required argument specifying the type of fragmentation point to select.

concave or **convex** — Selects a concave or convex corner (not fragment).

noncorner — Selects a fragmentation point that is not a corner, but can be caused by fragment_corner fragmentation. Ripples extend in both directions from the point.

any — Does not restrict selection to concave, convex, or noncorner.

- ***subtype***

An optional further constraint on the ***type***.

Subtypes that can be specified with **concave** or **convex**:

end_adjacent — Corners must be on fragments that satisfy the line end or space end conditions.

non_end_adjacent — Corners that are on edges that do not satisfy the line end or space end conditions.

angle_90 — Corner must be a right angle.

angle_non90 — Corner must not be a right angle.

Subtypes that can be specified with **noncorner** or **any**:

frag_corner — Selects fragment points created by [fragment_corner](#) statements.

frag_inter — Selects fragment points created by [fragment_inter](#) statements.

frag_island — Selects fragment points created by [fragment_island](#) statements.

frag_marker — Selects fragment points created by [fragment_marker](#) statements.

- ***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*] [*length2 constraint*]**

Optional keywords that specify the length of the edges at the first and second endpoint, respectively. The constraint takes the form of a DRC constraint and the value is specified in microns. Both of these constraints are optional, and either may be specified without the other. For example:

```
length1 <= 0.15
length1 > 0.2 length2 > 0.2
length2 < 0.24
```

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_ripple convex length <= 0.685 split 4
```

The *n* value must be an integer between 2 and 40, inclusive. If the resulting fragments would violate [fragment_min](#), the command generates an error. The above example would likely generate an error because lengths down to 0 would be split into four fragments. You can prevent this by providing a lower bound, as in this example:

```
fragment_min 0.050
fragment_ripple noncorner 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 an endpoint to the first fragmentation vertex that will be produced.

- ***lenN***

The distance in microns from the previous fragmentation vertex to the next fragmentation vertex, measured sequentially from the endpoint inward.

- **priority**

An optional flag for two-*type* rules indicating which endpoint has priority over the other. Typically, there is no fragmentation if the remainder (rem *r*) is violated after the first fragment at each end is created. With the priority option, the fragment at the priority end is created if the remainder (rem *r*) is not violated. See “[fragment_corner](#) priority” for an in-depth discussion.

- **rem *r***

An optional argument specifying the minimum allowed remainder. If fragmentation would produce a fragment with length smaller than *r*, then it will not fragment the edge. If not specified, the default value used 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 end 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 end fragment(s) 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

The fragment_ripple command is an optional fragmentation parameter. This keyword gives you precise control over the number and length of intrafeature fragmented edges at corners and from fragmentation points inserted by other types of fragmentation such as island crossings. The fragmentation process occurs automatically before OPC begins.

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 end point for both ends of the edge cannot be reliably determined.

The workaround is when fragmenting long edges or fragments is to specify fragment_ripple twice, one with an upper bound of the promotion radius and the other with it as a lower bound.

Examples

Example 1: Adding Ripples

This example shows combining fragment_corner with fragment_ripple in a fragment_layer block. It first performs basic interfeature fragmentation, then intrafeature fragmentation using fragment_corner, and then uses fragment_ripple to insert ripple fragments on any edges long enough.

```
fragment_layer main {
    fragment_inter -interdistance 0.900 -ripples 0 -cornerDist 0.09
    fragment_corner convex fragment 0.033
    fragment_corner concave fragment 0.035
    fragment_ripple noncorner frag_corner noncorner frag_inter
        length > 0.056 < 0.084 fragment 0.022 corner2 fragment 0.022
}
```

See also the examples in `fragment_corner`, which shares many of the same fragmentation controls.

Example 2: Simple edge_spec

A `fragment_ripple` command can be as simple as a single corner type and rule for one ripple; for example:

```
fragment_ripple convex 0.4
```

This identifies every fragment at a convex corner and adds a fragmentation vertex 0.4 microns from the corner if it fits.

Example 3: Corner Fragmentation

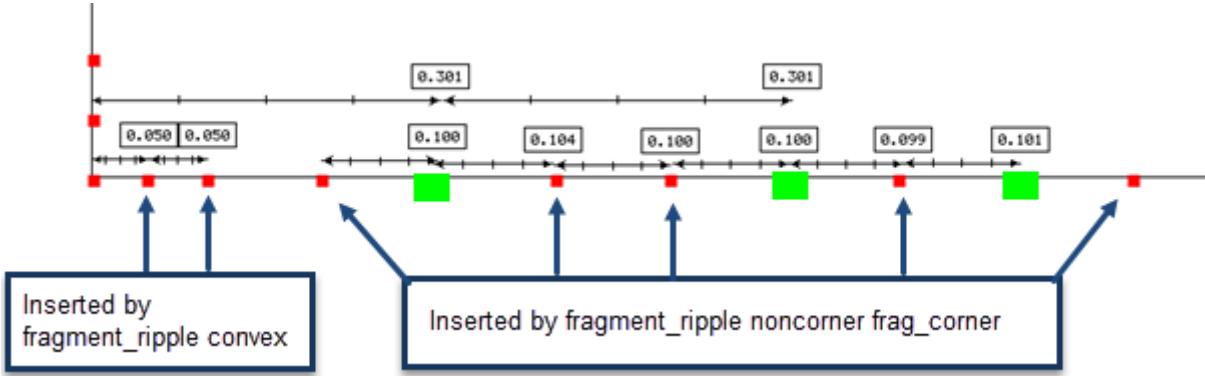
A `fragment_ripple` command works differently on a fragment depending on whether it is selected with a corner (“concave” or “convex”) or a “noncorner frag_corner” edge specification. The following code creates long corner fragments, and then applies ripples:

```
fragment_layer_order full

fragment_layer rx {
    fragment_min 0.05
    fragment_corner convex 0.3 0.3 0.2
    fragment_ripple convex 0.05 0.05
    fragment_ripple noncorner frag_corner length < 0.7 repeat 0.1
}
```

The result of this is that from a convex corner, the ripples are 0.05 um, but from the corner ripples themselves (indicated in green in the figure) there are additional ripples of 0.1 um.

Figure 4-32. `fragment_ripple` corner Versus `frag_corner`



Related Topics

[fragment_corner](#)

fragment_visible

[denseopc_options Keywords](#)

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 OPC.

freeze_skew_edges

[denseopc_options](#) Keywords

Freezes skew edges and neighboring fragments.

Usage

freeze_skew_edges [off | on] [is_error | no_error] [*max_log_limit*]]

Arguments

- off | on

Disables (off) or enables (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 nmOPC 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 nmOPC movement.

Note

 By default, **freeze_skew_edges** freezes skew edges in place (on) and continues the OPC 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

[denseopc_options](#) Keywords

Used to specify how 45-degree angles are moved during OPC.

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 OPC. 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 OPC.

```
grids_for_angle_45_snap 0.005
```

Related Topics

[min_dog_ear_length](#)
[step_size](#)

image

denseopc_options Keywords

Used to create an ideal image condition.

Usage

Syntax 1: Standard Model, Single Patterning

image

```
[mask_size {value | {layer value}...}]  
optical {opt_modelname [dose dose]}... [ddm [{layer model}]]...  
{resist resist_modelname | aerial threshold}  
[etch_model etch_modelname]
```

Syntax 2: Standard Model, Multi-Patterning

image

```
{[pattern num] [mask_size {value | {layer value}...}]  
optical {opt_modelname [dose dose]}... [ddm [{layer model}]]...  
{resist resist_modelname | aerial threshold}  
[etch_model etch_modelname]}...
```

Syntax 3: Litho Model, Single Patterning

```
image litho_model_directory [[mask N] [focus fhm] [dose d] [bias b]]...  
[aerial {val | model}] [thr_delta val[%]]  
[no_etch | with_etch] [no_flare] [no_shadow_bias][no_black_border]
```

Syntax 4: Litho Model, Multi-Patterning

```
image litho_model_directory [pattern num [[mask N] [focus fhm] [dose d] [bias b]]...  
[aerial {val | model}] [thr_delta val[%]]  
[no_etch | with_etch] [no_flare] [no_shadow_bias][no_black_border]]...
```

Arguments

- pattern *num*

An optional parameter used to specify to which pattern the image parameter belongs (used for multi-patterning). All the patterns must be specified as part of a single image command. For example:

```
image pattern 0 optical opticsfile_1 dose 1.0 resist resistfile1  
pattern 1 optical opticsfile_2 dose 1.0 resist resistfile2
```

Note

 For non-litho model implementations, pattern definitions must be the same between the [background](#) (for [denseopc_options](#) block), [image](#), and [pw_condition](#) statements.

Non-Litho Model Usage Arguments

- **mask_size {value | {layer value}...}**

An optional parameter that sizes the mask prior to performing simulation and correction. The mask sizing operation uses the following steps:

- a. The sizing specified with `mask_size` happens for all layers that are used in simulation.

If you give it one *value*, it performs x and y sizing with that *value* for all layers. Otherwise, it sizes each *layer* with the specified per-layer *value*. For example:

```
image mask_size 0.001 optical opt ...  
image mask_size poly 0.001 0.002 poly_sraf_e1 0.001 0 \  
poly_sraf_e2 0.001 ...
```

Note

 With asraf layers, positive bias makes the shapes defining the negative SRAFs *larger*. This means that the holes get larger. This is the reverse of the behavior when the negative SRAFs are included as holes in the main feature layer.

- b. The sizing specified with the `bias` keyword in the [layer \(for denseopc_options Block\)](#) command is applied to each layer. For example:

```
layer M1 opc clear bias 0.001 0.003
```

- c. Any corner chipping (using [mask_chip_corner](#)) is applied last.

```
mask_chip_corner 0.002
```

- **optical {opt_modelname [dose dose]}...**

A required parameter specifying the optical model exposures and doses used for this image operation. There can be up to three exposures specified. The default dose is 1.0.

When using process window conditions, all process window conditions (including the nominal) must have the same number of exposures, which must also match the number of exposures specified in the “background” keyword.

- **dose dose**

An optional parameter that specifies a dose for this image operation. It overrides the default dose specified in the `layer` command, if any. The default dose is 1.0.

- **ddm [{layer model}]...**

An optional argument that specifies a DDM model. DDM models are specified per layer. For example:

```
image optical o1 dose 1.01 resist cm1_res ddm layer1 ddm_model_1 \  
layer2 ddm_model_2
```

When using DDM models, make sure that each of the layers you intend to apply the DDM model to is explicitly defined, otherwise an error is issued. For example:

```
layer target opc clear
layer block correction clear
layer ph000 correction clear
layer ph180 correction phase180
...
image optical optmod resist rmod ddm ph000 ddm_ph000 /
    ddm ph180 ddm_ph180
```

In this case, the block, ph000, and ph180 layers are to have DDM applied. However, the run fails because the block layer is not explicitly defined in the image command for DDM. The correct example is as follows:

```
layer target opc clear
layer blk correction clear
layer ph0 correction clear
layer ph180 correction phase180
...
image optical optmod resist rmod ddm block ddm_blk /
    ddm ph000 ddm_ph000 ddm ph180 ddm_ph180
```

- **resist *resist_modelname***

A required argument for non-litho model usage that generates a print image contour using the specified resist model. Use this command to perform CD-checking applications only.

- **aerial *threshold***

A required argument for non-litho model usage that generates an aerial image threshold at *threshold* to produce the output contour. Use this command to perform critical feature detection.

- **etch _model *etch_modelname***

An optional parameter specifying the etch model used for this image operation.

Litho Model Usage Arguments

See “[Load Model Information With a Litho Model](#)” on page 48.

- ***litho_model_directory***

A required argument that specifies the name of a directory containing optical, resist, and mask models along with a *Lithomodel* file. The litho model is loaded automatically from the model path (see “[Lithomodel \(Litho Model Format\)](#)” on page 359).

- **mask *N***

An optional argument that specifies the mask number as defined in *Lithomodel* to which the focus, does, and bias parameters apply. The default if not specified is mask 0.

- **focus *fnm***

An optional argument that specifies a focus condition to be used for this mask. This parameter maps to the focus parameter used by the optical model specified in *Lithomodel*.

- dose d

An optional argument that specifies a dose to be used for this mask. Note that if a base dose for this mask was specified in *Lithomodel*, the actual dose is calculated by multiplying this value with the base dose in the litho model. The default is 1.0.

- bias b

An optional argument that specifies a bias in microns to be used for sizing this mask before using it in simulation. Note that if any biases were specified for individual mask layers in the litho model, those biases are added to the value specified here. Bias values are used only in simulation.

Note

When using asraf layers, a positive bias value increases the size of the shapes defining the negative SRAFs. This means the holes get increased in size as well. This is the reverse behavior when negative SRAFs are included as holes in the main feature layer. In that case, positive bias makes the holes smaller.

- aerial *val*

An optional argument that sets the threshold value for the aerial image for all process windows conditions (including nominal) to produce the simulation contours. This option overrides the resist model specified in the litho model.

- aerial *model*

An optional argument that is similar to aerial *val*, but the constant threshold value is taken from the image_threshold value specified in the litho model.

- thr_delta *val[%]*

An optional argument that specifies a signed offset to modify the image_threshold parameter in the litho model. If the percent sign is present, the offset is a relative percentage. Otherwise, the offset is an absolute delta.

This keyword is permitted only if aerial *model* is specified.

- no_etch

with_etch

If the litho model contains an etch model, one of “no_etch” or “with_etch” must be specified explicitly. If the litho model does not contain an etch model, these keywords are ignored.

no_etch — Disables using the etch model for this image and pattern only.

with_etch — Enables using the etch model for this image and pattern only.

- no_flare

If the litho model has an EUV flare model, this keyword disables using the flare model for all process window conditions including nominal.

- `no_shadow_bias`

If the litho model has an EUV shadow bias model, this keyword disables using the shadow bias model for all process window conditions including nominal.

- `no_black_border`

If the litho model has an EUV black border model, this keyword disables using the black border model for all process window conditions including nominal.

Description

A required command that sets the parameters for the “ideal” image condition, including dose, mask size, resist, and aerial threshold.

The single nominal image to correct is specified by the `image` command. Other images can also be added to the cost function using the [pw_condition](#) command.

Examples

The following example creates an aerial image contour using the optical model named o1, an override dose of 0.96, and a threshold value of 0.3:

```
image optical o1 dose 0.96 aerial 0.3
```

The following example creates an aerial image using two optical models (o2 and o3):

```
image optical o2 dose 0.96 aerial 0.3 o3 dose 1.02 aerial 0.3
```

jog_freeze

[denseopc_options](#) Keywords

Prevents a fragment from moving during OPC 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 OPC iterations.

Description

The jog_freeze command will prevent an edge from moving during OPC iterations if the edge is a jog (bounded by 1 convex and 1 concave corner) and its length is <= *value*.

jog_ignore_metric

denseopc_options Keywords

Changes fragment corner classification when shared with a short jog.

Usage

jog_ignore_metric *len*

Arguments

- *len*

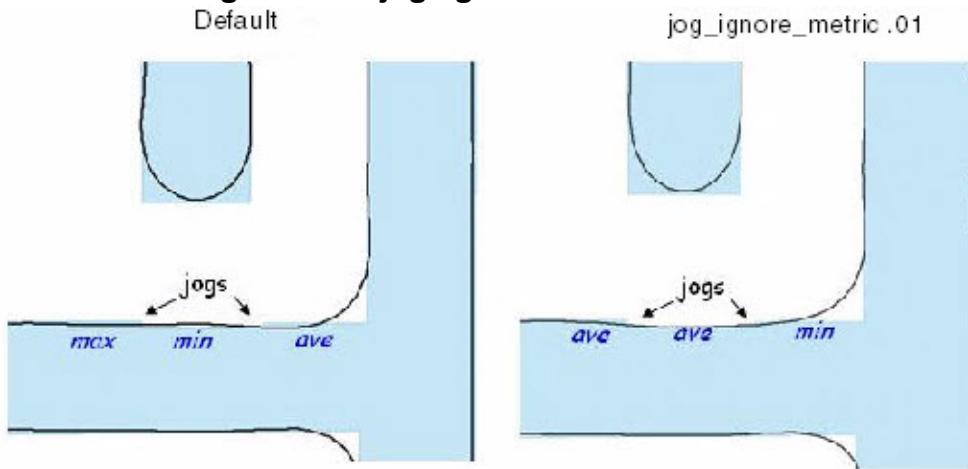
Define jog fragments with length $\leq \text{len}$ as jogs for this command. Valid values are > 0 .

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 4-33](#) shows how the metrics are evaluated using this command.

Figure 4-33. 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 OPC output that more closely match the original target.

Performance

$O(N)$ to runtime where N is number of fragments.

lapi_exec_tcl

[denseopc_options](#) Keywords

Enables tagging commands not accepted in the tag scripting section (formerly the <script>...<endscript> method).

Usage

```
lapi_exec_tcl
  {
    tcl_script
  }
  [-out tag_name]... [-outLayer name]...
  [-script_variable {name value}...]
```

Arguments

- ***tcl_script***
Specifies the custom Tcl script to execute at run time.
- **-out *tag_name***
Tag name declared as an output for lapi_exec_tcl. All tags that are created inside this section must be declared here, even if they are not used outside this section.
- **-outLayer *name***
Specifies the output layer name declared as an output for lapi_exec_tcl. All output layers that are written to inside this section must be declared here.
- **-script_variable {*name value*}...**
Makes the variable *name* with a *value* available to the Tcl interpreter executing the ***tcl_script*** section.

Description

This Tcl script can contain commands that are not accepted in the custom tag scripting section, (generally commands that work on a fragment-by-fragment basis), such as:

- [DBU2UU](#)
- [DENSE_CD](#)
- [DENSE_EPE](#)
- [FRAGMENT_GET_DISPLACEMENT](#)
- [FRAGMENT_ITERATOR first/next/release](#)
- [MEASURE](#)
- [MEASURE_PER_FRAGMENT](#)

- [MEASURE_PER_FRAGMENT_QUERY_COUNT](#)
- [MEASURE_PER_FRAGMENT_QUERY_FIRST](#)
- [MEASURE_PER_FRAGMENT_QUERY_NEXT](#)
- [TAG add | remove | ismember](#)
- [TAG and | or | subtract](#)
- [TAG fromlist](#)
- [TAG size](#)
- [TAG tolist](#)

This Tcl script is not fully checked for syntax errors at parse time, since its execution flow may depend on individual fragments.

Note

 Do not place any of the non-Tcl scripting commands inside the lapi_exec_tcl block (for example, [sraf_print_avoidance](#)).

However, partial checking is done by running it on a dummy layout at argument checking time. This can catch errors that do not depend on individual fragments. For example:

```
lapi_exec_tcl {  
    qqqqq # this error (invalid Tcl command) will get  
           # caught at parse time  
    FRAGMENT_ITERATOR first M1 -out fragid  
    if {$fragid == 2000 } {  
        qqqqq # this error (invalid Tcl command) may not get  
               # caught at parse time  
    }  
}
```

Tag sets from the tag script section, generated before the lapi_exec_tcl call, will automatically get imported into the lapi_exec_tcl section.

Tag sets generated in the lapi_exec_tcl section will have to be declared with -out. Multiple lapi_exec_tcl calls can exist within the same setup file.

Tcl Result

None

Performance

Depends on the commands called inside the lapi_exec_tcl section.

Scripts that are overly complicated and loop over fragments multiple times may degrade the performance.

The Tcl overhead incurred while executing these scripts is quantified by the **TCL_INTERPRETER_OVERHEAD** timer.

Examples

```
NEWTAG all M1 -out M1_all
lapi_exec user_c_lapi_1
set iter1 2
set iter2 4
OPC_ITERATION $iter1
lapi_exec_tcl {
    OPC_ITERATION $iteration
    for {set C [FRAGMENT_ITERATOR first M1 -out fragid]} { $C != 0} \
        set C [FRAGMENT_ITERATOR next M1 -out fragid]
    if { $fragid == 10 } {
        TAG add tag1 $fragid
    }
}
} -out tag1 -script_variable { iterations $iter2 }
```

layer (for denseopc_options Block)

[denseopc_options Keywords](#)

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

Usage

Syntax 1: Standard (Non-Litho) Model

```
layer layer_name layer_type transmission [mask n]... [pattern {0 | 1 | 2 | 3}] [bias x y]
```

Syntax 2: Litho Model

```
layer layer_name layer_type [mask n]... [pattern {0 | 1 | 2 | 3}] [mask_layer n]
```

Syntax 3: Topography Model Underlying Layer

```
layer layer_name underlying {active | poly | finfet | active2}
```

Syntax 4: VEB Model Underlying Layer

```
layer layer_name etch_underlying n
```

Arguments

- ***layer_name***

A required argument that specifies the name of the layer. Calibre nmOPC uses this name in the rest of the setup file to refer to this layer.

Layer names must be alphabetic or numeric strings (or both) less than 32 characters in length.

- ***layer_type***

A required argument that specifies how the layer is used by Calibre nmOPC and the aerial image simulator. The following table lists the valid layer_types and their related function.

Table 4-15. 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 OPC, unless correction layers are present. • The output (the OPC 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 input to the simulator to determine which edges are movable.</p> <p>When using the layer command with a litho model, do not map opc layers to mask layers.</p>
correction (optional)	<p>Correction layers are special layers whose edges are moved during OPC. Typically, you use correction layers when you need multiple masks to create the desired image, as with phase shifting masks. They are used only with the Calibre nmOPC batch tool or Calibre WORKbench application.</p> <p>During processing, a correction layer has:</p> <ul style="list-style-type: none"> • Its edges fragmented. • Its fragments are mapped to fragments on an opc layer. • Its edge fragments are 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. Correction layers can also be used with retarget layers.</p>
contact (optional)	<p>This layer is exactly like a hidden layer, with the additional property that if it is used with a wafer_enclose or wafer_enclosedby check, then its corners will be rounded by a predefined amount given by the Nyquist rate, automatically.</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 nmOPC, including serving as retargeting layers and marker layers, or for re-opc and “no opc” regions.</p>

Table 4-15. layer Command layer_types (denseopc_options) (cont.)

Layer Type	Function
target (optional)	<p>Target layers are for retargeting. Use them when the opc layer and desired target are too far apart.</p> <p>If a target layer is defined, the sites are shifted to it. This layer type supports fragmentation.</p> <p>Layers of this type should not be used in OPC iteration commands or MRC checks.</p>
visible (optional)	<p>Visible layers contain geometries that will not be 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 does not need 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>
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.
sraf (optional)	This layer behaves like an opc layer with respect to simulation and fragmentation, but the layer will not be affected by regular opc iterations. This is intended to be used for SRAF optimization commands such as sraf_print_avoidance . SRAF layers are unaffected by global settings and should have associated fragment_layer and mrc_rule definitions.
asraf (optional)	This layer represents holes on the mask and behaves like an opc layer with respect to simulation and fragmentation. However, it will not be affected by regular OPC iterations. This is intended to be used for SRAF optimization commands such as sraf_print_avoidance .
underlying (topography models only)	See the “underlying” keyword description.
etch_underlying (VEB retarget only)	See the “etch_underlying” keyword description.

- ***transmission***

For the first syntax, a required argument that defines the layer’s optical transmission.

The mask transmission is defined layer by layer, with the sum of all layers added to the background transmission to generate the ideal mask transmission. Layers which have the same transmission are automatically merged if they overlap, hence overlapping layers will not affect simulation. Layers with different transmissions are not merged, hence their transmissions will add if they overlap. The transmission values are ignored for hidden, contact, and island layer types.

The layer's optical transmission type must be the complement of the background, that is, you cannot specify clear features if you specify a clear background. The following table lists the valid *type* choices.

Table 4-16. layer Command transmission Argument Settings (denseopc_options)

Transmission	Description
clear	Defines the layer's optical transmission to be 100% (clear features). Transmission added to mask will be {0 - bg} for this layer.
dark	Defines the layer's optical transmission to be 0% (dark features). Transmission added to mask will be {1.0 - bg} for this layer.
<i>real imag</i>	<p>Defines the layer's optical transmission to be as specified by the real and imaginary value pair. Note that the attenuated syntax for the transmission is recommended over the REAL,IMAGINARY pair syntax.</p> <p>If you specify background with the “<i>real imag</i>” pair for a specified layer transmission value, the resulting transmission value for the output layer with a REAL,IMAGINARY pair is resolved relative to the background pair. The layer transmission value is resolved according to the following equation:</p> $\text{background } (\textit{real imag}) + \text{layer } (\textit{real imag}) = \text{transmission_value}$ <p>For example:</p> <pre>background 1 0 name opt type chrome visible 0 0</pre> <p>The background is specified with a (1,0) pair, while the chrome layer is defined with a transmission value of (0,0). With a clear background (1,0) and a layer specified with (0,0), the resulting transmission layer is (1,0) + (0,0) = clear (1,0).</p> <pre>background 1 0 name opt type p180 visible -1 0</pre> <p>In this case, the layer p180 is defined as (-1,0). With clear background (1,0) and a layer specified with (0,0), the resulting transmission layer is (1,0) + (-1,0) = dark (0,0).</p>

Table 4-16. layer Command transmission Argument Settings (denseopc_options)

Transmission	Description
attenuated <i>factor</i> atten <i>factor</i>	Defines the layer to be an attenuated phase shifting layer and the optical transmission to be a percentage of incident light, as specified by <i>factor</i> . When Calibre LITHO tools encounter the attenuated <i>factor</i> command, they translate the factor into a real imaginary pair such that the attenuated background has a transmission value: $RE(\text{layer}) = -\sqrt{\text{percent}/100}$ $IM(\text{layer}) = 0$ The maximum allowed attenuation factor is 36. Transmission added to mask will be $\{1.\text{value} - \text{background}\}$ for this layer.

- **underlying {active | poly | finfet | active2}**

Specifies a topography model underlying layer. This layer is similar to a hidden layer in that it is not fragmented and has no optical properties, but it is used when performing OPC with a topography model (refer to the *Calibre WORKbench Topography Modeling User's and Reference Manual*).

For OPC with topography models, the extra parameter specifies the type of layer. Supported layers are active, active2, poly, and finfet. The finfet layer is a special layer used only for topography modeling.

The following is an example of specifying an underlying layer type:

```
layer B1 underlying poly
```

Note

 The *layer_name* for underlying layer types should match the layer names defined in the topographical model “density underlying *layer1 layer2*” keyword.

In multi-patterning Calibre nmOPC runs, the same underlying layers are used for all patterns.

- **etch_underlying *n***

Specifies a VEB model underlying layer. This layer is similar to a hidden layer in that it is not fragmented and has no optical properties, but it is included in VEB simulations (refer to “[The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC](#)” appendix for information on VEB).

Note that an underlying layer has an additional parameter. For VEB models, it specifies a VEB underlying etch layer with an ID of *n*, pointing to the SKERNEL with the underlying=1 argument in the VEB etch model:

```
layer B1 etch_underlying <N>
```

- mask {0 | 1}

An optional argument for multiple exposures. The mask number must be entered separately for layers on each exposure. The default mask number is 0, hence this keyword is not needed for a single exposure simulation.

- pattern {0 | 1 | 2 | 3}

An optional argument that specifies the patterning number for multi-patterning. The default pattern number is 0. For single-patterning processes, there is no need to specify the pattern option.

The pattern argument enables certain changes to support multiple patterning. OPC is performed on all masks simultaneously. Unlike double exposure, each iteration includes a separate simulation step per pattern mask. (For example, for triple patterning, there would be three separate simulation steps per iteration.) EPEs are calculated separately.

- bias x y

An optional argument that sets the X and Y direction biasing to be applied to the given mask layer prior to simulation. The biasing is performed after the mask_size option in [pw_condition](#) (if present) and [mask_chip_corner](#) (if present). The biasing is applied to masks for all process window conditions.

- mask_layer n

Maps this layer to a mask_layer template in the *Lithomodel* file. The mask_layer specifies transmission, geometry preprocessing, ddm model, and so on. The default value is 0, so this parameter is only needed with models containing several mask_layer templates.

Note that if a layer is mapped to a mask_layer that specifies XBIAS, YBIAS, XSHIFT or YSHIFT, this will cause Calibre to clone all rotated or reflected cell transforms.

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:

- The transmission and *mask_num* values are only used with the opc, visible, and correction layer types.
- Calibre nmOPC does not simulate layers designated as the hidden layer type, but these layers can otherwise be used in all Calibre OPCverify commands.

Examples

In this example, POLY is the target layer, SRAF contains subresolution assist features, GATE is a marker layer, and CC is a layer that will be used for enclosure checks.

```
layer POLY opc dark
layer SRAF visible dark
layer GATE hidden dark
layer CC contact dark
```

line_end_fragment

[denseopc_options](#) Keywords

Enables or disables 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.

lint

denseopc_options Keywords

Checks a setup file or input geometries for suboptimal settings or other performance issues.

Usage

lint {none | setup}

Arguments

- **none**

Disables any lint messages.

- **setup**

Processes through the setup file (once per run) and flags settings that are not necessarily incorrect, but might produce undesirable results or poor performance. This is the default.

Description

The lint command allows you to run a check on either your setup file or input geometry for any potential problem areas that may affect performance. Items flagged during the check are not necessarily errors, but issues that, if corrected, can better optimize the performance of the run.

Settings in the setup file that are not necessarily incorrect, but might be a misuse of the options or might give suboptimal quality of results or performance are flagged as warnings. These warning usually have the following format:

<command>: <problem>. <recommended_solution>

If the Calibre OPCverify log_options command is used to disable parsing messages, the lint output is suppressed.

Custom lint checks can be added by creating a lint checking setup file and using the [lintmodel](#) command.

Possible Warnings

For [setlayer veb_retarget](#):

- “etch_imagegrid_microns” should be set to 0.01.
- “mask_chip_corner” should be set to 1.3*nyquist.

For [setlayer denseopc](#):

- The svrf PRECISION is too high. Set it to no more than 40000 to get correct opc results.
- Using svrf PRECISION of more than 10000 in ras mode may lead to incorrect opc results.

- “opc_grid_multiplier” was automatically reduced to prevent the effective opc precision from exceeding 10000 in ras mode.
- “tilemicrons” should be set to at most 35 microns for 32nm technology, and at most 50 microns for 45nm technology.
- “mask_sample_grid” should be set to “rsm”.
- simulation_consistency set to *<value>*: recommended setting for OPC is 2.
- “image”, “pw_condition”: all CM1 models used should have smooth neutralization set.
- “image”, “pw_condition”: At least one dose is set to less than 0.95 or more than 1.05.
- “image”, “pw_condition”: Using an etch model during simulations can slow down nmOPC. The recommended use model is to use veb_retarget, then set the output layer as a retarget layer.
- “pw_condition”: At least one process window condition has zero weight (so it is effectively ignored). You should set it to a non-zero weight and comment out the pw_condition statement.
- “pw_condition”: Using weight-based process window conditions is not recommended. Use tolerance based pw_condition specification instead.
- “max_opc_move”: The default maximum inside/outside edge movement is 0.08. You should decrease it, if possible, to get a speedup.
- “epe_spacing”: epe_spacing is too large. It should be smaller than half the minimum fragment length.
- “max_iterations”: By default, all iterations are done using the specified process window conditions. You should use the “nopw” option for “max_iterations” to get a speedup.
- “step_size”: step size is more than .5 nm. Decrease it to get more consistent opc output.
- “feedback”: The feedback is likely too high or too low. The suggested range is -0.8 to -0.05.
- Layer *<layer>* is corrected by opc, but there is no setlayer command (or SVRF command) that requires it.
- Some fragments can be smaller than 2*nyquist, according to fragmentation parameters. This might be suboptimal.
- “fragment_layer_order full” is used. User is responsible for specification of all of the fragmentation since this is a strict user-controlled mode.
- “opc_grid_multiplier”: You should use “opc_grid_multiplier” for better consistency.
- “mrc_rule”: At least one mrc rule doesn't have a good “length” spec. This means small jogs will get the same rule as larger fragments, which is probably undesired (might lead to undercorrection of the second fragment adjacent to a concave corner, for example).

- “fragment_max”: Maximum fragment length is set to *<value>*. It should be set to no more than *<value>* to avoid an increase in the interaction distance.
- fragment_max is not specified which may lead to large fragments which, in turn, may create bad opc results.
- fragment_corner: more than one length constraint was specified, only the last will be used.
- “jog_ignore_metric”: recommended setting is in the range nyquist/4 to nyquist/2.
- “jog_freeze”: recommended setting is in the range nyquist/4 to nyquist/2.
- “algorithm”: this setting will be ignored in double patterning mode.
- “algorithm”: this setting will be ignored in pbvband pwopc mode.
- algorithm 2 uses sites, not measurement locations; use SITES_DUMP command instead of OUTPUT_MEASUREMENT_LOCATIONS.
- fragment_corner length*<N>* must be specified and <*<value>*> when a corner*<N>*_next corner or length*<N>*_next is specified in order to limit fragmentation inconsistencies
- “SITES_*”: if too many user sites are created, the overall runtime will degrade because of intensity interpolation time increase.
- Tagscript command(s) increased interaction distance by *<value>*. Check NEWTAG sequence or other commands.
- One or more layers are defined as an "sraf" type. User defined fragment_layer block and mrc_rule block are required for these layers.
- “mrc_mode”: a setting of “0” is unsupported and will be ignored.
- “mrc_mode”: recommended setting is “2”
- “At least one MRC_RULE constraint for layer *<layer>* is greater than the minimum fragment length *<value>*. This can cause unpredictable OPC corrections.”
- “mrc_rule_priority last” would ignore earlier mrc rule(s) and only apply the last specified mrc_rule in case multiple rules apply to a fragment.
- unbounded ‘NEWTAG edge’ constraint with corner specifications will lead to the OPC output inconsistency
- Multiple fragment_inter commands are detected. Fragmentation is in the partially ordered mode and only the last fragment_inter command will be executed.
- non-Lithomodel syntax detected: switch to Lithomodel syntax to benefit from the rule file simplification and utilize full spectrum of imaging capabilities.
- fragment_optimize is used with the default processing order. Use “fragment_layer_order full” for the best control over fragmentation.

- None of the OPC or correction layer(s) are mapped out through LITHO DENSEOPC. Output may not be valid.
- FRAGMENT_CONFORM is detected without fragment_nearly_coincident in the fragmentation block. Small XOR's related to fragmentation differences are expected. Define fragment_nearly_coincident <marker_layer> -dist <distance> in the fragment_layer <target_layer> block.

For setlayer denseopc when invoked with EUV:

- optical_transform_size set to 768: for EUV OPCverify or dense OPC should be 512 for optimal performance.
- simulation grid sampling size set to '3/4/2': for EUV OPCverify or dense OPC should be set to '2/3/1' via 'imagegrid aerial' and 'final_upsample'.

Performance

Enabling setup lint makes no difference in run time.

Enabling runtime lint can cause a certain amount of slowdown, typically less than 10%.

Related Topics

[log_options \[Calibre OPCverify User's and Reference Manual\]](#)

mask_chip_corner

denseopc_options Keywords

Used to simulate corner reduction during lithography process.

Usage

Syntax 1: Non-Litho Model for OPC

```
mask_chip_corner convex_val [concave_val] [right_angles_only]
```

Syntax 2: Litho Model With VEB

```
mask_chip_corner -table '{' length convex_val concave_val  
                  length convex_val concave_val  
                  ...  
                  length convex_val concave_val '}' [right_angles_only]
```

Arguments

- ***convex_val***

A required argument that specifies a value for a convex corner that will be “chipped” by an isosceles right triangle inscribed at the corner (the length of which is specified by *cx*).

- ***concave_val***

An optional argument that specifies value for a concave corner that will be “chipped” by an isosceles right triangle inscribed at the corner (the length of which is specified by *cc*).

- **-table '{' {*length convex_val concave_val*} ... '}'**

A required argument for Syntax 2 that specifies different “chipping” values based on edge length in microns. There can be any number of entries, but the entire list must be enclosed with braces ({ }). The entries do not need to be on separate lines. Entries consist of three floating point numbers.

length — The length of the shorter edge in microns.

convex_value — As for non-litho model syntax.

concave_value — As for non-litho model syntax. Each entry in a table must include a value to use for concave corners, even if both corner types are to be chipped the same amount.

If an edge length falls between two ***length*** values, Calibre nmOPC calculates the linear interpolation between the two entries and uses that value. For lengths outside the table range, the corner values of the shortest or longest ***length*** are used without adjustment.

- ***right_angles_only***

If enabled, only chip corners with 90-degree angles.

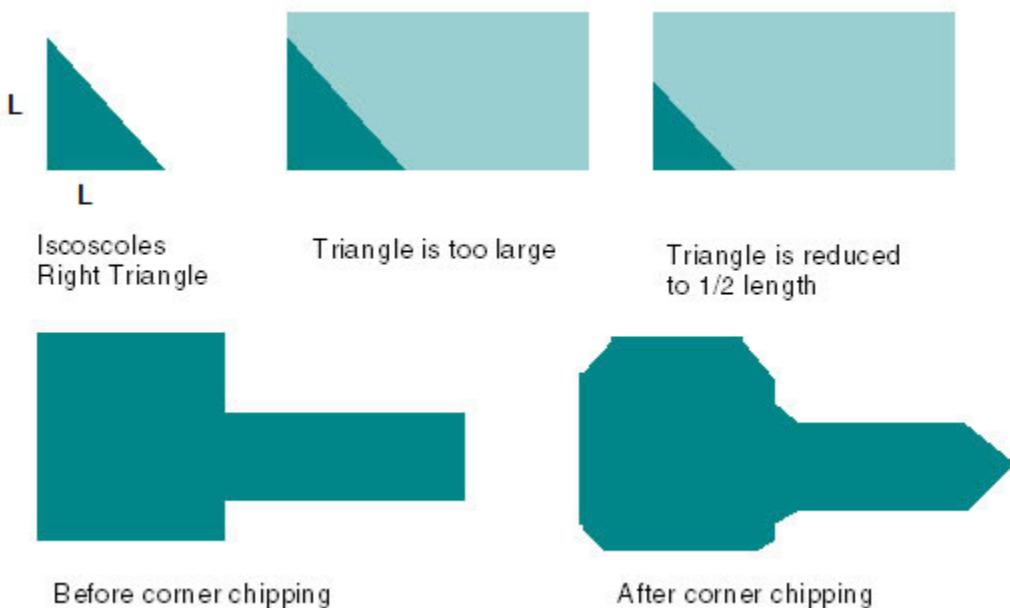
Description

When present, `mask_chip_corner` enables “mask corner chipping” during all OPC simulations. The corner chipping represents a 2D (X-Y plane) model of the effect of the mask writer on final mask corners on a real mask.

In effect, each convex corner has a isosceles right triangle inscribed in the corner and subtracted, and each concave corner has an isosceles right triangle filled into the corner. The isosceles triangles equal length edges have length, L, equal to the `cx` or `cc` value, making the hypotenuse $\sqrt{2} * L$.

If the shortest of the two edges attached to the corner is shorter than $(1/2 * L)$, then L is reset to be 1/2 the length of the shortest edges for that corner. The following figure shows the mask chipping process.

Figure 4-34. Mask Chipping Process



If you are using `mask_chip_corner` with OPC and SRAF layers to perform a Calibre OPCVerify or Print Image contour simulation, you must use the Calibre OPCVerify [cornerchop](#) command on both layers.

For example, in a Calibre nmOPC setup file containing:

```
mask_chip_corner 0.004 0.004
```

You should use the following to generate a chopped mask layer for the simulation to perform correctly:

```
setlayer opced_chop_lyr = cornerchop opced 0.004 0.004
setlayer sraf_chop_lyr = cornerchop sraf 0.004 0.004
```

Note

As a general rule, `mask_chip_corner` is not permitted with a litho model. The corner chipping should be specified in the litho model instead. However, for `setlayer veb_retarget` in and `setlayer veb_verify` commands used for VEB retargeting, corner chopping should always be specified with `mask_chip_corner` regardless of whether a litho model is referenced by the `image` command.

Syntax 2 is not restricted to VEB retargeting, but is generally not necessary for standard OPC runs.

Examples

Example 1

This chips all convex corners by 0.01 microns, and concave corners by 0.02.

```
mask_chip_corner 0.01 0.02
```

Example 2

This provides six different values of corner chipping based on the length of the shorter edge of the corner:

```
mask_chip_corner -table {  
    0.020  0.004  0.003  
    0.030  0.008  0.007  
    0.040  0.010  0.009  
}
```

If a convex corner's shorter edge is 0.035, Calibre nmOPC interpolates between 0.008 and 0.01 to chip the convex corner by 0.009.

If another corner had a short edge < 0.020 (for example, 0.015), the chop values of 0.004 or 0.003 would be used without change.

max_iterations

denseopc_options Keywords

Limits number of OPC iterations.

Usage

max_iterations *m* [nopw *number_nopw_it*]

Arguments

- ***m***

Specifies maximum number of OPC 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 OPC 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

 If any tagscript command such as NEWTAG is used in a setup file that also sets max_iterations, the warning "WARN: In tagscript mode, the value <*m*> of "max_iterations" is ignored." is produced. If you see this warning in the transcript, use OPC_ITERATION.

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. For example:

```
max_iterations 8 nopw 3
OPC_ITERATION 7
```

In this example, Calibre nmOPC will perform 3 non-process window iterations, followed by 4 more process window iterations, for a total of 7 iterations.

Examples

Run 7 iterations without process window conditions and 2 with:

```
max_iterations 9 nopw 7
```

max_iter_movement

[denseopc_options](#) Keywords

Limits edge movement distance for an OPC iteration.

Usage

max_iter_movement *val*

Arguments

- *val*

Specifies the maximum edge movement distance, expressed in microns. This value must be ≥ 0 . The default distance is Nyquist/2.

Description

This optional command controls the maximum distance an edge can be moved during an iteration of OPC. Use this keyword to

- improve OPC stability
- control convergence

max_opc_move

denseopc_options Keywords

Specifies maximum total distance all fragments are allowed to be moved by the entire OPC run.

Usage

max_opc_move *value* [*out_value*]

Arguments

- *value*

A required argument that specifies the maximum edge movement distance. If *out_value* is not specified, it applies to both inside and outside the shape.

The value must be ≥ 0 . The default distance is 80 nm.

- *out_value*

An optional argument that specifies the maximum edge movement outward in nanometers. The default is *value*.

Description

This optional command controls the total distance a fragment is allowed to move (inside or outside the polygon) during biasing for the entire dense OPC run. Specifying the smallest value possible can improve the speed of mrc_rule checking.

min_dog_ear_length

[denseopc_options](#) Keywords

Sets a minimum fragment length for diagonal fragments with two 135-degree corners.

Usage

min_dog_ear_length *minimum*

Arguments

- ***minimum***

A required argument specifying a lower bound in microns on the fragment length. Valid values are greater than zero.

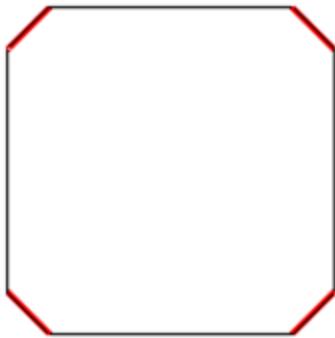
Description

This optional command sets a constraint on the length of fragments with 135-degree corners. The length of these fragments is not allowed to become smaller than this value.

If this command is not included in the rule file, trapped fragments are not protected from disappearing.

Examples

Consider a contact with corner chopping:



Sometimes the OPC algorithms adjust the horizontal and vertical edges to such an extent that the diagonal edges (red in the figure) are shrunk to the point of disappearing. The `min_dog_ear_length` protects the diagonal edges from aggressive neighbors.

Related Topics

[grids_for_angle_45_snap](#)

mpopc

[denseopc_options](#) Keywords

Adds special measurement sites to account for multi-patterning issues.

Usage

```
mpopc [filter_layer layer_name | filter_tag tag_name] criticalDistance val  
[minOverlay val] [maxLineEndWidth val]
```

Arguments

- **filter_layer** *layer_name*

An optional parameter that specifies a layer to be used as filter: only proper fragments touching this layer will be used in processing. All fragments are used by default (no filtering).

- **filter_tag** *tag_name*

An optional parameter that specifies a tag to be used as a filter. Only fragments from this tag set are used in processing. All fragments are used by default (no filtering).

- **criticalDistance** *val*

A required parameter that defines the size of the minimum feature to be resolved.

- **minOverlay** *val*

An optional parameter that specifies that the contour overlay across process window conditions should be at least the specified value. This parameter should match the value of min_overlay used in the [OPC_ITERATION](#) call. The default is 2*criticalDistance.

Note

 Overlay gauges or any attempt to conform to min_overlay only applies to rectangular overlaps or stitches. You should be careful when performing pre-OPC biasing to avoid creating non-rectangular overlaps or stitches.

- **maxLineEndWidth** *val*

An optional parameter that directs Calibre mpOPC to attempt to put special gauges on line ends that are smaller than maxLineEndWidth. Line ends and edges that are larger than maxLineEndWidth are ignored. The default is 2*criticalDistance.

Description

The mpopc command adds special measurement sites to account for mask-to-mask issues such as mask bridging or insufficient overlapping for stitching.

Layer Definitions

In a Calibre nmOPC setup file, create Calibre nmOPC input layers. Calibre mpOPC inputs are the decomposed mask layers defined using the [layer](#) command with the “pattern” keyword. The pattern keyword defines to which mask a certain layer belongs. For example:

```
layer mask0 opc      clear pattern 0
layer sraf0 visible  clear pattern 0
layer mask1 opc      clear pattern 1
layer sraf1 visible  clear pattern 1
```

Note that there are two defined OPC layers.

Required Dense Options Commands

Calibre mpOPC requires the use of retarget layers and process window OPC (PWOPC). Retarget (curve) layers for both masks can be defined as shown in the following example:

```
retarget_layer pattern 0 curve_0 pattern 1 curve_1 spline
```

Curve layers can be generated in the Calibre OPCverify section of the setup file using the [setlayer curve_target](#) command.

You will also be required to use [OPC_ITERATION](#) in PWOPC mode (-PW option). This is required in order to use the min_overlay option:

```
OPC_ITERATION val -PW min_overlay min_overlay_tag overlay
```

For example:

```
OPC_ITERATION 20 -PW min_overlay all 0.055
```

Best practices for Calibre nmOPC are available in “[Simplifying a Calibre mpOPC Setup File](#)” on page 667.

Verification

The different masks in multi-patterning do not affect each other in the printing process. You can safely print each mask separately and “OR” the resulting images together using setlayer commands to get the final wafer image. For example:

```
setlayer Img0 = image optical opt background dark \
    layer opc_0 clear resist_model res_model
setlayer Img1 = image optical opt background dark \
    layer opc_1 clear resist_model res_model
setlayer finalImg = OR Img0 Img1
```

Calibre mpOPC and SRAF Operations

Commands related to SRAF operations in Calibre nmOPC work the same way as all other correction calls (such as nominal, PWOPC, and so on). You do not need to explicitly specify

each sraf layer. For example, the following call to sraf_print_avoidance is valid in Calibre mpOPC.

```
sraf_sites_create
...
for { set i 1 } {$i <= 6 } {incr i } {
    OPC_ITERATION 1
    sraf_print_avoidance -step 0.002 -pw sraf_PA 0.225
}
```

Examples

The following shows a complete and annotated Calibre mpOPC setup file:

```
litho file mpOPC_setup [
    ##### Model Definitions #####
    modelpath ./models
    optical_model_load nom nom
    resist_model_load res res.mod
    optical_model_load fp50 fp50
    optical_model_load fn50 fn50
    background clear

    ##### Layer Definitions #####
    layer mask0 hidden      dark
    layer sraf0 visible     dark
    layer mask1 hidden      dark
    layer sraf1 visible     dark

    ##### OPCverify Definitions #####
    tilemicrons 25
    progress_meter on
    mask_sample_grid rsm

    denseopc_options mpOPC {
        version 1
        background clear
        layer mask0   opc      dark pattern 0
        layer sraf0   visible  dark pattern 0
        layer mask1   opc      dark pattern 1
        layer sraf1   visible  dark pattern 1
    }

    ##### Note the use of the "pattern" keywords #####
    layer curve0  hidden      dark
    layer curvel  hidden      dark

    ##### Nominal & Process Window Settings #####
    image optical nom resist res
        pw_condition inner optical fp50 dose 0.95 pvband
        pw_condition outer optical nom dose 1.05 pvband
        max_iterations 0
```

```
##### OPC Parameters #####
algorithm 2
    pwopc_mode 1
    opc_grid_multiplier on
    mrc_mode 1
    epe_spacing 0.005
    jog_ignore_metric 0.005
    jog_freeze 0.005

##### Fragmentation #####
fragment_min 0.03
fragment_max 0.2
fragment_layer_order full
line_end_fragment off

fragment_layer mask0 {
    fragment_corner convex concave length < 0.09 0.03
    fragment_corner convex concave length >= 0.09 0.04 breakinhalf
    fragment_inter -interdistance 0.15 -ripplelen 0.030 -num 2 \
        -shield 2 -distancePriority 1 -split 1 -skipcorners
    fragment_max 0.2
}
fragment_layer mask1 {
    fragment_corner convex concave length < 0.09 0.03
    fragment_corner convex concave length >= 0.09 0.04 breakinhalf
    fragment_inter -interdistance 0.15 -ripplelen 0.030 -num 2 \
        -shield 2 -distancePriority 1 -split 1 -skipcorners
    fragment_max 0.2
}

##### MRC Rules #####
mrc_rule internal mask0 mask0 {
    use 0.015 euclidean
}
mrc_rule external mask0 mask0 {
    use 0.015 euclidean
}
mrc_rule external mask0 sraf0 {
    use 0.015 euclidean
}
mrc_rule internal mask1 mask1 {
    use 0.015 euclidean
}
mrc_rule external mask1 mask1 {
    use 0.015 euclidean
}
mrc_rule external mask1 sraf1 {
    use 0.015 euclidean
}

##### Retarget Layers #####
retarget_layer pattern 0 curve0 pattern 1 curve1 emulate
```

```
##### Feedback and Tags #####
feedback -0.3 -0.2 -0.2 -0.2 -0.15 -0.15 -0.15 -0.1 -0.1 -0.2
NEWTAG all mask0 -out target0_all_frag
NEWTAG all mask1 -out target1_all_frag
TAG or target0_all_frag target1_all_frag -out all

##### mpopc command #####
mpopc criticalDistance 0.04 minOverlay 0.060 maxLineEndWidth 0.07

##### Specify OPC_ITERATION in -PW mode #####
OPC_ITERATION 10 -PW min_space all 0.020 \
    min_width all 0.036 \
    min_overlay all 0.060 \
}

##### Curve Target Creation #####
setlayer curve0 = curve_target mask0 criticalDistance 0.042
setlayer curve1 = curve_target mask1 criticalDistance 0.042

setlayer opc_0 = denseopc mask0 sraf0 mask1 sraf1 curve0 curve1 \
    MAP mask0 OPTIONS mpOPC
setlayer opc_1 = denseopc mask0 sraf0 mask1 sraf1 curve0 curve1 \
    MAP mask1 OPTIONS mpOPC

##### Simple Verification Setup #####
setlayer Image_0 = image optical nom background clear layer mask0 \
    dark layer sraf0 dark resist_model res
setlayer Image_1 = image optical nom background clear layer mask1 dark \
    layer sraf1 dark resist_model res
setlayer finalImg = or Image_0 Image_1
]
```

mrc_consistency

[denseopc_options](#) Keywords

Loosens MRC settings in particular cases.

Usage

mrc_consistency {0 | 1 | 2}

Arguments

- **0 | 1 | 2**

A required argument that specifies the level of MRC adjustment to use.

0 — Turns off the adjustment.

1 — Allows some fragments to move so that the overall edge achieves MRC constraints.
This is the default.

2 — Allows some frozen fragments to be moved if necessary in order to create a
consistent MRC result.

Description

This optional command can help in rare situations caused by frozen fragments receiving different tags in parent and child before MRC reconciliation. However, it is better to adjust the recipe so that the normal correction and MRC operations run as designed.

Use this keyword with caution. The more iterations a recipe uses, the greater the changes this keyword is likely to introduce.

mrc_mode

[denseopc_options Keywords](#)

Specifies a new fragmentation movement algorithm.

Usage

mrc_mode {1 | 2 | 3}

Arguments

- **1**

Calibre nmOPC will attempt to move the fragments towards their desired positions, making sure MRC rules are not violated. It will make the fragments move only between their starting and the desired position.

- **2**

This mode implements priority-based movement. Reducing the edge displacement error (EDE), the difference between the desired position of a fragment and its final position over all fragments is an important objective for this algorithm. Fragments are assigned dynamic priorities (based on their current EDE) and a higher priority fragment can lower its EPE at the cost of a low priority fragment. Fragments are not constrained to move between the starting position and the desired position (although this should happen rarely). The improved solution is achieved at the cost of a slight increase in run time. This is the default setting.

- **3**

This mode attempts to filter fragments with a small EDE threshold value (the default is 1 nm) and only optimizes fragments with EDE over this threshold. This mode is set automatically if you are performing EUV model-based OPC.

Description

An optional command that instructs Calibre nmOPC to use an alternate fragment movement algorithm. The benefit of this alternative algorithm is that you are much less likely to have MRC violations following OPC. The output should thus be cleaner with fewer MRC violations.

mrc_rule

denseopc_options Keywords

Used to create external and internal spacing constraints.

Usage

mrc_rule {external | internal | enclosure}

```
layer1 [{inside | outside | not_inside | not_outside} reglayer] [horizontal | vertical]
[layer2 [{inside | outside | not_inside | not_outside} reglayer] [horizontal | vertical]] {'}
[[projecting [dist] | not_projecting [dist]]
use d1 {metric | d2}
[{:length len use d1 {metric | d2}}...]]...
'}
```

Arguments

- **external | internal | enclosure**

A required argument that 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 enclosed frags enclosing_layer {
    use num
    ...
}
```

where:

- enclosed frags — Contact or via layer fragments that are enclosed by the enclosing layer.
- enclosing_layer — The enclosing layer fragments.
- use num — Measures the enclosure between enclosed frags and enclosing_layer.

- **layer1**

A required argument that specifies the first layer of the check. You can optionally specify a region of **layer1** or direction of edges for which this MRC rule is valid.

- *layer2*

An optional argument that 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 region or direction of edges.

Note

 When using several mrc_rule statements, the last statement in the sequence should not include an inside, not_inside, outside, not_outside, horizontal, or vertical declaration.

- {inside | outside | not_inside | not_outside} *reglayer*

An optional argument that restricts the region of *layer1* or *layer2*. Only one of inside, outside, not_inside, and not_outside can be specified. Only fragments whose shapes are completely inside, completely outside, completely “not inside,” or completely “not outside” the shapes on *reglayer* are used for the check. Collinear edges are considered to be both not inside and not outside.

- horizontal | vertical

An optional argument that restricts the check to considering only the horizontal or vertical edges on the layer.

- 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 both projecting and not_projecting, do not 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*). Generally, *dist* should be less than or equal to [fragment_min](#). If the projecting distance is less than or equal to *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*).

Note

Using metrics other than euclidean with not_projecting can cause inconsistencies, as discussed in “[OPC Output Consistency](#)” on page 650.

- **use *d1* {metric | *d2*}**

A required argument that specifies the minimum distance to allow for the rule. The body of the mrc_rule must contain at least one use argument.

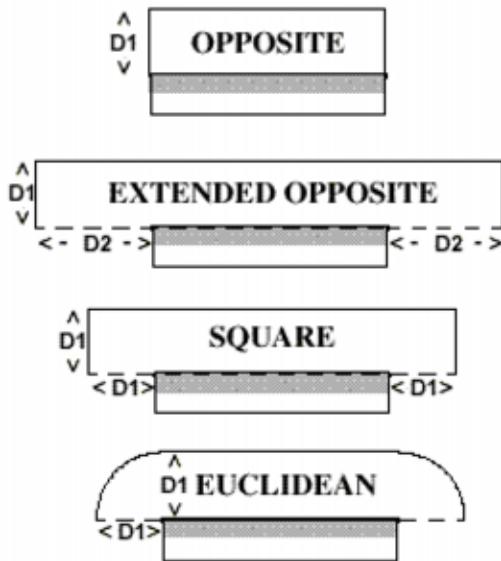
Distances that are not violations are $\geq d1$. The value for *d1* is required and is the default value used if none of the length-constraint rules apply (specified by either **metric** or *d2*). You must specify either **metric** or *d2*.

The possible values for **metric** are **euclidean**, **opposite**, and **square**. **euclidean** is recommended.

When *d2* is specified, it implies **opposite extended** with *d2* as the extension value.

The following figure shows the measurement regions created for each of the different metrics, and the dimensions used with each.

Figure 4-35. The Four Metrics



- length *len*

An optional keyword that specifies a maximum length criteria, *len*. The use argument is applied only when the shorter of the two edges being compared is less than or equal to *len*.

Description

The mrc_rule command block allows you to specify external and internal constraint checks between layer fragments based on the distance between fragments as well as fragment length. Multiple mrc_rule checks can be defined in a litho setup file; by default, when multiple checks apply to a situation the most constrained rule applies. This can be changed with [mrc_rule_priority](#).

Note

 For better performance and consistency, use region layers, smaller values, and length-based Euclidean checks.

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 OPC does not move the edges out. In another case, you want to move the edges outward when length constraints are met. When the check passes, OPC moves the edges as a result of the rule.

If you have multiple “projecting” rules, you can preface each with the keyword “projecting” for clarity. Note that there can be only a single “not projecting” keyword.

Examples

The mrc_rule command block allows the flexibility to perform internal and 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
}
```

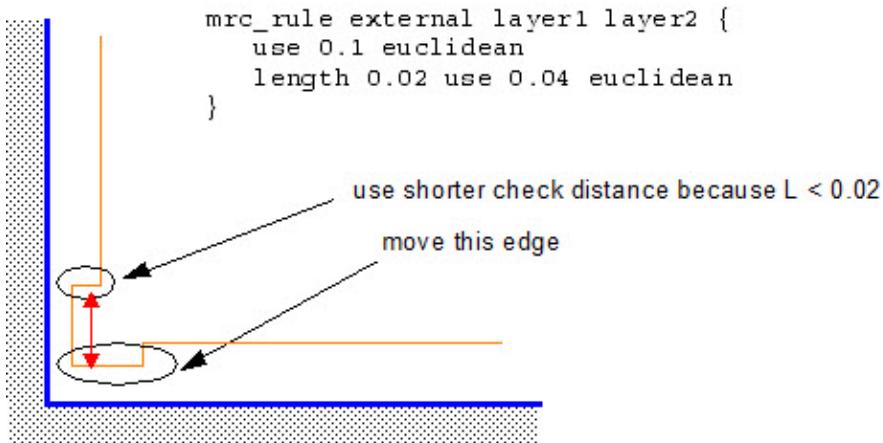
Area Constraints

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. 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
}
```

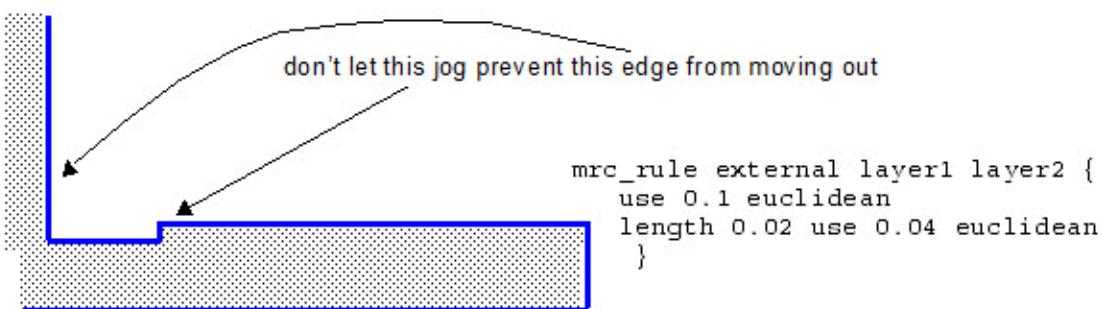
Moving an Edge From a Concave Corner

The following example allows an edge at a concave corner to move outwards.



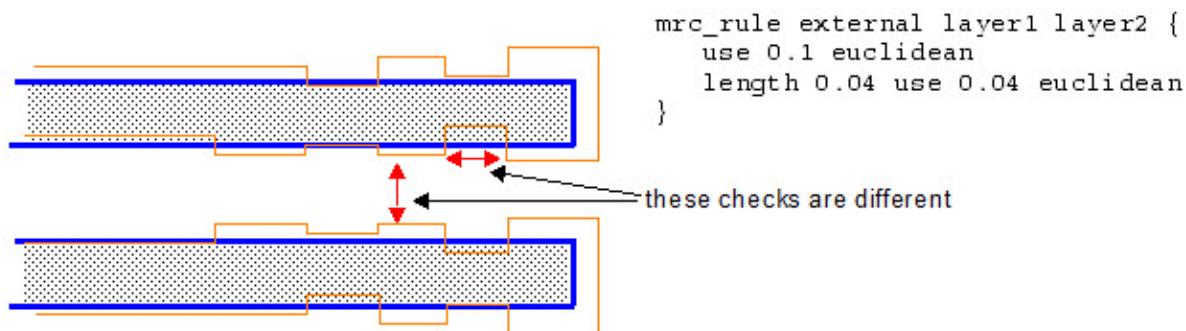
Jogs

In this example, a rule is set to ignore a small jog on the input layout when preventing an edge from moving during OPC.



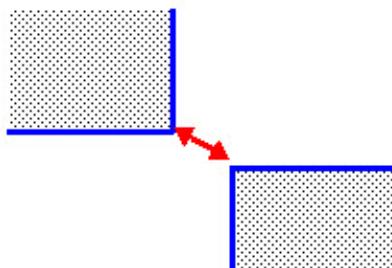
Notches and Feature-To-Feature Checks

In the following example, a distinction is made between notches and feature-to-feature checks.



Corner-to-Corner Checks

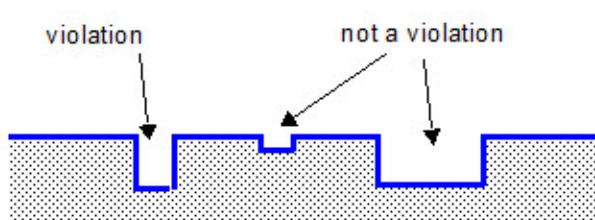
The `not_projecting` option can distinguish features such as corner-to-corner checks, which are not projecting onto each other, as in the following example.



```
mrc_rule external layer1 layer2 {
    projecting
    use 0.1 euclidean
    length 0.025 use 0.030
    not_projecting
    use 0.09 euclidean
    length 0.025 use 0.030
}
```

Stair-Stepping

The following example implements a “stair-stepped aspect ratio” of 2:1 when features are below a certain size.



```
mrc_rule external layer1 layer2 {
    use 0.1   euclidean
    length 0.02 use 0.04 euclidean
    length 0.03 use 0.06 euclidean
    length 0.04 use 0.08 euclidean
}
```

Related Topics

[MRC Rule Best Practices](#)

mrc_rule area

[denseopc_options](#) Keywords

Used to check SRAF layer spacing constraints for sraf_print_avoidance.

Usage

```
mrc_rule area sraf_layer {'  
    [square use area_square]  
    [use area_rectangle]  
'}
```

Arguments

- ***sraf_layer***

A required argument that specifies that an area rule check is to be performed on the designated SRAF layer. The input layer must be of type “sraf”.

- square use *area_square*

An optional argument that specifies the minimum area for squares. It operates similarly to the -min_square_area keyword in [sraf_print_avoidance](#). If the shape changes from a square to a rectangle during the rule check, this limit still applies.

- use *area_rectangle*

An optional argument that specifies the minimum area for rectangular SRAF shapes. It operates similarly to the -min_area keyword in [sraf_print_avoidance](#). If the shape changes from a rectangle to a square during the rule check, this limit still applies.

This limit is applied to square-shaped features if “square use” is not present.

Description

The mrc_rule area checks are used only when sraf_print_avoidance is also in the setup file. It applies only to shapes on the SRAF layer with four corners; other shapes like those caused by merging are ignored.

The parser issues the following error message when both sraf_print_avoidance and mrc_rule area specify a value:

The min_area value can be specified on both the command line or as an MRC rule, but not both.

To fix the error, either comment out the mrc_rule area lines or remove -min_area and -min_area_square from sraf_print_avoidance.

Related Topics

- [SRAFs and MRC Rules](#)

mrc_rule_priority

[denseopc_options Keywords](#)

Specifies how MRC rules are applied if multiple rules apply to an edge pair.

Usage

mrc_rule_priority {ALL | LAST}

Arguments

- **ALL | LAST**

A required argument that specifies the behavior or prioritizing MRC rules. ALL enforces all rules with the most constrained rule taking priority. LAST enforces the last applicable rule that was defined. The default setting is ALL.

Description

When multiple MRC rules apply to an edge pair, this command specifies which rule will be applied.

no_opc_region

denseopc_options Keywords

Creates a “non-opc” layer that designates areas where edges will not be moved by OPC (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 OPC.

Description

An optional command that defines an area in which no fragment is eligible for OPC and consequently remains in a fixed position. By default, all fragments are subject to OPC.

Though this command restricts the movement of fragments within the no_opc_region area, the no_opc_region command does not prevent simulation occurring on that layer.

Note

 All layers must be associated with patterns in layer statements. Each pattern can only have one no_opc_region layer specified or an error message is issued.

Performance

May be increased for a no_opc_region layer when a large area is specified

Examples

```
no_opc_region poly_no_opc
```

nominal_epe_limit

[denseopc_options](#) Keywords

Allows nominal EPE to converge to a value within a specified range.

Usage

nominal_epe_limit *tol_inner tol_outer*

Arguments

- ***tol_inner tol_outer***

Specifies that the nominal EPE is allowed to converge to any value within this range in order to improve PV band results. By default, ***tol_inner*** is set to -infinity and ***tol_outer*** to infinity (the nominal EPE is ignored and OPC only tries to improve the PV band).

Description

This command attempts to improve PV band quality by allowing the nominal EPE to converge to any value within a specified range. This command is only allowed in “PV Band PWOPC mode” (when [pw_condition](#) is specified with the “pvband” keyword).

one_time_site_pairing

[denseopc_options Keywords](#)

Performs site paring only in the first iteration during the execution of an OPC_ITERATION command.

Usage

one_time_site_pairing {on | off}

Arguments

- **on | off**

When set to on, site pairing is performed only in the first iteration during execution of an OPC_ITERATION command and dynamic adjustment is skipped in subsequent iterations.

Description

The one_time_site_pairing command is useful if you want to reduce the amount of time consumed by site pairing when performing PV Band PWOPC. This command only works for PV Band PWOPC and is ignored otherwise.

opc_grid_multiplier

denseopc_options Keywords

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 opc_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
```

pvband_tolerance

[denseopc_options](#) Keywords

Prevents further changes to a PV band if it is within a specified tolerance.

Usage

pvband_tolerance tol_inner tol_outer

Arguments

- ***tol_inner tol_outer***

Calibre nmOPC will not attempt to improve the PV band if it is within this specified tolerance.

Description

This command prevents further changes to a PV band if it is within a specified tolerance. This command is only allowed in “PV Band PWOPC mode” (when [pw_condition](#) is specified with the “pvband” keyword).

pw_condition

denseopc_options Keywords

Creates alternate image conditions for dose, focus, resist, and aerial threshold, as opposed to a nominal image.

Usage

Syntax 1: Standard (Non-Litho) Model, Single Patterning

```
pw_condition cond_name
  [mask_size sval] [{inside | outside | not_inside | not_outside} layer_name]
  {optical {opt_name [dose d]...} [ddm [{layer model}...]}
  {aerial threshold | resist res_name} [etch_model etch_name]
  [no_opc] [pvband] [3DSlice] [spa] [cd_only]
```

Syntax 2: Standard (Non-Litho) Model, Multi-Patterning

```
pw_condition cond_name
  [pattern num
    [mask_size sval] [{inside | outside | not_inside | not_outside} layer_name]
    {optical {opt_name [dose d]...} [ddm [{layer model}...]}
    {aerial threshold | resist res_name} [etch_model etch_name]
    [no_opc] [pvband] [3DSlice] [spa] [cd_only]]...
```

Syntax 3: Litho Model, Single Patterning

```
pw_condition cond_name
  [litho_model_2_name] [{[mask N] [focus fhm] [dose d] [bias b]...}
  [aerial {val | model}] [thr_delta val%]]
  [stochastic {high | low}] [stochastic_sigma multiplier]
  [plane {zval_um | SRAF_POSITIVE | SRAF_NEGATIVE}]
  [no_etch | with_etch] [{inside | outside | not_inside | not_outside} layer_name]
  [pvband] [3DSlice] [spa] [no_opc] [cd_only]
```

Syntax 4: Litho Model, Multi-Patterning

```
pw_condition cond_name
  [pattern num
    [litho_model_2_name] [{[mask N] [focus fhm] [dose d] [bias b]...}
    [aerial {val | model}] [thr_delta val%]]
    [stochastic {high | low}] [stochastic_sigma multiplier]
    [plane {zval_um | SRAF_POSITIVE | SRAF_NEGATIVE}]
    [no_etch | with_etch] [{inside | outside | not_inside | not_outside} layer_name]
    [pvband] [3DSlice] [spa] [no_opc] [cd_only]
  ]...
```

Arguments

- ***cond_name***

A required argument that specifies the name of the process window condition.

- pattern *num*

For multi-patterning, this optional argument specifies different backgrounds for each patterning step. If the pattern keyword is not used, the same backgrounds are used for all patterns.

If the image command specifies per-pattern settings, all pw_conditions must also specify them.

All the patterns must specified as part of a single pw_condition command, and must be specified in the following order: 0,1,2, and so on. The number of patterns specified must be the same as the number defined by the layer statements:

```
pw_condition Pinch
    pattern 0 focus 50nm pvbnd no_opc
    pattern 1 LITHOMODEL1 bias -0.00025 focus 50nm dose 0.959 pvbnd
```

Note

 For non-litho model implementations, pattern definitions must be the same between the [background \(for denseopc_options block\)](#), [image](#), and [pw_condition](#) statements.

Non-Litho Model Usage

- mask_size *sval*

An optional argument that specifies the mask size in microns. The default is 0.0.

- {inside | outside | not_inside | not_outside} *layer_name*

An optional argument that applies the defined process window condition only to fragments that meet one of the following conditions for the specified layer: completely “inside” the layer, completely “outside” the layer, completely “not_inside”, or completely “not_outside”.

- **optical** {*opt_name* [dose *d*]...}

A required argument when not using a litho model that specifies the name(s) of the optical model(s) for this process condition. The optional dose keyword specifies the dose for the optical model. The default is 1.0.

- ddm {*layer model*}...

An optional argument that specifies a DDM model. DDM models are specified per layer. For example:

```
ddm layer1 ddm_model_1 layer2 ddm_model_2
```

- **aerial threshold**

A required argument when not using a litho model that specifies the value of the aerial threshold for this process condition. Either aerial or resist but not both must be specified.

- **resist** *res_name*

A required argument when not using a litho model that specifies the name of the resist model for this process condition. The default is to use the nominal resist model.

- `etch_model etch_name`
An optional argument that specifies an etch model.
- `no_opc`
An optional argument that indicates to not use this process window condition to determine EPE (and to not simulate it during regular OPC iterations). This process window condition may still be used by [sraf_print_avoidance](#).
- `pvband`
An optional argument that specifies that this process window condition contributes to a PV band that OPC will improve. No resist or aerial threshold needs to be specified if the pvband option is present, though it can still be specified with a resist or aerial threshold present. This allows the use of different CTR and resist models for PV bands.
- `3DSlice`
An optional argument that specifies that this process window condition is a resist top-loss specification that contributes to a PV band that OPC will improve. In particular, this process window condition is checked only when corresponding 3D width and spacing constraints are specified during a PWOPC iteration.

This process window condition is not allowed to be used as a nominal condition.
- `spa`
An optional keyword that specifies that this process window condition is only used by [sraf_print_avoidance](#) internally. It is not simulated in OPC iterations and is not used to determine EPEs.

When specified with [sraf_print_avoidance](#), [sraf_print_avoidance](#) automatically recognizes this process window, simulates current mask(s) with dedicated sites, and adjusts assist features accordingly. You do not need to define a nominal condition. (This process window cannot be used as a nominal condition.)
- `cd_only`
An optional keyword that specifies that the process window condition is used only for a pinching or bridging check and correction. It will not be used for any other constraints, such as enclosure and PV band tolerances.

Litho Model Usage

See “[Load Model Information With a Litho Model](#)” on page 48.

- `litho_model_2_name`
An optional argument that specifies the name of a litho model to be used only for this pw_condition, in place of the one specified in the [image](#) command. If specified, this litho model must be identical to the one in the image command, with the exception that its resist or optical model files (or both) may be different. All other features such as mask definitions, mask layers, and numbers of focus conditions must be identical. This option enables a pw_condition to use resist or optical model files (or both) that are different from those in the

image command. It supports applications such as SRAF Print Avoidance or resist top loss modeling. By default, the litho model from the image command is used.

- **mask N**

An optional argument that specifies the mask number (as defined in the litho model file) to which the focus, dose, and bias arguments apply. The default if unspecified is mask 0.

- **focus fnm**

An optional argument that specifies a focus condition to be used for this mask. This argument maps to the focus parameter used by the optical model specified in the *Lithomodel* file.

- **dose d**

An optional argument that specifies a dose to be used for this mask. Note that if a base dose for this mask was specified in the *Lithomodel* file, the actual dose is calculated by multiplying this value with the base dose in the litho model. The default is 1.0.

- **bias b**

An optional argument that specifies a bias in microns to be used for sizing this mask before using it in simulation. Note that if any biases were specified for individual mask layers in the litho model, those biases are added to the value specified here. Bias values are used only in simulation.

- **aerial val**

For all process window conditions (including nominal), the aerial image is thresholded at val to produce the simulation contours. This option overrides the resist model specified in the litho model.

- **aerial *model***

This option is the same as aerial val , but the constant threshold value is taken from the *image_threshold* specified in the litho model.

Note



The aerial argument must be specified explicitly on each pw_condition for which an aerial image is required.

- **thr_delta $val\%[$**

An optional argument that specifies a signed offset to modify the *image_threshold* parameter in the litho model. This keyword is permitted only if aerial *model* is specified. If the percent sign is present, the offset is a relative percentage. Otherwise, the offset is an absolute delta.

- **stochastic {high | low}**

An optional argument that includes the EUV stochastic model. It accounts for EUV shot noise by modifying the simulation threshold for critical dimension (CD).

high — Use positive threshold deviation.

low — Use negative threshold deviation.

By default, stochastic model effects are not included in simulation even if the model is in the litho model.

This argument is typically used in conjunction with pvband; for example,

```
pw_condition pos_st dose 1.05 stochastic high stochastic_sigma 1 pvband
```

- stochastic_sigma *multiplier*

An optional argument that sets the confidence interval of the CD threshold change with the EUV stochastic model. It is ignored unless stochastic low or stochastic high is specified. The default for *multiplier* is 3.0. Values must be positive.

- plane {*zval_um* | SRAF_POSITIVE | SRAF_NEGATIVE}

An optional argument used for 3D resist imaging. It specifies the z-position within the resist film where the image is computed.

This keyword is permitted only if the optical model is 3D-enabled (using the imageplanes keyword). It also requires either a 3D-enabled CM1 resist model or an aerial threshold. If the plane keyword is not specified, the default is the usual non-3D convention of using the base focus plane from the optical model.

zval_um — The value is relative to 0.0 at the bottom of the resist, and must be ≥ 0.0 and \leq the resist film thickness. It can be used with either an aerial threshold or a 3D-enabled CM1 resist model.

SRAF_POSITIVE — Models the printing of positive SRAFs.

SRAF_NEGATIVE — Models the printing of negative SRAFs.

All values require an optical model that uses the imageplanes keyword. If a resist model is specified, it must be a CM1 3D resist model. The SRAF_POSITIVE and SRAF_NEGATIVE keywords cannot be used with aerial, and also require that the litho model contain a zplanes model.

For SRAF_POSITIVE and SRAF_NEGATIVE, the actual plane value calculated depends on whether the resist is negative or positive tone (NTD or PTD), and whether the mask is darkfield or clearfield.

- no_etch | with_etch

If a litho model has an etch model, either no_etch (do not use the etch model) or with_etch must be specified for the process window condition.

If a litho model does not have an etch model, the keywords are ignored.

- {inside | outside | not_inside | not_outside} *layer_name*

An optional argument that applies the defined process window condition only to fragments that meet one of the following conditions for the specified layer: completely “inside” the layer, completely “outside” the layer, completely “not_inside”, or completely “not_outside”.

- **pvband**

An optional keyword that specifies that this process window condition contributes to a PV band that OPC will improve. No resist or aerial threshold needs to be specified if the pvband option is present, though it can still be specified with a resist or aerial threshold present. This allows the use of different CTR and resist models for PV bands.

- **3DSlice**

An optional keyword that specifies that this process window condition is a resist top-loss specification that contributes to a PV band that OPC will improve. In particular, this process window condition is checked only when corresponding 3D width and spacing constraints are specified during a PWOPC iteration.

This process window condition is not allowed to be used as a nominal condition.

- **spa**

An optional keyword that specifies that this process window condition is only used by [sraf_print_avoidance](#) internally. It is not simulated in OPC iterations and is not used to determine EPEs.

When specified with [sraf_print_avoidance](#), [sraf_print_avoidance](#) automatically recognizes this process window, simulates current mask(s) with dedicated sites, and adjusts assist features accordingly. You do not need to define a nominal condition. (This process window cannot be used as a nominal condition.)

- **no_opc**

An optional argument that indicates to not use this process window condition to determine EPE (and to not simulate it during regular OPC iterations). This process window condition may still be used by [sraf_print_avoidance](#).

- **cd_only**

Specifies that the process window condition is used only for a pinching or bridging check and correction. It will not be used for any other constraints (such as enclosure and PV band tolerances).

Description

An optional command to specify alternate process window conditions to maximize performance of OPC for all conditions. EPE is measured at process window conditions, and process window conditions can be used with image constraints.

A process window is the tolerable change in dose and focus until the design fails to yield on silicon. Using the [image](#) command, you define a nominal process window condition, then use the **pw_condition** command to create alternate process window conditions, varying dose, focus, and mask sizing conditions.

Additional images to correct are specified by the [image](#) command. Each image will be simulated on each iteration, hence the overall run time is very much proportional to the number of **pw_conditions** present in the setup options.

In the case of gate CD control, for best CD fidelity, process window correction should not be done on gates. All pw_conditions can be used with marker layers to define where they are applied. By using the active region as a marker, pw_condition EPEs can be ignored for gates.

Examples

```
pw_condition D1 image dose 1.1
```

pwopc_mode (Deprecated in 2016.2)

[denseopc_options Keywords](#)

Defines how final EPE is calculated given nominal and process window condition EPEs.

Note

 This command is deprecated as of the 2016.2 release and will no longer be documented in future releases.

Usage

pwopc_mode {1 | 0}

Arguments

- **{1 | 0}**

Defines how final EPE is calculated given the nominal and process window condition EPEs.
The default is 1.

Description

Setting pwopc_mode to 1 enables the following to occur:

- If [algorithm](#) is set to 0, the process window and wafer constraint functions will behave as if algorithm is set to 1.
- All process [pw_condition](#) declarations must specify inner and outer tolerances.

Setting pwopc_mode to 0 reverts to the original behavior. This setting is ignored in algorithm 1.

Examples

```
pwopc_mode 1
```

read_layer_properties

denseopc_options Keywords

Makes numeric DFM properties accessible in Calibre nmOPC recipes.

Usage

read_layer_properties *layer* *property_name* [*property_name...*]

Arguments

- *layer*

A required argument that identifies the name of the layer containing DFM properties. In the denseopc_options block, the layer should be of type hidden.

- *property_name*

A required argument that specifies the DFM property to load from the layer. At least one property name must be specified. Names are case-sensitive.

Caution

 No error message is issued when the property name does not match any properties on the layer. The Calibre run completes but NEWTAG property does not annotate any fragments.

Description

The read_layer_properties command imports DFM properties associated with a layer. Only numeric DFM properties are supported; vector, string, and net ID properties are not.

After the layer properties are imported, other Calibre nmOPC commands such as NEWTAG property can access the properties in ways similar to an annotated tag set.

The read_layer_properties command may be specified multiple times in a setup file. Up to 100 properties per read_layer_properties command can be specified.

Examples

This loads the DFM properties bridge_count and pinch_count from the layer check_results. The properties are then separately added to annotated tag sets bridge_frags and pinch_frags.

```
read_layer_properties check_results bridge_count pinch_count
...
NEWTAG property allFrags check_results bridge_count -out bridge_frags
NEWTAG property allFrags check_results pinch_count -out pinch_frags
```

retarget_layer

[denseopc_options](#) Keywords

Defines a new target layer, leaving the fragmentation on the opc layers.

Usage

```
retarget_layer {[pattern num] layer}... [{emulate | spline} [concave_adjacent val]  
[convex_adjacent val]]
```

Arguments

- **pattern *num***
An optional argument for multi-patterning. Set *num* to indicate the pattern to which the following layer belongs.
- ***layer***
A required argument that specifies a new target layer or layers. 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. Smooth contours can be generated by the [setlayer curve](#), [setlayer curve_target](#), or [target_curve](#) command.

Emulate is similar to the spline option, except that it allows retarget edges to differ from the original OPC. 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 OPC recipes.

Note

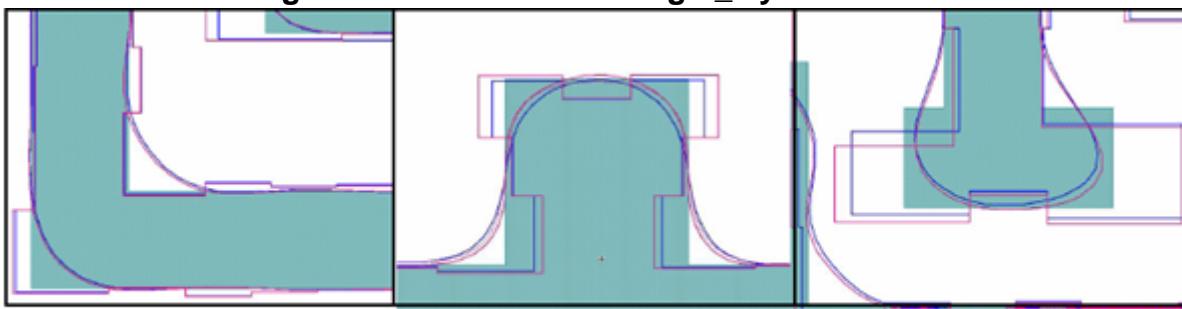
 The emulate option is particularly useful in scenarios where the spline option incorrectly handles the retarget edges and OPC might not even converge.

In general, retargeting should not modify the opc layer by more than the minimum feature width of OPC. Also, OPC will attempt to correct a retarget layer (you will need to ensure that PV bands 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 4-36](#) shows the effects of using retarget_layer emulate.

Figure 4-36. Effects of retarget_layer emulate



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

- `spline`

An optional flag that is used for target smoothing in which the input retarget layer is a smooth contour. Smooth contours can be generated by the `setlayer curve`, `setlayer curve_target`, or `target_curve` command. Spline differs from the emulate option in that it restricts retarget edges to stay close to the original OPC edges.

- `concave_adjacent val`

An optional keyword that uses the distance from a concave corner specified by *val* to tag fragments as concave adjacent. The default value is $5 * \text{frgu}$.

- `convex_adjacent val`

An optional keyword that uses the distance from a convex corner specified by *val* to tag fragments as convex adjacent. The default value is $5 * \text{frgu}$.

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 OPC flow.

Retarget layers are used to separate the target from the layer being corrected and help when the target is unsuitable for OPC. 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 (completely within 5^* Nyquist of the corner), 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 (completely within 5^* Nyquist of the corner), 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 4-37](#) and [Figure 4-38](#) illustrate how EPE is measured to determine how a fragment will be moved.

Figure 4-37. EPE Measurements at Corners (Example 1)

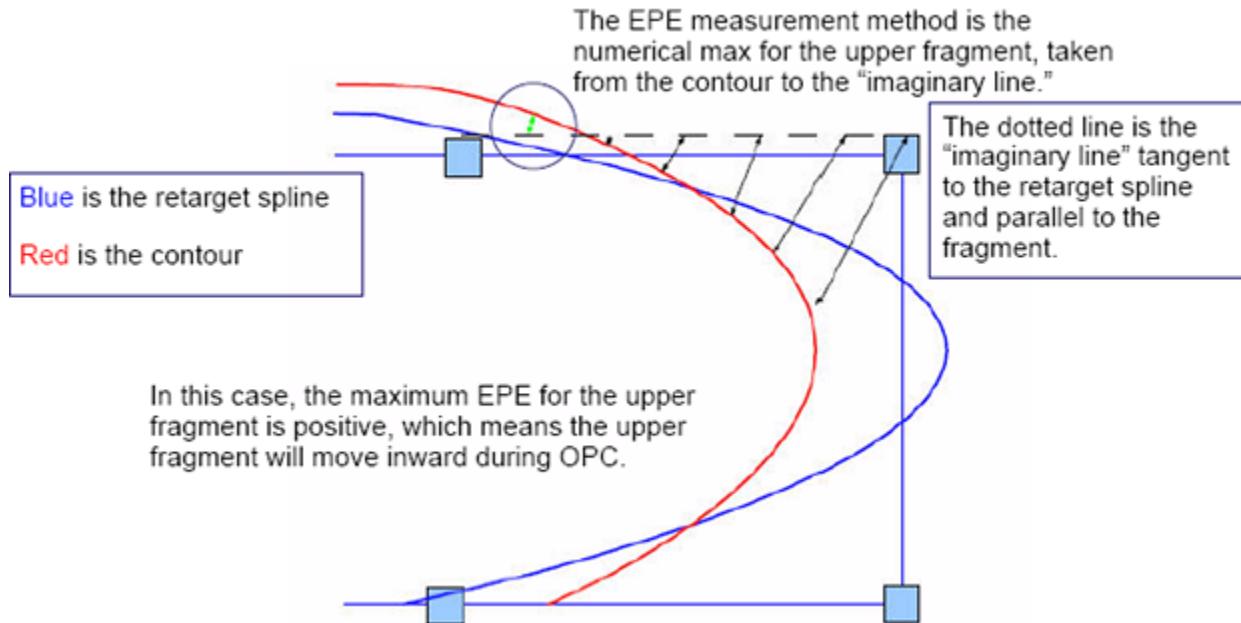
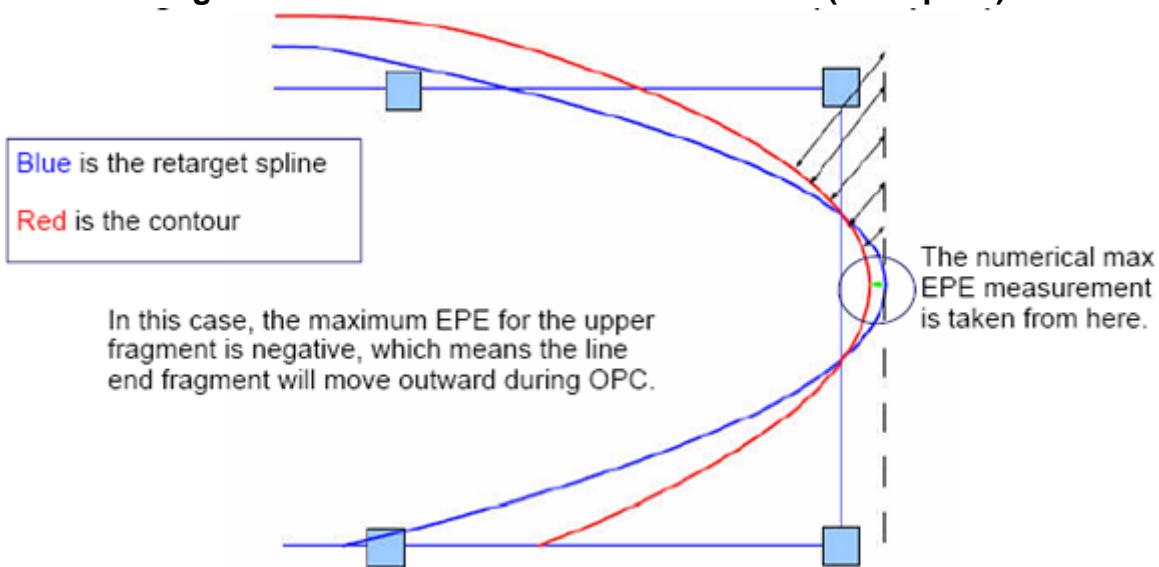


Figure 4-38. EPE Measurements at Corners (Example 2)



This command cannot be used with target_curve.

Examples

Example 1

In this example, the original drawn layer is the layer whose fragments move during OPC.

```
layer POLY      visible ... # original poly target
layer PREBIASED hidden ... # red layer above
retarget_layer  PREBIASED
```

Example 2

In this example, fragments at specified distances from corners are tagged as concave adjacent and convex adjacent.

```
retarget_layer L1 spline concave_adjacent 0.05 convex_adjacent 0.08
```

Setting concave_adjacent and convex_adjacent is recommended so that you control how many fragments next to the corner are adjusted using the minimum or maximum EPE. If you do not specify these settings, the transcript includes the warning “retarget_layer command is used without convex/concave_adjacent options. Potentially more than one fragment starting from a corner will use max/min as an EPE measurement metric.”

For example, if the frgu is 0.02 um, the default values for concave_adjacent and convex_adjacent are 0.1 um (100 nm). If corner fragmentation is set to 0.09, 0.10, and 0.11, the first two fragments are considered adjacent and adjusted using max EPE.

Related Topics

[target_curve](#)

retargeting_options

[denseopc_options](#) Keywords

Specifies values for matrix retargeting parameters.

Usage

```
retargeting_options [max_iter_target_bias iter val] [max_target_bias val]
    [impact_distance dist] [enhanced_epe_mode {on | off}]
```

Arguments

- *max_iter_target_bias iter val*
An optional argument that specifies the maximum retargeting change for one iteration of matrix retargeting, in user units. The default value is 3/10*Nyquist.
- *max_target_bias val*
An optional argument that specifies maximum total retargeting change for all iterations of matrix retargeting, in user units. The default value is 3/5*Nyquist.

Note



A runtime error halts execution if the two ranges $[-val, +val]$ and [lower nominal EPE limit, upper nominal EPE limit] do not overlap.

- *impact_distance dist*
An optional argument that specifies that only fragments within this distance from any violated fragments are affected by matrix retargeting calculations. Fragments outside this range are subject to retargeting based on the effective EPE. The default value is 10*Nyquist.
- *enhanced_epe_mode {on | off}*
An optional argument that controls whether retargeting is based on effective EPE for locations that do not violate any wafer specs and are far away from violated fragments.
 - on — Performs minimum retargeting if possible. This is the default.
 - off — Uses effective EPE to compute retargeting amount.

Description

This optional command specifies values for matrix retargeting parameters used by [OPC_ITERATION](#) and [OPC_XMEEF_ITERATION](#). Refer to the section “Retargeting in PWOPC” in the [OPC_ITERATION](#) description for details. This command only works for PV band PWOPC mode and is ignored otherwise.

scale

denseopc_options Keywords

Used to resize fragmentation lengths according to a scale multiplier.

Usage

scale *factor*

Arguments

- ***factor***

Specifies the scaling factor. The default is 1.0.

Description

This command is a multiplier that scales the initial fragmentation lengths. The scale command can be used to determine 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 default value for scale is 1.0.

The pseudo-Nyquist value is not only used for default fragmentation, but can also impact the fragment measurement metric setting. If the curve target is indicated with the spline or emulate keywords in the [retarget_layer](#) command, and the optional parameters convex_adjacent or concave_adjacent are not specified, then the unspecified fragment(s) have a default value of 5*frgu.

The default value for scale is 1.0. The default values for Calibre nmOPC keywords are listed in the following table.

Table 4-17. Calibre nmOPC Fragmentation Defaults

Calibre nmOPC Commands	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*(λ/NA) -num 0.5*(λ/NA)/ripplenlen -ripplelen 2*frgu
fragment_max	4*frgu
fragment_min	2*frgu
NEWTAG line_end space_end	4*frgu

spa_promotion

[denseopc_options](#) Keywords

Controls SRAF promotion during an OPC run.

Usage

spa {on | off | aggressive}

Arguments

- **on | off | aggressive**

A required argument that specifies SRAF promotion behavior. The default setting is **on**.

on — SRAFs are promoted using a larger promotion ring than OPC or correction layers.
The default distance in this case is 8 Nyquist.

off — SRAFs are not given special promotion treatment.

aggressive — SRAFs are promoted using a larger promotion ring than OPC or correction layers. The default distance is 9.5 Nyquist, typically about 160 nm.

Description

This command controls the behavior of SRAF promotion during an OPC run. Larger values can improve consistency but also increase runtime.

Performance

Large values increase the promotion distance and therefore runtime.

sraf_promotion_distance

[denseopc_options](#) Keywords

Set the distance from the main feature in which to check for SRAFs that may also need to be promoted.

Usage

sraf_promotion_distance *microns*

Arguments

- ***microns***

A required argument specifying the distance from promoted main features. Do not set the distance greater than 0.25.

The default is 0.1 microns.

Description

This optional command can be used to override the default promotion distances of `spa_promotion`. Typically “`spa_promotion on`” provides good quality results and `sraf_promotion_distance` is not required.

Performance

Large values increase the promotion distance and therefore runtime.

Examples

```
sraf_promotion_distance 0.180
```

Related Topics

[spa_promotion](#)

step_size

[denseopc_options Keywords](#)

Sets the minimum edge movement distance for OPC.

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 OPC fragment. This value is specified in user units and must be an integer multiple of the database unit.

Examples

```
step_size 0.00025
```

topo_model

[denseopc_options](#) Keywords

Applies previously-loaded topography models to OPC run.

Usage

topo_model *model_name*

Arguments

- ***model_name***

A required keyword that specifies the name of a topography model previously loaded using [topo_model_load](#).

Description

Applies a previously-loaded topo model to all process window conditions. This is not permitted when a litho model is used as the topography model.

Examples

```
topo_model topo_mod
```

topo_model_load

denseopc_options Keywords

Specifies the name of the topography model.

Usage

```
topo_model_load name {file | '{' inline_model_text '}' }
```

Arguments

- ***name***

A required keyword that specifies a model name.

- ***file***

Specifies the file containing the model. The file be inside the current directory or in the model path. The file can also contain an inlined model file.

- ***inline_model_text***

Specifies a topography model text file instead of the model file path inside braces ({}).

Description

This command reads in a topography model and assigns a name to it for later use in setlayer commands such as [setlayer denseopc](#). When a litho model is used, the topo_model_load command is not needed and is ignored.

Examples

```
topo_model_load topo_mod ./user/topo_mod_file
```

veb_corner_site_placement

Variable Etch Biasing (VEB) Commands

denseopc_options Keyword

Specifies where along a corner fragment to place the VEB evaluation point.

Usage

veb_corner_site_placement *value*

Arguments

- *value*

Specifies one of the following values:

default — The same as not specifying this option. The current default would be written as 50%.

conservative — Place as far away from any corner as possible.

number — The distance from a corner, in microns, at which to place an evaluation point. This is an absolute distance and gets clipped if outside the range [0,*flen*] where *flen* is the fragment length.

percentage% — The distance from a corner, expressed as percentage of fragment length, at which to place an evaluation point. This is a relative measure and must be within the range [0,100].

Description

Due to corner rounding, this command influences how much correction gets applied at corners. This option is only applied to fragments which have exactly one endpoint at a corner; any fragment with both endpoints at corners is unaffected by this option, and its evaluation point remains at the center.

Examples

The following example setup file shows the corner sites set to conservative (as far away from the corner as possible):

```
litho_target = LITHO DENSEOPC etch_target FILE mbr.setup MAP litho_target
LITHO FILE mbr.setup /*

# ----- Simulation models -----
modelpath /test/models
etch_model_load veb_mod veb.mod
tilemicrons      40 adjust
etch_imagegrid 0.01

# ----- Layer info -----
background clear
layer etch_target hidden atten 0.064 0 1
```

```
denseopc_options MBROPTS {
    version 1
    etch_model veb_mod
    background clear
    layer etch_target opc atten 0.064
    max_iterations 10
    feedback -1
    mask_chip_corner 0.015 0.015

    # ----- Fragmentation control -----
    veb_pseudo_nyquist 0.032

veb_corner_site_placement conservative
    veb_evalpts_per_framgment      3

    fragment_min 0.030
    fragment_inter -interdistance 0.50 -num 3 -ripplelen 0.035 -shield 1
    fragment_corner convex 0.040 0.040 0.040 breakinhalf
    fragment_corner concave 0.040 0.040 0.040 breakinhalf
    fragment_max 0.20
}

setlayer litho_target = veb_retarget etch_target MAP etch_target \
    OPTIONS MBROPTS
*/]
```

veb_evalpts_per_fragment

Variable Etch Biasing (VEB) Commands

denseopc_options Keyword

Specifies how many VEB evaluation points to place on each non-corner fragment.

Usage

veb_evalpts_per_fragment {default | 1 | 2 | 3}

Arguments

- **default | 1 | 2 | 3**

Sets the number of evaluation points (sites). Specifying default is the same as not specifying this option; the default is 1.

Description

The default of this command is to place one evaluation point at the center of each fragment. If two points are specified, they are placed at the one-third and two-thirds positions. If three points are specified, they are placed at the one-fourth, one-half, and three-fourths positions. The bias for the fragment as a whole is the average of the per-point biases.

Performance

Performance impact is model-dependent. For a model with only Gaussian kernels, there should be essentially no impact. A model with a large-diameter visibility kernel has more of an effect. Performance is at worst 2X slower if two points per fragment are selected, and 3X slower if three points per fragment is selected.

Examples

The following example setup file shows three sites being set per fragment:

```
litho_target = LITHO DENSEOPC etch_target FILE mbr.setup MAP litho_target
LITHO FILE mbr.setup /*

# ----- Simulation models -----
modelpath /test/models
etch_model_load veb_mod veb.mod
tilemicrons          40 adjust
etch_imagegrid 0.01

# ----- Layer info -----
background clear
layer etch_target hidden atten 0.064 0 1
```

```
denseopc_options MBROPTS {
    version 1
    etch_model veb_mod
    background clear
    layer etch_target opc atten 0.064
    max_iterations 10
    feedback -1
    mask_chip_corner 0.015 0.015

    # ----- Fragmentation control -----
    veb_pseudo_nyquist 0.032

    veb_corner_site_placement conservative
    veb_evalpts_per_fragment 3

    fragment_min 0.030
    fragment_inter -interdistance 0.50 -num 3 -ripplelen 0.035 -shield 1
    fragment_corner convex 0.040 0.040 0.040 breakinhalf
    fragment_corner concave 0.040 0.040 0.040 breakinhalf
    fragment_max 0.20
}

setlayer litho_target = veb_retarget etch_target MAP etch_target \
OPTIONS MBROPTS
*/]
```

veb_pseudo_nyquist

Variable Etch Biasing (VEB) Commands

denseopc_options Keyword

Defines a pseudo-Nyquist value for automatic fragmentation.

Usage

veb_pseudo_nyquist *value*

Arguments

- ***value***

Specifies the pseudo-Nyquist value to be used for fragmentation.

Description

The veb_pseudo_nyquist command defines a “pseudo-Nyquist” value where:

pseudo-Nyquist = **scale * *value***

Typically, there are three ways to specify fragmentation for VEB retargeting:

1. Fragmentation based on veb_pseudo_nyquist.

In this case, Nyquist is assumed to be equal to veb_pseudo_nyquist; it is then used to determine the fragmentation base value.

2. Automatic fragmentation, based on the [image](#) command.

3. Automatic fragmentation, based on the loaded etch model. If neither methods 1 or 2 are specified, then 2.5 * etch mode pixel size (set by the [etch_imagegrid](#) command) is used as a basis for fragmentation.

Examples

```
veb_pseudo_nyquist 0.06
```

version

denseopc_options Keywords

Specifies the Calibre nmOPC 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 Calibre nmOPC algorithm to use. Since this is a required parameter, all setup files contain this keyword. This version applies only to the OPC 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

This command specifies that the default settings for the 2012.1 release version of the OPC algorithm be used.

```
version 2012.1
```

This command specifies that the defaults for the current running version of Calibre be used (it may or may not be 2012.1). Note that specifying “version 1” can produce changes in OPC behavior if defaults have changed between releases.

```
version 1
```

Performance

No impact

wafer_enclose

denseopc_options Keywords

Creates a layer-based cost function to check if a shape is enclosed within an image contour.

Usage

wafer_enclose *layer* [by *val*] [pw_condition *name*] [weight *w* | constrain]

Arguments

- *layer*

A layer to use for the cost function, which is compared to the image contour (either nominal contour or the pw_condition). This layer can be any layer defined in the dense OPC layer setup, including hidden, visible, or opc layers. If a contact layer is specified, then automatic corner rounding on the contacts will occur when computing the distance for the enclosure cost.

- by *val*

The amount of enclosure or internal distance requested for OPC. This can be a negative number. OPC tries to ensure at least this amount of enclosure, internal, or external distance between the contour and the *layer*. When this is omitted, it is assumed to be “by 0”.

- pw_condition *name*

The process window condition (defined by the [pw_condition](#) command) to apply the given cost function to. When the pw_condition name is omitted, it is assumed to be the nominal condition.

Note

 In PV Band PWOPC mode, these constraints cannot be specified for individual process windows using this option. Consequently, the pw_condition option will result in an error in PV band mode. The constraint value is uniformly enforced for all process window conditions in PV band mode.

- weight *w*

The weight to give to this cost function, relative to other cost functions already present. When this is omitted, the weight defaults to 1.

Note

 Weight-based wafer constraints and constrain are deprecated. They are used with algorithm 0 and algorithm 1.

- constrain

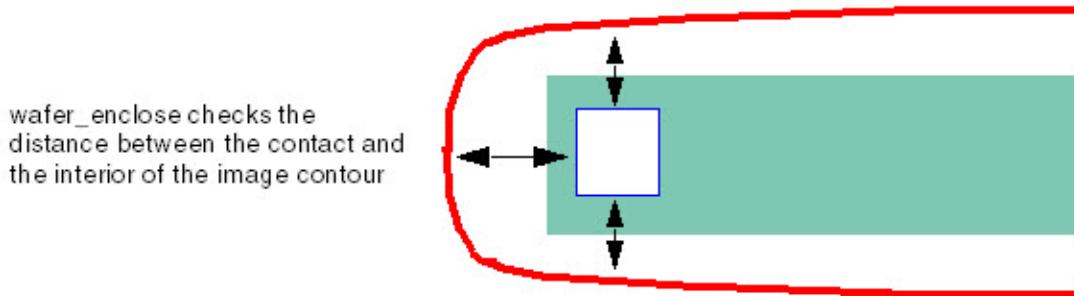
If this option is used, no weighting is done between the EPE and the wafer constraint EPE. Instead, the EPE is ignored when the wafer constraint must be obeyed.

In addition, the wafer constraint cost function becomes smoother (for example, no more big jumps around 0) to improve convergence.

Description

This command allows you to create layer-based cost functions to check if your polygons on a specified layer are properly enclosed within the image contour. [Figure 4-39](#) shows an example where a contact on a separate layer is compared to the image contour.

Figure 4-39. wafer_enclose Example

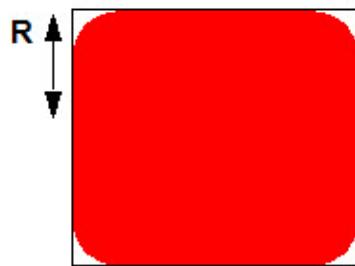


You can use wafer_enclose to specify a distance between the polygon edges and the interior of the contour in the cost function check. The image contour can either be based on the default nominal image (as specified by the image command), or a variant process window condition with alternate dose and focus (as defined using the [pw_condition](#) command). You can specify multiple wafer_enclosed cost functions.

If a contact layer is specified, then automatic corner rounding occurs on the contacts when computing the distance for the enclosure cost. The corner rounding is performed by considering the contact with perfect radial corner rounding as shown in [Figure 4-40](#). At points closer than a distance R to the corners, the target is rounded by a perfectly inscribed circle. At points farther than R from the corners, the target is flat.

The value for the radius R is set automatically to the Nyquist sampling distance which is equal to the largest sampling distance of aerial intensity that does not introduce aliasing errors.

Figure 4-40. Corner Rounding (wafer_enclose)



Examples

```
image optical opticsfile resist resistfile
pw_condition HIGHDOSE optical opticsfile dose 1.05 resist resistfile
wafer_enclose contact by 0
wafer_enclose contact by -0.010 pw_condition HIGHDOSE
```

wafer_enclosedby

denseopc_options Keywords

Creates a layer-based cost function to check if a shape is enclosed within a polygon on another layer.

Usage

wafer_enclosedby *layer* [by *val*] [pw_condition *name*] [weight *w* | constrain]

Arguments

- *layer*

A layer to use for the cost function, which is compared to the image contour (either nominal contour or the [pw_condition](#)). This layer can be any layer defined in the dense OPC layer setup, including hidden, visible, or OPC layers. If a contact layer is specified, then automatic corner rounding occurs on the contacts when computing the distance for the enclosure cost.

- by *val*

The amount of external distance requested for OPC. This can be a negative number. OPC tries to ensure at least this amount of enclosure, internal, or external distance between the contour and the *layer*. When this is omitted, it is assumed to be “by 0”.

- pw_condition *name*

The process window condition (defined by the [pw_condition](#) command) to apply the given wafer cost function to. When the pw_condition name is omitted, it is assumed to be the nominal condition.

Note

 In PV Band PWOPC mode, these constraints cannot be specified for individual process windows using this option. Consequently, the pw_condition option will result in an error in PV band mode. The constraint value is uniformly enforced for all process window conditions in PV band mode.

- weight *w*

The weight to give to this cost function, relative to other costs present. When this is omitted, the weight is assumed to be 1.

Note

 Weight-based wafer constraints and constrain are deprecated. They are used with algorithm 0 and algorithm 1.

- constrain

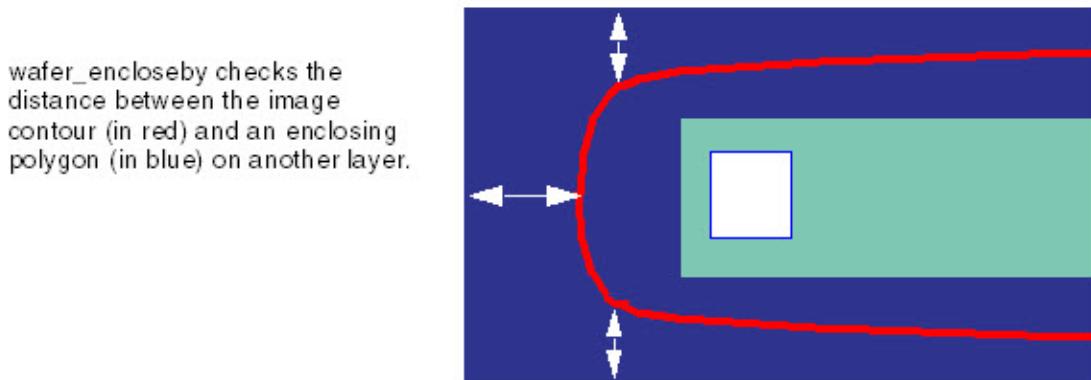
If this option is used, no weighting done between the EPE and the wafer constraint EPE. Instead, the EPE is ignored when the wafer constraint must be obeyed.

In addition, the wafer constraint cost function becomes smoother (for example, no more big jumps around 0) to improve convergence.

Description

This command allows you to create layer-based cost functions to check if your image contour is properly enclosed within a polygon on a specified layer (see [Figure 4-41](#)).

Figure 4-41. wafer_enclosedby Example



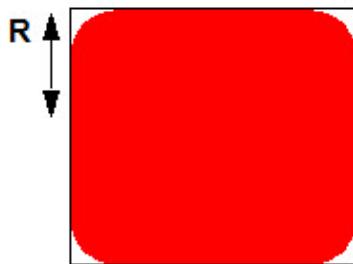
You can use wafer_enclosedby to specify a distance between the exterior of the image contour and the interior of the polygon in the cost function check. The image contour can either be based on the default nominal image (as specified by the image command), or a variant process window condition with alternate dose and focus (as defined using the [pw_condition](#) command). You can specify multiple wafer_enclosedby costs.

By default the wafer-based image costs apply to the nominal image, but can apply to any named pw_condition. The default weight is 0.

If a contact layer is specified, then automatic corner rounding occurs on the contacts when computing the distance for the enclosure cost. The corner rounding is performed by considering the contact with perfect radial corner rounding as shown [Figure 4-42](#). At points closer than a distance R to the corners, the target is rounded by a perfectly inscribed circle. At points farther than R from the corners, the target is flat.

The value for the radius R is set automatically to the Nyquist sampling distance which is equal to the largest sampling distance of aerial intensity that does not introduce aliasing errors.

Figure 4-42. Corner Rounding



Examples

```
image optical opticsfile resist resistfile  
pw_condition HIGHDOSE optical opticsfile dose 1.10 resist resistfile  
wafer_enclosedby OUTER
```

wafer_exclude

[denseopc_options Keywords](#)

Used in conjunction with PV Band PWOPC, prevents the image from getting too close to shapes on a specified layer.

Usage

wafer_exclude *layer* [by *value*]

Arguments

- *layer*

A required argument specifying the layer whose geometries should be avoided by the wafer image.

- by *value*

An optional argument that specifies a distance in user units to keep the image from the shapes on the layer. The default is 0.0.

Description

This optional keyword defines a wafer exclude constraint to force the image to stay away from the defined layer of geometry. The exclusion is only supported in PV Band PWOPC mode and ignored in other modes. The distance is uniformly enforced for all process window conditions.

The constraint is used when creating WAFER_EXCLUDE sites. By default, one site is created per fragment. To customize WAFER_EXCLUDE sites, use SITES_CREATE.

Examples

This keyword prevents wafer images from touching shapes on the m1_blue layer:

```
wafer_exclude m1_blue by 0.05
```

Related Topics

[Using PV Band PWOPC](#)

[SITES_CREATE](#)

Calibre nmOPC File Formats

Calibre nmOPC supports file formats listed in the following table.

Table 4-18. Calibre nmOPC File Formats

Command	Description
Lithomodel (Litho Model Format)	Loads modeling information to Calibre nmOPC that describes all aspects of a simulation including multiple exposure.

Lithomodel (Litho Model Format)

Calibre nmOPC File Formats

The litho model consolidates all of the model and mask information into a single object in order to define a manufacturing technology configuration. It is intended to unify simulation conditions across Calibre tools to reduce the possibility of errors.

A litho model is loaded when its directory name is detected within the `denseopc_options` block. The directory describes all aspects of a simulation, including multiple exposure. The litho model directory contains a file called *Lithomodel* that defines the specifics.

The current methods of specifying the information contained in the litho model are backwards compatible, but may be deprecated in a future release.

Many Calibre nmOPC keywords have a different syntax for litho models. Within a setup file, you cannot mix litho model syntax and traditional syntax.

- Any `background`, `ddm_model_load`, `etch_model_load`, `flare_model_load`, `optical_model_load`, `resist_model_load`, and `topo_model_load` commands are ignored. The values in the litho model are used instead.
- The setup file must use litho-model syntax for the [image](#), [layer](#) (for [denseopc_options Block](#)), and [pw_condition](#) commands.
- The outer [layer](#) command is only used to set the association-by-order of SVRF input layers to the names used in the setup file, and requires only the name alias of the layer. Other parameters for transmission and type are ignored.
- The setup file cannot include [flare_model](#) or [topo_model](#).

Format

A litho model must conform to the following restrictions:

- Keywords in the *Lithomodel* file can be in nearly any order; the exception is that **version** must be the first line. Only mask blocks may occur more than once:
 - There can be multiple mask *n* blocks. Each must have a unique mask number *n*.
 - Within a mask block, there can be multiple `mask_layer` statements. Each must have a unique `mask_layer` index *m*.
 - Within a mask block, there can be multiple optical statements. Each must have a unique focus value.
- The model files that comprise the litho model must be linked from or preferably in the same directory as the *Lithomodel* file. (Use the name of the directory to load litho models.)

- A *Lithomodel* file uses the following format:

```
version 1
resist filename
[image_threshold value]
[etch filename]
[topo filename]
[flare filename
[black_border filename] ]
[shadow_bias filename]
[cdsa filename]
[stochastic filename]
[zplanes filename]
mask n {
    background trans
    mask_layer m [TRANS transmission] [CX val] [CC val] [ALL_ANGLES]
        [BIAS val] [XBIAS xval] [YBIAS yval]
        [XSHIFT xval] [YSHIFT yval]
        [DDM filename]
        [CATEGORY name]
    {optical filename[focus fnm] }+
    [dose val]
}
[zplanes filename]
```

Parameters

- **version 1**

A required keyword that specifies the version. This keyword must always be the first parameter specified. Currently, the only available version is 1.

- **resist *filename***

A required keyword that specifies the name of a resist model file, without path information. The same resist model is used for all exposures, and only one may be specified. However, the Calibre OPCverify [setlayer image](#) command provides an override to generate constant-threshold aerial contours

- **image_threshold *value***

An optional keyword that specifies a “best” constant threshold value, typically derived from the calibration model flow. If specified, this value may be used in Calibre OPCverify [setlayer image](#) command through the “aerial model” parameter. The same value is used for all mask exposures.

- **etch *file***

An optional keyword that specifies the name of a VEB etch model file, without path information. If specified the model is used for all exposures.

- **topo *filename***

An optional keyword that specifies the name of an topographical model file without path information. If specified the model is used for all exposures.

See “[Topography Modeling](#)” on page 50 and the *Calibre WORKbench Topography Modeling User’s and Reference Manual* for information on topo models.

- **flare *filename***

An optional keyword that specifies the EUV (flare) model name. If specified the model is used for all exposures. EUV models require in LITHO EUV OPCVERIFY and LITHO EUV DENSEOPC.

See “[Models in EUV Lithography](#)” on page 50 and “[Flare Model File Format](#)” in the *Calibre WORKbench User’s and Reference Manual* for information on flare models. If you are also using long range information for the flare model, you must also use the [flare_longrange](#) command.

If you are using global litho models, also see “[EUV Through-Slit Litho Model Extension](#)” in the *Calibre OPCverify User’s and Reference Manual*.

- **black_border *filename***

An optional keyword that specifies the EUV black border model name. See “[Black Border Model File Format](#)” in the *Calibre WORKbench User’s and Reference Manual* for information on black border models.

The file must also specify “*flare filename*” if it specifies *black_border*.

- **shadow_bias *filename***

An optional keyword that specifies the shadow bias model name. See “[Models in EUV Lithography](#)” on page 50 for further information on shadow bias models. Note that if a shadow bias model is used in either simulation or setlayer *shadow_bias*, the Calibre OPCverify setup command *euv_slit_x_center* must be specified, even if it is zero.

- **cdsa *filename***

An optional keyword that specifies the compact DSA model name. See the *Calibre Directed Self-Assembly User’s and Reference Manual* for further information on DSA models.

- **stochastic *filename***

An optional parameter that specifies a stochastic compact model file for EUV modeling.

- **zplanes *filename***

An optional parameter that specifies a filename containing additional information for special imaging conditions used for SRAF print avoidance or 3D images for toploss models.

- **mask *n* { }**

A required command block that defines different masks used in multiple exposures. These define different masks used in multiple exposures. The braces ({}) are required. All mask numbers must be sequential. The following must always be defined in a mask command block:

- Background transmission defined with the **background** keyword.

- Nominal-focus optical mode using the **optical** keyword. (Off-focus models are always optional.)
 - mask_layer 0
- **background trans**

A required keyword for the **mask** command block that defines the background imaging characteristics and is used instead of any background statement in the setup file. Refer to [background \(for denseopc_options block\)](#) for information on possible transmission values.
 - **mask_layer m**

A required keyword for the **mask** command block. The mask_layer parameter specifies the characteristics of a mask layer template. The template contains a list of information such as transmission, geometry preprocessing, DDM model, and so on, and replaces the values specified by the layer statement.

For each mask, “mask_layer 0” is required. Other mask_layer values are optional, and if defined must be numbered sequentially. When coding mask_layer assignments in a setup file, it is not required that all templates in the mask model be used.

Note
If you use **mask_layer** with [setlayer veb_retarg](#)t, do not specify geometry parameters (CC, CX, XSHIFT, YSHIFT, BIAS, XBIAS, YBIAS).
 - **TRANS transmission**

A parameter for **mask_layer** that specifies one of the following mask transmission values: dark, clear, atten x, phase90, phase270, phase180, or (re im) pair. See the [layer \(for denseopc_options Block\)](#) command for information on transmission values.

The TRANS argument is required except when this layer is an asraf layer, in which case it is prohibited because asraf layers are holes that use the background transmission value.
 - **[CX val] [CC val]**

An optional parameter for **mask_layer** that specifies convex (CX) and concave (CC) cornerchop values (in microns) to be applied to the mask layer before using them in a simulation.

Note
For litho model flows, corner-chopping behavior defaults to chopping right-angled corners only. This is different from the non-litho model flows, in which “setlayer cornerchop” and “mask_chip_corner” by default chip all-angle corners.
 - **ALL_ANGLES**

Optionally specifies that corner chopping occur on all-angle corners, not just right-angled corners. This is only permitted if CX or CC is also specified.

Note



This option can create skewed edges, which are not permitted with some features such as DDM.

- [BIAS *val*] [XBIAS *xval*] [YBIAS *yval*] [XSHIFT *xval*] [YSHIFT *yval*]

Optional parameters for **mask_layer** that specify bias and shift values (in microns) to be applied to the mask layer before using them in a simulation. If BIAS is specified, XBIAS and YBIAS are not permitted, as well as the converse. Note the following:

- When BIAS is specified in the setlayer image command as well as in this litho model, Calibre OPCverify adds the values of BIAS together.
- How BIAS affects holes and negative SRAFs depends on the layer type. For asraf layers, positive values for BIAS makes the shapes defining the negative SRAFs larger. This means that the holes created by the asraf cutouts will also be larger. However, if the negative SRAFs are included as holes in the main feature layer, increasing the BIAS value makes the holes smaller.
- If a mask_layer specifying XBIAS, YBIAS, XSHIFT or YSHIFT is used in a setlayer image command in Calibre OPCverify or in Calibre nmOPC, Calibre clones all rotated or reflected cell transforms.
- Geometry preprocessing before simulation is applied in the following order:
 - i. BIAS
 - ii. XSHIFT and YSHIFT
 - iii. XBIAS and YBIAS
 - iv. Corner chopping.
- Large values of XSHIFT and YSHIFT are not permitted for performance reasons. It is recommended that shifts be ≤ 1.0 micron. For larger values, use the SVRF SHIFT command instead.

- DDM *filename*

An optional parameter for **mask_layer** that specifies the name of a DDM model without path information. The DDM model must be in the litho model directory (or may be a link in that directory). DDM models cannot be used with an asraf layer.

- CATEGORY

An optional parameter for **mask_layer** specifies a comment (typically a name) documenting the usage of this mask_layer template. This comment will not be referenced in setup files. For example:

```
CATEGORY sraf
```

- **optical *filename***

A required keyword for the **mask** command block that specifies the name of an optical model file, without path information. The optical model must be located in the litho model directory (or may be a link in that directory).

At least one optical parameter is required inside a mask block to specify the nominal focus optical model. A mask block can have multiple optical statements, but each one must have a unique focus value. If no focus parameter is specified, the file is treated as the nominal optical model.

- **focus *fnm***

An optional keyword for the **mask** command block that defines a symbolic name for the focus condition under which the optical model was created. This symbolic name will be referenced in various setup file commands.

The optional focus parameter is a symbolic name for an off-focus optical model, where *f* is a positive or negative integer in nanometers. (Note the “nm” is a literal part of the argument, such as “-20nm”, and the decimal form of the integer is also accepted, such as “20.000nm”, but “20.334nm” is treated as 20 nanometers).

If not specified, nominal focus (focus 0nm) is assumed. In setup files, simulations may use only the specific focus conditions defined in the litho model.

- **dose *val***

An optional keyword for the **mask** command block that defines a relative base dose for a mask. It is intended for multiple exposure setups, where the mask exposures may have different relative doses.

Caution

 When doses are specified in setup files, in the setlayer image or in the image or **pw_condition** commands in the denseopc_options block, the actual dose used is derived by multiplying together the setup file dose and the litho model base dose for that mask. If not specified, the default is 1.0.

- **zplanes *filename***

An optional parameter that specifies a zplanes model that is used for all exposures. (See “[ZPlanes Model Format](#)” in the *Calibre OPCverify User’s and Reference Manual* for more information.)

Examples

The following is a basic example of the *Lithomodel* contents and directory structure:

```
% cat duv10/Lithomodel

version 1
resist mymodel.mod
mask 0 {
    background clear
    mask_layer 0 TRANS dark CATEGORY main
    optical duv193_nom
}
% ls duv10
    Lithomodel
    duv193_nom
    mymodel.mod
```

The following is a double exposure model that uses the same optical model on both exposures with a multiple focus and a CTR threshold:

```
% cat duv10_2exp/Lithomodel

version 1
resist mymodel.mod
image_threshold 0.125
mask 0 {
    background clear
    mask_layer 0 TRANS atten 0.06 CATEGORY main
    optical duv193_nom_x
    optical duv193_m20x focus -20nm
    optical duv193_p20x focus 20nm
}
mask 1 {
    background clear
    mask_layer 0 TRANS atten 0.06 CATEGORY main
    optical duv193_nom_y
    optical duv193_m20y focus -20nm
    optical duv193_p20y focus 20nm
    dose 1.1
}
```


Chapter 5

Calibre nmOPC Custom Tcl Scripting Reference

Calibre nmOPC supports a Tcl-based scripting language which provides a highly customizable environment for Calibre nmOPC with a Tcl interface.

Custom Tcl-Based Script Creation	369
Variables in Scripting	371
Outputs in Calibre nmOPC Scripts	371
Tag Scripting	372
Creating New Tag Sets	372
NEWTAG Output	373
Tag Set Operations	373
Tag Set Variables	373
Visible Layers and Tagging	374
Tagging Controls	374
Fragmentation Controls	376
Fragment-Based Operations	376
Measurement Controls	378
Dense Simulation Controls	378
Output Controls	379
Target Modification	379
OPC Controls	380
Unit Conversions and Miscellaneous	380
Site Controls	381
Calibre nmOPC Tcl Scripting Commands	382
auto_target_control	389
CLASSIFY_FRAGMENTS	392
DBU2UU	397
DENSE_CD	398
DENSE_EPE	400
DENSE_SIMULATE	402
DISPLACEMENT	404
EUV_MODEL_REDUCTION	409
FEEDBACK	413
fragmaxedge	416
FRAGMENT delete	418

FRAGMENT end	419
FRAGMENT modify	420
FRAGMENT split	422
FRAGMENT_CONFORM	424
FRAGMENT_EPE_MEASURE_METHOD	426
FRAGMENT_GET	427
FRAGMENT_GET_DISPLACEMENT	429
FRAGMENT_ITERATOR first/next/release	431
FRAGMENT_MAX_DISPLACEMENT	433
FRAGMENT_MOVE	435
FRAGMENT_SET_DISPLACEMENT	437
fragtypetag	439
GET_FRAGMENT_EPE_INFO	441
MEASURE	442
MEASURE_PER_FRAGMENT	444
MEASURE_PER_FRAGMENT_QUERY_COUNT	447
MEASURE_PER_FRAGMENT_QUERY_FIRST	448
MEASURE_PER_FRAGMENT_QUERY_NEXT	450
ML_OP_C	452
ML_VECTOR_CAPTURE_END	454
ML_VECTOR_CAPTURE_INIT	458
MRC	460
MRC_RULE (Custom Scripting Command)	463
NEWTAG all	466
NEWTAG area	467
NEWTAG blocked	469
NEWTAG cd	470
NEWTAG complete	472
NEWTAG corner	474
NEWTAG correction_map	476
NEWTAG displacement	477
NEWTAG edge	479
NEWTAG epe	483
NEWTAG expression	485
NEWTAG external internal enclose enclosing	490
NEWTAG facing_pattern	496
NEWTAG fragment	498
NEWTAG line_end space_end	503
NEWTAG mrcViolation	504
NEWTAG neighbor	505
NEWTAG property	507
NEWTAG sadp	509
NEWTAG sequence	512
NEWTAG sites	514
NEWTAG topological	518

NEWTAG vertical horizontal all_angle 45_angle	520
NMBIAS_MEASURE_TAG	521
NYQUIST	528
OPC_ITERATION	529
OPC_XMEEF_ITERATION	541
OUTPUT_IMAGE	549
OUTPUT_MEASUREMENT_LOCATIONS	550
OUTPUT_OPCT	551
OUTPUT_RETARGET	553
OUTPUT_SHAPE fragment	554
OUTPUT_TARGET_CONTROL	558
PW_RETARGET	559
SET_EPE	560
SET_MEDIAN_EPE	564
SITES_CREATE	565
SITES_DELETE	570
SITES_DUMP	571
SITES_SHIFT (Deprecated in 2020.3)	573
sraffragalign	575
sraf_print_avoidance	577
sraf_sites_create	584
TAG add remove ismember	586
TAG and or subtract	588
TAG fromlist	590
TAG size	591
TAG tolist	592
TAG_FRAG_SRC	593
TAG_MRC_SRC	595
TARGET_CONTROL	597
target_curve	600
TARGET_FUNCTION	606
TDBIAS	609
UU2DBU	611

Custom Tcl-Based Script Creation

The Tcl-based script can be placed anywhere within a denseopc_options block. This allows for the script to be precompiled and checked for errors at parse time, and may enable a better controlled run time (the Tcl script becomes a string of valid Calibre nmOPC script commands, so there is no user Tcl code being executed per tile, at run time).

An example of this method is as follows:

```
denseopc_options dense_opts {
    version 1
    background clear
    layer POLY opc dark mask 0
    layer SX visible dark mask 0
    layer SY hidden dark mask 0
    max_iterations 9
    image optical opt aerial 0.20
    scale 1.5

### The following is a tagging script
    NEWTAG all POLY -out A
    NEWTAG edge A len < 0.3 corner1 convex corner2 convex -out line_end
    NEWTAG neighbor both line_end len < 0.4 corner convex -out line_end_adj
    NEWTAG neighbor both line_end_adj len < 0.4 corner no_corner -out \
        secondfrag
    NEWTAG neighbor prev line_end len < .4 corner convex -out line_end_adj0
    NEWTAG neighbor next line_end len < .4 corner convex -out line_end_adj1
    FRAGMENT modify pt1 line_end_adj0 0.05 0.1 0.1
    FRAGMENT delete pt1 line_end_adj0
    NEWTAG all POLY -out A
    NEWTAG edge A len < .3 corner1 convex corner2 convex -out line_end
    NEWTAG neighbor prev line_end len < 0.4 corner convex -out \
        line_end_adj0
    FRAGMENT split line_end_adj0 0.4 0.1
    OPC_ITERATION 8

    NEWTAG all POLY -out A
}
```

The following conditions apply to this method of tagging:

- Many of these commands trigger “tagscript” mode. This mode changes the following tool behavior:
 - The interaction distance typically increases.
 - The max_iterations command does not trigger simulation. You must use OPC_ITERATION or OPC_XMEEF_ITERATION.
- If you use “list” type variables like:

```
setenv algo "algorithm 1"
```

you must instead use eval “\$env(algo)” in the setup file.
- All “puts” statements will work in tag script mode, but since the tag script is evaluated at parse time, you will see “puts” outputs at that time (not at run time).

Note

 When specifying layers for Tcl-based customization commands (such as [TDBIAS](#) or [NEWTAG all](#)), use layer names only and not layer numbers. Using layer numbers in Tcl customization commands will generate an error.

Related Topics

[Tcl Library in Calibre \[Calibre Administrator's Guide\]](#)

Variables in Scripting

Variables in Calibre nmOPC custom scripting are defined by the VARIABLE SVRF command and passed in.

The [svrf_var_import](#) command, defined outside the [denseopc_options](#) block, is used to import variables from SVRF.

Outputs in Calibre nmOPC Scripts

Calibre nmOPC scripts can create multiple output layers. These layers can serve a number of different purposes, including:

- OPC corrections
- Fragments
- Contours
- Target modifications
- Measurement locations

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. Some ways you might use tagging would be:

- Identify fragments that will or will not be subject to OPC.
- Identify fragments requiring OPC.
- Limit or otherwise control the movement for specific fragments.
- Highlight key fragments for inspection after processing.
- Define different width and spacing rules for different types of fragments.

Creating New Tag Sets	372
NEWTAG Output	373
Tag Set Operations	373
Tag Set Variables	373
Visible Layers and Tagging	374

Creating New Tag Sets

In Calibre nmOPC, the NEWTAG command is used to create new tag sets.

These include the following types of sets, as shown in [Table 5-1](#).

Table 5-1. NEWTAG Operations

Predefined Fragments	OPC Analysis	Geometric
NEWTAG all	NEWTAG blocked	NEWTAG complete
NEWTAG vertical horizontal all_angle 45_angle	NEWTAG cd	NEWTAG edge
NEWTAG line_end space_end	NEWTAG epe NEWTAG expression NEWTAG displacement NEWTAG sequence	NEWTAG fragment NEWTAG external internal enclose enclosing NEWTAG neighbor

For example, the following code tags all edges on layer L1, placing them in a labeled tag set named T0.

```
NEWTAG all L1 -out T0
```

NEWTAG Output

Output from a NEWTAG operation is a unique name that refers to a set of fragments or edges. To reference a tag set:

- Tag sets are typically referenced by their names only.
- A tag set can also be referred to by using `$<name>`. The `$<name>` syntax is required when passing tag sets to Tcl processes, and when creating a Tcl list of tag sets.

Tag Set Operations

Once you have created your tags, you can control the effect the RET tools have on the tagged edges.

You can use tagging commands to control the movement of edges during OPC. You can set limits on how far edges can move in OPC or specify the type of OPC to perform on them.

The TAG command is used to manipulate existing tag sets, as shown in [Table 5-2](#).

Table 5-2. Tag Set Operations

Supported in denseopc_options block		
TAG and or subtract		
Supported in Calibre nmOPC Tcl Scripting block		
TAG fromlist	TAG add remove ismember	TAG and or subtract
TAG tolist	TAG size	

Tag Set Variables

As the tagging script is Tcl-based, the variables that hold tag sets are passed by name into functions that use tag sets. This eliminates the need to use large numbers of `$tagname` references in the code, but can cause unusual behavior when passing tag sets into processes.

For example, the following example sets the displacement for a tag set.

```
NEWTAG all L0 -out mytagset
FRAGMENT_SET_DISPLACEMENT mytagset 0.002 (CORRECT)
```

The following code is incorrect:

```
NEWTAG all L0 -out mytagset
FRAGMENT_SET_DISPLACEMENT $mytagset 0.002 (NOT CORRECT)
```

Tcl functions that use tag sets as input do not require a dollar sign (\$) before the variable name. This is because the tag variable name is dereferenced internally in the function to obtain the tag set (in other words, the dollar sign is applied internally). This is similar to how array variables

are treated in Tcl. For example, to convert an array to a list, you do not need to specify a dollar sign for the array variable name:

```
array get A
```

However, there are limits to not using *\$tagname*. In order to pass a tag set into “proc,” the dereferencing operator (\$) must be used. This is shown in the following example:

```
proc apply_displacement {tag_set value} {
    FRAGMENT_SET_DISPLACEMENT tag_set $value
}
NEWTAG all L0 -out mytagset
apply_displacement $mytagset 0.002
```

The tag set is dereferenced to pass it into the processes, but when the tag set is actually inside the process, you no longer need to dereference the variable.

As a general rule, tags should not be treated as if they are names. Only tagging commands can automatically apply the dollar sign to tag variables in order to access them. Typically, Tcl commands such as “proc” cannot apply a dollar sign to tag variables. Instead, you must place the variables in a list as in the following example:

```
set tag_list [$line_end $concave $convex]
```

Visible Layers and Tagging

Tagging is only supported for the opc or correction layers, thus tagging visible layers directly will generate an error. However, as layers can be specified in the tag-based MRC rules, you can effectively specify a visible layer using this method. This is the correct way to achieve MRC rules between tags that are not on the same layer (for example, an opc feature and an sraf layer).

Tagging Controls

There are a number of tagging-related commands legal for Calibre nmOPC Tcl-based custom scripting.

Table 5-3 lists the available tagging commands.

Table 5-3. Calibre nmOPC Tcl Custom Script Tagging Commands Summary

Command	Description
NEWTAG all	Creates a fragment set consisting of all fragments.
NEWTAG area	Returns a set of fragments based on the width and length of the polygon.
NEWTAG blocked	Checks for fragments that had movement blocked by an MRC rule.

Table 5-3. Calibre nmOPC Tcl Custom Script Tagging Commands Summary

Command	Description
<code>NEWTAG cd</code>	Measures the CD and outputs a given bin.
<code>NEWTAG complete</code>	Outputs all fragments on the edge (if the edge meets length and corner constraints).
<code>NEWTAG corner</code>	Performs a search for fragments based on distance from a corner.
<code>NEWTAG correction_map</code>	Outputs correction layer fragments that are mapped to OPC layer fragments.
<code>NEWTAG displacement</code>	Returns a set of fragments whose displacements are in the given range which is specified in user units.
<code>NEWTAG edge</code> <code>NEWTAG fragment</code>	Performs a search for a fragment whose length and corner types match a constraint.
<code>NEWTAG epe</code>	Measures the EPE and outputs a given bin.
<code>NEWTAG external internal enclose enclosing</code>	Computes distance as internal, external, or enclosure from fragments in one layer to fragments in another.
<code>NEWTAG facing_pattern</code>	Generates fragment tag sets of same and opposite facing patterns.
<code>NEWTAG line_end space_end</code>	Computes pre-defined line end and space-end definitions.
<code>NEWTAG mrcViolation</code>	Outputs fragments that violate MRC.
<code>NEWTAG neighbor</code>	Creates a new set of fragments containing the previous, next, or both neighbors.
<code>NEWTAG sadp</code>	Classifies fragments according to the relationship between associated cut masks and target layers.
<code>NEWTAG sequence</code>	Creates new tag set(s) holding fragments from a sequence of edges.
<code>NEWTAG topological</code>	Performs the indicated topological operation on the given input layers.
<code>NEWTAG vertical horizontal all_angle 45_angle</code>	The output set contains fragments whose edge slopes match the specification.

Table 5-3. Calibre nmOPC Tcl Custom Script Tagging Commands Summary

Command	Description
TAG add remove ismember	Individual element membership query and manipulation.
TAG fromlist	Converts a list of fragment ids (long integer) into a tag set.
TAG size	Returns the size of a tag set.
TAG tolist	Convert a tag set to a list.
TAG and or subtract	Boolean set operations on multiple tag sets.
TAG_FRAG_SRC	Creates a new tag set based on which fragmentation rule created the fragment.
TAG_MRC_SRC	Creates an annotated tag set based on which MRC_RULE is blocking fragment movement.

Fragmentation Controls

You can create extensions to existing fragmentation schemes as well as create entirely new fragmentation algorithms using the Calibre nmOPC customization script.

The following table lists the Calibre nmOPC Tcl-based fragmentation controls.

Table 5-4. Fragmentation Controls Summary

Command	Description
FRAGMENT delete	Deletes either a single fragment, or set of fragments.
FRAGMENT end	Splits fragments into multiple 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-Based Operations

You can apply different operations to fragments, including measurements, retrieving EPE information, and setting displacements.

[Table 5-5](#) lists the Calibre nmOPC Tcl-based fragmentation-based operations for performing operations on fragments during OPC.

Table 5-5. Fragment-Based Operations Commands Summary

Command	Description
<code>auto_target_control</code>	Automatic target control used to adjust a small four-point polygon for convergence
<code>fragmaxedge</code>	Applies the Calibre OPCpro minedgelength operation, but only to the fragments in the specified tag set.
<code>fragtypetag</code>	Creates a tag set based on a specified fragmentation operation type.
<code>FRAGMENT_EPE_MEASURE_METHOD</code>	Allows you to select between minimum, maximum, or average values of EPE for a tag set.
<code>FRAGMENT_GET</code>	Dumps the raw fragment information into a Tcl associative array named by the caller.
<code>FRAGMENT_GET_DISPLACEMENT</code>	Gets the fragment displacement for a single fragment or for an entire set of fragments (in array mode).
<code>FRAGMENT_SET_DISPLACEMENT</code>	Sets a “desired displacement” for one or more fragments.
<code>FRAGMENT_ITERATOR first/next/release</code>	Allows the caller to iterate over active fragments.
<code>GET_FRAGMENT_EPE_INFO</code>	Retrieves the EPE measurement method and value for specified fragment(s).
<code>NEWTAG expression</code>	Evaluates numerical properties about or attached to fragments.
<code>NEWTAG property</code>	Annotates a tag set with DFM properties.
<code>PW_RETARGET</code>	Applies PWOPC-based retargeting for OPC fragments in either a tag set or for all fragments.
<code>SET_EPE</code>	Sets EPEs on fragments, based on an input Tcl formula or expression.
<code>SET_MEDIAN_EPE</code>	Sets median EPE for a tag set of fragments.
<code>TAG_FRAG_SRC</code>	Gets the fragmentation rule that created the fragment.
<code>TDBIAS</code>	Sets the displacement (requires a call to <code>FRAGMENT_MOVE</code> unless the <code>-force</code> flag is used).

Measurement Controls

Create custom measurements for fragment length, distances, EPE, and CD using the Tcl commands.

The measurement controls are listed in [Table 5-6](#).

Table 5-6. Calibre nmOPC Tcl-Based Measurement Commands Summary

Command	Description
MEASURE	Calculates distances from fragments on the first input layer to nearest fragments on the second input layer.
MEASURE_PER_FRAGMENT	Calculates distances from the specified fragment to the specified input layer.
MEASURE_PER_FRAGMENT_QUERY_COUNT	Returns the total number of retrievable edges.
MEASURE_PER_FRAGMENT_QUERY_FIRST	Initiates retrieval of edge data recorded by MEASURE_PER_FRAGMENT.
MEASURE_PER_FRAGMENT_QUERY_NEXT	Retrieves subsequent edge data recorded by MEASURE_PER_FRAGMENT (following MEASURE_PER_FRAGMENT_QUERY_FIRST).

Dense Simulation Controls

Calibre nmOPC bases all of its operations on a pixel-based dense simulation grid. Measuring EPE and CDs are performed on a dense simulation of the layout.

The customization script supports several commands (listed in [Dense Simulation Controls](#)) that enable you to control the computations for dense simulations images as well as EPE and CD measurements.

[Table 5-7](#) lists the Calibre nmOPC dense simulation commands.

Table 5-7. Dense Simulation Commands Summary

Command	Description
DENSE_SIMULATE	Concurrently computes aerial or resist images for several dose values and inserts them into the user-specified array of image contour registers.

Table 5-7. Dense Simulation Commands Summary (cont.)

Command	Description
DENSE_CD	Measures CDs on a pre-computed contour for locations selected automatically on a given fragment.
DENSE_EPE	Measures EPEs on a pre-computed contour for locations selected automatically on a given fragment.

Output Controls

Several commands are also available to output shapes, images, and other information from your layout.

Table 5-8 lists the output shapes commands.

Table 5-8. Output Commands Summary

Command	Description
OUTPUT_IMAGE	Outputs the given image register to a layer.
OUTPUT_MEASUREMENT_LOCATIONS	Outputs EPE measurement locations.
OUTPUT_OPCT	Outputs OPC layout of specified layer.
OUTPUT_RETARGET	Outputs retargeted opc layers.
OUTPUT_SHAPE fragment	Outputs simple shapes around fragments.
OUTPUT_TARGET_CONTROL	Outputs shapes representing the modified target, for the tagged fragments which have had TARGET_CONTROL applied.
NYQUIST	Outputs the Nyquist value in either database units or user units.

Target Modification

Several target function commands are also available to assign fragments in tag sets to particular functions and bind them to function slots.

Table 5-9 lists the target function commands.

Table 5-9. Target Function Controls Summary

Command	Description
TARGET_CONTROL	Associates fragments in a tag set to a specific target function.

Table 5-9. Target Function Controls Summary (cont.)

Command	Description
target_curve	Creates a curve from the target based on the command settings, then moves the EPE sites to the new curve.
TARGET_FUNCTION	Defines a target function and binds it to a target function slot.

OPC Controls

Several commands are also available to optimize your OPC performance.

[Table 5-10](#) lists the Calibre nmOPC utility commands.

Table 5-10. OPC Controls Summary

Command	Description
Layer Operations	
DISPLACEMENT	Sets FROZEN/FIXED status, displacement, desired displacement, and displacement limits for all fragments in the given tag.
FRAGMENT_MOVE	Moves all fragments to their desired displacements simultaneously, while obeying mask constraints.
OPC_ITERATION	Performs a “standard” iteration as defined by all setup file user settings.
OPC_XMEEF_ITERATION	Performs one or more Matrix OPC iterations, using an alternative algorithm for faster calculations of XMEEFs.
Tag Set Operations	
FEEDBACK	This command sets the feedback for one or more fragments.
MRC	Establishes tag-specific adjustment of width or spacing between fragments after OPC movement.
MRC_RULE (Custom Scripting Command)	Creates external and internal spacing constraints based on tag sets.

Unit Conversions and Miscellaneous

Output data from Calibre nmOPC script commands contain coordinates and distances, measured in either microns or in database units.

The following commands listed in [Table 5-11](#) allow you to convert values from one form to the other.

Table 5-11. Unit Conversion and Miscellaneous Summary

Command	Description
Unit Conversions	
DBU2UU	Converts the user-supplied database units into user units (usually microns) which is returned in the Tcl result.
UU2DBU	Converts the user-supplied user units (usually microns) into database units which is returned in the Tcl result.
Miscellaneous	
NYQUIST	Outputs the Nyquist value in either database units or user units.
sraffragalign	This command aligns fragment breakpoints on sraf layers for sraf_print_avoidance .
sraf_print_avoidance	Prevents selected Sub-Resolution Assist Feature (SRAF) from printing.
sraf_sites_create	Creates sites for SRAF layers.

Site Controls

Several commands have been implemented for you to optionally add and modify the position of intensity sampling points, otherwise known as sites.

This allows you to define intensity sites before running the simulation and then get intensity information after the simulation is done. [Table 5-12](#) lists the site control commands.

Table 5-12. Site Controls Summary

Command	Description
SITES_CREATE	Create sites centered on the target layer.
SITES_DELETE	Deletes all sites associated to fragments on the input tag set.
SITES_DUMP	Outputs all sites associated to fragments on the input tag set.
SITES_SHIFT (Deprecated in 2020.3)	Shifts all sites (or sites of the given type) associated to fragments on the input tag set.

Calibre nmOPC Tcl Scripting Commands

Tcl-based custom scripting commands can be used within a denseopc_options scripting block.
In the following figure, a Tcl scripting block is highlighted in red.

Figure 5-1. Setup File Location for Tcl Scripting Commands

```
m1_out = LITHO DENSEOPC met1 sraf
FILE m1_opc MAP m1_opc
LITHO FILE m1_opc /*

modelpath ./models
optical_model_load opt m1.opt
resist_model_load res m1.cml

layer m1 hidden clear
layer sraf hidden clear

denseopc_options m1_opt {

    version 1
    background poly opc atten 0.06
    layer m1 opc clear
    ...

    NEWTAG all POLY -out A
    NEWTAG edge A len < 0.3 corner1 convex corner2 convex
        -out line_end
    NEWTAG neighbor both line_end len < 0.4 corner convex \
        -out line_end_adj
    ...
    OPC_ITERATION 8
    NEWTAG all POLY -out A

}

setlayer m1_opc = denseopc m1 sraf MAP m1 OPTIONS m1_opt
```

Calibre nmOPC Tcl Scripting
Commands

The following table lists all Tcl-based custom scripting commands supported by Calibre nmOPC.

Table 5-13. Calibre nmOPC Custom Scripting Commands Summary

Command	Description
auto_target_control	Automatic target control used to adjust a small four-point polygon for convergence.

Table 5-13. Calibre nmOPC Custom Scripting Commands Summary (cont.)

Command	Description
CLASSIFY_FRAGMENTS	Classifies fragments on OPC layers into sets based on surrounding edges. The command can also adjust fragment EPE and displacement to conform to a median range.
DBU2UU	Converts database units (dbus) into user units.
DENSE_CD	Measures CDs on a contour and outputs information into an array.
DENSE_EPE	Measures EPEs on a contour and outputs information into an array.
DENSE_SIMULATE	Computes aerial or resist images and outputs information into an array.
DISPLACEMENT	Sets FROZEN/FIXED status, displacement, desired displacement, and displacement limits for all fragments in the given tag set.
EUV_MODEL_REDUCTION	Toggles model reduction mode for EUV Dense OPC. May be specified at most twice.
FEEDBACK	Sets the feedback factor used to determine edge movement for specific fragments, overriding the global value set by the feedback command.
fragmaxedge	Applies sparse OPC minedgelength operation only to fragments in specified tag set.
FRAGMENT delete	Deletes fragments.
FRAGMENT end	Splits fragments into multiple fragments.
FRAGMENT modify	Shifts the ordered endpoint of a given fragment.
FRAGMENT split	Splits fragments.
FRAGMENT_CONFORM	Moves fragments to the position of similarly oriented fragments within a short distance on a marker layer.

Table 5-13. Calibre nmOPC Custom Scripting Commands Summary (cont.)

Command	Description
<code>FRAGMENT_EPE_MEASURE_METHOD</code>	Allows you to select between minimum, maximum, or average values of EPE for a tag set.
<code>FRAGMENT_GET</code>	Obtains fragment information and outputs it to an array.
<code>FRAGMENT_GET_DISPLACEMENT</code>	Obtains fragment displacement information.
<code>FRAGMENT_ITERATOR first/next/release</code>	Iterates OPC over active fragments.
<code>FRAGMENT_MAX_DISPLACEMENT</code>	Sets a maximum limit on how far a target fragment will move.
<code>FRAGMENT_MOVE</code>	Moves fragments.
<code>FRAGMENT_SET_DISPLACEMENT</code>	Sets a specific displacement for a fragment.
<code>fragtypetag</code>	Creates a tag set containing all fragments created by a specified fragmentation type.
<code>GET_FRAGMENT_EPE_INFO</code>	Retrieves EPE measurement method and value for specified fragment(s).
<code>MEASURE</code>	Measures distances from fragments on one layer to the nearest fragments on another layer.
<code>MEASURE_PER_FRAGMENT</code>	Measures against all the fragments in the search range.
<code>MEASURE_PER_FRAGMENT_QUERY_COUNT</code>	Returns number of retrievable fragment edges.
<code>MEASURE_PER_FRAGMENT_QUERY_FIRST</code>	Initiates a query to retrieve fragment edges.
<code>MEASURE_PER_FRAGMENT_QUERY_NEXT</code>	Queries data from subsequent fragment edges.
<code>ML_OPCT</code>	Moves or retargets fragments using machine learning models.
<code>ML_VECTOR_CAPTURE_END</code>	Marks the end of the data capture segment of the setup file used for machine learning.
<code>ML_VECTOR_CAPTURE_INIT</code>	Marks the beginning of the data capture segment of the setup file used for machine learning.

Table 5-13. Calibre nmOPC Custom Scripting Commands Summary (cont.)

Command	Description
MRC	Establishes tag-specific adjustment of width or spacing between fragments after OPC movement.
MRC_RULE (Custom Scripting Command)	Creates external and internal spacing constraints based on tag sets.
NEWTAG all	Tags all fragments.
NEWTAG area	Tags fragments based on the width and length of the polygon.
NEWTAG blocked	Tags fragments blocked by an MRC rule.
NEWTAG cd	Measures CDs from a contour.
NEWTAG complete	Outputs all fragments on the edge (if the edge meets length and corner constraints).
NEWTAG corner	Performs a search for fragments based on the distance from a corner.
NEWTAG correction_map	Creates a mapping between correction layer fragments and OPC layer fragments.
NEWTAG displacement	Returns fragments within a specified displacement range.
NEWTAG edge	Searches an edge for fragments within a given set of constraints.
NEWTAG epe	Measures EPE on a given contour.
NEWTAG expression	Works with numerical properties on fragments. Can be used to annotate fragments or create new tag sets based on existing annotations.
NEWTAG external internal enclose enclosing	Computes distance from one layer to another using one of three metrics.
NEWTAG facing_pattern	Generates fragment tag sets of same and opposite facing patterns.
NEWTAG fragment	Searches for fragments within a given set of constraints.
NEWTAG line_end space_end	Computes line ends and space ends.
NEWTAG mrcViolation	Outputs fragments that violate MRC.

Table 5-13. Calibre nmOPC Custom Scripting Commands Summary (cont.)

Command	Description
NEWTAG neighbor	Returns a set of fragments containing previous, next, or both neighbors.
NEWTAG property	Copies DFM properties into a tag set for use by dense OPC commands.
NEWTAG sadp	Classifies fragments according to the relationship between associated cut masks and target layers.
NEWTAG sequence	Creates new tag set(s) holding fragments from a sequence of edges.
NEWTAG sites	Outputs fragments that follow the specified constraint.
NEWTAG topological	Performs a topological operation on the specified input layers.
NEWTAG vertical horizontal all_angle 45_angle	Generates a set of fragments that match an angle specification.
NMBIAS_MEASURE_TAG	Creates a tag set with numeric properties.
NYQUIST	Returns the Nyquist value in either database units or user units.
OPC_ITERATION	Instructs Calibre nmOPC to perform one or more OPC iterations.
OPC_XMEEF_ITERATION	Instructs Calibre nmOPC to perform one or more Matrix OPC iterations, using an alternative method for faster calculations of XMEEFs.
OUTPUT_IMAGE	Instructs Calibre nmOPC to output the given image register to a layer.
OUTPUT_MEASUREMENT_LOCATIONS	Outputs EPE measurement locations.
OUTPUT_OPCT	Outputs OPC layout of specified layer.
OUTPUT_RETARGET	Outputs retargeted opc layers.
OUTPUT_SHAPE fragment	Outputs simple shapes around fragments on a layer and attaches tag-set properties.
OUTPUT_TARGET_CONTROL	Outputs shapes from a target for fragments placed under TARGET_CONTROL.

Table 5-13. Calibre nmOPC Custom Scripting Commands Summary (cont.)

Command	Description
PW_RETARGET	Applies PWOPC-based retargeting for OPC fragments in either a tag set or for all fragments.
SET_EPE	Sets EPEs on fragments based on an input Tcl formula or expression.
SET_MEDIAN_EPE	Sets a fragment set to the median EPE value.
SITES_CREATE	Create sites centered on the target layer.
SITES_DELETE	Deletes all sites associated to fragments on the input tag set.
SITES_DUMP	Outputs all sites associated to fragments on the input tag set.
SITES_SHIFT (Deprecated in 2020.3)	Shifts all sites (or sites of the given type) associated with fragments on the input tag set.
sraffragalign	Aligns fragmentation vertices on sraf layers for sraf_print_avoidance.
sraf_print_avoidance	Prevents selected Sub-Resolution Assist Feature (SRAF) from printing.
sraf_sites_create	Creates sites on an SRAF layer.
TAG add remove ismember	Adds, removes, or identifies members of a tag set.
TAG and or subtract	Performs boolean operations on tag sets.
TAG fromlist	Converts a fragment ID list to a tag set.
TAG size	Returns tag set size.
TAG tolist	Converts a tag set to a list of fragment IDs.
TAG_FRAG_SRC	Finds all fragments tagged with a specific fragment_corner source number and places them in a tag set.
TAG_MRC_SRC	Identifies which MRC rule blocked a fragment when used in conjunction with MRC_RULE ... source.
TARGET_CONTROL	Changes the target value (ideal placement) of specified fragments.

Table 5-13. Calibre nmOPC Custom Scripting Commands Summary (cont.)

Command	Description
target_curve	Creates a curve from the target based on the settings and moves sites to the new curve.
TARGET_FUNCTION	Associates a target function with a function slot.
TDBIAS	Sets a rule-based fragment displacement.
UU2DBU	Converts user units into database units.

auto_target_control

Calibre nmOPC Tcl Scripting Commands

Target Modification

Automatic target control used to adjust a small four-point polygon for convergence.

Usage

```
auto_target_control [tag_in tag] [max_image_factor max_image_factor]
[min_image_factor min_image_factor] [limit inward_limits outward_limits]
[limit_tag tag_name inward_limits outward_limits]...
[limit_delta_cd inward_limits outward_limits]... [exclude tag_name]
[min_space bridging_distance] [enclosing_layer layer_name [enclosed_by distance]]
```

Arguments

- tag_in tag

An optional keyword to take an input tag set. If specified, the auto target control process will be applied only to polygons with fragments that are members of this tag set. Otherwise, all four-point simple polygons are considered.

- max_image_factor max_image_factor

An optional keyword that defines a floating number. If the difference between the aspect ratio of the target and the aspect ratio of the image is greater than *max_image_factor*, no auto_target_control is applied to the polygon.

- min_image_factor min_image_factor

An optional keyword that defines a floating number. If the difference between the aspect ratio of the target and the aspect ratio of image is smaller than *max_image_factor*, no auto_target_control is applied to the polygon. One recommendation is if the aspect ratio and current image ratio are close enough (smaller than specified *min_image_factor*), you do not need to perform retargeting.

- limit inward_limits outward_limits

A required keyword that specifies the default retarget limits (in microns). Both *inward_limits* and *outward_limits* should be positive or zero. The *inward_limits* parameter specifies the maximum distance a fragment can shrink and *outward_limits* specifies the maximum distance a fragment can be extended.

- limit_tag tag_name inward_limits outward_limits

An optional argument that takes a tag set and retargets limits (in microns). It restricts the movement and re-targeting of fragments in the specified tags set to the specified limits. The maximum number of limit_tags you can specify is 16. If a fragment is a member of multiple tag sets, the most restricted one is used. For example, if you have the following:

```
limit_tag tag1 0.002 0.003
limit_tag tag2 0.001 0.005
```

if fragment 1 is a member of both tag1 and tag2, the actual move limit for the fragment is 0.001 (in) and 0.003 (out).

- **limit_delta_cd *inward_limits outward_limits***

An optional argument that specifies CD limits (in microns) on inward and outward fragment movement.

- **exclude *tag_name***

An optional keyword that specifies a tag set to be excluded from all modification on the target, including the ratio-calculation and trimming for the purpose of bridge avoidance. If the polygon contains any fragment that is member of the exclude-defined tag set, the polygon is not processed. If a fragment is a member of both the tag_in tag set and the exclude tag set, the exclude tag set wins.

- **min_space *bridging_distance***

An optional keyword that defines a minimum space (in microns) for the purpose of bridge avoidance.

- **enclosing_layer *layer_name***

An optional keyword that defines a safety zone layer. An existing layer name should be provided as the safety zone layer.

- **enclosed_by *distance***

An optional keyword, only valid when enclosing_layer is declared, that specifies the “enclosed by” spacing. Target control is offset inside from the safety zone layer by this value (only positive or zero is allowed). If not specified, zero is the default value.

Description

This is an automatic target control used to adjust a small four-point polygon with an image that may be very difficult to converge. If there is no other tag set and max_image_factor and min_image_factor are specified, only four-point polygons with edges containing no more than two fragments in the tag set are processed.

Use the min_space keyword to avoid bridging issues.

The auto_target_control command checks the retargeting for all applied polygons and trims the edges if a bridge is found. The back edge is then adjusted to compensate for the trimming if there is enough space. This command can be called with max_image_factor (and min_image_factor) in a loop with **OPC_ITERATION** after auto_target_control. This should be followed by a final call to auto_target_control without max_image_factor declared to retarget all of the problematic polygons.

Examples

```
# OPC_ITERATION loop with auto target control
for {set i 0 } { $i < 5 } { incr i} {
    OPC_ITERATION -simulate_sites -set_epe
    OPC_ITERATION -apply_feedback
    FRAGMENT_MOVE
# retarget qualified polygons
    auto_target_control max_image_factor 0.15 limit 0 0.000 \
        limit_tag tag_in0 0 0.000 \
        limit_tag tag_in1 0 0.002 \
        limit_tag tag_in2 0 0.003 \
        limit_tag tag_in3 0 0.003 \
        limit_tag tag_iso 0 0.004 \
        min_space 0.030 enclosing_layer SAFE_ZONE_LYR \
        enclosed_by 0.011

# Delete all sites and create new sites
    SITES_DELETE all
    SITES_CREATE frag_1S -conservative -ignore_jogs 0.020 \
        -numx 9 -spacing 0.005
    SITES_CREATE frag_mS -multi -numx 9 -spacing 0.005
    SITES_CREATE CONTACT_NEAR_POLY -bylayer SAFE_ZONE_LYR \
        -type WAFER_ENCLOSEDBY -numx 9 -spacing 0.005 -offset -0.011
}

# Retarget all polygons that didn't get retargeted in last few iterations
    auto_target_control limit 0 0.000 limit_tag tag_in_0 0 0.000 \
        limit_tag tag_in1 0 0.002 \
        limit_tag tag_in2 0 0.003 \
        limit_tag tag_in3 0 0.003 \
        limit_tag tag_iso 0 0.004 \
        min_space 0.030 enclosing_layer SAFE_ZONE_LYR \
        enclosed_by 0.011
```

CLASSIFY_FRAGMENTS

Calibre nmOPC Tcl Scripting Commands

Classifies fragments on OPC layers into sets based on surrounding edges. The command can also adjust fragment EPE and displacement to conform to a median range.

Usage

```
CLASSIFY_FRAGMENTS distance [-cleanup_tol tolerance] [-compresskey value]
[-dist_for_sraf sraf_distance] | {-layers "name value..."}
[-out [-annotated] tag]
[-symmetryx {1 | 0}] [-symmetryy {1 | 0}]
[-tag name] [-tol tolerance] [-tol_for_disp tolerance]
```

Arguments

- *distance*

A required argument specifying the distance in user units (typically microns) to search around the fragment. The total length of the search box is fragment length + 2**distance*. The width of the search box is 2**distance*.

Set *distance* to a value less than the interaction distance. When *distance* is too large, patterns are likely to be misclassified. The parser warns with the following message, and internally adjusts distance to the maximum value, but it may not be the value you intended:

The distance <*distance*> for CLASSIFY_FRAGMENTS is too large. This will result in mis-classification and inconsistent displacement for some fragments. Reset it to <*new_value*> based on internal calculation.

- -cleanup_tol *tolerance*

An optional argument that triggers a second pass to move all fragments that vary up to the specified tolerance in microns to have the same displacement. To use this argument, the litho setup file must be called from **LITHO CELLARRAY DENSEOPC** instead of LITHO DENSEOPC.

When using -cleanup_tol, the area to be corrected should be restricted to highly repetitive layouts such as memory arrays. Additionally, only litho models are supported.

- -compresskey *value*

An optional argument that activates key compression, resulting in a smaller memory footprint. The default is 1. To disable, set to 0.

- -dist_for_sraf *sraf_distance*

An optional argument that specifies a distance in user units (typically microns) to search around the fragment on non-opc layers. Edges beyond *distance* from the central fragment are not considered when classifying patterns.

If this option is not specified, the same distance is used for all layers included in pattern classification. If *sraf_distance* is greater than *distance*, *distance* is used on non-opc layers.

This option is ignored if the command also specifies -layers.

- **-layers " {name value} ... "**

An optional argument that explicitly specifies the layers to consider when classifying patterns. By default, only the layers used for simulation are considered when classifying. If there are additional layers, such as those used in MRC or tagging that might differentiate otherwise similar contexts, they can be included using -layers.

name — The layer name.

value — A positive integer used to group layers together. It is only used for identifying layers assigned to the same group and has no other significance.

The first layer for each group of *val* layers must be the OPC layer containing the fragments to classify. For example, to consider a retarget layer that would otherwise not be included when classifying the fragments on layer M1:

```
-layers "{M1 1} {M1_SRAF 1} {M1_SMOOTHED 1}"
```

When using -layers, classification does not consider the default layers unless they are explicitly listed.

- **-out [-annotated] tag**

An optional argument that creates a tag set containing the grouped fragment markers. If -annotated is specified, the tag set contains classification IDs as properties.

When using CELLARRAY (-cleanup_tol is specified), classification IDs are global; identical values imply the fragments are from the same pattern.

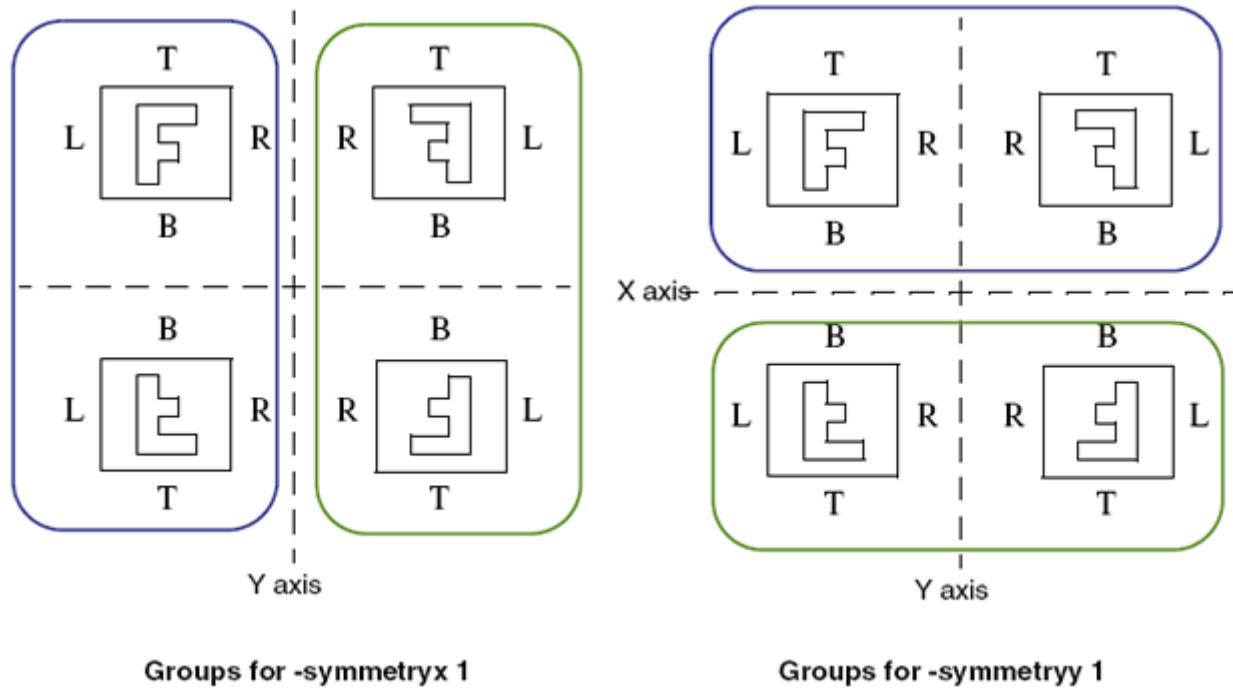
If -out -annotated is specified without -cleanup_tol, the classification IDs are tile-based. The values are only meaningful for fragments within a filter region. The same value may be used for different patterns in other tiles.

- **-symmetryx {1 | 0}**

An optional argument controlling the matching behavior for symmetrical-X patterns. A setting of 1 indicates that patterns that are identical but reflected around the X-axis are considered a match (that is, the top and bottom of the pattern is reversed, as shown on the left in [Figure 5-2](#)). A setting of 0 indicates that they are not to be considered identical.

The default value is automatically identified from the optical source, but if the value cannot be identified from the source, the default is 1.

Figure 5-2. Symmetrical X and Y Groups



- **-symmetry {1 | 0}**

An optional argument that controls the matching behavior of symmetrical-Y patterns. A setting of 1 indicates that patterns which are identical but reflected around the Y-axis are considered a match (that is, the left and right of the pattern is reversed as shown on the right in [Figure 5-2](#)). A setting of 0 indicates that they are not considered identical.

The default value is automatically identified from the optical source, but if the value cannot be identified from the source, it is set to 1.

Note

Both **-symmetryx** and **-symmetryy** can be enabled simultaneously. Setting both to 1 would match all four cells in [Figure 5-2](#).

- **-tag name**

An optional argument restricting the fragments to classify to only those in the tag set *name*. By default, all fragments may be classified.

- **-tol tolerance**

An optional argument that sets the EPE tolerance. The *tolerance* value is in user units and must be non-negative. Fragments within *distance* that have $EPE \pm tolerance$ from the median EPE are modified to have the median EPE. The default is equivalent to 2 dbu.

- `-tol_for_disp tolerance`

An optional argument that sets default displacements of selected fragments having displacements less than or equal to the median displacement $\pm tolerance$ to the median displacement. The *tolerance* value is in user units and must be non-negative. Only fragments in the group that have displacements in the tolerance band have their displacement modified. The default is equivalent to 2 dbu.

Description

This operation classifies fragments on each OPC layer. Fragments are grouped into sets based on having the same pattern of edges surrounding them. The edges that are considered are all of the edges that contribute to an optical simulation for that fragment, consisting of edges on visible layers as well as OPC or correction layers.

When EPE is computed during iterations, a median EPE is computed for each group. The EPE of a fragment in the group is set to this median if it is within (less than or equal to) a specified tolerance. By making EPE the same, the resulting fragment displacements tend to be the same. The purpose of this command is to improve consistency for repeated patterns.

This operation can also perform a median operation for displacement tolerance, resulting in more consistent fragment displacements.

CLASSIFY_FRAGMENTS must be placed before any DISPLACEMENT commands or the following error is produced:

```
ERROR: inside options block <name>
      CLASSIFY_FRAGMENTS <recipe settings>
      DISPLACEMENT command should not be before CLASSIFY_FRAGMENTS.
```

Note

 Unless used with `-cleanup_tol`, which requires LITHO CELLARRAY DENSEOPC, this command does not guarantee consistency for the entire layout. The version that is used with LITHO DENSEOPC does not address consistency between patterns that are not part of the same tile.

Inconsistent results can be caused by several cases:

- Manipulating tag sets of fragments using the FRAGMENT Tcl scripting commands. For the most consistent displacements, use `fragment_layer` blocks.
- Excessive distance settings. Keep *distance* less than the interaction distance and the optical model's optical diameter.
- Post-CLASSIFY_FRAGMENT adjustments to position, such as by DISPLACEMENT or tag-based MRC_RULE. Run CLASSIFY_FRAGMENT after other adjustments so that the consistency is not disturbed.

CLASSIFY_FRAGMENTS may be specified once per setup file.

Examples

Example 1: Minimum Specification

Runs classification of fragments with a search distance of 0.5 dbu.

```
CLASSIFY_FRAGMENTS 0.5
```

Example 2: In Relation to Other Commands

The following lines from a setup file demonstrate the proper relationship of CLASSIFY_FRAGMENTS to other commands. Notice that OPC iterations happen after fragments are classified.

```
#Create a tag set
NEWTAG topological inside_edge M1_Tgt check_marker -out tgCheck

#Group fragments
CLASSIFY_FRAGMENTS 0.5 -tag tgCheck -tol 0.0006

#Perform OPC
OPC_ITERATION 8

#Adjust SRAFs
sraf_print_avoidance -step 0.001

#Show fragments
OUTPUT_SHAPE fragment tgCheck tgCheck 0.001 0.001
```

DBU2UU

Calibre nmOPC Tcl Scripting Commands

Unit Conversions and Miscellaneous

Converts database units (dbus) into user units.

Usage

DBU2UU *database_units*

Arguments

- *database_units*

Specifies the database units (typically in microns) to be converted.

TCL Results

The converted number of user units as a floating point.

Description

This command converts the user-supplied *database_units* into user units (usually microns), which is returned in the Tcl result.

Performance

O(1) constant time

Rating = fast

Examples

The following example prints the XY coordinates of a fragment endpoint in microns:

```
puts [DBU2UU frag(x1)] [DBU2UU frag(y1)]
```

DENSE_CD

Calibre nmOPC Tcl Scripting Commands

Dense Simulation Controls

Measures CDs on a contour and outputs information into an array.

Usage

DENSE_CD *id* *contourid* [*max maxsearch*] **-out** *outvar*

Arguments

- ***id***
The fragment id of the fragment to calculate CDs.
- ***contourid***
A contour register id coming from a previous simulation call. The available registers are named IM0... IM32.
- ***max maxsearch***
An optional keyword that sets the maximum search distance in user units to look for CDs.
- ****-out** *outvar***
The output array variable name.

Description

This command measures CDs on a pre-computed contour for locations selected automatically on a given fragment. The output associative array contains the following info:

```
General info:  
outvar(count)      : total count of measurements CD-space measurements:  
outvar(spacemin)   : min CD space or negative number if invalid  
outvar(spacemax)   : max CD space or negative number if invalid  
outvar(spaceav)    : average CD space or negative number if invalid  
outvar(space1) ... outvar(spaceN) : CD space or negative number if invalid  
outvar(targetspace1) ... outvar(targetspaceN) : target width or negative  
CD-width measurements:  
outvar(widthmin)   : min CD width or negative number if invalid  
outvar(widthmax)   : max CD width or negative number if invalid  
outvar(widthav)    : average CD width or negative number if invalid  
outvar(width1) ... outvar(widthN) : CD width or negative number if invalid  
outvar(targetwidth1) ... outvar(targetwidthN) : target width or negative
```

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

None

Performance

$O(\log N)$ where N is the number of vertices in the simulation contour

`time(OP) >> time(fragment_get)`

Rating = slow for a per-fragment operation

Alternative Commands

[NEWTAG cd](#) is a faster way to compute an iso-CD set of fragments.

DENSE_EPE

Calibre nmOPC Tcl Scripting Commands

Dense Simulation Controls

Measures EPEs on a contour and outputs information into an array.

Usage

DENSE_EPE *id* *contourid* [max *maxsearch*] -out *outvar*

Arguments

- ***id***
Specifies the ID of the fragment used to calculate EPEs.
- ***contourid***
Specifies a contour ID number from a previous simulation call. The available registers are in the set IM0... IM32.
- **max *maxsearch***
Optional argument that sets the maximum distance (in user units) to search for EPEs.
- **-out *outvar***
Specifies the output array variable name.

Description

This command measures EPEs on a pre-computed contour for locations selected automatically on a given fragment.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

The output associative array (`outvar`) contains the following info:

```

General info:
  outvar(count)      : total count of measurements
  outvar(status)     : 0 = the fragment has a valid epe
                      : 1 = the fragment has no valid epe, but the
                      :       center measurement point is inside the contour,
                      : 2 = the fragment has no valid epe, but the center
                      :       measurement point is outside the contour
                      : 3 = at least two EPEs are invalid (a possible
                      :       pinch-bridge situation)

EPE measurements:
  outvar(epemin)    : minimum of all valid EPE measurements for fragment
  outvar(epemax)    : maximum of all valid EPE measurements for fragment
  outvar(peeav)     : average of valid EPE measurements for fragment
  outvar(x1) ... outvar(xN) : x-location for measurements on fragment
  outvar(y1) ... outvar(yN) : y-location for measurements on fragment
  outvar(epel) ... outvar(epen) : EPEs for measurements on fragment
                                Set to -1001 for invalid EPE

```

Note

 The `epemin` and `epemax` values are defined in the standard mathematical definition for signed values. If the various EPEs on a given fragment are (in nm) -2, 10, 7, 2, and 1, then `epemin` = -2 and `epemax` = 10.

Tcl Results

None

Performance

$O(\log N)$ where N is the number of vertices in the simulation contour time(`OP`) >> `time(fragment_get)`

Rating = slow for a per-fragment operation

Examples

```
DENSE_EPE $fragid IM0 max 0.2 -out simvar
```

Alternative Commands

[NEWTAG cd](#) is a faster way to compute an iso-CD set of fragments.

DENSE_SIMULATE

Calibre nmOPC Tcl Scripting Commands

Dense Simulation Controls

Computes aerial or resist images and outputs information into an array.

Usage

```
DENSE_SIMULATE -name pw_name -dose1 dose_list [-dose2 dose_list [-dose3 dose_list]]  
-registers register_list
```

Arguments

- **-name *pw_name***
Specifies the name of the process window condition (or “nominal” window condition) to use for this simulation.
- **-dose1 *dose_list***
Specifies the list of doses to calculate for the first exposure.
- **-dose2 *dose_list***
Optionally specifies the list of doses to calculate for the second exposure.
- **-dose3 *dose_list***
Optionally specifies the list of doses to calculate for the third exposure. (This only occurs with topological models.)
- **-registers *register_list***
Specifies the list of output register names in which to store the simulation contours for the given doses. The length of this list must match the length of the dose list(s). Registers are specified by names in the set IM0...IM32.

Description

This command concurrently computes aerial or resist images for several dose values and inserts them into the user-specified array of image contour registers. The available registers are named IM0... IM32. Use of a given register will delete the previous contents (if any) of the register, and insert the new simulation contour.

Note

 This command does not set the fragments' desired displacements.

Tcl Results

None

Performance

$O(D)$, where D is the dose count. Run time doubles for double exposure. Run time also increases with optical diameter and kernel count.

Rating = very slow

Examples

```
DENSE_SIMULATE -name nominal -dose1 1 -registers IM0  
NEWTAG epe gate IM0 -out epe_pos epeav > 0.001 -out epe_neg epeav < -0.001
```

DISPLACEMENT

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Sets FROZEN/FIXED status, displacement, desired displacement, and displacement limits for all fragments in the given tag set.

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.
The value of *val* can be either a floating point number or read from an annotated tag set *tag_fO* using “property(*tag_fO*)[:default(*value*)”, for example, property(tfixedOffset) or property(tfixedOffset):default(0.0). If using the property syntax and a fragment is not annotated, the default value is assigned. If there is no default value, the fixed offset is 0.

- **-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. As with -fixedOffset, *val* can be set either as a floating point number or read from an annotated tag set using “property(*tag_fOI*)[:default(*value*)].” The default is 0.

- **-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 OPC, 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.

The value of *val* can be either a floating point number or read from an annotated tag set *tag_hO* using “property(*tag_hO*)[:default(*value*)]”, for example, property(thintOffset) or property(thintOffset):default(0.0). If using the property syntax and a fragment is not annotated, the default value is assigned. If there is no default value, the desired displacement is set to 0.

- **-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. As with -hintOffset, the value of *val* can be either a floating point number or read from an annotated tag set using “property(*tag*)[:default(*value*)].” The default with the optional :default(*value*) is 0.

- **-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 .

The values for *inner* and *outer* can be either floating point numbers or read from an annotated tag set using “property(*tag*)[:default(*value*)]; for example,

```
-limit property(tLimitInner) :default(0) property(tLimitOuter)
```

If an input fragment is not found in the annotated tag set, the limit is set to INT_MAX (effectively no limit) unless a default value is provided. (In the example, tLimitInner has the custom default 0, but any fragments not in tLimitOuter have a limit of INT_MAX.)

- **-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.

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 OPC 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.)

If multiple **DISPLACEMENT** statements attempt to set limits on a fragment, the effect depends on **displacement_limit_mode**. When not set or set to LAST, the instance before the movement command applies. When set to STRICT, the limit closest to 0 is used.

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, OPC 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 OPC 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.

Related Topics

[BIASRULE \[Calibre Rule-Based OPC User's and Reference Manual\]](#)

[FRAGMENT_MOVE](#)

[NMBIAS_MEASURE_TAG](#)

[NEWTAG expression](#)

EUV_MODEL_REDUCTION

Calibre nmOPC Tcl Scripting Commands

EUV Command

Toggles model reduction mode for EUV Dense OPC. May be specified at most twice.

Note

 A single “EUV_MODEL_REDUCTION off” instance implies all statements above it are to be treated as though “EUV_MODEL_REDUCTION on” were specified at the beginning of the block. This may have undesired results.

Usage

EUV_MODEL_REDUCTION {off | on | {x1 y1 x2 y2}}

Arguments

- **on**

x1 y1 x2 y2

A required argument that activates model reduction until a subsequent EUV_MODEL_REDUCTION off statement. This affects OPC_ITERATION and OPC_XMEEF_ITERATION commands. When specified, this preliminary call must precede all OPC iterations.

Note

 Although explicitly activating model reduction is optional, it is strongly recommended.

on — Turns on model reduction across entire layout.

x1 y1 x2 y2 — A set of coordinates identifying the lower-left and upper-right corners of a layout extent from which representative model values are calculated to be used throughout the layout.

By default, the LAYOUT WINDOW settings in the SVRF file are used, if provided; if LAYOUT WINDOW is not set, the TOPCELL extent is used.

- **off**

A required argument that deactivates model reduction. Further OPC_ITERATION or OPC_XMEEF_ITERATION commands are performed with full process models.

If an “off” statement is found without an EUV_MODEL_REDUCTION “on” or coordinates statement above it, the run treats all OPC iteration statements until that point as if “EUV_MODEL_REDUCTION on” had been specified at the top of the code block.

Description

This optional command controls the mode for EUV process models. The mode is used for the EUV dense OPC advanced hierarchical flow to process full chip layouts more quickly. There may be some minor changes to fragmentation compared to the standard processing mode.

The advanced hierarchical flow requires that the setup file include “processing_mode flat”. You must also specify either ULTRAflex or TURBOflex mode.

Restricted Commands

The EUV model reduction mode only allows a subset of the Calibre nmOPC commands in its operations. The following setup file commands generate errors in a setup file that also contains an EUV_MODEL_REDUCTION call:

- auto_target_control
- flare_model
- flare_model_load
- FRAGMENT delete, FRAGMENT end, FRAGMENT modify, and FRAGMENT split
- fragment_inter (unless fragment_interlayers is also specified)
- fragment_optimize
- lapi_exec_tcl
- ML_OPc
- MRC
- OUTPUT_IMAGE
- OUTPUT_OPc
- OUTPUT_RETARGET
- OUTPUT_TARGET_CONTROL
- TARGET_CONTROL
- TARGET_FUNCTION

Recommended Usage

The recommended usage for the EUV advanced hierarchical flow is to use EUV_MODEL_REDUCTION twice in the setup file. The first instance should activate model reduction with an explicit layout extent (EUV_MODEL_REDUCTION *x1 y1 x2 y2*). After calls to one or both of OPC_ITERATION and OPC_XMEEF_ITERATION, the second instance should deactivate the mode (EUV_MODEL_REDUCTION off).

This usage model is shown as [Example 1 — Recommended Usage](#).

An alternate usage for testing is similar to the recommended usage, but has “EUV_MODEL_REDUCTION on” instead of “EUV_MODEL_REDUCTION *x1 y1 x2 y2*.” This style is not recommended except for testing, because if LAYOUT WINDOW is provided the model reduction will differ from that of a full chip run.

This usage model is shown as [Example 2 — Testing Usage](#).

If you specify EUV_MODEL_REDUCTION off twice, the first instance is treated as EUV_MODEL_REDUCTION on.

Changes to Transcript

When the rule file includes EUV_MODEL_REDUCTION, it adds lines before and after the corresponding LITHO RUN banner, as in the following example:

```
-----  
***** STEP 1 OF ADVANCED HIERARCHICAL FLOW (EUV_MODEL_REDUCTION on) *****  
*****  
===== START DENSEOPC EXECUTION (LITHO RUN 1) =====  
=====  
...  
...  
===== END DENSEOPC EXECUTION (LITHO RUN 1) =====  
=====  
*****  
***** STEP 2 OF ADVANCED HIERARCHICAL FLOW (EUV_MODEL_REDUCTION off) *****  
*****  
===== START DENSEOPC EXECUTION (LITHO RUN 2) =====  
=====
```

If the best practice of providing layout coordinates was not followed, the transcript additionally includes the calculated settings in the HIER MODE CELL PROCESSING section:

```
INFO: Missing 'EUV_MODEL_REDUCTION <x1> <y1> <x2> <y2>' .  
Reduced model may depend slightly on 'LAYOUT WINDOW' settings.  
EUV_MODEL_REDUCTION 8027.91 6622.02 8029.26 6623.34  
should be sufficient to reproduce approximately the same model reductions in  
subsequent runs with different 'LAYOUT WINDOW' settings.  
The best practice is to always use the above syntax with  
the correct layout extents (not necessarily the ones printed here) .
```

Examples

Example 1 — Recommended Usage

The following example shows how to incorporate EUV_MODEL_REDUCTION in a tag script:

```
...
EUV_MODEL_REDUCTION 0.0 3000.0 4000.0 4000.0
OPC_ITERATION 3
OPC_XMEEF_ITERATION 5
EUV_MODEL_REDUCTION off
OPC_XMEEF_ITERATION 2
...
```

The reduced models use representative values from (0, 3000) to (4000, 4000) for the first 8 iterations. The last 2 iterations are simulated with the full details of the process models.

Example 2 — Testing Usage

In this example, the coordinates of LAYOUT WINDOW determine the area for the model reductions. This is not recommended for production flows.

```
...
EUV_MODEL_REDUCTION on
OPC_ITERATION 8
EUV_MODEL_REDUCTION off
OPC_ITERATION 2
...
```

Related Topics

[OPC_ITERATION](#)

[OPC_XMEEF_ITERATION](#)

[Transcripts for Post-Tapeout Operations \[Calibre Post-Tapeout Flow User's Manual\]](#)

FEEDBACK

Calibre nmOPC Tcl Scripting Commands

OPC Controls

Sets the feedback factor used to determine edge movement for specific fragments, overriding the global value set by the feedback command.

Usage

Syntax 1

FEEDBACK [-keep] [-control] *gid* *feedbackvalue*

Syntax 2

**FEEDBACK [-keep] [-control] *tag* {*feedbackvalue*
| **property**(*annotated_tag*)[:default(*value*)]} ...**

Syntax 3

FEEDBACK -remove *tag*

Arguments

- **-keep**

An optional argument that locks the last feedback value to the value set by FEEDBACK. The OPC algorithm will not adjust the user-set feedback. To unlock, issue FEEDBACK again without the -keep switch.

- **-control**

An optional argument that uses the last user-set feedback value as a seed when the user-set feedback has expired or is canceled. By default, it starts with the system default feedback values.

- ***gid***

For syntax 1, a required argument that specifies a generalized fragment ID, which can be a single fragment ID.

- ***tag***

For syntax 2 and 3, a required argument that specifies the name of a tag set.

- ***feedbackvalue***

For syntax 1 and 2, a required argument that sets the feedback for an iteration on the fragments indicated. This value should be between -2 and +2. Typically, the feedback is a negative number between -1 and 0.

For syntax 2, ***feedbackvalue*** can be specified multiple times. Each feedback value corresponds to an OPC iteration. If the OPC iteration count is more than the given number of feedback values, then further iterations use the last value set by the feedback command. When -keep is in effect, the last feedback value is not adjusted between iterations for further iterations.

feedbackvalue cannot be used with **-property**.

- **property(*annotated_tag*)[:default(*value*)]**

For syntax 2, a required argument that can be used to set feedback values based on the properties of an annotated tag set. It cannot be used with **gid** or intermixed with *feedbackvalue*.

If the property value is less than -2 or greater than 2, it is treated as 0 and generates a warning.

The annotated tag set can be the same as **tag**, but does not need to be. If a fragment in **tag** is not found in **annotated_tag**, the optional default value is used. If :default has not been specified, the fragment feedback is set to -0.4.

- **-remove**

For syntax 3, a required argument that removes the feedback values previously assigned to the tag set. This is only applicable when multiple feedback values are set.

Tcl Results

None

Description

This optional command sets the feedback for one or more fragments. The feedback is used to calculate fragment movement on the next OPC iteration. Note the following:

- FEEDBACK overrides the feedback command in the denseopc_options block for the specified fragments only.
- Only the subsequent call to OPC_ITERATION or OPC_XMEEF_ITERATION uses the FEEDBACK setting. The next call after that to OPC_ITERATION, OPC_XMEEF_ITERATION, or FRAGMENT_MOVE resets the user feedback.
- If you use FEEDBACK locally for specified fragments, you must explicitly use -keep to continue using the final specified FEEDBACK value, otherwise:
 - If the feedback command is explicitly set, then it reverts to the last specified value for that command.
 - If the feedback command is not set, then it reverts to the default -0.4.

Setting a fragment's feedback to 0 freezes it in its current location.

The order of priority for determining feedback values is as follows:

1. If a fragment occurs in a FEEDBACK command, that value is used.
2. If FEEDBACK does not apply, if you set **feedback** it is used.
3. If you do not explicitly set FEEDBACK or feedback, the default is a global feedback value of -0.4.

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

Example 1

This example shows setting feedback specific to particular tag sets before performing OPC:

```
FEEDBACK line_end -0.2
FEEDBACK convex    -0.4
FEEDBACK gate      -0.6
OPC_ITERATION      5
```

Example 2

This example shows creating separate annotated tag sets to be used for each iteration:

```
NEWTAG all e1_target -out e1_all
for {set i 1} { $i <= 10 } { incr i } {
    NEWTAG expression e1_all { -1.0 + 0.1*$i } -aout fb_$i
}
FEEDBACK -keep e1_all property(fb_1) property(fb_2) property(fb_3) \
           property(fb_4) property(fb_5) property(fb_6) property(fb_7) \
           property(fb_8) property(fb_9) property(fb_10)
OPC_ITERATION 10
```

Related Topics

[feedback](#)

fragmaxedge

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Applies sparse OPC minedgelength operation only to fragments in specified tag set.

Usage

fragmaxedge *layer tagname maxedgelength*

Arguments

- ***layer***

Specifies the name of the layer which contains the fragments.

- ***tagname***

Specifies the name of the tag set containing the fragments.

- ***maxedgelength***

Specifies the maximum edge length used for trimming the fragments.

Description

This function applies the Calibre OPCpro minedgelength operation, but only to the fragments in the tag set indicated by *tagname*. This functionality is equivalent to the Calibre OPCpro command “[fragmentTag ... maxedgelength -split 1](#).”

Note that the only -split option supported is “-split 1.” (This means that fragments larger than maxedgelength are broken, unless this would violate the Calibre OPCpro minedgelength specification.) For most typical values of maxedgelength and minedgelength, this means that the longest remaining fragment can be no longer than maxedgelength.

Examples

The following example uses the following steps:

First, tag all fragments from the target layer as t1.

```
NEWTAG all mx.opc.e1_yes -out t1
```

Second, tag t1 fragments that are long enough to need maxedgelength fragmentation as t2. In this case, it tags fragments longer than 0.07 microns.

```
NEWTAG fragment t1 len >= 0.07 -out t2
```

Finally, break the fragments that are long enough to need it, which are in the t2 tag set.

```
fragmaxedge mx.opc.e1_yes t2 0.035
```

Note that although the layer “mx.opc.e1_yes” was specified in “NEWTAG all” it must also be specified for fragmaxedge or results are incorrect.

FRAGMENT delete

[Calibre nmOPC Tcl Scripting Commands](#)

[Fragmentation Controls](#)

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.

Description

This command can delete either a single fragment, or set of fragments.

Note

 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. Siemens EDA recommends against using this command in hierarchical applications due to the necessity of Calibre nmOPC to control the fragmentation behavior around tile boundaries. Using FRAGMENT delete may disable full control over fragment insertion.

Tcl Results

None

Performance

O(N) where N is the number of fragments to delete.

Rating = slow, use rarely

FRAGMENT end

Calibre nmOPC Tcl Scripting Commands

Fragmentation Controls

Splits fragments into multiple fragments.

Usage

```
FRAGMENT end gid [-first d1 d2 ... dn] [-last d1 d2 ... dn [-force]]  
[gid [-first d1 d2 ... dn] [-last d1 d2 ... dn [-force]] ...]
```

Arguments

- *gid*
Specifies a generalized fragment ID which can be the tag name or single fragment ID.
- -first *d1 d2 ... dn*
Specifies that lengths to split fragment from first end point (up to 10 can be given).
- -last *d1 d2 ... dn*
Specifies lengths to split fragment from last end point (up to 10 can be given).
- -force
If given, the command ignores minimum fragment length when splitting. Otherwise, a split is not performed if it will create fragment(s) shorter than min fragment length.

Description

This command splits fragments into multiple fragments. The ordering of the splits alternates between -first and -last lengths to create a more balanced split when a short fragment cannot accommodate all the split points. For example, given a fragment of 10 units long and “-first 1 3 5 -last 2 4 6,” the fragment is split into the following fragments lengths: 1, 3, 4, 2.

When -force is not given, the -first and -last lengths cannot be shorter than min fragment length for the layer. This command invalidates all previously-computed tag sets, setting them to empty sets.

Tcl Results

None

Performance

O(N) where N is the number of fragments to split.

Rating = slow, use rarely

Examples

```
FRAGMENT end mytagset -first 0.02 0.03 -last 0.03 -force
```

FRAGMENT modify

Calibre nmOPC Tcl Scripting Commands

Fragmentation Controls

Shifts the ordered endpoint of a given fragment.

Usage

FRAGMENT modify {pt1 | pt2} *gid shift_by 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.

- **shift_by**

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 **shift_by**, **min**, and **min_neighbor**. Entering keywords such as tag names will generate an error.

Description

This command shifts the ordered endpoint of a given fragment by the given amount while preserving minimum resulting fragment lengths.

Tcl Results

None

Performance

O(N) where N is the number of fragments to modify.

Rating = medium

FRAGMENT split

Calibre nmOPC Tcl Scripting Commands

Fragmentation Controls

Splits fragments.

Usage

FRAGMENT split {*gid location min* | -3fragments *gid dist_pt1 dist_pt2 min*} ...

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 point 1 to split.
- ***dist_pt2***
Specifies the distance from point 2 to split.

Description

This command splits a fragment (or set of 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 tag set parameters.

Note

 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.

Tcl Results

None

Performance

O(N) where N is the number of fragments to split.

Rating = slow, use rarely

FRAGMENT_CONFORM

Calibre nmOPC Tcl Scripting Commands

Moves fragments to the position of similarly oriented fragments within a short distance on a marker layer.

Usage

FRAGMENT_CONFORM -target *layer* [-tag *name*] -marker *layer* -distance *microns*

Arguments

- **-target *layer***

A required argument specifying the layer with fragments to move. This layer should be of type “opc.”

- **-tag *name***

An optional argument restricting the fragments to move to only those in the tag set *name*. By default, all fragments may be moved.

- **-marker *layer***

A required argument specifying the layer with the desired fragment placement. The target layer and the marker layer must be different layers.

- **-distance *microns***

A required argument specifying how close the marker and target layer fragments must be in order for the target layer fragments to move to the position of the equivalent marker layer fragment. Values must be positive.

The value of -distance must be large enough to cover the space between all pairs of target and marker fragments, but small enough to not include any neighboring polygons. If the value is too large, FRAGMENT_CONFORM may map marker layer fragments to the wrong polygon, such as a neighboring via.

Description

This optional command moves the fragments on the target layer to nearby fragments on the marker layer if they are similarly oriented. It can be useful for improving consistency, such as when the marker layer contains a preferred OPC correction.

For best results, the target layer fragmentation should begin with [fragment_nearly_coincident](#) based on the marker layer. This improves consistency in dense layouts. If a target fragment projects onto multiple marker edges within -distance, it conforms to the edge with which it has maximum overlap.

Examples

Consider a regular array of shapes, such as commonly occur in memory designs:



The difference between the post-OPC layer and the consistent placement shown in the marker layer is only 1 or 2 dbu. The following command moves the post-OPC edges to the marker position, so that the array is consistent:

```
FRAGMENT_CONFORM -target post_opc -marker marker -distance 0.01
```

FRAGMENT_EPE_MEASURE_METHOD

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

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.

Description

This command allows you to select between minimum, maximum, or average values for EPE for a tag set.

Tcl Results

None

Performance

O(1), constant time to get single fragment.

Rating = fast

FRAGMENT_GET

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Obtains fragment information and outputs it to an array.

Usage

FRAGMENT_GET *id* [-moved] [top] -out *fragvar*

Arguments

- ***id***
Specifies a single fragment ID.
- **-moved**
When this optional argument is set, the fragment is output in its current moved position. The corner type information reflects the unmoved fragment corner types.
- **-top**
If this is specified, the fragment coordinates are transformed to top level coordinates before being output. Note that if any of the cells in the path from the currently processed cell to the top are instantiated more than one time, the transform is not unique. In this case, one of those transforms is (randomly) chosen.
- **-out *fragvar***
Specifies the name of an output Tcl array to place the fragment info.

Description

This command dumps the raw fragment information into a Tcl associative array named by the caller. The output associative array will contain:

```

fragvar(prev) : fragment id for previous fragment (counter-clockwise)
fragvar(next) : fragment id for next fragment (clockwise)
fragvar(x1) : First point x-coordinate in database units
fragvar(x2) : Second point x-coordinate in database units
fragvar(y1) : First point y-coordinate in database units
fragvar(y2) : Second point y-coordinate in database units
fragvar(c1) : Corner type for first point.
    0 = not corner, 1 = convex, 2 = concave
fragvar(c2) : Second point y-coordinate
    0 = not corner, 1 = convex, 2 = concave
fragvar(L) : Length of the current fragment in database units
fragvar(Luu) : Length of the current fragment in user units
fragvar(disp) : Fragment displacement in database units
fragvar(dispuu) : Fragment displacement in user units.
fragvar(goal) : Fragment desired displacement
fragvar(goaluu) : Fragment desired displacement in user units
fragvar(state) : Fragment state (0 = invalid, 1 = valid, 2 = partially valid)
fragvar(feedback) : Fragment current feedback

```

To obtain the correct current feedback value, **FRAGMENT_GET** must be called prior to the application of feedback. The following call sequence yields the correct current feedback value:

```
OPC_ITERATION -set_epe  
FRAGMENT_GET $fragid -out fragvar  
OPC_ITERATION -apply_feedback
```

FRAGMENT_GET_DISPLACEMENT is a faster way to get only the displacement.

Tcl Results

None

Performance

O(1), constant time to get single fragment. It is slightly slower than **FRAGMENT_GET_DISPLACEMENT**.

Rating = fast

Examples

```
#dump out the EPE info for the fragment
FRAGMENT_GET $fragid -out fragdata
puts $dbg "Fragment ID $fragid X1 $fragdata(x1)    Y1 $fragdata(y1)      \
          X2 $fragdata(x2)    Y2 $fragdata(y2)"
puts $dbg "All EPEs $epe_list"
puts $dbg   "EPE used $epe  "
puts $dbg  "
```

FRAGMENT_GET_DISPLACEMENT

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Obtains fragment displacement information.

Usage

FRAGMENT_GET_DISPLACEMENT *id* [-desired] -out *dispvar*

FRAGMENT_GET_DISPLACEMENT *array tagname* [-desired] -out *disarray*

Arguments

- **array *tagname***

Enables array mode, where the command will get the displacement for an entire set of fragments.

- ***id***

Specifies the single fragment ID.

- -desired

Retrieves the desired displacement instead of the current actual displacement.

- **-out *dispvar***

Specifies a variable name to place the output displacement. This is a non-array variable for non-array mode.

- **-out *disarray***

Specifies the array variable name for array mode. Outputs are stored in a displacement associative array (*disarray*(fragid)).

Description

This command gets the fragment displacement for a single fragment or for an entire set of fragments (in array mode). The displacement is returned through a variable whose name is passed in. The displacement is output in user units.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

None

Performance

O(1), for non-array mode (constant time for single fragment displacement).

O(N), for array mode where N is the size of the input set.

Rating = medium

Alternative Commands

- [NEWTAG displacement](#) is a faster way to get an equally displaced set of fragments
- [FRAGMENT_GET](#) returns the displacement in an associative array and is slightly slower

FRAGMENT_ITERATOR first/next/release

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Iterates OPC over active fragments.

Usage

FRAGMENT_ITERATOR first {layer | tag} [-out variable]

FRAGMENT_ITERATOR next {layer | tag} [-out variable]

FRAGMENT_ITERATOR release {layer | tag}

Arguments

- **first**

Calls the iterator first to start the traversal of the indicated layer.

- **next**

Calls the next iterator to traverse to the next layer.

- **release**

Once an iterator is invoked on a given tag or layer with “first”, no other iterator on the same layer or tag can be invoked until it is released. The release occurs automatically if “first” or “next” gets to the end.

- **layer | tag**

Specifies the layer or tag to iterate over. The argument is recognized as a layer if it is an integer layer index between [0,N-1]. If the argument is a string, it is assumed to be a tag variable name.

- **-out variable**

Optional argument that specifies an output variable. When this argument is set, the fragment is output in its current moved position. The corner type information reflects the unmoved fragment corner types.

Description

This command allows the caller to iterate over active fragments. The caller can iterate over all fragments in a given layer or all fragments in a given set. It outputs the current fragment id into a Tcl variable named by the caller. It returns a boolean value indicating whether the iteration is still active or not.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

You can use the following commands alternatively:

- The **TAG tolist** command converts a tag set into a list of fragment IDs.
- The FRAGMENT ITERATOR is considered a more efficient way to loop over a tag set, in comparison to first converting to a list.

Tcl Results

1 = Valid single fragment ID returned.

0 = Done with iterator, invalid fragment id returned release mode: None

Performance

O(1), constant time.

Rating = fast

Examples

The following example shows when to use the release command:

```
for {set C [FRAGMENT_ITERATOR first L1 -out fragid]} { $C != 0 }  
    {set C [FRAGMENT_ITERATOR next L1 -out fragid]} {  
puts $fragid  
if { some condition is true } {  
    break  
}  
}  
FRAGMENT_ITERATOR release L1
```

Erroneous usage of an iterator that is already being used:

```
FRAGMENT_ITERATOR first L1 -out fragid  
FRAGMENT_ITERATOR first L1 -out fragid <-- this is an error
```

Use of iterators simultaneously for different layers:

```
FRAGMENT_ITERATOR first L1 -out fragid  
FRAGMENT_ITERATOR first L2 -out fragid
```

FRAGMENT_MAX_DISPLACEMENT

Calibre nmOPC Tcl Scripting Commands

OPC Controls

Sets a maximum limit on how far a target fragment will move.

Note

 This command is retained for backward compatibility. For new litho setup files, use **DISPLACEMENT -limit** instead.

Usage

FRAGMENT_MAX_DISPLACEMENT -frag *fid* *x1* [*x2*] (for Tcl-based scripting only)

FRAGMENT_MAX_DISPLACEMENT -frag *fid* -clear (for Tcl-based scripting only)

FRAGMENT_MAX_DISPLACEMENT -tag *tagname* *x1* [*x2*]

FRAGMENT_MAX_DISPLACEMENT -tag *tagname* -clear

Arguments

- ***fid***

Specifies a single fragment ID.

- ***tagname***

Specifies the tag name to identify all target fragments for operation.

- **-clear**

Removes any previously-set displacement limits on the target fragments.

- ***x1***

Sets the maximum outer displacement. If optional *x2* is not given, then *x1* is taken as the maximum inner displacement. Valid values are ≥ 0 .

- ***x2***

Sets the maximum inner displacement. Valid values are ≥ 0 .

Description

This command sets a limits on how far a target fragment or set of fragments can move.

Tcl Results

None

Performance

O(*T*), where *T* is the size of the input set

Rating = medium

Examples

```
FRAGMENT_MAX_DISPLACEMENT -tag frags 0.01
```

FRAGMENT_MOVE

[Calibre nmOPC Tcl Scripting Commands](#)

[OPC Controls](#)

Moves fragments.

Usage

FRAGMENT_MOVE [-freeze_nmoving] [-freeze_tag *tagname*]

Arguments

- **-freeze_nmoving**

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*.

Description

This command moves all fragments to their desired displacements simultaneously, while obeying mask constraints. This command only moves fragments whose desired displacement differs from the current actual displacement.

In other words, fragments that were moved using [FRAGMENT_SET_DISPLACEMENT](#) -force do not require a subsequent call to this command to move the fragments. There are two ways to set desired displacements:

- OPC_ITERATION
- FRAGMENT_SET_DISPLACEMENT (without the -force option)

The implication is that calling either of those commands would typically require a subsequent call to FRAGMENT_MOVE. The maximum movement per iteration is also enforced (the default is Nyquist/2), which can also 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.

Tcl Results

None

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 requires 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
```

Related Topics

[DISPLACEMENT](#)

[FRAGMENT_SET_DISPLACEMENT](#)

[OPC_ITERATION](#)

FRAGMENT_SET_DISPLACEMENT

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Sets a specific displacement for a fragment.

Usage

FRAGMENT_SET_DISPLACEMENT *tagname disp* [-force] [-incremental]

For `lapi_exec_tcl` scripting only:

FRAGMENT_SET_DISPLACEMENT *gid disp* [-force] [-incremental]

FRAGMENT_SET_DISPLACEMENT *array assocArray* [-force] [-incremental]

Arguments

- ***tagname***

Specifies a tag name.

- ***gid***

Specifies a generalized fragment ID, which can be a single fragment ID or a tag variable name. This is used when not in array mode.

- Non-array mode

In non-array mode, where ***gid*** is specified, the displacement for one or more fragments is set to a single displacement. The input displacement should be in user units.

- ***disp***

Specifies the displacement value in user units. This is used when not in array mode.

- ***array assocArray***

Enables array mode, where the associative array has fragment IDs for keys and displacements in user units as values.

- Array mode

In array mode (when array is specified), a set of fragments can be simultaneously set to different desired displacements.

- -force

Forces the displacement to this value, even if it violates mask constraints.

- Force Mode

When using -force, the fragment displacements are set to the desired values immediately. In this mode, there is no need to call FRAGMENT_MOVE afterwards,

because the fragments move immediately. This mode also overrides the default [max_iter_movement](#) value.

- Non-Forced Mode

If -force is not specified, the desired fragment displacements are set. The fragment displacement is not actually moved until the next call to FRAGMENT_MOVE, which will move all fragments to their desired displacement while obeying mask constraints. As the mask constraints are obeyed, setting a displacement may not result in the fragment actually moving to the displacement if it violates a mask constraint.

When using FRAGMENT_SET_DISPLACEMENT, the desired fragment displacement value overrides the max_iter_movement *val*. Note that the max_iter_movement command applies to an OPC iteration-directed edge movement.

- -incremental

Specifies that the displacement value is incremental to the current displacement.

Description

This command sets a desired displacement for one or more fragments. This displacement can be changed incrementally, or set to an absolute number. If the displacement is “forced,” it is set regardless of mask constraints such as external or internal.

Tcl Results

None

Performance

O(N), where N is the number of fragments in the input.

For best performance, it is recommended that you set entire fragment tag sets at once, instead of setting single fragments at a time.

Rating = fast

Examples

This example first places displacements into an array then sets them back again (using the -force option):

```
FRAGMENT_GET_DISPLACEMENT array A -out displacement_array
FRAGMENT_SET_DISPLACEMENT array displacement_array -force
```

fragtypetag

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Creates a tag set containing all fragments created by a specified fragmentation type.

Usage

fragtypetag *layer* *fragtype* *outTagName*

Arguments

- ***layer***

Specifies the layer.

- ***fragtype***

Specifies one of the following fragment types:

- fragtype_endpt — Selects all fragments with endpoints from the original layout, prior to fragmentation operations. These fragments are the only kind of fragments that were not created with a fragmentation operation.
- fragtype_builtin — Selects all fragments whose initial breakpoint was created by implicit fragmentation (implicit fragmentation is fragmentation performed by the hierarchical engine to help manage promotion of data between levels of the hierarchy). This includes fragments that were created to accommodate cell boundaries and tile boundaries, or promoted fragments from cells lower in the hierarchy.
- fragtype_corner — Selects all fragments whose initial breakpoint was created by a [fragment_corner](#) command.
- fragtype_interfeature — Selects all fragments whose initial breakpoint was created by an [fragment_inter](#) command.
- fragtype_layer — Selects all fragments whose initial breakpoint was created by an SRAF, correction, external, island, or visible layers, as well as layers declared with a command such as [fragment_layer](#).
- fragtype_max — Selects all fragments initially created by [fragment_max](#).
- fragtype_tag — Selects all fragments whose initial breakpoint was created from a tagging operations. This includes the interfeature, intersection, and other specialized forms.
- fragtype_other — Selects all fragments whose initial breakpoint was created by a custom operation, such as LAPI.

- *outTagName*

Specifies the output tag. The *outTagName* tag set contains all fragments whose first point was created by the fragmentation operation listed for *fragtype*.

Description

The *fragtypetag* command creates a tag set containing all fragments on the specified layer which were created by the fragmentation operation specified by *fragtype*.

Examples

In this example, *OUTPUT_SHAPE* creates outlines of all fragments that are on layer “*mx.opc.target_fill*” and whose first point was created by the [fragment_corner](#) operation.

```
fragment_layer mx.opc.target_fill {  
    . . .  
}  
. . .  
fragtypetag mx.opc.target_fill fragtype_corner fragByCorner  
OUTPUT_SHAPE fragment fragByCorner fragByCorner 0.001 0.002
```

GET_FRAGMENT_EPE_INFO

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Retrieves EPE measurement method and value for specified fragment(s).

Usage

GET_FRAGMENT_EPE_INFO *gid* -out *outVar*

Arguments

- ***gid***
Specifies the generalized fragment ID. It can be a single fragment ID, or a tag variable.
- **-out *outVar***
Specifies an output of one of the following (where *fragid* is the fragment ID number the information is associated with):
 - *outVar(method,fragid)*: EPE measurement method (max, min, or ave)
 - *outVar(epc,fragid)*: Chosen EPE value

Description

This command retrieves the EPE measurement method and value for fragment(s).

The 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 OPC uses to derive the fragment movement amount, and is set by OPC_ITERATION.

When calling OPC manually with the OPC_ITERATION and FRAGMENT_MOVE commands, GET_FRAGMENT_EPE_INFO returns the EPE before the FRAGMENT_MOVE command.

Tcl Results

None

Performance

O(*N*), where *N* is the number of fragments requested.

Examples

```
GET_FRAGMENT_EPE_INFO tags_neg -out epe_info
```

MEASURE

[Calibre nmOPC Tcl Scripting Commands](#)

[Measurement Controls](#)

Measures distances from fragments on one layer to the nearest fragments on another layer.

Usage

```
MEASURE op fromLayer toLayer metric max maxsearch [-moved]  
      -out_d measArray [-out_id meastoId]
```

Arguments

- ***op***
Specifies a type of operation: internal, external, enclose, or enclosing.
- ***fromLayer toLayer***
Specifies layers or tag sets to measure from and to. For internal checks, ***toLayer*** is not allowed. Measurements are always to the same layer.
- ***metric***
Specifies a measurement metric: opposite, euclidean, or opposite extended value.
- ***max maxsearch***
Specifies the maximum search distance in user units to look for CDs.
- ***-moved***
An optional flag that measures to/from the moved fragments.
- ***-out_d measArray***
Specifies an output associative array indexed by fragment ID containing the distance measurement in database units. A distance < 0 indicates no measurement ***toLayer*** fragment was found.
- ***-out_id meastoId***
An optional argument that specifies an output associative array indexed by fragment ID containing the fragment ID to which the distance is calculated. If the associative array entry for a given fragment is < 0, no ***toLayer*** fragment was found.

Description

This command calculates distances from fragments on the first input layer to nearby fragments on the second input layer, according to the specified measurement type (INTERNAL, EXTERNAL, ENCLOSE, or ENCLOSING). You can specify a type of measurement metric (opposite, euclidean, or opposite extended). You can also measure from a specified tag to another tag.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

None

Performance

$O(N)$ where N is the number of fragments in *fromLayer*

Alternative Commands

[NEWTAG external | internal | enclose | enclosing](#) is an easier and faster method to calculate sets of fragments based on distance measurements.

MEASURE_PER_FRAGMENT

Calibre nmOPC Tcl Scripting Commands

Measurement Controls

Measures against all the fragments in the search range.

Usage

```
MEASURE_PER_FRAGMENT op from_frag toLayer metric max maxsearch [-moved]
[-record mode] -out_d meas [-out_id frag_index]
```

Arguments

- ***op***
Specifies a type of operation: internal, external, enclose, or enclosing.
- ***from_frag***
Specifies the fragment ID of the “from” layer to measure from.
- ***toLayer***
Specifies the layer to measure to.
- ***metric***
Specifies a measurement metric: opposite, euclidean, or opposite extended value.
- ***max maxsearch***
Specifies the maximum search distance in user units.
- **-moved**
An optional flag that measures to/from the moved fragments.
- **-record *mode***
An optional directive that directs the system to record all comparable edges in the “from” layer within ***maxsearch*** from ***from_frag***. The *mode* keyword is a string to specify the record mode as follows:
 - no: No recording of the edges is performed (default).
 - sort_xy: Recording is performed and the resultant edges are sorted based on the coordinates.
 - sort_dist: Recording is performed and the resultant edges are sorted based on the distance from ***from_frag***.
- **-out_d *meas***
An output variable containing the closest distance measurement in user units. A distance < 0 means no measurement is found.

- `-out_id frag_index`

An optional output variable containing the fragment id to which the distance is calculated. If the output fragment is < 0, it indicates no “to” fragment was found.

Description

This command calculates distance from the specified fragment to the specified input layer according to the measurement type, which can be internal, external, enclose, or enclosing. The MEASURE_PER_FRAGMENT command shares the same measurement method as that of MEASURE, and the measurement is performed exactly the same way as a result. However, if you would like to measure distances between all fragments in the “from” layer and “to” layer, it is recommended that you use [MEASURE](#).

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

This command focuses on the measurement between the specified single fragment and all fragments of the “to” layer. It also has an option to record all comparable edges of the “to” layer within the maximum search distance from the specified fragment. The number of such comparable edges can be enormous even for just one fragment’s measurement, depending on the search distance, and whether or not the information on the comparable edges is stored internally.

The user can review the comparable edge information through

- [MEASURE_PER_FRAGMENT_QUERY_FIRST](#)
- [MEASURE_PER_FRAGMENT_QUERY_NEXT](#)
- [MEASURE_PER_FRAGMENT_QUERY_COUNT](#)

Shielding does not apply to those recorded edges. The system simply records all edges that have the right spatial relationship with the specified fragment (for example, if the external side of an edge is facing the external side of the specified fragment within the search range, then the edge is comparable).

Tcl Results

None

Performance

The function focuses on the single fragment measurement, and it would be faster than running LAPI_MEASURE and filtering out all information other than the particular single fragment. However, if all fragments are to be measured between the “from” layer and the “to” layer, it would be faster to use LAPI_MEASURE than to run LAPI_MEASURE_PER_FRAGMENT for each individual fragment.

Examples

```
NEWTAG topological inside_edge target marker -out tin
for { set C [FRAGMENT_ITERATOR first tin -out fragid] } { $C != 0 }
{set C [FRAGMENT_ITERATOR next tin -out fragid]}
{
    MEASURE_PER_FRAGMENT external $fragid target euclidean max 0.64 \
        -moved -record sort_dist -out_d meas -out_id frag_index
    set count [MEASURE_PER_FRAGMENT_QUERY_COUNT target]
if { 0 >= $count } { continue }
for { set EC [MEASURE_PER_FRAGMENT_QUERY_FIRST target -out_d \
    meas -out_id frag_index -out_pt pt_array] } { 0 != $EC }
{ set EC [MEASURE_PER_FRAGMENT_QUERY_NEXT target -out_d \
    meas -out_id frag_index -out_pt pt_array] } {
# do something here using $meas, $frag_index, $pt_array(x1),
# $pt_array(y1), $pt_array(x2), $pt_array(y2)
} }
```

MEASURE_PER_FRAGMENT_QUERY_COUNT

Calibre nmOPC Tcl Scripting Commands

Measurement Controls

Returns number of retrievable fragment edges.

Usage

MEASURE_PER_FRAGMENT_QUERY_FIRST toLayer

Arguments

- *toLayer*

Specifies the “to” layer identified by [MEASURE_PER_FRAGMENT](#).

Description

This command returns the total number of the retrievable edges, used in conjunction with [MEASURE_PER_FRAGMENT](#).

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

The number of the edges of the “to” layer measured with [MEASURE_PER_FRAGMENT](#)

Performance

Rating = fast. The bulk of the work is done by [MEASURE_PER_FRAGMENT](#).

Examples

```
NEWTAG topological inside_edge target marker -out tin
for { set C [FRAGMENT_ITERATOR first tin -out fragid] } { $C != 0 } \
    {set C [FRAGMENT_ITERATOR next tin -out fragid] } \
    {      MEASURE_PER_FRAGMENT external $fragid target euclidean max 0.64 \
        -moved -record sort_dist -out_d meas -out_id frag_index
        set count [MEASURE_PER_FRAGMENT_QUERY_COUNT target]
    if { 0 >= $count } { continue }
    for { set EC [MEASURE_PER_FRAGMENT_QUERY_FIRST target -out_d \
        meas -out_id frag_index -out_pt pt_array] } { 0 != $EC }
    { set EC [MEASURE_PER_FRAGMENT_QUERY_NEXT target -out_d meas -out_id \
        frag_index -out_pt pt_array] } {
        # do something here using $meas, $frag_index, $pt_array(x1),
        # $pt_array(y1), $pt_array(x2), $pt_array(y2)
    } }
```

MEASURE_PER_FRAGMENT_QUERY_FIRST

Calibre nmOPC Tcl Scripting Commands

Measurement Controls

Initiates a query to retrieve fragment edges.

Usage

```
MEASURE_PER_FRAGMENT_QUERY_FIRST toLayer -out_d meas  
[-out_id frag_index] [-out_pt pt_array]
```

Arguments

- ***toLayer***
Specifies the “to” layer identified by [MEASURE_PER_FRAGMENT](#).
- **-out_d *meas***
An output variable containing the closest distance measurement in user units. A distance < 0 means no measurement is found.
- **-out_id *frag_index***
An optional output variable containing the fragment id to which the distance is calculated. If the output fragment is < 0, it indicates no “to” fragment was found.
- **-out_pt *pt_array***
An optional output array containing the edge location. The contents of the array are as follows:
 - *pt_array(x1)*: the x-coordinate of the first end point of the edge in clockwise orientation.
 - *pt_array(y1)*: the y-coordinate of the first end point of the edge in clockwise orientation.
 - *pt_array(x2)*: the x-coordinate of the second end point of the edge in clockwise orientation.
 - *pt_array(y2)*: the y-coordinate of the second end point of the edge in clockwise orientation.

Description

This command initiates the retrieval of edges recorded by [MEASURE_PER_FRAGMENT](#). This command should be called once at the beginning of the edge data retrieval process. The rest of the edge data can be retrieved with [MEASURE_PER_FRAGMENT_QUERY_NEXT](#).

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

1 = Valid measured edge information returned

0 = No edge information returned

Performance

Rating = fast. The bulk of the work is done by MEASURE_PER_FRAGMENT.

Examples

```
NEWTAG topological inside_edge target marker -out tin
for { set C [FRAGMENT_ITERATOR first tin -out fragid] } { $C != 0} \
{set C [FRAGMENT_ITERATOR next tin -out fragid]}
{    MEASURE_PER_FRAGMENT external $fragid target euclidean max 0.64 \
-moved -record sort_dist -out_d meas -out_id frag_index
set count [MEASURE_PER_FRAGMENT_QUERY_COUNT target]
if { 0 >= $count } { continue }
for { set EC [MEASURE_PER_FRAGMENT_QUERY_FIRST target -out_d \
meas -out_id frag_index -out_pt pt_array] } { 0 != $EC } \
{ set EC [MEASURE_PER_FRAGMENT_QUERY_NEXT target -out_d \
meas -out_id frag_index -out_pt pt_array] }
# do something here using $meas, $frag_index, $pt_array(x1),
# $pt_array(y1), $pt_array(x2), $pt_array(y2)
} }
```

MEASURE_PER_FRAGMENT_QUERY_NEXT

Calibre nmOPC Tcl Scripting Commands

Measurement Controls

Queries data from subsequent fragment edges.

Usage

```
MEASURE_PER_FRAGMENT_QUERY_NEXT toLayer -out_d meas  
[-out_id frag_index] [-out_pt pt_array]
```

Arguments

- ***toLayer***
Specifies the “to” layer identified by [MEASURE_PER_FRAGMENT](#).
- ****-out_d** *meas***
An output variable containing the closest distance measurement in user units. A distance < 0 means no measurement is found.
- ****-out_id** *frag_index***
An optional output variable containing the fragment id to which the distance is calculated. If the output fragment is < 0, it indicates no “to” fragment was found.
- ****-out_pt** *pt_array***
An optional output array containing the edge location. The contents of the array are as follows:
 - *pt_array(x1)*: the x-coordinate of the first end point of the edge in clockwise orientation
 - *pt_array(y1)*: the y-coordinate of the first end point of the edge in clockwise orientation
 - *pt_array(x2)*: the x-coordinate of the second end point of the edge in clockwise orientation
 - *pt_array(y2)*: the y-coordinate of the second end point of the edge in clockwise orientation

Description

This command retrieves the next edge data recorded by [MEASURE_PER_FRAGMENT](#). This command can be called multiple times as long as there is edge data remaining.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

1 = Valid measured edge information returned

0 = No edge information returned

Performance

Rating = fast. The bulk of the work is done by MEASURE_PER_FRAGMENT.

Examples

```
NEWTAG topological inside_edge target marker -out tin
for { set C [FRAGMENT_ITERATOR first tin -out fragid] } { $C != 0} \
{set C [FRAGMENT_ITERATOR next tin -out fragid]} \
{    MEASURE_PER_FRAGMENT external $fragid target euclidean max 0.64 \
    -moved -record sort_dist -out_d meas -out_id frag_index \
    set count [MEASURE_PER_FRAGMENT_QUERY_COUNT target]
if { 0 >= $count } { continue }
for { set EC [MEASURE_PER_FRAGMENT_QUERY_FIRST target -out_d \
    meas -out_id frag_index -out_pt pt_array] } { 0 != $EC } \
{ set EC [MEASURE_PER_FRAGMENT_QUERY_NEXT target -out_d meas \
    -out_id frag_index -out_pt pt_array] } {
# do something here using $meas, $frag_index, $pt_array(x1),
# $pt_array(y1), $pt_array(x2), $pt_array(y2)
} }
```

ML_OPC

Calibre nmOPC Tcl Scripting Commands

ML OPC only

Moves or retargets fragments using machine learning models.

Usage

Syntax 1:

ML_OPC *tag ml_model* [*retarget_model*] [euv | duv]

Syntax 2:

ML_OPC -tag tag -model ml_model [-layer name] [-retargetModel *retarget_model*]
[-euv | -duv] [-incremental]

Arguments

- ***tag***
-tag *tag*
A required argument that specifies the fragments to act upon.
- ***ml_model***
-model *ml_model*
A required argument that specifies the machine learning model to use. The ***ml_model*** value must match a name specified by *ml_model_load*.
- **-layer *name***
An optional argument that specifies the name of a layer that contains the source geometry. If -layer is not specified, the opc layer is used.
You must use Syntax 2 to specify a layer.
- ***retarget_model***
-retargetModel *retarget_model*
An optional argument that specifies a second machine learning model to use. If this argument is not specified, no retargeting is done.
- **euv | duv**
-euv | -duv
An optional argument that specifies the type of the optical model.
 - euv — For EUV runs. This is the default.
 - duv — For non-EUV runs.

- -incremental

An optional argument that specifies to include previous corrections with the ML_OPCT corrections.

Without the -incremental option, ML_OPCT removes any prior OPC corrections. With -incremental, the ML_OPCT offset is added to the previous correction.

Description

The ML_OPCT command uses machine learning models to move or retarget fragments. Syntax 1 requires that all arguments be specified in the order given. For Syntax 2, keyword-value pairs can be specified in any order.

Examples

This example shows loading two models, their associated metadata, and using ML_OPCT to retarget fragments. Note that the rules must still define fragmentation and MRC constraints.

```
modelpath models:/models/ml

ml_model_load disp.mod L128_displacement/saved_model/frozen_model.pb
ml_model_load retr.mod L128_retargeting/saved_model/frozen_model.pb

ml_featurevector_load disp.mod mxopc.mod
ml_featurevector_load retr.mod mxopc.mod
...

denseopc_options mx_options {
    version 1
    layer opc.in      hidden
    layer opc.frag    opc      mask_layer 0
    layer sraf        sraf     mask_layer 0
    ...

    # Standard layer fragmentation and MRC setup
    fragment_layer_order full
    fragment_layer opc.frag {
        ...
    }
    mrc_rule internal opc.frag {
        ...
    }...

    # ML OPC
    NEWTAG all opc.frag -out AllTags
    ML_OPCT AllTags disp.mod retr.mod duv
    OUTPUT_RETARGET opc.frag -out retarget_out

    # Post-processing. Notice that fewer iterations are needed.
    sraf_sites_create
    sraf_print_avoidance ...
    OPC_XMEEF_ITERATION 1
    sraf_print_avoidance ...
    OPC_XMEEF_ITERATION 2
}
```

ML_VECTOR_CAPTURE_END

Calibre nmOPC Tcl Scripting Commands

ML OPC only

Marks the end of the data capture segment of the setup file used for machine learning.

Usage

Syntax 1:

ML_VECTOR_CAPTURE_END *tag* [*calibration_filename*]

Syntax 2:

ML_VECTOR_CAPTURE_END **-tag** *tag* [-cal *calibration_filename*]
[-gauges *gauge_filename*] [-layer *name*] [-model *ml_model*]
[-old *calibration_filename* -oldmodel *ml_model*] {[-record *values*]...} [-weightTag *tag*]

Arguments

- *tag*

-tag *tag*

A required argument specifying the name of a tag set that identifies the fragments to consider for calibration vectors. This should be the same tag set supplied to **ML_VECTOR_CAPTURE_INIT**.

- *calibration_filename*

-cal *calibration_filename*

An optional argument specifying a name for the calibration file. The calibration file, also called the normalized file, is used when training the neural network. The default name is *calibration_vectors.csv*.

If **-old** is specified, the calibration file contains the results of both this run and the one supplied by **-old**.

- **-gauges** *gauge_filename*

An optional argument that reads a gauge file and causes the run to only record values for fragments that cross the gauges.

This argument cannot be combined with **-old**.

- **-layer** *name*

An optional argument that specifies the name of a layer that contains the source geometry. The same layer should be used in **ML_VECTOR_CAPTURE_INIT**; if not, the run exits with an error that **ML_VECTOR_CAPTURE_INIT** must be specified before **ML_VECTOR_CAPTURE_END**.

If **-layer** is not specified, the **opc** layer is used.

When using `-layer`, the command must also supply a calibration file name with `-cal` and an output model name with `-model`.

- `-model ml_model`

An optional argument that specifies a name for the model that will be created. The default name is `phv0.mod`.

- `-old calibration_filename -oldmodel ml_model`

An optional set of arguments that includes the results of a previous run with the current data collection. The calibration file and model must have been created from an earlier run of the same SVRF and OPC rules as the current run. (The runs can use different layouts.)

- `-record values`

An optional argument that adds one or more columns to the calibration file. It can be specified multiple times.

When used with `-old`, the old calibration file must have been created with identical `-record` settings, including `column_name`.

There are two different syntaxes for `values`:

`column_name tag` — The keyword `-record` followed by the name for a column and the name of the tag set from which to populate it. Only a single column name-tag pair can be specified per `-record` keyword.

`(column_name tag column_name tag ...)` — The keyword `-record` followed by parentheses enclosing a list of column name-tag pairs. There can be any number of pairs.

These syntaxes can be combined. For example, any of these could be used:

```
-record angle aTag -record angle_prev a1Tag -record angle_next a2Tag  
-record angle aTag -record (angle_prev a1Tag angle_next a2Tag)  
-record (angle aTag angle_prev a1Tag angle_next a2Tag)
```

The columns contain property values from the specified tag set. If a fragment is not in `tag`, it receives a value of 0 in the column.

- `-weightTag tag`

An optional argument that specifies the name of a tag set that has been annotated with numeric properties representing weights. Fragments that do not have an annotation receive a weight of 1.0. Use [NEWTAG expression](#) to annotate the fragments, for example:

```
NEWTAG expression T1 {0.5} -aout weights
```

Access these values using “`-weightTag weights`.”

Description

The **ML_VECTOR_CAPTURE_INIT** and **ML_VECTOR_CAPTURE_END** commands mark the data capture region within a litho setup file and generate a calibration file. The calibration file is used in model training after Calibre exits.

Syntax 1 requires that all arguments be specified in the order given. For Syntax 2, keyword-value pairs can be specified in any order.

The **ML_VECTOR_CAPTURE_INIT** and **ML_VECTOR_CAPTURE_END** commands can be specified at most once per layer. If the layer is captured more than once, the run exits with a runtime error of “lapi_user_error: Two objects with same key.”

Examples

Example 1: Basic Syntax

This example shows the command to end a data capture region that considered all fragments. The results are written to the normalized file, *calibration_vectors.csv* (not shown).

```
ML_VECTOR_CAPTURE_END allFrags
```

Example 2: Adding Vectors to a Training Set

When you want to add features to an existing normalized file, use the **-old** and **-oldmodel** arguments. The SVRF rule file must be identical to the one used for creating the existing normalized file except for the layout specification and the addition of these arguments to the **ML_VECTOR_CAPTURE_END** command.

In the following example, the existing files have been renamed *run1_cal_vectors.csv* and *run1_phv0.mod* as the original **ML_VECTOR_CAPTURE_END** command relied on default names. The file loaded by **ml_featurevector_load** remains the same.

```
modelpath models:/models/ml
ml_featurevector_load nmod nmod/phv_final.mod
...
denseopc_options mx_options {
    ...
ML_VECTOR_CAPTURE_INIT allFrags nmod
    ...
ML_VECTOR_CAPTURE_END -tag allFrags -old run1_cal_vectors.csv \
    -oldmodel run1_phv0.mod
    ...
}
```

After the modified file is run, the new calibration file and machine learning model contain the data from both the old data and the current run. Because the output files are not specified in the **ML_VECTOR_CAPTURE_END** command, they are named *calibration_vectors.csv* and *phv0.mod*.

Related Topics

[Output Options from Gauge Objects \[Calibre WORKbench User's and Reference Manual\]](#)

ML_VECTOR_CAPTURE_INIT

Calibre nmOPC Tcl Scripting Commands

ML OPC only

Marks the beginning of the data capture segment of the setup file used for machine learning.

Usage

Syntax 1:

ML_VECTOR_CAPTURE_INIT tag ml_model [euv | duv]

Syntax 2:

ML_VECTOR_CAPTURE_INIT -tag tag -model ml_model [-layer name] [-euv | -duv]

Arguments

- **tag**

-tag tag

A required argument specifying the name of a tag set that identifies the fragments to consider for calibration vectors. This should be the same tag set supplied to ML_VECTOR_CAPTURE_END.

- **ml_model**

-model ml_model

A required argument specifying the name of the model used during training.

- **-layer name**

An optional argument that specifies the name of a layer that contains the source geometry. The same layer should be used in ML_VECTOR_CAPTURE_END; if not, the run exits with an error that ML_VECTOR_CAPTURE_INIT must be specified before ML_VECTOR_CAPTURE_END.

If -layer is not specified, the opc layer is used.

- **euv | duv**

euv | -duv

An optional argument specifying the type of optical model.

euv — For EUV runs. This is the default.

duv — For non-EUV runs.

Description

The **ML_VECTOR_CAPTURE_INIT** and **ML_VECTOR_CAPTURE_END** commands mark the data capture region within a litho setup file and generate a calibration file. The name of the calibration file is set by **ML_VECTOR_CAPTURE_END**. The calibration file is used for model training after Calibre exits.

Syntax 1 requires that all arguments be specified in the order given. For Syntax 2, keyword-value pairs can be specified in any order.

The ML_VECTOR_CAPTURE_INIT and ML_VECTOR_CAPTURE_END commands can be specified at most once per layer. If the layer is captured more than once, the run exits with a runtime error of “lapi_user_error: Two objects with same key.”

Examples

This example starts a data capture segment. It considers all fragments for use with the model named “nmod.”

```
ML_VECTOR_CAPTURE_INIT allFrags nmod
```

This is the equivalent Syntax 2 version:

```
ML_VECTOR_CAPTURE_INIT -tag allFrags -model nmod
```

MRC

Calibre nmOPC Tcl Scripting Commands

OPC Controls

Establishes tag-specific adjustment of width or spacing between fragments after OPC movement.

Note

 The MRC command is not recommended to define rules as it does not adhere to existing MRC rules, jog freeze definitions, and so on. Use [MRC_RULE \(Custom Scripting Command\)](#) instead to define rule checks.

Usage

```
MRC {internal | external} dist tag1 tag2 priority1 priority2 [opposite [extended_dist]]  
[projecting dist | not _projecting dist]
```

Arguments

- **internal | external**

A required keyword that sets the direction of the check.

- ***dist***

A required value in microns that specifies the minimum distance between fragments for this check.

- ***tag1 tag2***

A required pair of tag names identifying fragments to check.

For the check to apply to any two fragments, one fragment must be in the ***tag1*** set and the other must be in the ***tag2*** set.

- ***priority1 priority2***

A required pair of values indicating how to move constrained fragments.

Assigns a priority for which fragment can move more, when the fragments are constrained. The fragments move in the following proportion:

- The ***tag1*** fragment moves by $\text{priority1}/(\text{priority1}+\text{priority2})$
- The ***tag2*** fragment moves by $\text{priority2}/(\text{priority1}+\text{priority2})$

- **opposite**

An optional keyword that specifies to use the opposite measurement metric instead of the default (Euclidean).

- ***extended_dist***

An optional value in microns that triggers the extended opposite 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.

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 Calibre nmOPC 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.

Tcl Results

None

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.
- The [MRC_RULE \(Custom Scripting Command\)](#) sets MRC rules *between tag sets*.

MRC_RULE (Custom Scripting Command)

Tagging Controls

OPC Controls

Creates external and internal spacing constraints based on tag sets.

Usage

```
MRC_RULE {external | internal | enclosure} tag1 [tag2] [-tagOnly] '{'
{
[[projecting [dist] | not_projecting [dist]]
{
[length len] use d1 metric [source number]
}...
}...
}'
```

The braces ({{}}) with quotation marks (‘’) around them are meant as literals (without the quotes).

Arguments

- **external | internal | enclosure**

A required keyword that specifies the check type. These keywords allow specific mask constraint rules to be applied to fragments specified in the tag sets. The check types are the same used by the [mrc_rule](#) setup file command.

- **tag1 [tag2]**

A required argument that specifies a tag set or tag sets to be used by the MRC rule. The second tag set is optional.

- **-tagOnly**

An optional keyword that constrains the MRC check to only those fragments that are members of the tag sets. Jogs that grow into fragments are not checked for MRC violations.

- **projecting dist**

An optional argument 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*).

A single MRC_RULE may have multiple projecting clauses.

- **not_projecting dist**

An optional argument 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*).

Note

 Using metrics other than euclidean with not_projecting can cause inconsistencies, as discussed in “[OPC Output Consistency](#)” on page 650.

Only one not_projecting clause may be specified per MRC_RULE.

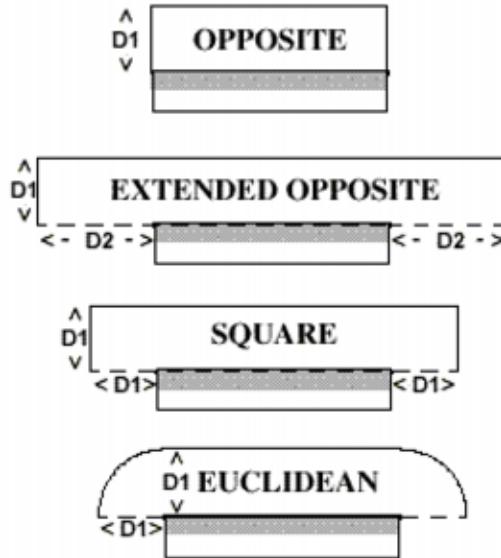
- **use *d1 metric***

A required argument set that 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 5-3](#) shows the measurement regions created for each of the different metrics, and the dimensions used with each.

Figure 5-3. The Four Metrics for Tag-Based MRC



- **length *len***

An optional argument 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.

- **source *number***

An optional argument that marks fragments blocked from moving by the “use” criteria. The source number is used by TAG_MRC_SRC when creating tag sets. Set *number* to an integer from 2 to 1024, inclusive.

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.

The maximum rule count for MRC_RULE is 64.

Tcl Results

None

Performance

Very fast, but will slow down [FRAGMENT_MOVE](#) and [OPC_ITERATION](#).

Examples

Example 1

This MRC_RULE checks external constraints between fragments tagged with line_end and short_side.

```
MRC_RULE external line_end short_side {  
    use 0.016 SQUARE  
    projecting 0.01 use 0.018 EUCLIDEAN  
    projecting 0.03 use 0.020 EUCLIDEAN  
}
```

Example 2

This example shows using MRC_RULE ...source with TAG_MRC_SRC to identify which rule is blocking a fragment.

```
MRC_RULE internal tAll tAll {  
    projecting 0.014 use 0.043 opposite source 2  
    not_projecting use 0.041 euclidean source 3  
}  
  
MRC_RULE external tAll tAll {  
    projection 0.014 use 0.042 opposite source 4  
    not_projecting use 0.041 euclidean source 5  
}  
  
TAG_MRC_SRC tAll 1 -aout mrc_rule_src
```

Related Topics

[mrc_rule](#)

[TAG_MRC_SRC](#)

NEWTAG all

[Calibre nmOPC Tcl Scripting Commands](#)

[Tagging Controls](#)

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.

Description

Creates a fragment set consisting of all fragments.

Tcl Results

None

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 area

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Tags fragments based on the width and length of the polygon.

Usage

NEWTAG area *layer* **-maxExtent** *X* [*Y*] [-unmoved] *tag_sets*... [-outRest *tag2*]

Arguments

- ***layer***
A required argument that specifies the name of a layer containing polygons. The layer type must be one of correction, opc, or sraf.
- ***-maxExtent X* [*Y*]**
A required keyword and one or two numeric values that specify the maximum extent of polygons to consider. If *Y* is not specified, it defaults to *X*. The values should be less than the interaction distance, and are in microns.
- **-unmoved**
An optional keyword that specifies to use the initial coordinates when computing maximum extent and area. By default, the current coordinates are used.
- ***tag_sets***
A required set of arguments that specifies one or more tag sets. The set(s) cannot be followed by any arguments except -outRest.

Each tag set definition uses the following syntax:

-[a]out *tag* range *constraint*

where

- The set is introduced with either “-out” or “-aout.” Use -aout to attach area as a property to the fragments of the tag set.
 - The set’s name is given by *tag*.
 - The ***constraint*** applies to the area (not the extent) of the polygon. Fragments on polygons satisfying the constraint are added to the tag set. Area is calculated in user units, typically square microns.
- **-outRest *tag2***
An optional argument that specifies a tag set for all fragments on the layer that did not meet any of the defined ranges. Fragments on polygons that exceed **-maxExtent** are assigned to the *tag2* set.

Description

Creates one or more sets of fragments associated with polygons of specified areas. By default, only horizontal and vertical fragments are tagged. Skew edges are ignored.

For “binning” fragments into multiple tag sets, it is recommended that you define a single NEWTAG area with multiple -out options rather than multiple calls to NEWTAG area.

Examples

This example creates four tag sets. The -outRest set contains fragments from those polygons with at least one side greater than -maxExtent.

```
NEWTAG area poly.in -maxExtent 0.1 0.4 \
    -out tagAreaS range > 0 < 0.005 \
    -out tagAreaM range >= 0.005 < 0.01 \
    -out tagAreaL range >= 0.01 \
    -outRest tagArea
```

NEWTAG blocked

[Calibre nmOPC Tcl Scripting Commands](#)

[Tagging Controls](#)

Tags fragments blocked by an MRC rule.

Usage

NEWTAG blocked *tin* -out *tag* [-outRest *tag2*]

Arguments

- ***tin***
A required argument that identifies the initial tag set of fragments.
- **-out *tag***
A required argument that names the output tag set.
- **-outRest *tag2***
An optional argument that creates a second tag set containing all fragments from ***tin*** that are not blocked.

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](#).

Tcl Results

None

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
```

Related Topics

[TAG_MRC_SRC](#)

NEWTAG cd

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Measures CDs from a contour.

Usage

```
NEWTAG cd tag_in contour_ID {-out tag mode range constr} ... [-outRest tag2]
```

Arguments

- ***tag_in***
A required argument that identifies the initial tag set of fragments.
- ***contour_ID***
A required argument that 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***.

Description

Given a pre-simulated contour with the input ***contour_ID***, which takes on the values of IM0...IM32, this command measures the CD and outputs a given bin.

Tcl Results

None

Performance

O(N) where N is the size of *tag_in*

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
```

Alternative Commands

[DENSE_EPE](#) computes the EPE and CD statistics for a single fragment and returns the information in an associative array.

NEWTAG complete

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

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.

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.

Tcl Results

None

Performance

$O(N)$ where N is the size of *tin*.

NEWTAG corner

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Performs a search for fragments based on the distance from a corner.

Usage

```
NEWTAG corner tin corner_desc [dist] dist_constr  
    [len constr] [len1 constr] [-jog_tol value] -out tag [-outRest tag2]
```

Arguments

- ***tin***
A required argument that specifies the name of the tag set from which to perform the calculations.
- ***corner_desc***
A required argument to identify the corner. The argument has two different forms:
 - type*** — One of the following literal keywords identifying the corner type: **convex**, **concave**, **any**.
 - angle constraint*** — The keyword **angle** followed by a degree-based constraint such as ≥ 90 . The angle is measured on interior abutting edges.
- **[*dist*] *dist_constr***
A required parameter that specifies the constraint describing the distance from the corner. The keyword “*dist*” is required when separating an **angle constraint** from *dist_constr*.
- ***len constr***
An optional keyword that specifies the length criteria for the current edge.
- ***len1 constr***
An optional keyword that specifies the length criteria for the neighboring edge at the corner.
- **-jog_tol *value***
An optional argument that ignores, or tolerates, jogs shorter than *value* when computing *len* and *len1*.

Note

 If -jog_tol is specified, jogs with lengths less than or equal to the tolerance are ignored in both *len* and *len1* calculations.

- **-out *tag***

A required keyword specifying the name of a tag set. Each fragment meeting the input tag criteria is placed into 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.

Description

This command performs a search for fragments that satisfy a distance constraint from the corner of an edge. This allows multiple fragments to be selected near a corner. One use of NEWTAG corner is to select fragments near the end of a long edge.

Tcl Results

None

Performance

O(N) where N is size of input set *tin*

Rating = fast

Examples

```
NEWTAG corner ALL convex < 0.600 len > 0.8 -out near_convex_corner
```

NEWTAG correction_map

Calibre nmOPC Tcl Scripting Commands

Creates a mapping between correction layer fragments and OPC layer fragments.

Usage

NEWTAG correction_map *corr_tag* -opc frags *opc_tag* -out *tag* [-outRest *tag2*]

Arguments

- ***corr_tag***

A required argument that specifies an input tag set that contains correction fragments.

- **-opc frags *opc_tag***

A required argument specifying an input tag set that contains OPC fragments that the correction fragments are matched against.

- **-out *tag***

A required argument that specifies an output tag set that contains the subset of correction fragments that are mapped to given OPC fragments.

- **-outRest *tag2***

An optional argument that creates a second tag set containing all fragments from ***corr_tag*** that did not correspond to a fragment in ***opc_tag***.

Description

This command is used to create a mapping between correction layer fragments and OPC layer fragments.

The output tag set ***tag*** consists of correction layer fragments that belong to the input correction tag set that are mapped to corresponding OPC fragments in the given OPC tag set.

If a correction fragment maps to only one OPC fragment, then that OPC fragment needs to be a part of the input OPC tag set. However, a correction fragment can map to multiple OPC fragments, in which case only one of those OPC fragments needs to be in the input OPC tag set to generate an output.

Examples

Given a correction tag set “all_corr” and an OPC tag set “all_opc”, the following command maps the correction layer tags to the matching OPC layer tags, and writes the results to the “mapped_correction frags” tag.

```
NEWTAG all correction_layer -out all_corr
NEWTAG all opc_layer -out all_opc
NEWTAG correction_map all_corr -opc frags all_opc \
    -out mapped_correction frags
```

NEWTAG displacement

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

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.

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.

Tcl Results

None

Performance

O(*N*) where *N* is size of input set *tin*

Rating = fast

Alternative Commands

[FRAGMENT_GET_DISPLACEMENT](#) is more suited towards getting the displacement of a single fragment.

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

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Searches an edge for fragments within a given set of constraints.

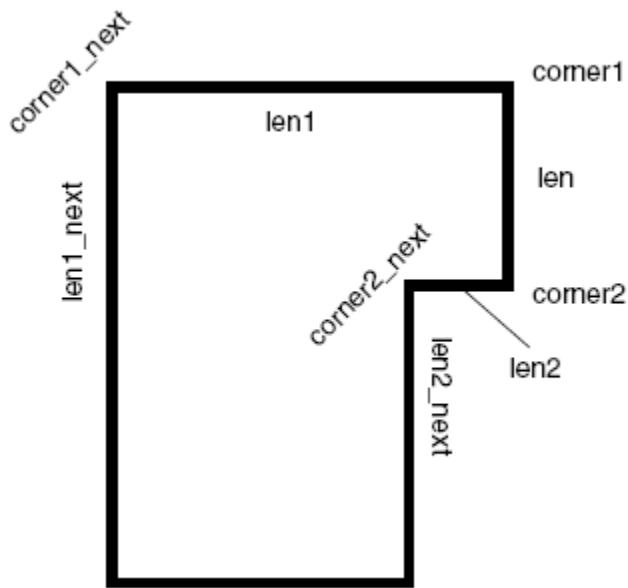
Usage

```
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 5-4. 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 specify 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 results 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 OPC 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

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Measures EPE on a given contour.

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 OPC 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 OPC.

Note

 When using **OUTPUT_SHAPE** fragment with NEWTAG epe, note that the **OUTPUT_SHAPE** generates a contour based on the EPE from the *prior* OPC iteration, while NEWTAG epe obtains its EPE from the *final* iteration of the OPC 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

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.

Tcl Results

None

Performance

O(N) where N is the size of **tag_in**

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 expression

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Works with numerical properties on fragments. Can be used to annotate fragments or create new tag sets based on existing annotations.

Usage

NEWTAG expression *tag_in* '{' *expression* '}' {-aout | -out} *new_tag*

Arguments

- ***tag_in***

A required argument that specifies a tagged set of fragments to evaluate.

- **{*expression*}**

A required argument enclosed in braces. The expression can be a number, an arithmetic or Boolean calculation, or one of the functions listed in the Description section.

The expression can span multiple lines. See “Examples.”

- **{-aout | -out} *new_tag***

A required argument specifying a name for the tag set.

-aout — Creates an annotated tag set. Each fragment is associated with its result from expression.

This argument can also be written as “-out -annotated *new_tag*”.

-out — Creates a tag set. This set is not annotated, and requires that *expression* return a constant. See “[Example 5: Constant Expressions](#)” on page 489

You must specify -aout or -out but not both.

Description

Many of the tagging operations can put a numeric value into a tag set. This command can manipulate these values and place the result in an output tag set.

Operators

Most common Tcl operators are supported: arithmetic, relational, logical, and ternary (?:). The precedence is also as in Tcl.

The modulus operator % and bitwise operations are not supported.

The full list of supported operators by precedent is

1. ! (Boolean not) and unary - (negate)
2. * (multiply) and / (divide)

3. + (add) and - (subtract)
4. < > <= >= == != (relational operators)
5. && (logical AND)
6. || (logical OR)
7. ?: (ternary operator, functions as “if ? then : else”)

When an expression has operators of the same precedence on either side of an operand, it is evaluated right to left. Use parentheses or subexpressions for clarity.

Subexpressions and Filters

Subexpressions are named expressions. They can be included in the *expression* or called using the expr() function. Subexpressions are defined using the following syntax:

```
subexpression name = expression ;
```

Filters determine whether a fragment is included in *new_tag*; any filter that evaluates to 0 excludes that fragment. Filters generally appear in *expression* after any subexpressions. They are defined using the following syntax:

```
filter expression ;
```

Functions

Following are a list of available functions. They can be called directly in *expression* or as part of a subexpression or filter. Names are case-sensitive.

- **ceil(*expression*, *step_size*)**
Returns the smallest multiple of *step_size* that is equal to or greater than *expression*.
- **expr(*subexpression_name*)**
References the value of the named subexpression. Subexpressions can also be referenced without expr() by using the name in another expression.
- **floor(*expression*, *step_size*)**
Returns the largest multiple of *step_size* that is less than or equal to *expression*.
- **has_tag(*tag*)**
Returns true (nonzero) if the current fragment is in *tag*.
- **maximum(*expression*[, *expression*]...)**
minimum(*expression*[, *expression*]...)
Evaluates all expressions and returns the maximum or minimum result. At least one *expression* must be specified.

- **neighbor(*tag* [, *constraint*]...)**

Evaluates to the ***tag*** value of a neighbor of the current fragment. Only the fragments that share an endpoint with the current fragment are considered. Use *constraint* to define which neighbor (for example, the neighboring fragment that is also a member of a third tag set). When both neighbors satisfy the same constraint, a resolution constraint decides the return value.

There can be up to 9 constraints. The constraint keywords are as follows:

- corner:convex — selects the neighboring convex corner fragment
- corner:concave — selects the neighboring concave corner fragment
- corner:any — selects the neighboring corner fragment of either type
- corner:noncorner — selects the neighboring fragment that does not have an end on a corner
- resolution:avg
- resolution:max
- resolution: min — if both neighbors satisfy the preceding constraint, resolves to the average, larger, or smaller value.
- withtag: *tag3* — selects the closest fragment that is also in *tag3*.
- default:*expression* — returns the result of *expression* if no neighbor satisfies the constraint(s).

- **property(*tag* [,*else*])**

property(:*prop* [,*else*])

If the first argument is a tag set name, property returns a value based on whether the current fragment is a member of ***tag***. If it is, the value in ***tag*** for that fragment is returned. If it is not, the result of evaluating *else* is returned.

If the first argument begins with a colon (:), it is treated as a built-in property. The available properties are

- :desired_disp — displacement goal of the fragment.
- :disp — current displacement for the fragment.
- :disp_limit_inner — inner displacement limit.
- :disp_limit_outer — outer displacement limit.
- :epe — EPE value from the last simulation.
- :epe_measurement_method — EPE measurement method (0: average, -1: minimum, 1: maximum).

- :feedback — feedback for the current iteration.
- :rl_control — amount of site shift. If there are multiple sites on a fragment, the value to return is specified by :epc_measurement_method.
- :target_control — amount of target shift.

“else” defaults to the **tag_in** value in both cases.

- **round(*expression, step_size*)**

Returns the nearest multiple of **step_size** to **expression**. When the multiples of **step_size** are equally near, the one with the larger absolute value is returned.

- **space_pair(*max_distance* [,property:*tag*] [,default:*expr*] [,resolution:{avg|min|max}])**

Computes the distance between a fragment and the opposing fragment on a different polygon provided it is within **max_distance**. **max_distance** should not exceed the interaction distance. The units can be specified as part of the value, but default to microns.

The optional property:*tag* causes the value in the tag set *tag* for the opposing fragment to be returned instead of the computed distance.

The optional default:*expr* is evaluated when no opposing fragment is found in **max_distance**.

The optional resolution setting determines the returned value when there are multiple opposing fragments at the same distance. It has no effect unless property:*tag* is also specified. The default is resolution:avg, the average of the values.

- **width_pair(*max_distance* [,property:*tag*] [,default:*expr*] [,resolution:{avg|min|max}])**

Computes the distance between a fragment and the opposing fragment on the same polygon provided it is within **max_distance**. **max_distance** should not exceed the interaction distance. The units can be specified as part of the value, but default to microns.

The optional property:*tag* causes the value in the tag set *tag* for the opposing fragment to be returned instead of the computed distance.

The optional default:*expr* is evaluated when no opposing fragment is found in **max_distance**.

The optional resolution setting determines the returned value when there are multiple opposing fragments at the same distance. It has no effect unless property:*tag* is also specified. The default is resolution:avg, the average of the values.

Examples

Example 1: select Function

The following example doubles the width measurement for special cases. It uses ?: to determine if the current fragment of width_tag is also in the set “special_frags.” If it is, the value is doubled. If it is not, its current value is returned.

```
NEWTAG expression width_tag {special_frags?tagval()*2:tagval()} \
-aout new_width_tag
```

Example 2: Subexpressions

This example uses subexpressions to make the overall expression more readable. Notice how each subexpression starts on a new line and is terminated with a semicolon.

```
NEWTAG expression in_tag {
    subexpression frag_pitch = property(pitch_tag);
    subexpression opposite_frag_pitch = width_pair(0.2um, \
        property: pitch_tag, default: frag_pitch);
    pitch_tag?(frag_pitch + opposite_frag_pitch) / 2 : 0.0um
} -aout average_pitch_tag
```

Example 3: Filters

This example selects the fragments that are also in tag1 or tag2. Notice that filters, like subexpressions, are terminated by a semicolon.

```
NEWTAG expression in_tag {
    filter has_tag(tag1) || has_tag(tag2);
    property(in_tag)
} -aout out_tag
```

Example 4: Filtering the Results of a Subexpression

This example finds all fragments that have a combined width and space greater than 400 nm.

```
NEWTAG expression in_tag {
    subexpression result = property(width_tag) + property(space_tag);
    filter result > 0.4;
    result
} -aout out_tag
```

Example 5: Constant Expressions

This expression creates a tag set of all fragments that have a combined width and space greater than 400 nm, but does not annotate the output. Instead of returning “result”, as in the preceding example, it returns a constant 0 and uses “-out” instead of “-aout.”

```
NEWTAG expression in_tag {
    subexpression result = property(width_tag) + property(space_tag);
    filter result > 0.4;
    0
} -out out_tag
```

NEWTAG external | internal | enclose | enclosing

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

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_also | -perpendicular_only} [-corner_block length]]  
[-c2c [=45 | ==45 | >=0]] [-moved [-tag_only]] [-drc_style]]  
[-shielding_check {none | simple | all}]  
[-out tag range constr]... [-outRest tag2]
```

Arguments

- **external | internal | enclose | enclosing**

A required argument that specifies if the area to be measured is to be external, internal, enclosed, or enclosing. Specify only one of the four keywords.

Setting “enclosing” specifies that the layers or tags specified first (“from”) *enclose* layers or tags specified second (“to”). For example, the following code outputs fragments from metal1, where the metal1 to contact1 enclosure is within the given constraint.

```
NEWTAG enclosing metal1 contact1 ...
```

Setting “enclose” specifies that tags or layers specified first (“from”) are *enclosed by* the tags or layers specified second (“to”). For example, the following code outputs fragments from contact1, where the metal1 to contact1 enclosure is within the given constraint:

```
NEWTAG enclose contact1 metal1 ...
```

- **from**

A required argument that specifies a layer or tag name.

- **to**

A required argument that specifies a layer or tag name.

- **metric**

A required argument that specifies a measurement metric: opposite, euclidean, or opposite extended *value*.

- **-perpendicular_also [-corner_block length]**

An optional argument that measures perpendicular fragments in addition to other orientations.

The -perpendicular_also and -perpendicular_only options cannot be specified together. See -perpendicular_only for the description of -corner_block.

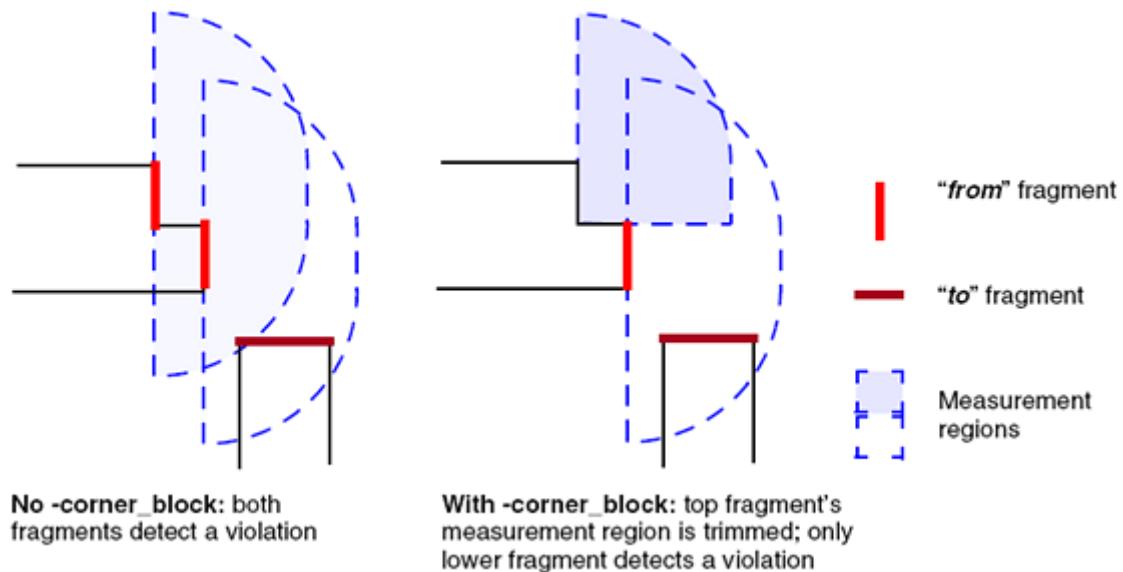
- **-perpendicular_only [-corner_block *length*]**

An optional argument that indicates to only measure 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 ***to*** 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 $\geq \text{length}$ 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 will be in the **-out** tag set.

Figure 5-5. Effect of -perpendicular_only and -corner_block



- **-c2c [!=45 | ==45 | >=0]**

An optional argument that measures fragments when they are in a corner-to-corner configuration. This configuration is defined as fragments being parallel and not projecting, with additional qualifiers:

- **external:** both fragments are on convex 90-degree corners.
- **internal:** both fragments are on concave 90-degree corners.
- **enclose:** the ***from*** fragment is on a convex 90-degree corner and the ***to*** fragment is on a concave 90-degree corner.
- **enclosing:** the ***to*** fragment is on a convex 90-degree corner and the ***from*** fragment is on a concave 90-degree corner.

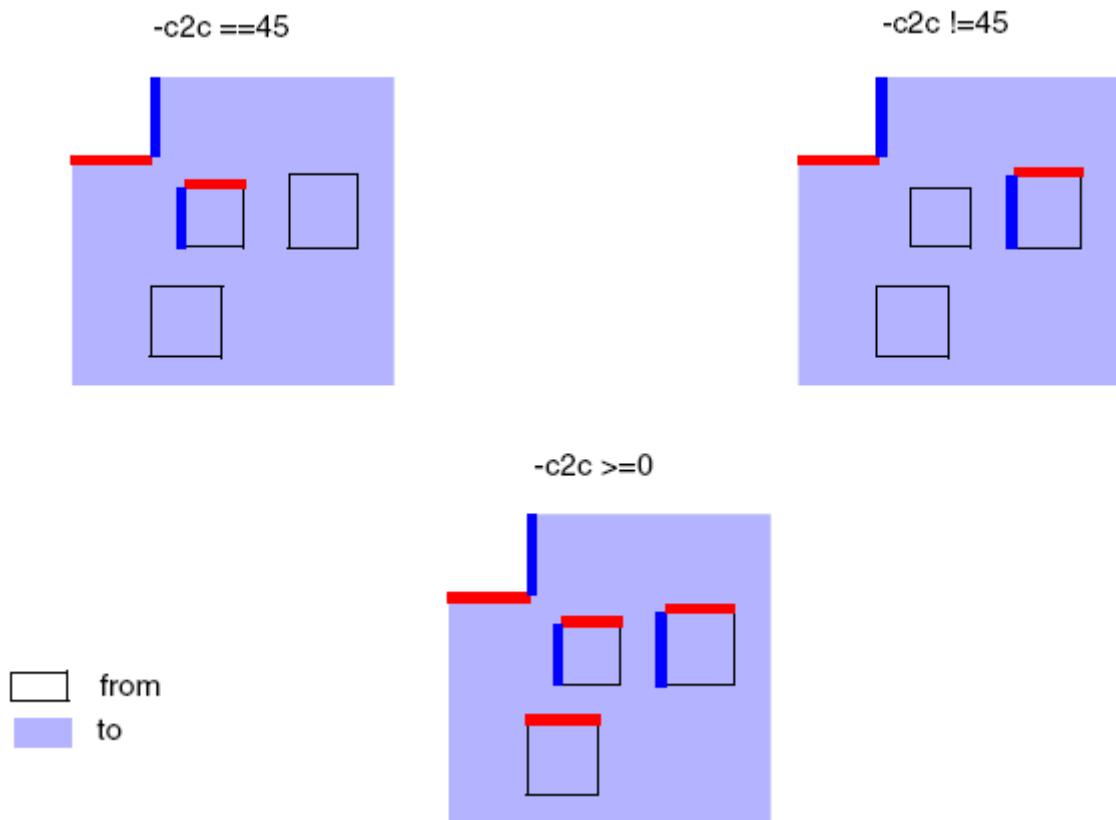
By default, projection is not allowed between the fragments or corners. You can permit projection using one of the angle constraints, visualized as being on a line drawn between the two corners relative to the x-axis.

$!=45$ — This is the default behavior when no constraint is specified. Projection is not allowed, but the two corners do not need to have equal vertical and horizontal separation.

$==45$ — Only corners that would describe a line segment at a 45-degree angle are measured.

$>=0$ — Any orientation is allowed, but projection must be zero length.

Figure 5-6. Enclosing -c2c Constraints



- **-moved [-tag_only]**

An optional argument that measures to or from the distance of the moved fragments.

The optional `-tag_only` keyword constrains the measurement to only those fragments that are original members of `to`; jogs created by fragment movement are not considered.

- **-drc_style**

An optional argument that specifies that the OPC 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}`

An optional argument that 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.

Note

 If the command relies on a shielding check for correct results, use derived layers or tag sets that use no more than two layers.

- `-not_projecting`

An optional argument that restricts output to **from** fragments that do not project on any of the **to** fragments in the range specified by “**range constr**.”

- `-projecting [all] [constr]`

An optional argument that restricts output to **from** fragments that project onto at least one of the **to** fragments in the range specified by “**range constr**.”

If a local *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. If “all” is specified, a **from** fragment must project on all **to** fragments in the range specified by *constr* in order to be output.

- `-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.

As of version 2020.3, all constraints must include an upper bound or the run exits with an error message similar to the following:

```
ERROR: The constraints for Tagscript NEWTAG external/internal/
enclose/enclosing must have an upper bound.
```

- `-outRest tag2`

An optional argument that creates a tag set of all **from** fragments that did not satisfy any of the **constr** conditions.

Description

This command computes distance as internal, external, or enclosure from fragments in the “from” layer or tag set to fragments in the “to” layer or tag set. 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 time 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 (in other words, not included) when it violates the constraint with any fragment. Shielding is not considered during projection comparisons.

Tcl Results

None

Performance

$O(N)$ where N is the number of fragments in the “from” layer.

Alternative Commands

[MEASURE](#) can be used to obtain the actual measurement value on a per-fragment basis.

[NMBIAS_MEASURE_TAG](#) can make many of the same measurements, and also allows annotating the fragments with the results.

Examples

Example 1: Two Ranges

The following example performs a simple check:

```
NEWTAG external L0 L1 opposite -out dense range < 0.065 -out iso range \
> 0.065 < 0.2
```

Example 2: Performance Comparison

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:

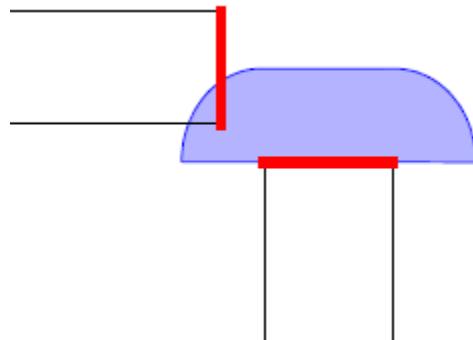
```
NEWTAG external L1 L2 opposite -out T1 range < 0.1 \
-out T2 range >= 0.1 < 0.2 -out T3 range >= 0.2 < 0.3
```

Example 3: Corner-to-Corner Line-End Checks

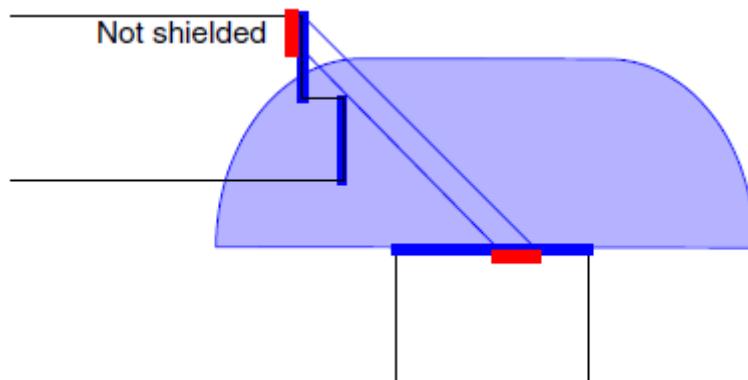
The -perpendicular_only option is ideal for measuring spacing between space ends or line ends in two-dimensional layouts such as shown in the following figure and example code:

```
# Collect the line ends
NEWTAG edge tAll len < 0.12 corner1 convex corner2 convex \
len1 > 0.3 len2 > 0.3 -out tgLE
```

```
# Tag line ends that violate corner-to-corner spacing
NEWTAG external tgLE tgLE euclidean -perpendicular_only \
-out tgCloseLE range < 0.05
```



The shielding behaves similarly to DRC-style shielding in that interaction and shielding are checked separately. If the vertical line end were in multiple parts and the upper part still intersected the measurement region, as shown below, if any part of the upper fragment was “visible” from the source fragment, the upper fragment would be returned even though the visible part is not interacting with the measurement region.



In the figure, the solid blue lines represent the fragments being checked, and the red represents the visible portion. Notice how the red is outside the measurement region.

NEWTAG facing_pattern

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Generates fragment tag sets of same and opposite facing patterns.

Usage

```
NEWTAG facing_pattern pattern_spec tag -layer layer_name [layer_name2...]  
-out output_tag range constraint [metric]
```

Arguments

- *pattern_spec*

Specifies filtering options to select fragments for the tag set based on pattern IDs. Valid options are as follows:

ANY — The pattern IDs of the fragment and edge pair are ignored (no check or filtering of the pattern IDs is performed).

SAME — The fragment is selected only if its pattern ID is the same as the pattern ID of the edge the fragment is facing.

DIFF — The fragment is selected only if its pattern ID not equal to the pattern ID of the edge the fragment is facing.

patternId1 patternId2 — The fragment is selected only if its pattern ID is equal to *patternId1* and the pattern ID of the edge the fragment is facing is equal to *patternId2*.

- *tag*

Specifies the tag set from which fragments facing each other are selected.

- *-layer layer_name* [*layer_name2...*]

Specifies the layer name from which fragments facing each other are selected.

- *-out output_tag*

Specifies the destination output tag to store the fragments facing each other (or facing layer(s) edges) to.

- *-range constraint*

Specifies the maximum distance between fragments. The range can be specified using the following operators: < and <= *value*.

- *metric*

Optionally specifies the metric to apply when measuring the distance between the fragment and facing edge to determine if the distance is within the range constraint. Valid options are: opposite, euclidean, or opposite extended *value*.

Description

This command is used to find fragments facing each other with optional filtering based on the fragment's pattern ID. Fragments from the input tags are scanned. Fragments are selected to be stored in the specified tag set if:

- The fragment “can see” (is facing without obstruction) any edge on the specified layer(s) and is not obstructed by other fragments or edges,
- The distance between the fragment and edge is within the user-specified range,
- Pattern IDs (of the fragment or edge pair) fit the user-specified pattern specification.

Tcl Results

None

NEWTAG fragment

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

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] [-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 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 501.) 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 specifies 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.

The arguments cannot be specified with -jog_tol.

- `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 len1 and len2. It also ignores jogs when computing len unless two concave or two convex corners are specified. To ignore jogs in this case, use -jog_tol_main.

You must specify either two identical corner constraints or a single constraining convex or concave length.

This argument cannot be specified with any of the options referring to moved fragments, such as mlen1.

- `-jog_tol_main value`

An optional argument that ignores jogs for len 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.

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.

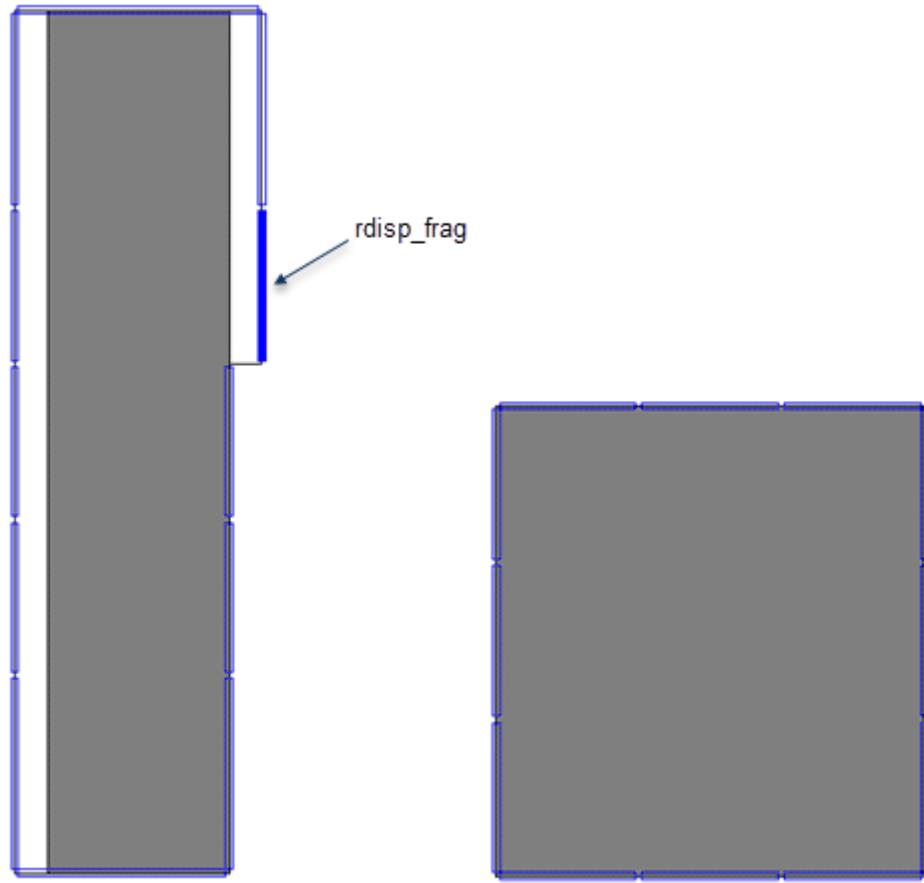
```
NEWTAG fragment ALL len < 0.01 corner1 convex len1 > 0.065 -out tag1
```

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 5-7. 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

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

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 second tag set containing all fragments from *tin* that are not part of *tag*.

Description

This command is a fast method to compute pre-existing line end and space-ends.

A line end is an edge that is \leq lineEndLength, has 2 convex corners, and the lengths of the adjacent edges are \geq lineEndLength. A space-end is the same except there are 2 concave corners.

In Calibre nmOPC, lineEndLength = 4*fragmentation unit (frgu). The term “frgu” is equivalent to “fragmentation base value”, as reported by the nmOPC and nmBIAS run log files.

Tcl Results

None

Performance

$O(N)$ where N is the size of *tin*

Examples

```
NEWTAG line_end all -out line_end
```

NEWTAG mrcViolation

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Outputs fragments that violate MRC.

Usage

NEWTAG mrcViolation *tin* -out *tag* [-outRest *tag2*]

Arguments

- ***tin***
The tag variable name for which to do the MRC check.
- **-out *tag***
Each fragment in the input tag that violated any MRC rule will be placed into the output tag.
- **-outRest *tag2***
An optional argument that creates a second tag set containing all fragments from ***tin*** that are MRC clean.

Description

This command outputs fragments that violate MRC when detected. This is typically used to tag all fragments that have pre-existing MRC violations.

Tcl Results

None

Performance

$O(N)$ where N is the number of input fragments

Examples

```
NEWTAG all input -out tAll
NEWTAG mrcViolation tAll -out tMRCViolating
```

NEWTAG neighbor

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Returns a set of fragments containing previous, next, or both neighbors.

Usage

```
NEWTAG neighbor {prev | next | both} tin [edge] [len constraint] [corner type]  
[dist constraint] [count n] [jog_tol max_jog] -out tag
```

Arguments

- **prev | next | both**

A required argument that specifies if the neighbor fragment returned is the previous fragment, the next fragment, or both neighbors of the fragments in *tin*.

- ***tin***

A required argument that identifies the name of the input tag set.

- **edge**

An optional keyword that causes the **len** and corner constraints to apply to the edge instead of the fragment.

- **len *constraint***

An optional argument that specifies the length constraint for the neighboring fragment or, if edge is specified, edge.

This argument is ignored if dist or count is specified.

- **corner *type***

An optional argument that specifies a corner constraint for the neighboring fragment or edge. The *type* is one of no_corner, convex, or concave.

- **dist *constraint***

An optional argument that tags the nearest neighbor fragments within the range specified by *constraint* and optionally by corner *type*. The length (len) constraint is ignored.

- **count *n***

An optional argument that extends tagging to *n* neighbors. The corner constraint is applied but the len constraint is ignored.

- **jog_tol *max_jog***

An optional argument that causes fragments less than or equal to max_jog to be ignored (not included in the output tag set).

- **-out *tag***

A required argument that specifies a name for the output tag set.

Description

Given a set of fragments, this command creates a new set which contains the previous, next, or both neighbors, subject to the specified constraints. The “previous” fragment is on the pt1 side of the initial fragment. The “next” fragment is on the pt2 side.

At least one of len, dist, or count must be specified.

Tcl Results

None

Performance

$O(N)$ where N is the size of *tin*.

Examples

In the following example, the second fragment from a line end is found and added to the secondfrag tag set.

```
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 property

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Copies DFM properties into a tag set for use by dense OPC commands.

Usage

```
NEWTAG property tag_in layer property [-dist value] [-resolution mode] [-inside]  
      -out tag_out [-outRest tag_rest]
```

Arguments

- *tag_in*

A required argument specifying a tag set. The fragments in the tag set are checked for nearby DFM property markers.

- *layer*

A required argument specifying the name of the layer containing the DFM properties. The layer must also appear in a [read_layer_properties](#) statement.

- *property*

A required argument specifying the name of the DFM property to read. Only numeric properties are supported. The name must match what is specified in [read_layer_properties](#), including case.

- -dist *value*

An optional argument specifying how far to search from the fragment for DFM property markers. The default is 0.001 microns.

- -resolution *mode*

An optional argument indicating how to handle the case of multiple markers. Only one value can be annotated per fragment in the tag set. The possible modes are

AVERAGE — This is the default behavior if the argument is not specified. The properties in the region are averaged to provide the value for the fragment in *tag_out*.

MAX — Uses the largest value.

MIN — Uses the smallest value.

- -inside

An optional argument that transfers the *layer* property to all fragments that interact with shapes on the layer. The -dist and -resolution arguments are ignored if specified.

- -out *tag_out*

A required argument specifying a name for the new tag set. The *tag_out* tag set is always annotated.

- **-outRest tag_rest**

An optional argument that tags those fragments of tag_in that did not have nearby DFM property markers as members of tag_rest.

Description

The NEWTAG property command creates an annotated tag set from a combination of a layer containing DFM properties and a set of tagged fragments. Although the layer may contain multiple properties, only one property can be searched for at a time.

Note

 To use NEWTAG property, you must first load the DFM properties with [read_layer_properties](#).

Examples

This shows reading in a DFM Property layer with property “bridge” and writing out the annotated tags along with their properties:

```
//The bridge_count layer has the DFM properties, and the bridgeFrags
// is the layer containing fragments from the annotated tag set.
propFrags =
    LITHO DENSEOPC M1 M1_sraf bridge_count FILE dopc.in MAP bridgeFrags

...
LITHO FILE dopc.in /**
    layer target
    layer sraf
    layer bridge_count

...
denseopc_options opc_opt {
    ...
    layer target      opc      mask_layer 0
    layer sraf        sraf     mask_layer 0
    layer bridge_count hidden mask_layer 0

    ...
    read_layer_properties bridge_count bridge
    NEWTAG all target -out allFrags
    NEWTAG property allFrags bridge_count bridge -out bFrags

    OUTPUT_SHAPE fragment bFrags bFrags -property bridge int max
}
...
setlayer bridgeFrags = denseopc target sraf bridge_count \
    OPTIONS opc_opt MAP bFrags property { bridge }
*/]
```

NEWTAG sadp

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Classifies fragments according to the relationship between associated cut masks and target layers.

Usage

Functional Usage

```
NEWTAG sadp functional -tag input_tag -layer opc_layer -cutMask cutMask_layer
    -target target_layer [-length microns] -functional_separation microns -out FC -out FNC
```

Critical Usage

```
NEWTAG sadp critical -tag input_tag -layer opc_layer -cutMask cutMask_layer
    -target target_layer [-length microns] -critical_separation microns -out CS
```

Arguments

- **-tag *input_tag***

A required argument that specifies the initial tag set to check for functional and critical fragments.

- **-layer *opc_layer***

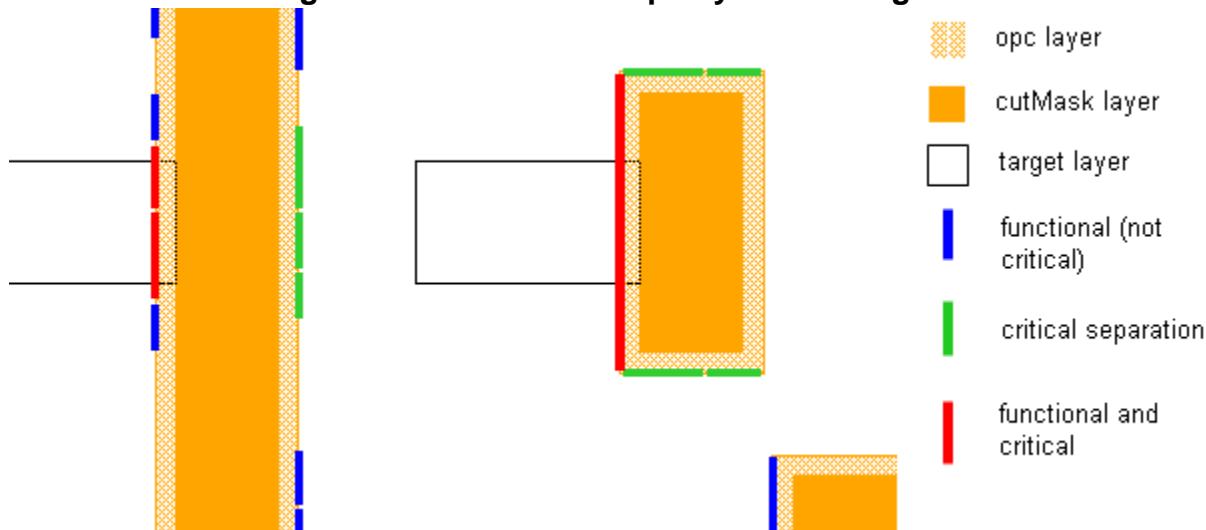
A required argument that specifies a layer containing *input_tag* fragments. The layer must be of type opc. You can specify either a layer name or number.

Typically this layer is derived by oversizing the cut mask layer.

- **-cutMask *cutMask_layer***

A required argument that specifies the layer name or number of the cut mask. Fragments on this layer are not tagged.

Figure 5-8. NEWTAG sadp Layers and Tag Sets



- **-target *target_layer***

A required argument that specifies the layer name or number of the layer containing the desired final shapes (after the mandrel is cut). Fragments on this layer are not tagged.

- **-length *microns***

An optional argument that specifies the maximum length of target edge to consider when tagging “functional not critical” and “critical separation” fragments. It is used to differentiate the line side from the line end. The default is 0.030 microns.

- **-functional_separation *microns***

- critical_separation *microns***

Specify only one.

A required argument that specifies the threshold distance between cut mask and target edges.

- functional_separation *microns*** — The maximum distance between a cut mask edge and a target edge for the corresponding fragment to be classified as “functional not critical.”

- critical_separation *microns*** — The minimum distance between a cut mask edge and a target edge for the corresponding fragment to be classified as “critical separation.”

- **-out *tag_name***

A required argument specifying a name for the produced tag set.

For the functional usage, the first -out argument names the “functional critical” (FC) tags and the second -out argument names the “functional not critical” (FNC) tags.

For the critical usage, specify only one -out argument. The name is applied to the “critical separation” tags.

Description

This optional tagging command classifies a fragment into the following tag sets:

- **Functional Critical** — The opc layer fragment encloses a cut mask edge that touches the target layer edge. The fragments are at a location where cut placement must be as accurate as possible.
- **Functional Not Critical** — The opc layer fragment encloses a cut mask edge that is within the functional separation distance from a target layer edge, and the target layer edge is less than or equal to -length. The fragments are at a location where a cut must be made, but the placement has some freedom.
- **Critical Separation** — The fragment encloses a cut mask edge that is within the critical separation distance from a target layer edge, and the target layer edge is less than -length. The fragment is at a location where separation between cut mask and target must be maintained.

Note

- ▀ In some cases, a fragment may be tagged as both functional critical and functional not critical; for example, when a sized-up OPC layer overlaps partially with the target layer as with the left red tags in [Figure 5-8](#). It is the responsibility of the setup file to handle these special cases.
-

Examples

This example shows the relevant code used to produce [Figure 5-8](#).

```
denseopc_options sadp {
    version 1

    layer cutMask.orig      hidden    mask_layer 0
    layer design_target     hidden    mask_layer 0
    layer cutMask.opc       opc      mask_layer 0

    image lithomodel_fast

    fragment_min 0.015
    fragment_max 0.090
    line_end_fragment off

    fragment_layer cutMask.opc {
        fragment_min 0.015
        fragment_corner convex convex length == 0.050 \
            length1 > 0.060 length 1 > 0.060 fragment 0.050
        fragment_max 0.090
    }

    NEWTAG all cutMask.opc -out allCutTags
    NEWTAG sadp functional -tag allCutTags -layer cutMask.opc \
        -cutMask cutMask.orig -target design_target \
        -functional_separation 0.020 -out FuncCrit -out FuncNotCrit
    NEWTAG sadp critical -tag allCutTags -layer cutMask.opc \
        -cutMask cutMask.orig -target design_target \
        -critical_separation 0.075 -out CritSep

    OUTPUT_SHAPE fragment FuncCrit criticalCutTags 0.001 0.001
    OUTPUT_SHAPE fragment FuncNotCrit funcNotCritTags 0.001 0.001
    OUTPUT_SHAPE fragment CritSep criticalSeparationTags 0.001 0.001
}
```

NEWTAG sequence

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Creates new tag set(s) holding fragments from a sequence of edges.

Usage

NEWTAG sequence *tin*

{**len *length_constraint* angle *angle_constraint***}... **len *length_constraint***
-out *tag* [-ordered] [-output_all] [-outRest *tag2*]

Arguments

- ***tin***
A required argument that identifies the initial tag set of fragments.
- **len *length_constraint* angle *angle_constraint***
A required set of arguments 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***
A required argument that names the output tag set.
- **-ordered**
An optional argument restricting the direction of the sequence.
If this optional flag is set, then the pattern match is performed only in the clockwise direction and not counter-clockwise direction. The default behavior is to check the pattern match in both directions.
- **-output_all**
An optional argument that creates extra output tag sets. The extra tag sets contain fragments belonging to the same edge in the sequence, but that are not part of ***tin***. These tag set names are formed by appending “_*n*” to the “-out” tag set name, where *n* is a positive integer representing edge number of the fragment (for example, fragments of edge 1 go into _1).
- **-outRest *tag2***
An optional argument that creates a tag set containing all fragments from ***tin*** that did not match the sequence.

Description

This command creates at least one new tag set of fragments that are part of a sequence of edges (multiple fragments per edge). All sequence fragments must be in the initial tag set ***tin*** for those fragments to be output.

Tcl Results

None

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 sites

[Calibre nmOPC Tcl Scripting Commands](#)

[Tagging Controls](#)

Outputs fragments that follow the specified constraint.

Usage

```
NEWTAG sites tin -imax pw_name constr [-type type] -out tag
NEWTAG sites tin -imin pw_name constr [-type type] -out tag
NEWTAG sites tin -contrast pw_name constr [-type type] -out tag
NEWTAG sites tin -has_site [-type type] -out tag
NEWTAG sites tin -epe pw_name constr -out tag
NEWTAG sites tin -printing pw_name -out tag
NEWTAG sites tin -nonprinting pw_name -out tag
NEWTAG sites tin -pvband {width constr | inner constr outer constr} -out tag
NEWTAG sites tin -wafer [-pw pw_name] [min_space constr] [min_width constr]
[min_space_3D constr] [min_width_3D constr] -out tag
NEWTAG sites tin -gradient [slope constr] [pvband_width constr] [nominal_epe constr]
-out tag
NEWTAG sites tin -epe_delta pw_name constr -out tag
NEWTAG sites tin -slope constr -out tag
```

Arguments

- ***tin***
Specifies the tag name to perform the check.
- **-type *type***
Specifies the site type: EPE, WAFER_ENCLOSE, WAFER_ENCLOSEDBY. The default is to create EPE sites.
- **-out *tag***
Specifies that each fragment in the input tag that meets the criteria will be placed into the output tag.
- **-imax *pw_name constr***
Specifies that the fragments' imax (maximum intensity) for process window condition *pw_name* is checked against the constraint (*constr*). No fragments will be output if the imax value is not available (if the fragment does not have sites, or they were not simulated at the last [OPC_ITERATION](#), or [OPC_ITERATION](#) -simulate_sites_epe was used).

Only fragments that have sites (of type *type*, if specified, otherwise of any type) are checked.

- **-imin *pw_name constr***

Specifies that the fragments' imin (minimum intensity) for process window condition ***pw_name*** is checked against the constraint (***constr***). No fragments will be output if imin is not available (if the fragment does not have sites, or they were not simulated at the last **OPC_ITERATION**, or **OPC_ITERATION -simulate_sites_epe** was used).

Only fragments that have sites (of type *type*, if specified, otherwise of any type) are checked.

- **-contrast *pw_name constr***

Specifies that the fragments' contrast for the process window condition ***pw_name*** is checked against the constraint ***constr***, where the contrast is defined as $((\text{imax} - \text{imin}) / (\text{imax} + \text{imin}))$. No fragments will be output if the contrast is not available (if the fragment does not have sites, or they were not simulated at the last **OPC_ITERATION**, or **OPC_ITERATION -simulate_sites_epe** was used).

Only fragments that have sites (of type *type*, if specified, otherwise of any type) are checked.

- **-has_site**

Specifies that fragments that have sites (of type *type*, if specified, otherwise of any type) are output.

- **-epe *pw_name const***

Specifies that the fragments' site EPE for process window condition ***pw_name*** is checked against the constraint (***const***). No fragments will be output if the EPE is not available (if the fragment does not have EPE sites, or they were not simulated at the last **OPC_ITERATION**). If the fragment has more than one site, all site EPEs must adhere to the constraint.

- **-printing *pw_name***

Specifies that fragments where least one sampling point is above the printing threshold for process window condition ***pw_name*** are output. Only EPE sites are checked.

- **-nonprinting *pw_name***

Specifies that fragments where at least one sampling point is below the printing threshold for process window condition ***pw_name*** are output. Only EPE sites are checked.

- **-pvband {*width constr* | *inner constr outer constr*}**

Specifies that fragments where the PV band adheres to the given constraint are output. No fragments will be output if the PV band is not available (if the fragment does not have EPE sites, or they were not simulated at the last **OPC_ITERATION**). At least one constraint has to be specified.

- **-wafer** [-pw *pw_name*] [*min_space constr*] [*min_width constr*] [*min_space_3D constr*] [*min_width_3D constr*] **-out tag**

Outputs fragments for which the minimum CD (across all regular process window conditions or 3D process window conditions if *min_width_3D* or *min_space_3D* are specified) obeys the given constraint, and for which the minimum space (across all process window or 3D process window conditions) obeys the given constraint. No fragments are output if one of the following is true:

- The width constraint is specified but the CD is not measured.
- The space constraint is specified but the space is not measured (possibly, because the fragment does not have EPE sites, if none of the sites are paired, or they were not simulated at the last [OPC_ITERATION](#)).

The -pw *pw_name* option is used to tag fragments that violate litho constraints for specific process conditions to apply corrections based on those conditions.

Note

 The -pw option cannot be used with the *min_space_3D* and *min_width_3D* options.

At least one constraint must be specified. 3D process window conditions are typically used in defining top loss resist models (see “[Resist Top Loss Correction](#)” on page 75 for further information).

Note

 3DSlice width and space tagging cannot be performed in the same tagging command with regular width and space tagging.

- **-gradient** [*slope constr*] [*pvband_width constr*] [*nominal_epe constr*]-**out tag**

Specifies that fragments that are output obey the constraint set by slope, *pvband_width*, or *nominal_epe* gradient. If the fragment has more than one site, only the gradient value at the first, last or middle site (depending on the measurement metric) is considered. The last [OPC_ITERATION](#) before this command must have been executed with at least one gradient optimizer option (with the *pvband_width* option, if *pvband_width* is specified here), otherwise no fragments will be output. The constraint values are unitless for *pvband_width* and *nominal_epe*, and units of I/sq um for slope. I/sq um for slope is defined as 1.0 / uu (user units²).

- **-epe_delta pw_name constr -out tag**

Calculates $\text{abs}(\text{max_epe} - \text{min_epe})$, calculated for all EPE sites. If it obeys the constraint, the fragment is output in the tag. This can be used to quantify how much “ringing” is on a fragment. For corner fragments, the epe_delta will typically be larger because of corner rounding.

- **-slope *constr* -out *tag***

Outputs all fragments that have the image slope less than the constraint. If the fragment has more than one site, only the slope value at the middle site or the site with maximum EPE or minimum EPE, depending on the measurement metric, are considered.

For 2012.2, the -slope option only works for PV band mode (PWOPC), where [algorithm](#) is automatically set to 2, and otherwise will not produce output.

The last [OPC_ITERATION](#) before this command must have been executed with at least one gradient optimizer option, otherwise no fragments will be output. The *constr* value is in units of (I/threshold * um). This is 1.0 / uu (user units) where uu is usually um. Internally, Calibre normalizes by threshold, scales by dbu, etc. This can be interpreted as:

unit of Intensity / uu (user units)

Note that the Intensity value is unitless.

Description

This command outputs fragments that adhere to the associated constraint, either by minimum intensity (imin), maximum intensity (imax), those that contain sites, those that are above or below a set printing threshold, or by PV band.

Tcl Results

None

Performance

O(N) where N is the size of the set *tin*.

NEWTAG topological

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Performs a topological operation on the specified input layers.

Usage

```
NEWTAG topological [standard] operation layer1 layer2 [-moved] [-tag tin] [-pct value]  
-out tag [-outRest tag2]
```

Arguments

- standard

An optional keyword that instructs **inside_edge** and **outside_edge** to not include partially inside or outside edges. This behavior is consistent with other uses of inside/outside edges in Calibre. This switch has no effect on operation other than **inside_edge** and **outside_edge**.

- **operation**

A required argument that specifies one of the following topological keywords:

- | | |
|---|---|
| <ul style="list-style-type: none">• inside_edge• outside_edge• touch_edge• coincident_edge• coincident_inside_edge• coincident_outside_edge• interact_edge | <ul style="list-style-type: none">• 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**

A required argument naming the layer that contains the tagged edges to act upon.

- **layer2**

A required argument identifying the layer that selects edges on **layer1**.

- **-moved**

An optional argument that specifies to use the current fragment positions when performing the operation. If **-moved** is not specified, measurements use the original positions on **layer1** and **layer2**.

- **-tag *tin***

An optional argument that restricts the parts of **layer2** to consider. Only those fragments within the *tin* tag set are used. By default, all fragments on the second layer are used.

This option is only used when *operation* is coincident_edge, and ignored for all other operations.

- -pct *value*

An optional argument specifying what percentage of the fragment must satisfy *operation* before being included in the tag set. The value must be between 0 and 100.

This option is only supported when operation is inside_edge or not_inside_edge. It cannot be used with the “standard” option.

- -out *tag*

Specifies the variable 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 *layer1* that did not satisfy the condition.

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 topological operations 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.

Tcl Results

None

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 that are partially inside the filter.

```
NEWTAG topological not_outside_edge L0 L1 -out set_1
NEWTAG topological outside_edge L0 L1 -out set_2
TAG and set_1 set_2 -out set_3; # set_3 contains partially inside/outside
```

NEWTAG vertical | horizontal | all_angle | 45_angle

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

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**

A required keyword specifying that the fragments match one of the following angle specifications: vertical, horizontal, all_angle, 45_angle.

- ***tin***

A required argument that identifies the initial tag set of fragments.

- **-out *tag***

A required argument that names the output tag set.

- **-outRest *tag2***

An optional argument that creates a second tag set containing all fragments from *tin* that did not match the angle.

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).

When -outRest is used, the command classifies all fragments in *tin* as *tag* or *tag2*.

Tcl Results

None

Performance

O(N) where N is the size of *tin*.

Rating = fast

NMBIAS_MEASURE_TAG

Calibre nmOPC Tcl Scripting Commands

Creates a tag set with numeric properties.

Usage

```
NMBIAS_MEASURE_TAG layer [reference_layer [reference_layer]] [-overlapLayer layer]  
[-tag tin] [-toTag to] [-unmoved] [-jog_ignore max_jog] [-projecting [length]]  
[-projecting_space [length]] [-ignore_singularity] [-default value]  
[-resolution {sum | min | max}] [-c2c [!=45 | ==45 | >=0]]  
-measurement_type type1 -dist range [metric] -aout tag1  
[-measurement_type type2 -dist range [metric] -aout tag2]  
[-measurement_type type3 -dist range [metric] -aout tag3 ]
```

Arguments

- ***layer***
A required argument that specifies the name of the layer to analyze.
- ***reference_layer* [*reference_layer*]**
An optional argument that specifies the name of one or two additional layers for space measurements.
All layer arguments must be specified before any keyword/value arguments.
- **-overlapLayer *layer***
An optional argument that specifies the name of a layer to be used for overlap and enclosure measurements.
- **-tag *tin***
An optional argument that specifies a tag set to consider. By default, all fragments on *layer* are analyzed.
- **-toTag *to***
An optional argument that specifies a tag set to measure against. Supported measurement types are projlen, space, and width. Only fragments in *to* from *layer* are used.
- **-unmoved**
An optional argument that performs the calculation using original fragment positions. If this argument is not specified, measurements are performed with the current position.
- **-jog_ignore *max_jog***
An optional argument that ignores jogs that do not exceed *max_jog* when computing length.

- -projecting [*length*]

An optional argument that includes projecting fragments greater than *length* microns for -measurement_type width, width2, or width3. If *length* is not specified, all projecting fragments are included.

- -projecting_space [*length*]

An optional argument that includes projecting fragments greater than *length* microns for -measurement_type space, space2, or space3. If *length* is not specified, all projecting fragments are included.

- -ignore_singularity

An optional argument that inhibits measurement of fragments projecting onto each other at a single point, also referred to as kissing corners. It requires either -projecting or -projecting_space.

- -default *value*

An optional argument that changes how fragments that do not satisfy criteria are handled.

By default, if a fragment does not return a value for a measurement (for example, it is out of range), it is not included in the -aout tag set. When -default is specified, the fragment *is* included in the -aout tag set, and annotated with *value*. All -measurement_type sets use the same -default setting.

Specify *value* as a floating point number.

- -resolution {sum | min | max}

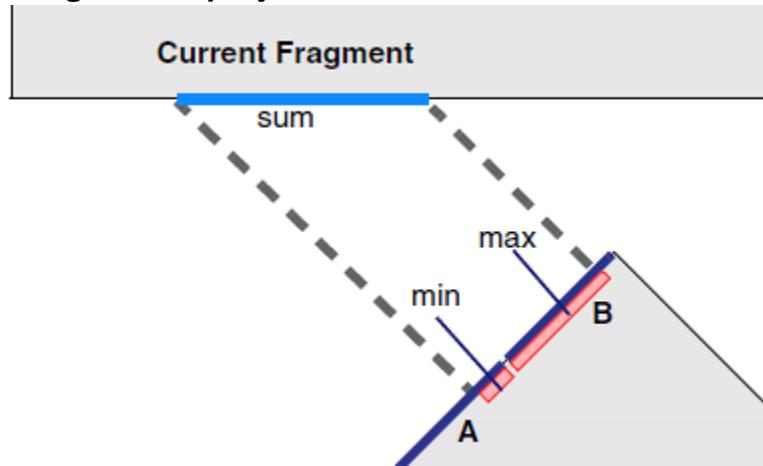
An optional argument only valid when *type1* is projlen, space, or width. It specifies which measurement to return when the current fragment projects onto multiple opposing fragments.

sum — The length of the current fragment that faces an opposing edge. This is the default for projlen, and cannot be specified for space or width.

min — The shortest of the fragment lengths in the projection. In [Figure 5-9](#), this is the small highlighted segment of fragment A. This is the default for space and width.

max — The longest of the fragment lengths in the projection. In the figure, this is the highlighted segment of fragment B.

Figure 5-9. projlen Measurement Resolution



- `-c2c [!=45 | ==45 | >=0]`

An optional argument that measures fragments when they are in a corner-to-corner configuration. This configuration is defined as fragments being parallel and not projecting, with additional qualifiers:

- **space**: both fragments are on convex 90-degree corners.
- **width**: both fragments are on concave 90-degree corners.
- **enclosure**: the *layer* fragment is on a convex 90-degree corner and the *overlapLayer* fragment is on a concave 90-degree corner.
- **enclosing**: the *overlapLayer* fragment is on a convex 90-degree corner and the *layer* fragment is on a concave 90-degree corner.

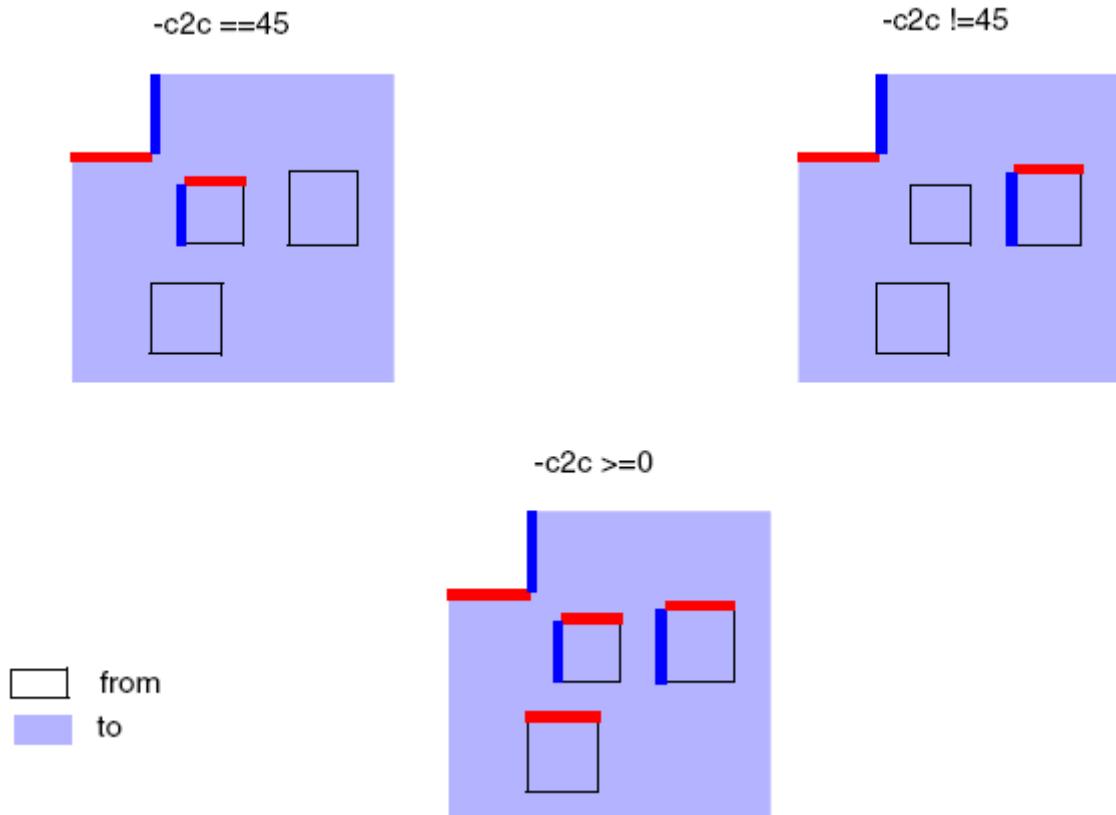
By default, projection is not allowed between the fragments or corners. You can permit projection using one of the angle constraints, visualized as being on a line drawn between the two corners relative to the x-axis.

`!=45` — This is the default behavior when no constraint is specified. Projection is not allowed, but the two corners do not need to have equal vertical and horizontal separation.

`==45` — Only corners that would describe a line segment at a 45-degree angle are measured.

`>=0` — Any orientation is allowed, but projection must be zero length.

Figure 5-10. Enclosing -c2c Constraints



- **-measurement_type type1**
 $[-\text{measurement_type } \text{type2} \dots]$
 $[-\text{measurement_type } \text{type3} \dots]$

A required argument that specifies the calculation to perform. The second and third -measurement_type arguments are optional, and can place restrictions on the first.

The type can be one of the following:

enclosing — Measured from the inside of the fragment on *layer* to the outside of a fragment on -overlapLayer.

enclosure — Measured from the outside of the fragment on *layer* to the inside of a fragment on -overlapLayer.

fraglength — Measured along the length of the current fragment.

length1 — Measured along the edge containing the current fragment. It can span multiple fragments. (There is no length2 keyword.)

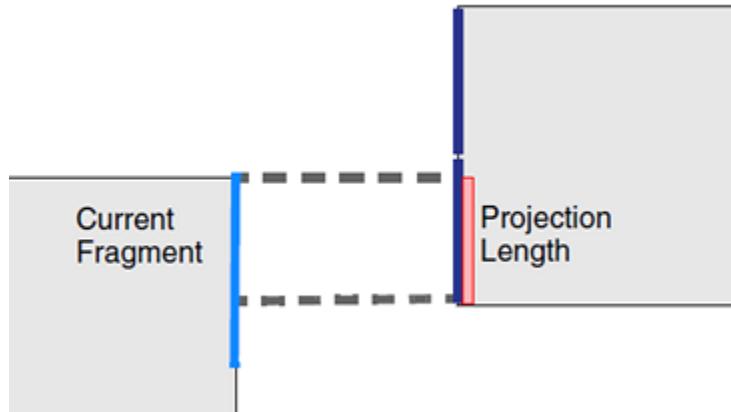
overlap — Measured from the inside of the fragment on *layer* to the inside of a fragment on -overlapLayer.

pitch — Returns **space + width**.

projlen — Measures the projection of fragments on *layer* or in the -tag tag set to a facing fragment or fragments. The facing fragments are from a reference layer or the

layer fragments in the -toTag tag set. (Fragments from other layers are ignored.) When both -tag and -toTag are specified shielding is ignored.

Figure 5-11. Projecting Length (projlen)



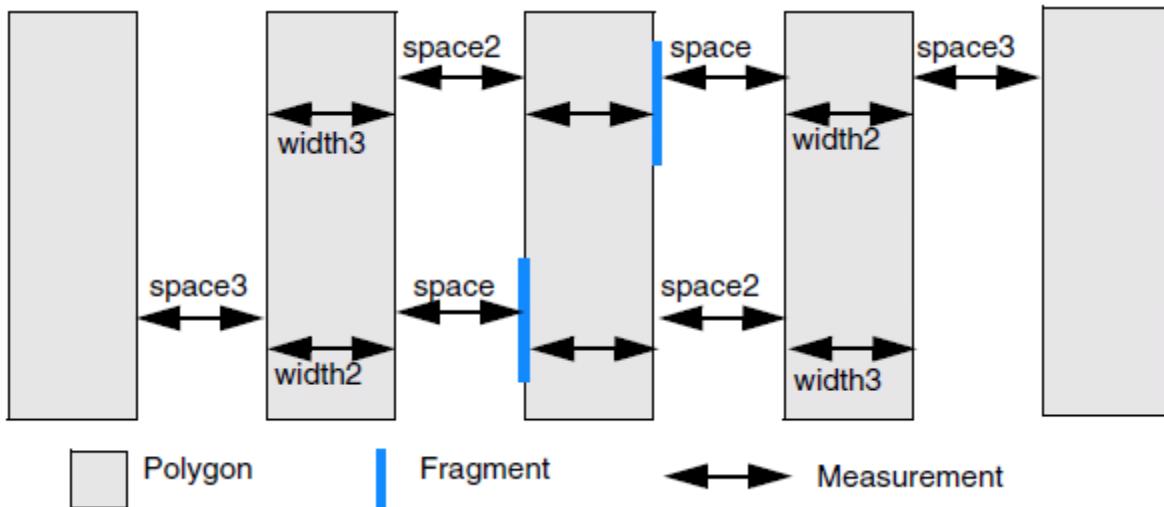
The projlen measurement type may optionally specify -resolution to determine which measurement is returned when the projection covers more than one facing fragment. (See [Figure 5-9](#) on page 523.)

sidespace — Measures the space for the edge to the side or sides of the current fragment. If the current fragment is not corner adjacent, sidespace is 0. If the current fragment is a line end and has two corners, sidespace is the smaller of the two values.

space, space2, or space3 — Measured from the outside of the fragment to the outside of a facing fragment.

width, width2, or width3 — Measured from the inside of the fragment to the inside of a fragment on the same polygon.

Figure 5-12. Space2, Space3, Width2, and Width3



type2 can be only space2 or width2, and requires that *type1* be width or space respectively.

type3 can be only space3 or width3, and requires that *type2* be width2 or space2, respectively.

- **-dist range**

A required argument that specifies how far to search for the opposing fragment. Values are in microns. The range applies to fragments in the *tin* tag set, or if -tag is not specified, all fragments on *layer*. It does not apply to fragments in the *to* tag set or reference layers.

The -dist settings when using two -measurement_type specifications can be different.

- **metric**

An optional argument that specifies the shape of the measurement region. Allowed values are opposite, opposite extended *distance*, and euclidean. The default is opposite.

If used with *type1* of “space” or “width” and -resolution, only the opposite metric is permitted.

- **-aout tag**

A required argument that specifies the name to give the tag set containing fragments from *layer* that satisfied the measurement. The fragment is also annotated with the corresponding measurement value in microns.

This argument can also be written as “-out -annotated *tag*.”

Description

This optional command classifies fragments within the input tag set (by default, all fragments on the layer) with a specified measurement type. If the measurement is not consistent along the fragment’s length, the smallest measurement is annotated to the fragment.

Annotations can be output to DFM RDBs and OASIS files. Annotations can also be read by [NEWTAG expression](#).

Examples

See also “[Example 3](#)” on page 557 in OUTPUT_SHAPE for the use of NMBIAS_MEASURE_TAG with OUTPUT_SHAPE and setlayer denseopc.

Example 1: Layer Only

This example measures the width of all polygons on the layer poly and places the property in the tag set “width_annotation.”

```
NMBIAS_MEASURE_TAG poly -measurement_type width dist 0.07 opposite \
-aout width_annotation
```

Example 2: Layer and Tag

This example measures the length of the edges for the fragments in the tag set “inTag” and places the measurement in the tag set “length1_annotation.” The tag set “inTag” must be created by a NEWTAG command before NMBIAS_MEASURE_TAG.

```
NMBIAS_MEASURE_TAG poly -tag intag -measurement_type length1 dist 0.035 \
-aout length1_annotation
```

Example 3: Two-Layer Measurement

This example measures the offset between a main feature and SRAF by doing a space measurement with two layers, poly and assist.

```
NMBIAS_MEASURE_TAG poly assist -measurement_type space dist 0.1 \
-aout offset
```

Related Topics

[NEWTAG external | internal | enclose | enclosing](#)

NYQUIST

[Calibre nmOPC Tcl Scripting Commands](#)

[Output Controls](#)

Returns the Nyquist value in either database units or user units.

Usage

NYQUIST {-dbu | -uu}

Arguments

- **-dbu**

Specifies database units.

- **-uu**

Specifies user units.

Description

This command returns the Nyquist value in either database units or user units.

Tcl Results

The Nyquist value in either database units or user units.

Performance

O(1)

Rating = fast

Examples

```
set nyq [NYQUIST -uu]
NEWTAG fragment all len <= [expr 3.5 * $nyq] corner1 convex corner2 \
    convex len1 > [expr 4 * $nyq] len2 > [expr 4 * $nyq] -out line_end
```

OPC_ITERATION

Calibre nmOPC Tcl Scripting Commands

OPC Controls

Instructs Calibre nmOPC to perform one or more OPC iterations.

Usage

Syntax 1:

```
OPC_ITERATION [count] [-registers register_list] [pvband_pwopc_options]  
[matrix_retargeting_options] [gradient_optimizer_options] [-coarse num_of_simulations]  
[-simulate_sites | -simulate_sites_epe] [-simulate_no_opc_pw]
```

Syntax 2:

```
OPC_ITERATION -set_epe [pvband_pwopc_options] [matrix_retargeting_options]  
[gradient_optimizer_options] [-coarse num_of_simulations]  
[-simulate_sites | -simulate_sites_epe] [-simulate_no_opc_pw]
```

Syntax 3:

OPC_ITERATION -apply_feedback

where *pvband_pwopc_options* (also known as pvband_pwopc mode) are:

-PW [exclude *exclude_tag*]
[nominal_epe_limit *nominal_epe_limit_tag tol_inner tol_outer*]...
[pvband_tolerance *pvband_tolerance_tag tol_inner tol_outer*]...
[corner_ignore *corner_ignore_tag value*]...
[min_width *min_width_tag width*]...
[min_space *min_space_tag space*]...
[min_space_interpattern *min_space_interpattern_tag space*]...
[min_overlay *min_overlay_tag overlay*]...
[min_width_3D *min_width_3D_tag width_3D*]...
[min_space_3D *min_space_3D_tag space_3D*]...
[min_width_pw *min_width_pw_name min_width_pw_tag width*]...
[min_space_pw *min_space_pw_name min_space_pw_tag space*]...
[exclude_pw *excluded_pw_name*]...

matrix_retargeting_options are:

```
[-schedule {retarget riter_1 riter_2 ... [pw iter_1 iter_2 ...]}]  
[-rt_tag tagname]...  
[-target_bias_limit tagname lower_limit upper_limit]...  
[-output_overconstrained iter iter_num output_layer_name]...  
[-output_violated iter iter_num output_layer_name]...
```

gradient_optimizer_options are:

```
[opt_slope opt_slope_tag min_slope]...
```

Arguments

- **-set_epe**
Syntax 2 only. Triggers 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**
Syntax 3 only. 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.
- *count*
Enables the “iteration count” mode of this command, specifying a number of iterations to be performed. If *count* is unspecified, OPC_ITERATION will default to *count* = 0.
- **-registers register_list**
An optional keyword that saves the computed images into the registers specified by *register_list*.
- **-coarse num_of_simulations**
An optional argument that specifies the first *num_of_simulations* simulations use both a coarse grid specified by “[imagegrid](#) ... coarse” for the resist and a coarse grid specified by [rasterizer_upsample_factor](#) for the mask. (See the *Calibre OPCverify User’s and Reference Manual* for [rasterizer_upsample_factor](#) and [imagegrid](#) details.)
- **-simulate_sites**
If this is specified, sites created with [SITES_CREATE](#) are simulated. This is not allowed in “iteration count” mode, unless *pvband_pwopc_options* are used.
- **-simulate_sites_epe**
If this is specified, sites created with [SITES_CREATE](#) are simulated, but only their EPE or printing status (or both) is available. If enabled, this will prevent some scripting commands such as [NEWTAG sites -imin](#) from working. This is not allowed in “iteration count” mode, unless *pvband_pwopc_options* are used.
- **-simulate_no_opc_pw**
If this is specified, process window conditions that are marked as “no_opc” in the [pw_condition](#) definition are also simulated (but not used for OPC). This is used with producing intensities with the “no_opc” [pw_condition](#) that are used by [sraf_print_avoidance](#) after the call to OPC_ITERATION.

Pvband_pwopc Options

For all numeric values for *pvband_pwopc options*, the values can be expressed either as a number or read from an annotated tag set using the following syntax:

```
property(annotated_tag_set) :default(value)
```

For example, both of these are acceptable:

```
nominal_epe_limit all_frags \
    property(tol_inner):default(-0.003) \
    property(tol_outer):default(0.003)

nominal_epe_limit all_frags -0.003 0.003
```

The default value is used when a fragment is not in the annotated tag set. Setting a default is optional, but strongly recommended.

- exclude *exclude_tag*

Specifies that PV band for fragments belonging to this tag will be ignored, and only nominal EPE will be used.

- nominal_epe_limit *nominal_epe_limit_tag tol_inner tol_outer*

Specifies that PV band for fragments belonging to this tag will be considered, but Calibre nmOPC attempts to prevent the nominal EPE from exceeding the *tol_inner tol_outer* tolerance values.

- pvband_tolerance *pvband_tolerance_tag tol_inner tol_outer*

Specifies that the PV band for fragments belonging to this tag will be considered only if it is outside the tolerance specified using the *tol_inner tol_outer* values.

- corner_ignore *corner_ignore_tag value*

Specifies that PV band measurements closer than *value* to a corner will be ignored.

- min_width *min_width_tag width*

Specifies that OPC will try to ensure that the contour width across process window conditions for this tag will be at least be *width* value.

- min_space *min_space_tag space*

Specifies that OPC will try to ensure that the contour spacing across process window conditions for this tag will be at least *space* value.

- min_width_3D *min_width_3D_tag width_3D*

Specifies a minimum width for a 3D process window (top loss process window). See “[Resist Top Loss Correction](#)” on page 75 for further information.

- min_space_3D *min_space_3D_tag space_3D*

Specifies a minimum space for a 3D process window (top loss process window). See “[Resist Top Loss Correction](#)” on page 75 for further information.

- min_width_pw *pw_name min_width_pw_tag width*

Establishes the minimum contour width for the named tag and process window condition. The *pw_name* should not refer to a “spa” or “no_opc” condition.

- **min_space_pw *pw_name* *min_space_pw_tag* *space***
Specifies that Calibre nmOPC attempt to ensure that the contour spacing (for the process window condition specified by *pw_name*) for this tag will be at least *space*. The *pw_name* should not refer to a “spa” or “no_opc” condition.
- **min_space_interpattern *min_space_interpattern_tag* *space***
Specifies that Calibre nmOPC attempt to ensure that the contour spacing across process window conditions for this tag will be at least *space* size for inter-pattern spacing checks. The min_space requirement is ignored if min_space_interpattern is specified. The min_space_interpattern constraints override the min_space_pw constraints for inter-pattern spacing.
- **min_overlay *min_overlay_tag* *overlay***
Specifies that OPC will try to ensure that the contour overlay across process window conditions for this tag will be at least be of *overlay* value.
- **exclude_pw *excluded_pw_name***
Specifies that Calibre nmOPC will not simulate the specified process window and use it for corrections.

Matrix Retargeting Options

- **-schedule {retarget *riter_1* *riter_2*... [pw *iter_1* *iter_2*...]}**
An optional keyword that enables matrix retargeting and specifies the how iterations are treated.
 - The retarget keyword specifies during which OPC iterations matrix retargeting will be performed.
 - If the pw keyword is specified, the iterations specified (*iter_1 iter_2...*) are PWOPC iterations and the rest are treated as nominal iterations that override the -PW keyword. If the pw is not specified, all OPC iterations in the OPC_ITERATION command are treated as PWOPC iterations.
- **-rt_tag *tagname***
An optional keyword that specifies which tag set (and related neighboring fragments) should be considered for matrix retargeting. If not present, all fragments are considered for matrix retargeting.
- **-target_bias_limit *tagname* *lower_limit* *upper_limit***
An optional keyword that specifies lower and upper bias limits for a specific tag set in user units. The specified range applies to the specific tag set and overrides the global limit that can be specified with the max_target_bias keyword in the retargeting_options block. The range is not allowed to exceed the global range specified with max_target_bias keyword.
- **-output_overconstrained iter *iter_num* *output_layer_name***
An optional keyword that, if specified for an iteration (*iter_num*), outputs markers for fragments with unresolved constraints to specified layer (*output_layer_name*).

For pair constraints, markers are placed in-between the fragment pairs, with a width that gives a hint about how much violation is seen. For enclosure constraints, markers are centered at the fragments, with a width that gives a hint similar to pair constraints.

- `-output_violated iter iter_num output_layer_name`

An optional keyword that, if specified for an iteration (*iter_num*), fragments that fail to satisfy PV band-PWOPC constraints and are inputs to matrix retargeting are output to the specified layer (*output_layer_name*).

Gradient Optimizer Options

- `opt_slope opt_slope_tag min_slope`

Specifies that fragments belonging to this tag will be optimized for slope, if the slope is below the given *min_slope* (in units/microns). In this case, *min_slope* is defined in units of 1.0 / uu (user units). As this is a gradient optimizer, the gradient must be calculated. This is a very computation-heavy operation, which may lead the OPC_ITERATION runtime to increase by up to 100%.

Description

This command performs a “standard” OPC iteration as defined by all setup file user settings. All options in the setup file which apply to iterations will be obeyed:

- `pw_condition`
- `active_layer`
- `retarget_layer`
- contact layer type (in the layer command)
- `check_movement`
- `feedback`
- `epe_spacing`
- `mrc_rule`
- `max_iter_movement`

The following modes are available 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.

- PV band mode (see the section “[PV Band Mode](#)” on page 535 for a description).
- 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 feedback values after simulation and after EPEs are calculated, but before the feedback is applied.

Site simulation is done by default if [algorithm 2](#) is set.

Note

 The PV band mode uses sites to determine EPEs and is enabled automatically if at least one of the following is true:

- All PW conditions are declared with the `pvband` keyword (in [pw_condition](#)), otherwise an error message is issued.
 - An `OPC_ITERATION` command in the tagging script has the `-PW` parameter.
 - An `opt_slope` keyword is present.
-

Note that if you specify both `max_iterations` and `OPC_ITERATION` in the setup file, the number of iterations specified in `OPC_ITERATION` will take precedence. For example:

```
max_iterations 8 nopw 3
OPC_ITERATION 7
```

In this example, Calibre nmOPC will perform three non-process window iterations, followed by four more process window iterations, for a total of seven iterations.

An alternative command, `OPC_XMEEF_ITERATION`, is available if any of the following circumstances occur:

- Ripples occur on metal lines
- Out-of-specification post-OPC EPE occur for contact and poly layers
- If you are having difficulties tuning feedback for certain tag sets. High MEEF on a fragment usually results in difficulties tuning feedback and bad convergence.

Constraints for WIDTH/SPACE

Contour width and spacing are calculated by pairing sites across a feature. Pairing is done only for sites that are properly oriented (the angle between them is smaller than some internal constant).

Pairing is done only for sites that are not too far apart (their midpoints should be closer than the specified width + Nyquist for CD pairs, and closer than the specified space + Nyquist for space pairs).

- Width pairing will be done to EPE sites belonging to fragments on the same patterning layer
- Space pairing will be done to EPE sites belonging to fragments on the same, or different patterning layers
- Site pairs can be output using [SITES_DUMP](#) -type PAIR

For every fragment that belongs to a `min_width_tag`, `min_space_tag`, or `min_overlay_tag`, the process window EPEs are ignored, and the nominal EPE is considered only if the contour CD/space is within the given tolerance (the width/space has a higher priority than the process window tolerance).

The maximum site pairing distances for `min_width` and `min_space` are as follows:

$$\text{maximum_width_pair_dist} = \text{min_width_tol} + \min(3 * \text{Nyquist}, \max(2 * \text{Nyquist}, -\text{nominal_tol_inner}))$$

$$\text{maximum_space_pair_dist} = \text{min_spacing_tol} + \min(3 * \text{Nyquist}, \max(2 * \text{Nyquist}, +\text{nominal_tol_outer}))$$

where:

- `min_width_tol` is specified by min_width `min_width_tagwidth`
- `min_space_tol` is specified by min_space `min_space_tagspace`
- `nominal_tol_inner` and `nominal_tol_outer` are the nominal EPE tolerances specified by the respective **OPC_ITERATION** -PW options:

```
OPC_ITERATION -PW ... [nominal_epe_limit nominal_epe_limit_tag tol_inner tol_outer]
```

PV Band Mode

This mode is both for PV Band PWOPC and cases using [algorithm 2](#). This mode uses sites to determine EPEs and is enabled automatically if at least one of the following is true:

- A [pw_condition](#) is declared with the “pvband” keyword
- An **OPC_ITERATION** in the tagscript has the -PW parameter
- An `opt_slope` is present

If this mode is enabled, default sites are created. To alter the automatically-created sites, delete the automatically created ones first using [SITES_DELETE](#). The constraint parameters for this mode behaves as follows:

- If this mode is enabled and [algorithm](#) is explicitly specified in the transcript, the algorithm setting will be ignored, and a warning will be output in the transcript.
- The first priority is to bound to nominal EPE, if nominal EPE limits exist (set by [nominal_epe_limit](#)). By default, nominal EPE limits are set to very loose values.
- The second priority is to make sure that the minimum CD/space across process window conditions is obeyed (or for individual process window conditions, if [min_width_pw](#) or [min_space_pw](#) are used). By default, there is no limit on the CD/space. The second priority is also to obey the [wafer_enclose](#) or [wafer_enclosedby](#) constraints, across all process window conditions.
- Third priority is to drive the PV band to be within the given tolerance (set by [pvband_tolerance](#)). By default, the tolerance is zero, so the middle of the PV band will be driven to be on the OPC target.
- Fourth priority is to improve the slope (set by [min_slope](#)) or PV band width (if specified, and out of the specified tolerance).

If some constraints cannot be obeyed, then all constraints will be violated by the same amount.

For example, if there are two PW conditions + nominal, and the PV band tolerance is set to 0, then [pw1_epe](#) will be exactly equal to [-pw2_epe](#).

This assumes that:

$$\text{abs}(\text{nominal_epe}) \leq \max(\text{abs}(\text{pw1_epe}), \text{abs}(\text{pw2_epe}))$$

if OPC converges properly. The pseudo-EPE function is designed to obey the constraints as previously stated, and optimize nominal EPE (always), image slope (if specified) and PV band width (if specified).

If [opt_slope](#) is specified, the nominal EPE is only optimized if the slope exceeds the specified [min_slope](#) and the [pvband](#) width is smaller than the specified [max_pvband_width](#).

Application points:

- By default, [pvband_tolerances](#) are set such that the PV band is optimized to be centered around the target. If this is not the case, the [pvband_tolerance](#) constraint should be specified accordingly, or relaxed so it does not negatively affect other constraints.
- For a specific keyword in [pvband_pwopc_options](#), if a fragment is specified multiple times with the same keyword (commonly when tag sets overlap), then the last

specification is used. For example, if fragment A appears in both tag1 and tag2, in the following command:

```
OPC_ITERATION 5 -PW min_space tag1 0.030 min_space tag2 0.028 ...
```

the final min_space spec for fragment A is 0.028.

Retargeting in PWOPC

Process Window OPC works using the concept of effective EPE. This means that:

- EPE is calculated using a nominal process model (the nominal EPE).
- Off-target models provide additional information about process window variations.
- The additional information is used to modify the nominal EPE.

With this in mind, there are different methods used by Calibre nmOPC to improve accuracy and run time in determining optimal EPE for edge movement and correction.

- **Process Window Retargeting** — With the PW_RETARGET function, you can translate the effective EPE into target adjustments necessary to resolve process window violations. PW_RETARGET was designed to provide a new adjusted target that allows you to switch results from process window iterations and nominal iterations (selecting the more accurate EPE result) on the new target
- **Matrix Retargeting** — Matrix retargeting is an evolution of process window retargeting, using a matrix solver instead (rather than manually choosing from PW conditions) to resolve multiple conflicting wafer constraints for improved accuracy, visualization, and run time. In Matrix retargeting:
 - If matrix retargeting is requested (for instance, -schedule), matrix retargeting is performed and OPC_ITERATION must be in count mode. Retargeting iteration numbering starts from, and cannot exceed the total number of iterations in this OPC_ITERATION command. Global retargeting control parameters are specified with the [retargeting_options](#) command.
 - Use the following keywords to generate marker and fragment layers: [output_overconstrained](#) and [output_violated](#).
 - Each matrix retargeting iteration may consume up to half of a nominal simulation. It is recommended that you engage matrix retargeting when significant retargeting is needed. You can use a few matrix retargeting iterations at a later stage, to allow a few more OPC iterations to converge to new retargeted positions.
 - Generate incremental retargeting shapes using [OUTPUT_TARGET_CONTROL](#)

Tcl Results

None

Performance

This command invokes one or more full tile simulations. It is best to call this function very rarely.

Rating = very slow

Examples

Example 1

The following example shows four iterations:

```
for {set i 0} {$i < 4} {incr i} {
    OPC_ITERATION
    FRAGMENT_MOVE
}
```

Example 2

The following example is a simpler method of iterating four times:

```
OPC_ITERATION 4
```

Example 3

The following is an example of fine-grained control:

```
OPC_ITERATION -set_epe
NEWTAG ...
FEEDBACK ...
OPC_ITERATION -apply_feedback
FRAGMENT_MOVE
```

Example 4

The following example shows three “nominal” iterations and four PV band iterations excluding gates:

```
OPC_ITERATION 3      # only does nominal simulations in pvbnd_pwopc mode
OPC_ITERATION 4 -PW exclude gates nominal_epe_limit -0.02 0.02
```

Example 5

The following is a setup file that uses OPC_ITERATION along with [sraf_print_avoidance](#) and [pw_conditions](#):

```
denseopc_options DOPC {
    ...
    background clear
    layer TARGET      opc      atten 0.06
    layer MBSRAF     sraf      atten 0.06
    ...

    pw_condition spa optical OPT dose 1.15 resist RESIST pvbnd spa
    ...
}
```

```

fragment_layer CONTACT_VSB_PRE {
fragment_min 0.03
fragment_max 0.12
}
...
mrc_rule area sraf_layer {
use areal
square use area2
}
...
sraf_sites_create

#OPC iteration
    OPC_ITERATION 5

#OPC and SRAF print avoidance
    For {set i 0} {$i < 5} {incr i} {
        OPC_ITERATION 1
        sraf_print_avoidance -step 0.001
    }

    OPC_ITERATION 3
}

```

Example 6

The following is an example of Matrix Retargeting (the iteration numbering starts from 1):

```

OPC_ITERATION 6 -PW nominal_epe_limit all frags -0.005 0.005 \
    min_width all frags 0.030 \
    min_space all frags 0.030 \
    pvband_tolerance all frags -0.100 0.100 \
    -schedule { pw 1 3 5 retarget 1 3 5 } \
    -output_overconstrained iter 1 overconstrained_1 \
    -output_overconstrained iter 3 overconstrained_3

```

Example 7

The following example shows using annotated tag sets for arguments. The annotated tag sets are constructed with [NEWTAG expression](#).

```

NEWTAG expression all frags { has_tag(tag1) ? -0.010 \
    : (has_tag(tag2) ? -0.010 : -0.005) } \
    -aout nominal_epe_limit_inner_tag
NEWTAG expression all frags { has_tag(tag1) ? 0.010 \
    : (has_tag(tag2) ? 0.010 : 0.005) } \
    -aout nominal_epe_limit_outer_tag

NEWTAG expression tag1 { -0.030 } -aout pvband_tolerance_limit_inner_tag
NEWTAG expression tag1 { 0.030 } -aout pvband_tolerance_limit_outer_tag

```

```
OPC_ITERATION 4 -PW \
    nominal_epe_limit \
        property(nominal_epe_limit_inner_tag) \
        property(nominal_epe_limit_outer_tag) \
    pvband_tolerance \
        property(pvband_tolerance_limit_inner_tag):default(-0.020) \
        property(pvband_tolerance_limit_outer_tag):default(0.020) \
    min_width all_frags 0.034 \
    min_space all_frags 0.037 \
    min_space tag4 0.034 \
    min_width tag4 0.035
```

OPC_XMEEF_ITERATION

Calibre nmOPC Tcl Scripting Commands

OPC Controls

Instructs Calibre nmOPC to perform one or more Matrix OPC iterations, using an alternative method for faster calculations of XMEEFs.

Usage

```
OPC_XMEEF_ITERATION [opcLayer layer] [[iterations] count] [coarse num_simulations]  
[iterativeXmeef {true | false}] [recycleXmeef {true | false}] [xmeefRadius dist]  
[pvband_pwopc_matrix_retarget_and_gradient_opt_options]
```

Arguments

- opcLayer *layer*

An optional argument that specifies a single layer on which to perform OPC. By default, OPC is performed on the layer of type opc on pattern 0.

This argument does not allow for simulating multiple layers on the same pattern. Only one layer of type opc per pattern can be simulated. There is no method to simultaneously simulate multiple layers on the same pattern.

- [iterations] *count*

An optional argument that specifies the number of iterations to perform.

- coarse *num_simulations*

An optional argument that specifies the first *num_of_simulations* simulations use both a coarse grid specified by “[imagegrid ... coarse](#)” for the resist and a coarse grid specified by [rasterizer_upsample_factor](#) for the mask. (See the *Calibre OPCverify User's and Reference Manual* for [rasterizer_upsample_factor](#) and [imagegrid](#) details.)

This argument can also be specified as -coarse *num_simulations*.

- iterativeXmeef {true | false}

An optional argument that, if set to true, a two-step process XMEEF computation process called Matrix OPC is enabled for each iteration:

- a. Calibre nmOPC performs a fast computation of XMEEF for all fragments on the full chip.
- b. An XMEEF mathematical matrix (see the following figure) is constructed and utilized as a solving method. As this method is fast, every fragment on a full chip layout can use this matrix solver.

Figure 5-13. XMEEFs Matrix

$$\begin{matrix} \text{xmeef matrix} & \text{fragment movements} & \text{Fragment EPEs} \\ \left[\begin{array}{ccc} \frac{dEpe_1}{df_1} & \dots & \frac{dEpe_1}{df_n} \\ \vdots & & \vdots \\ \frac{dEpe_i}{df_1} & \dots & \frac{dEpe_i}{df_n} \\ \vdots & & \vdots \\ \frac{dEpe_n}{df_1} & \dots & \frac{dEpe_n}{df_n} \end{array} \right] \left[\begin{array}{c} f_1 \\ \vdots \\ f_n \end{array} \right] = \left[\begin{array}{c} -Epe_1 \\ \vdots \\ -Epe_n \end{array} \right] \end{matrix}$$

If set to false, the two-step computation process is performed for the first iteration only. For subsequent iterations, only the second computation step is performed. This is the default setting.

- recycleXmeef {true | false}

An optional argument that controls whether the fast XMEEF computations (step [a](#) in `iterativeXmeef`) can be reused when “`iterativeXmeef true`” is set. This option is ignored when `iterativeXmeef` is not specified or is set to false.

`true` — The default when `iterativeXmef` is set. The fast XMEEF computations are re-used in the iteration provided fragmentation has not changed. This can improve performance. (If the fragmentation has changed between iterations, the XMEEFs are recalculated.)

false — Always recalculate XMEEFs for each iteration.

- xmeefRadius *dist*

An optional argument that is used to identify which neighbor fragments that should have XMEEFs computed against a given fragment. The neighbors are found by searching at the specified distance away from the center of the target fragment. The default value is 5*Nyquist value.

- *pvband_pwopc_matrix_retarget_and_gradient_opt_options*

An optional set of keywords used to define PV Band PWOPC, matrix retargeting, and gradient optimization. These include the following:

-PW [exclude *exclude_tag*]

[nominal_epe_limit nominal_epe_limit_tag tol_inner tol_outer]....

[pvband tolerance *pvband_tolerance_tag* *tol_inner* *tol_outer*]...

[corner ignore *corner_ignore_tag* value] ...

[min width *min_width_tag width*]...

[min space *min_space_tag* space]...

[min space interpattern *min_space*]

```
[min_overlay min_overlay_tag overlay]...
[min_width_3D min_width_3D_tag width_3D]...
[min_space_3D min_space_3D_tag space_3D]...
[min_width_pw pw_name min_width_pw_tag width]...
[min_space_pw pw_name min_space_pw_tag space]...
[exclude_pw excluded_pw_name]...
[-schedule {retarget riter_1 riter_2 ... [pw iter_1 iter_2 ...]}]
[-rt_tag tagname]...
[-output_overconstrained iter iter_num output_layer_name]...
[-output_violated iter iter_num output_layer_name]...
[opt_slope opt_slope_tag min_slope]...
```

These options perform the same function as used in [OPC_ITERATION](#).

Description

The OPC_XMEEF_ITERATION command instructs Calibre nmOPC to perform OPC iterations, but using a process called Matrix OPC to make faster calculations of XMEEFs to speed convergence for the current fragment. This command is an alternative to [OPC_ITERATION](#) if any of the following circumstances occur:

- Ripples occur on metal lines
- Out-of-specification post-OPC EPE occur for contact and poly layers
- If you are having difficulties tuning feedback for certain tag sets. High MEEF on a fragment usually results in difficulties tuning feedback and bad convergence.

Note

 OPC_XMEEF_ITERATION requires that the [algorithm](#) command be set to 2.

OPC_XMEEF_ITERATION can operate on every fragment on a chip (not just as a local solver).

What is MEEF?

The Mask Error Enhancement Factor (MEEF) represents the magnification of mask errors in the printed image on the wafer as a result of nonlinear factors in the pattern transfer process. This can lead to increased proximity effects and bias problems.

When a mask is made with a certain deviation compared to the designed dimension, by using the MEEF value, you can calculate and know whether the printed wafer has the potential to pinch or bridge. In high-MEEF regions of the layout, movement of neighbor fragments can slow

convergence of the current fragment. In the worst case scenario, the fragment cannot converge without taking neighbor movement into account properly.

A common way of measuring MEEF is by performing a CTR or Resist contour simulation on the original OPC mask and a new OPC mask by sizing up the original mask by 1 nm (in other words, every fragment is moved outward by 1 nm).

From location to location, you can measure the change in contour and divide by a 1 nm mask change to derive the MEEF value. For example, if the contour changed by 3 nm, then the MEEF at this location is $3\text{ nm}/1\text{ nm} = 3$. MEEF, as it is a mathematical factor, is essentially unitless.

What is XMEEF?

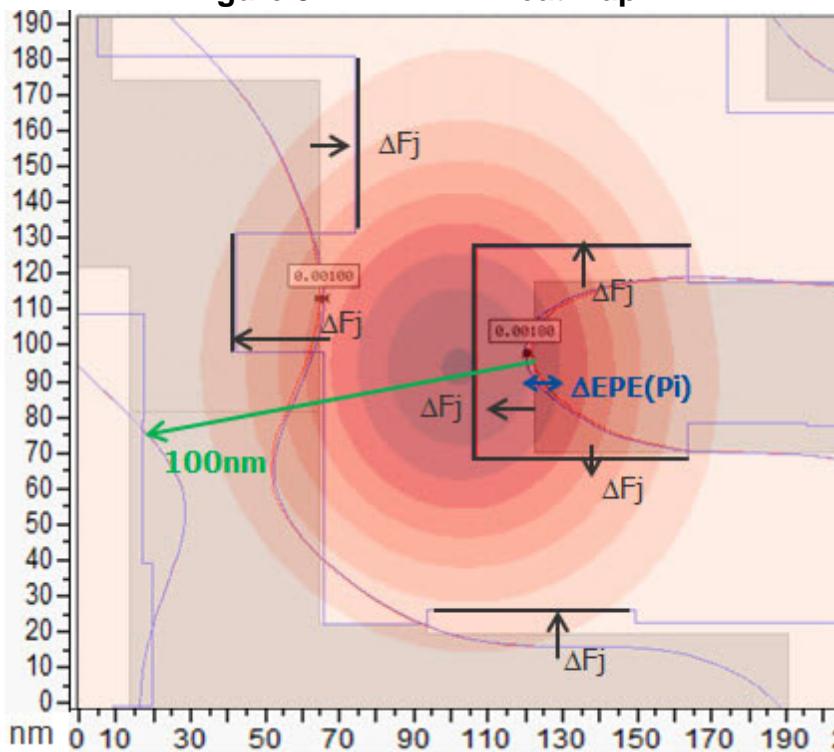
Cross MEEFs, also known as XMEEFs, are a representation of one OPC fragment's movement to EPE changes induced on its surrounding neighboring fragments. This is essentially a fragment pair; one fragment is the initiator that is moved, and another fragment is where the effects on the movement are observed. Like MEEFs, XMEEFs are unitless parameters computed using EPE/movement.

As computing XMEEFs by moving each fragment can be time consuming, OPC_XMEEF_ITERATION uses an alternative calculation process called Matrix OPC specifically for XMEEFs without the need to physically move fragments.

For example, one method to measure XMEEF is to move a single target fragment out by 1 nm and observe the EPE change on all surrounding fragments including itself.

For each fragment on the chip, the impact of every neighbor fragment's movement (within a 100 nm radius) is computed. For example, [Figure 5-14](#) shows a graphical representation of the EPE impact of XMEEFs for a line end fragment as a “heat map.” Note how the strongest interactions are within a 100 nm distance where XMEEF interactions are computed.

Figure 5-14. XMEEF Heat Map



Examples

The following examples illustrates how to upgrade an OPC_ITERATION call to OPC_XMEEF_ITERATION.

Basic Conversion

The following examples show how to convert OPC_ITERATION calls to OPC_XMEEF_ITERATION. Note the following:

- It is recommended that you run a couple of regular OPC iterations prior to using OPC_XMEEF_ITERATION
- OPC_XMEEF_ITERATION needs fewer iteration to converge

Example 1: Single Call

```
OPC_ITERATION 10
```

This call converts to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 5
```

Example 2: Multiple Calls

```
OPC_ITERATION 3
...
OPC_ITERATION 4
...
OPC_ITERATION 3
```

These calls convert to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 5
```

Example 3: Multiple PWOPC Calls

If PWOPC is used after a nominal iteration, there is no need to convert the OPC_ITERATION of a nominal portion of the run into OPC_XMEEF_ITERATION. The reason for this is the nominal OPC and PWOPC iterations often have conflicting convergence targets.

```
OPC_ITERATION 3
...
OPC_ITERATION 7 -PW ...
```

These calls convert to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 5 -PW ...
```

Example 4: Using with sraf_print_avoidance

```
OPC_ITERATION 3
set i 0
while {$i < 3} {
    OPC_ITERATION 1
    OPC_ITERATION -simulate_sites -set_epc -simulate_no_opc_pw
    sraf_print_avoidance ...
    incr i
}
OPC_ITERATION 3
```

This converts to the following:

```
OPC_ITERATION 3
set i 0
while {$i < 3} {
    OPC_XMEEF_ITERATION iterations 1
    OPC_ITERATION -simulate_sites -set_epc -simulate_no_opc_pw
    sraf_print_avoidance ...
    incr i
}
OPC_XMEEF_ITERATION iterations 3
```

Optimized for Accuracy

In the following examples, the OPC_XMEEF_ITERATION calls are optimized to favor accuracy over run time. This includes the following:

- XMEEF value does change slightly as OPC edge placement changes.
- Each call to OPC_XMEEF_ITERATION triggers re-computation of XMEEF value based on previous iteration OPC results.
- Use two to three OPC_XMEEF_ITERATION calls to trigger re-computation two to three times within the entire OPC flow.
- XMEEF value re-computation can become time consuming.

Example 1: Single Call

```
OPC_ITERATION 10
```

This converts to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 2
OPC_XMEEF_ITERATION iterations 3
```

Example 2: Multiple Calls

```
OPC_ITERATION 3
...
OPC_ITERATION 4
...
OPC_ITERATION 3
```

This converts to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 2
OPC_XMEEF_ITERATION iterations 3
```

Example 3: Multiple PWOPC Calls

```
OPC_ITERATION 3
...
OPC_ITERATION 7 -PW ...
```

This converts to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 2 -PW ...
OPC_XMEEF_ITERATION iterations 3 -PW ...
```

Optimized for Maximum Accuracy

The following examples show OPC_XMEEF_ITERATION optimized for maximum accuracy over run time.

- Use the iterativeXmeef switch to enable re-computation of XMEEF values at every iteration.
- XMEEF value re-computation is most time consuming in this case as they are evaluated at each OPC iteration.

Example 1: Single Call

```
OPC_ITERATION 10
```

This converts to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 5 iterativeXmeef true
```

Example 2: Multiple Calls

```
OPC_ITERATION 3
...
OPC_ITERATION 4
...
OPC_ITERATION 3
```

This converts to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 5 iterativeXmeef true
```

Example 3: Multiple PWOPC Calls

```
OPC_ITERATION 3
...
OPC_ITERATION 7 -PW ...
```

This converts to the following:

```
OPC_ITERATION 3
OPC_XMEEF_ITERATION iterations 5 iterativeXmeef true -PW ...
```

OUTPUT_IMAGE

Calibre nmOPC Tcl Scripting Commands

OPC Controls

Instructs Calibre nmOPC to output the given image register to a layer.

Usage

OUTPUT_IMAGE *name register*

Arguments

- ***name***

Name of the output layer to place the shape on.

- ***register***

An output register name to take the image from; must be one of the predefined IM0...IM32 set.

Description

The command outputs the given image register to a layer.

Tcl Results

None

Performance

O(N) where N is the number of edges in the output register

Rating = fast

Examples

This outputs “IMAGE” from the register 0:

```
OUTPUT_IMAGE IMAGE IM0
```

OUTPUT_MEASUREMENT_LOCATIONS

[Calibre nmOPC Tcl Scripting Commands](#)

[Output Controls](#)

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. If omitted, this option defaults to 16/3 of the Nyquist sampling interval (the length used in the actual EPE calculations).

Description

This command is used to output one box per EPE measurement location. Note that it handles only measurement locations and not sites, and so cannot be used with [algorithm 2](#).

Tcl Results

None

Performance

O(N) where N is the number of measurements.

Rating = depends on the tag set. 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
```

Related Topics

[SITES_DUMP](#)

OUTPUT_OP

Calibre nmOPC Tcl Scripting Commands

OPC Controls

Outputs OPC layout of specified layer.

Usage

OUTPUT_OP *offset_type opc_layer1 [...opc_layerN]* **-out** *output_layer1 [...output_layerN]*

Arguments

- *offset_type*

A required argument that specifies how to offset the edges. Specify one of the following:

- curvilinear** — OPC displacement as curvilinear shapes.
- epe** — Effective EPE.
- nomepe** — Nominal EPE.
- opc** — OPC displacement as rectilinear shapes.

Do not use -opc with curved input layers. (It can be used with the output from [setlayer](#) [layer_simplify](#).)

- *opc_layer1 [... opc_layerN]*

A required argument that specifies at least one opc input layer for which the current layout state is to be saved to its corresponding output layer.

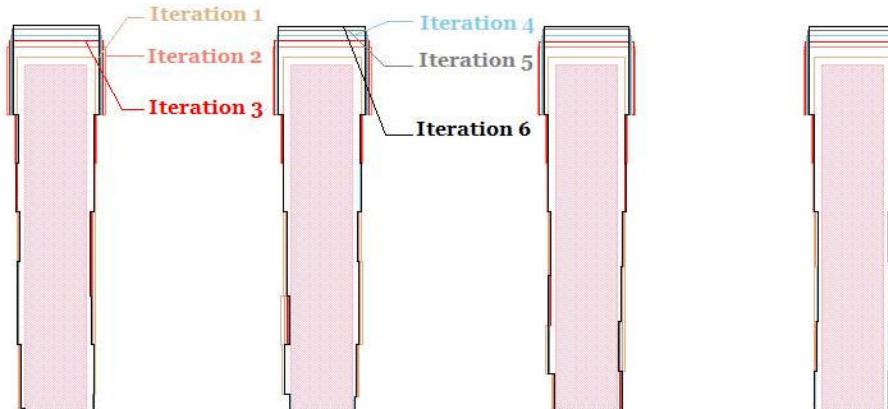
- **-out** *output_layer1 [... output_layerN]*

A required argument that specifies at least one destination output layer to copy the opc layer's layout to. Each output layer is cleared of layout before copying the opc layout.

Description

The command outputs the OPC layout of the specified layer. This can be useful to output the OPC results per iteration, as shown in [Figure 5-15](#). You can then analyze how edges are fragmented and moved each time the OPC recipe iterates.

Figure 5-15. OPC Output Per Iteration Example



Tcl Results

None

Performance

Theta (N) where N is the number of edges in design.

OUTPUT_RETARGET

Calibre nmOPC Tcl Scripting Commands

Output Controls

Outputs retargeted opc layers.

Usage

OUTPUT_RETARGET *opc_layer1* [... *opc_layerN*] **-out** *output_layer1* [... *output_layerN*]

Arguments

- *opc_layer1* [... *opc_layerN*]
A required argument that 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*]
A required argument that specifies the destination output layer(s) where the retargeted opc layer is copied. Each output layer is cleared of any layout before copying the retargeted OPC layout.

Description

This command outputs retargeted opc layers created with the [TARGET_CONTROL](#) (constant function type or -constant mode) or [PW_RETARGET](#) commands. If layers of type “target” are present, the command preferentially maps retargeting from opc layers to target layers and output retargeting on top of target layers.

Tcl Results

None

Performance

Theta (N) where N is the number of edges in design.

OUTPUT_SHAPE fragment

[Calibre nmOPC Tcl Scripting Commands](#)

[Output Controls](#)

Outputs simple shapes around fragments on a layer and attaches tag-set properties.

Usage

Syntax 1: Geometric Output

OUTPUT_SHAPE fragment *name* *gid*
[-moved | -epe | -nomepe | -pwepe *name* | -point | -point1 | -point2]
***halfwidth* [*trimends* [*offset*]]**

Syntax 2: Annotated Output

OUTPUT_SHAPE fragment *name* *gid* [-point | -point1 | -point2]
-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 OPC 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.
The name can be "nominal" or match a [pw_condition cond_name](#). Other values cause the run to exit with an error.

- **-point**
 An optional flag that marks each endpoint with a box. The box is $2 * \text{halfwidth}$ on each side and centered on the endpoint.
- **-point1**
 An optional flag that marks the first endpoint of each fragment with a box.
- **-point2**
 An optional flag that marks the second 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](#) [denseopc](#) to read the 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**. The default is max.
- The property is attached to a 1-dbu marker. The marker is placed at the center of the fragment unless -point, -point1, or -point2 is specified; when any of those arguments are present, the marker is attached to the endpoint box.
- **-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 defined by fragments. Typically, these simple shapes can be used for debugging or visualization of fragmentation results. An entire tag set can be output to shapes.

As of version 2019.1, the command can also be used to annotate layouts with numeric properties that Calibre nmOPC has stored in tag sets.

Note

- When using OUTPUT_SHAPE with [NEWTAG epe](#), note that the OUTPUT_SHAPE generates a contour based on the EPE from the *prior* OPC iteration, while NEWTAG epe obtains its EPE from the *final* iteration of the OPC run.
-

Tcl Results

None

Performance

$O(N)$ where N is the number of shapes to output.

Rating = fast

Examples

Example 1

The following example outputs all line ends:

```
OUTPUT_SHAPE fragment DEBUG line_end 0.05 0.004
```

Example 2

The following example outputs each the EPE for each process window condition to a separate layer. It uses a litho model for the nominal condition and layer transmission.

```
denseopc_options dopc {
    ...
    layer target      opc      mask_layer 0
    layer sraf        sraf     mask_layer 0
    layer target_curve hidden  mask_layer 0

    image lmodel
    pw_condition pw1 dose 0.95
    pw_condition pw2 dose 1.05

    retarget_layer target_curve spline
    ...

    OPC_ITERATION 8
    OPC_XMEEF_ITERATION 2 -PW mind_width_pw pw1 allFrags 0.030
    PW_RETARGET

    OUTPUT_SHAPE fragment frag_nom allFrags -pwepe nominal 0.001 0.001
    OUTPUT_SHAPE fragment frag_pw1 allFrags -pwepe pw1      0.001 0.001
    OUTPUT_SHAPE fragment frag_pw2 allFrags -pwepe pw2      0.001 0.001
    ...
}
```

The process window condition “nominal” refers to the default condition specified by the litho model.

Example 3

The following example demonstrates how to output shapes with a width property attached. The property is calculated by [NMBIAS_MEASURE_TAG](#) and placed in the tag set “width_prop,” highlighted in green.

```
NMBIAS_MEASURE_TAG poly -measurement_type width dist 0.07 -aout width_prop
...
OUTPUT_SHAPE fragment width_layer width_prop \
    -property width float max 0.002
...
}
...
setlayer width_annotation = denseopc poly assist contact MAP width_layer \
    OPTIONS opts property { width }
...
width_annotation = LITHO DENSEOPC FILE property.in ...
width_annotation {COPY width_annotation}
DRC CHECK MAP width_annotation 19 PROPERTIES
```

The OUTPUT_SHAPE command reads the value from the tag set width_prop (in green) and gives it the name “width” (in orange) when writing the layer. From here, property passing works the same as it does for Calibre OPCverify: the setlayer denseopc command uses a property block to read the property and pass it to the SVRF. The SVRF file copies the layer to the output with a DRC CHECK MAP command, specifying the keyword “PROPERTIES” to annotate the layer. See “[Writing Properties to Outputs](#)” in the *Calibre OPCverify User’s and Reference Manual* for more details.

OUTPUT_TARGET_CONTROL

Calibre nmOPC Tcl Scripting Commands

Output Controls

Outputs shapes from a target for fragments placed under TARGET_CONTROL.

Usage

OUTPUT_TARGET_CONTROL *tag* *outLayer* [-range *constr*]

Arguments

- ***tag***
Specifies an tag set name.
- ***outLayer***
Specifies the name of the output layer to place the shape on.
- **-range *constr***
Specifies an optional range for target control values. Range values are in user units. The range must be specified as the last argument.

Description

This will output shapes representing the modified target, for the fragments in ***tag*** which have had **TARGET_CONTROL** applied.

- Constant functions are represented by a rectangle.
- Linear functions are represented by a trapezoid or two triangles.
- Quadratic functions are represented by up to three trapezoids or triangles.

If a range is specified, a fragment is output only when it has a target control value in the specified range. Only constant target control values or constant-type target control functions are considered for the range check.

Tcl Results

None

Performance

O(N) where N is the size of the input set.

Rating = medium

PW_RETARGET

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Applies PWOPC-based retargeting for OPC fragments in either a tag set or for all fragments.

Usage

PW_RETARGET [*tagname*]

Arguments

- *tagname*

An optional keyword restricting a tag set to consider OPC fragments exclusively.

Description

This command applies PWOPC-based retargeting for OPC fragments in the tag set if specified, or for all OPC fragments. SRAF fragments are always excluded.

TCL Result

None

Performance

O(N) where N is the size of the input set.

Rating = fast

SET_EPE

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Sets EPEs on fragments based on an input Tcl formula or expression.

Usage

SET_EPE tag -sites {-tcl_formula "formula" | -expression "expression"}

Arguments

- **tag**

Specifies the tag name.

- **formula**

Specifies a TCL script that returns a numeric value to be used for setting the EPE. The **formula** must be enclosed in double quotation marks (""). The script may contain the following special variables listed in the following table.

Table 5-14. Special Variables for SET_EPE

Variable	Description
These variables are valid if there is at least one EPE site (for each variable, <i>name</i> is the name of a defined process window condition).	
\$epe_min_nominal and \$epe_min_name	These are floating point values representing the EPEs calculated at EPE sites. If a fragment has more than one site, the minimum value will be used.
\$epe_max_nominal and \$epe_max_name	These are floating point values representing the EPEs calculated at EPE sites. If a fragment has more than one site, the maximum value will be used.
\$epe_avg_nominal and \$epe_avg_name	These are floating point values representing the EPEs calculated at EPE sites. If a fragment has more than one site, the average value will be used.
\$printing_nominal and \$printing_name	These are true if any of the intensity value sampled at EPE sites are above the printing threshold.
\$imax_nominal and \$imax_name	These are the maximum intensity (imax) values. These are not available if the previous OPC_ITERATION used -simulate_sites_epe instead of -simulate_sites. Only EPE sites are checked.
\$imin_nominal and \$imin_name	These are the minimum intensity (imin) values. These are not available if the previous OPC_ITERATION used -simulate_sites_epe instead of -simulate_sites. Only EPE sites are checked.

Table 5-14. Special Variables for SET_EPE (cont.)

Variable	Description
These variables are valid if there is at least one WAFER_ENCLOSE or WAFER_ENCLOSEDBY site.	
\$constr_epe_min_nominal and \$constr_epe_min_name	The minimum WAFER_ENCLOSE EPE (if larger than zero, the constraint is obeyed).
\$constr_epe_max_nominal and \$constr_epe_max_name	The maximum WAFER_ENCLOSEDBY EPE (if larger than zero, the constraint is obeyed).

- **expression**

Specifies an expression to use for setting the EPE. The **expression** must be enclosed in double-quotes (""). An expression is a sequence of tokens representing the following Backus-Naur Format (BNF):

Table 5-15. Expressions for Setting EPE

<i>expr</i> -> <i>expr</i> + <i>term</i> -> <i>expr</i> - <i>term</i> -> <i>term</i>	<i>term</i> -> <i>term</i> / <i>prim</i> -> <i>term</i> * <i>prim</i> -> <i>prim</i>	<i>prim</i> -> <i>number</i> -> <i>variable</i> -> '-' <i>prim</i> -> '+' <i>prim</i> -> '!' <i>prim</i> -> '~' <i>prim</i> -> '('('expr')')' -> <i>func</i> '('('expr')')
<i>func</i> -> 'exp' -> 'log' -> 'sqrt' -> 'sin' -> 'cos' -> 'tan' -> 'min' -> 'max'	<i>variable</i> -> same special variables as in the <i>formula</i> specification, except they don't have to be prefixed by "\$"	

Description

This command sets EPEs on fragments based on an input Tcl formula or expression. Fragments that do not have a site will be ignored. Fragments that do not have any valid sites (for example, simulated with -simulate_sites at the last [OPC_ITERATION](#)) will be ignored. Fragments that have some valid sites will be considered, but the invalid sites will be ignored.

Notes

The special variables in the `-tcl_formula` must be prefixed with a backslash (\), otherwise they are evaluated outside SET_EPE and might produce undesired results.

For example:

```
set epe_nominal 0.05
SET_EPE all -sites -tcl_formula "return $epe_nominal"
    # sets the EPEs of all fragments to 0.05 (probably undesired), or
    # error if epe_nominal is not defined
SET_EPE all -sites -tcl_formula "return \$epe_nominal"
    # sets the EPEs of all fragments to their nominal epe,
    # calculated at the sites
```

There is no such requirement for the `-expression` variant.

Use caution when defining a formula or expression, to avoid run time problems like division by 0. For example:

```
SET_EPE all -sites -expression "1/imin_nominal"
```

will probably work correctly in most cases, until “imin_nominal” is 0, at which point Calibre will stop the run with an error message.

Run time problems can also occur if a formula/expression is valid but contains variables that are not available at the evaluation time. For example:

```
OPC_ITERATION -simulate_sites_epe
SET_EPE all -sites -expression "1/imin_nominal"
```

Although this will pass the syntax check, it will fail at run time because “imin_nominal” is not available. It is recommended that you use [OPC_ITERATION -simulate_sites](#) instead.

Tcl Results

None

Performance

Proportional to the complexity of the input tcl_formula script.

Very fast if `-expression` is used instead of `-tcl_formula`.

Examples

```
SITES_CREATE all -conservative -numx 11 -spacing 0.01
for { set i 0 } { $i < 10 } {set i [expr $i+1]} {
    OPC_ITERATION -set_epe -simulate_sites
    SET_EPE all -sites -tcl_formula "return \$epe_nominal"
    OPC_ITERATION -apply_feedback
    FRAGMENT_MOVE
}
```

Related Topics

[SET_MEDIAN_EPE](#)

SET_MEDIAN_EPE

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Sets a fragment set to the median EPE value.

Usage

SET_MEDIAN_EPE *tag*

Arguments

- *tag*

Specifies a tag set containing edges to set EPE values to their median.

Description

This command loops through the fragments in the tag set, and sets the EPE value for all fragments to their median. This should be used to help OPC consistency.

Note that it will be the user's decision to select only fragments that should have the same EPE, otherwise the OPC results will be incorrect.

Tcl Results

None

Performance

O($N \log N$), where N is the number of fragments in the input.

Rating = fast

Examples

The following example derives a tag set consisting of high MEEF line ends with the same environment:

```
NEWTAG all M1 -out M1_all
...
...
NEWTAG ... ... -out high_meef_line_ends
# run iterations
for { set i 0 } { $i < $iter } { incr i }
    # set regular epe
    OPC_ITERATION -set_epe
    # make sure the high MEEF line end EPEs are consistent
    SET_MEDIAN_EPE high_meef_line_ends
    # apply feedback, and move
    OPC_ITERATION -apply_feedback
    FRAGMENT_MOVE
}
```

SITES_CREATE

Calibre nmOPC Tcl Scripting Commands

Site Controls

Create sites centered on the target layer.

Usage

SITES_CREATE *tag* **-bylayer** *layer* [-multi] [-type *type*] **-numx** *x* **-spacing** *s*

[-min_cornerdist *dist*] [-offset *offset*] [-epe_spacing *es*]
[-frag {pt1 | pt2 | midpt} [-shift [ratio] *shift_value*]] [-replace]

SITES_CREATE *tag* **-bymarker** *layer* **-numx** *x* **-spacing** *s*

[-min_cornerdist *dist*] [-offset *offset*] [-dist *md*] [-replace]

SITES_CREATE *tag* **-conservative** [-ignore_jogs *jog*] **-numx** *x* **-spacing** *s*

[-offset *offset*] [-replace]

SITES_CREATE *tag* **-frag** {pt1|pt2|midpt} **-numx** *x* **-spacing** *s*

[-shift [ratio] *shift_value*] [-min_cornerdist *dist*]
[-{offsetPropTag *tag2*} | -offset *offset*] [-noretarget] [-rotate *angle*] [-replace]

SITES_CREATE *tag* **-multi** **-numx** *x* **-spacing** *s* [-epe_spacing *es*] [-replace]

Arguments

- ***tag***

A required argument that specifies the tag name.

- **-bylayer** *layer*

A required argument for “by layer” usage that specifies that sites are created on the edges of the shapes on the given layer (but associated with the fragments in the input *tag*). The layer type must not be opc or correction.

Note

 If -offset is specified, the offset value is used for site placement and overrides any values specified with [wafer_enclose](#), [wafer_enclosedby](#), or [wafer_exclude](#).

- **-numx** *x*

A required argument that specifies the number of points per site.

The value of *x* can be either a floating point number or read from an annotated tag set *tag_x* using “**property(*tag_x*)[:default(*value*)]**”, for example, **property(tNumProp)** or **property(tNumProp):default(15)**. If using the property syntax and a fragment is not annotated, the default value is assigned. If there is no default value, the site has 15 points.

- **-spacing** *s*

A required argument that specifies the distance between points.

The value of *s* can be either a floating point number or read from an annotated tag set *tag_s* using “**property(*tag_s*)[:default(*value*)]**”, for example, **property(tSpaceProp)** or **property(tSpaceProp):default(0.01)**. If using the property syntax and a fragment is not annotated, the default value is assigned. If there is no default value, the distance is Nyquist/3.

- **-bymarker *layer***

A required argument for “by marker” usage that specifies sites are created by projecting the markers on *layer* to the nearest fragment.

- **-conservative**

A required argument for “conservative” usage that places the site at the end point opposite to a corner (for one-corner fragments) and in the middle of the fragment for none or two-corner fragments.

- **-frag {pt1 | pt2 | midpt}**

A required argument for “frag” usage that specifies the location of the site. The **pt1** and **pt2** indicate placing sites at endpoints of a fragment, and **midpt** places a site at the center of a fragment.

- **-multi**

A required argument for “multi” usage and an optional argument to “by layer” usage that creates multiple sites per fragment, in the manner of **algorithm 0** cutlines (as specified by the **epe_spacing** setup file command).

Note

 Optional arguments are listed in alphabetical order.

- **-dist *md***

An optional argument for “by marker” usage that specifies the maximum distance between a polygon on *layer* and the nearest fragment for the fragment to receive a site.

- **-epe_spacing *es***

An optional argument that specifies the spacing between sites. If *es* is greater than 0, this overrides the global **epe_spacing** setting. The default value is -1, which means that the global **epe_spacing** value is used.

The value of *es* can be either a floating point number or read from an annotated tag set *tag_es* using “**property(*tag_es*)[:default(*value*)]**”, for example, **property(tEPEProp)** or **property(tEPEProp):default(0.01)**. If using the property syntax and a fragment is not annotated, the default value is assigned. If there is no default value, the distance is the global **epe_spacing**.

When specified in a **-bylayer** statement, **-multi** is required.

- **-ignore_jogs *jog***

An optional argument that specifies that fragments adjacent to jogs up to this length are treated as though there is no corner on the side with the jog.

- **-min_cornerdist *dist***

An optional argument that specifies that sites are not allowed to get closer than *dist* from a corner.

The value of *dist* can be either a floating point number or read from an annotated tag set *tag_dist* using “property(*tag_dist*)[:default(*value*)]”, for example property(tCD) or property(tCD):default(0.03). If using the property syntax and a fragment is not annotated, the default value is assigned. If there is no default value, the distance is 0.

- **-noretarget**

An optional argument that specifies that the retarget layer is not used when determining where to place the site. This is ignored if no retarget layer is present.

- **-offset *offset***

An optional argument that specifies that sites are offset from the target, retarget, or **-bylayer** layer by this value (outside if positive, inside if negative). Note that various commands that use the sites consider the offset target for EPE purposes.

offset — A floating point number. All sites are offset by the same amount.

property(*offsetTag*) — A keyword and annotated tag set. Site offset is read from the property value attached to fragments in *offsetTag*.

Use “:default(*value*)” to provide a default value for fragments that do not have an *offsetTag* annotation. For example, property(tOffset):default(-0.002) would use the annotated value in tOffset if present and if not, an offset of 0.002 to the inside. If :default(*value*) is not present, fragments without an annotation are offset by 0.

applyProperty — A keyword that indicates the offset value is set by the property attached to *tag*. When using applyProperty, *tag* must be an annotated tag set. Both -shift and -offset accept applyProperty, but only one of them may use it at a time. To have both -shift and the offset value read from an annotated tag set, use “-offsetPropTag” instead of “-offset applyProperty.”

When used with **-bylayer** the -offset value is interpreted as follows:

- With -type WAFER_ENCLOSE *offset* is the amount to enclose a layer.
- With -type WAFER_ENCLOSEDBY *offset* is measured in the other direction and should be the negative of the amount to be enclosed by the layer.
- With -type WAFER_EXCLUDE *offset* is the distance to keep away from **-bylayer** and is measured outward and normal to the polygons on *layer*.

- **-offsetPropTag *tag2***

An optional argument for specifying a second annotated tag set to be used with “**SITES_CREATE *tag -frag ... -shift applyProperty***”. If a fragment is in **tag** but not **tag2**, the offset is 0.

The following example creates sites on fragments in the shiftTag tag set. The shift value is read from the properties annotated on the fragments. The offset properties are read from a second annotated tag set, offsetTag. Old sites are deleted before the new ones are created.

```
SITES_CREATE shiftTag -frag midpt -numx 15 -spacing 0.005 \
              -shift applyProperty -offsetPropTag offsetTag -replace
```

- **-replace**

An optional argument that specifies to delete existing sites of the same type before new sites are created on the fragment.

- **-rotate *angle***

An optional argument that specifies the amount of rotation in degrees to apply to the newly created site. The default value is 0, or no rotation.

Site rotation is done counterclockwise from the original direction. The original direction of the control site is perpendicular to the fragment.

The value of *angle* can be either a floating point number or read from an annotated tag set *tag_r* using “property(*tag_r*)[:default(*value*)],” for example, property(tRot) or property(tRot):default(15).

- **-shift [*ratio*] *shift_value***

An optional argument that specifies the amount in dbus to shift the sites. A negative value shifts towards **pt1**, a positive value shifts towards **pt2**.

The option “*ratio*” causes the value to be treated as a ratio to the length of the fragment. The absolute value must be less than or equal to 1.0 when *ratio* is specified.

The value can be specified in the following ways:

shift_value — A floating point number. All sites are shifted by the same amount.

property(*shiftTag*)[:default(*value*)] — A keyword and annotated tag set with an optional default. The value is read from the property attached to fragments in *shiftTag*.

Fragments that are not in the *shiftTag* set receive a value of 0 when a default value is not provided.

applyProperty — A keyword that indicates the shift value is set by the property attached to **tag**. When using **applyProperty**, **tag** must be an annotated tag set. Both **-shift** and **-offset** accept **applyProperty**, but only one of them may use it at a time. To have both **-shift** and the offset value read from an annotated tag set, use “**-offsetPropTag**” for the offset.

When specified in a **-bylayer** statement, **-frag** is required.

- **-type** *type*

An optional argument that specifies the site type: EPE, OVERLAY, WAFER_ENCLOSURE, WAFER_ENCLOSEDBY, or WAFER_EXCLUDE. The default is to create EPE sites.

Description

This command creates sites centered on the target or retarget layer. A site is similar to a cutline that measures image intensity at several points. The type determines how Calibre nmOPC includes the site measurements when optimizing correction, and the site placement in regards to the contour or fragment.

A site should be able to cover all contours, especially when there are bridging or pinching constraints. The maximum EPE detected by a site is approximately $(\text{numx}-1)/(2*\text{spacing})$. By default, numx is 15 and spacing is Nyquist/3. If some process conditions can significantly overprint or underprint, adjust these accordingly.

Tcl Results

None.

Performance

No impact; however, simulation time may be affected by the number of sites.

Examples

```
SITES_CREATE all -conservative -numx 11 -spacing 0.01
SITES_CREATE line_ends -bylayer contact -type WAFER_ENCLOSURE -offset 0.02
# to enforce 20nm contact enclosure at line ends
```

SITES_DELETE

Calibre nmOPC Tcl Scripting Commands

Site Controls

Deletes all sites associated to fragments on the input tag set.

Usage

SITES_DELETE *tag* [-type *site_type*]

Arguments

- ***tag***
Specifies the tag name.
- **-type *site_type***
An optional argument that specifies particular types of sites for deletion.

Description

Deletes all sites associated to fragments on the input tag set.

Tcl Results

None

Performance

No impact.

Examples

```
SITES_DELETE all
```

SITES_DUMP

Calibre nmOPC Tcl Scripting Commands

Site Controls

Outputs all sites associated to fragments on the input tag set.

Usage

```
SITES_DUMP tag name [-type type] [ALL | EXTERNAL | INTERNAL]  
[-epe pw_name] output_site_half_width [output_site_half_length]
```

Arguments

- **tag**
A required argument that specifies the tag name.
- **name**
A required argument that specifies the name of the output layer to place the shape on.
- **-type type**
An optional argument that specifies the site type. Valid types are PAIR and any created by SITES_CREATE. If -type is specified, only sites of this type in the tag set are output.
- **-epe pw_name**
An optional argument that causes boxes to be output instead of the sites themselves. The boxes are of width $2 * \text{output_site_half_length}$ and output at the location where the site intersects the *pw_name* (virtual) contour.
This argument cannot be specified if -type is PAIR.
- **output_site_width** [**output_site_length**]
Specifies the length and width of the shape to be output wherever sites are. If the length is not specified, the actual site length is used.
- **ALL | EXTERNAL | INTERNAL**
An optional argument that can only be specified with -type PAIR. If this is specified, only pairs of this type are output.

Description

This command outputs all sites associated to fragments on the input tag set.

Tcl Results

None

Performance

No impact.

Examples

The following is a basic example of SITES_DUMP:

```
SITES_DUMP all outLayer 0.002
```

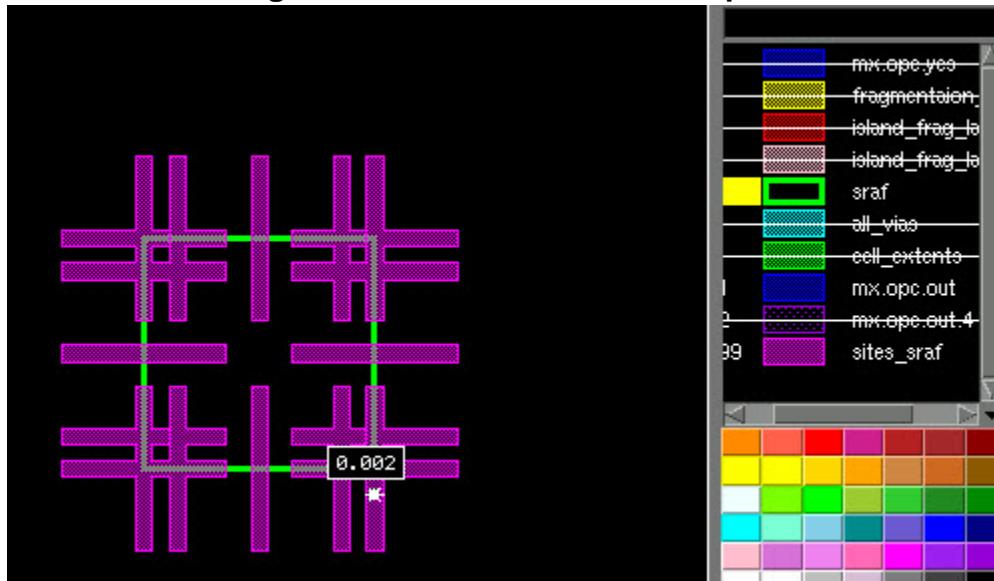
This example uses SITES_DUMP to output all sites associated with an SRAF.

```
SITES_DUMP all_sraf sites_sraf 0.001 0.01
SITES_CREATE all_sraf -multi -numx 11 -spacing 0.003
SITES_CREATE all_sraf -frag pt1 -numx 11 -spacing 0.003
SITES_CREATE all_sraf -frag pt2 -numx 11 -spacing 0.003
...
sites_sraf =
    LITHO DENSEOPC mx.opc.yes fragmentation_layer island_frag_layer_1.1
        island_frag_layer_2.1 sraf negsraf mx.opc.no all_vias no_opc_marker
        MAP sites_sraf FILE "mx.opc.setup"
sites_sraf {COPY sites_sraf} DRC CHECK MAP sites_sraf 9999

setlayer sites_sraf = denseopc opc.yes opc.frag.yes island_frag_layer_1 \
    island_frag_layer_2 sraf negsraf opc.no mx.opc.yes.curve \
    all_vias_chopped no_opc_marker no_opc MAP sites_sraf \
    OPTIONS m1_options
```

The following figure illustrates the output. The SRAF is a box rendered in green, and the sites are output on a separate layer (in purple).

Figure 5-16. SITES_DUMP Example



Related Topics

[SITES_CREATE](#)

SITES_SHIFT (Deprecated in 2020.3)

Calibre nmOPC Tcl Scripting Commands

Site Controls

Shifts all sites (or sites of the given type) associated with fragments on the input tag set.

Note

 This command is deprecated as of the 2020.3 release and will not be documented in future releases. Use [SITES_CREATE -shift](#) and [-offset](#) instead.

Usage

```
SITES_SHIFT tag {value | property(tag2)[:default(value)] | applyProperty}  
[-min_cornerdist {dist | property(tag3)[:default(value)]}] [-type type]
```

Arguments

- **tag**
A required argument that specifies the tag name.
- **value | property(tag2)[:default(value)] | applyProperty**
A required argument that specifies the amount to shift the sites. A negative value shifts towards **pt1**, a positive value shifts towards **pt2**.
 - value** — A floating point number. All sites are shifted by the same amount.
 - property(tag2)[:default(value)]** — Uses the property value attached to fragments in the **tag2** set. Fragments that are not in the **tag2** set use the value specified by the optional “**:default(value)**” and 0 otherwise.
 - applyProperty** — A keyword that indicates the shift value is set by the property attached to **tag**. When using **applyProperty**, **tag** must be an annotated tag set.
- **-min_cornerdist {dist | property(tag3)[:default(value)]}**
An optional argument that specifies that sites are not moved to closer than *dist* from a corner. *dist* can be either a floating point number or read from an annotated tag set with **property(tag3)[:default(value)]**. Fragments that are not in the **tag3** set use the value specified by the optional “**:default(value)**” or if not specified, 0.
- **-type type**
An optional argument that specifies the site type: EPE, WAFER_ENCLOSE, WAFER_ENCLOSEDBY, or WAFER_EXCLUDE. If **-type** is specified, only sites of this type are shifted.

Description

Shifts all sites (or sites of the given type) associated with fragments on the input tag set.

Tcl Results

None

Performance

No impact.

Examples

```
# shifts sites away from corners
SITES_SHIFT corn_pt1 0.002
SITES_SHIFT corn_pt2 -0.002
```

sraffragalign

[Calibre nmOPC Tcl Scripting Commands](#)

[Unit Conversions and Miscellaneous](#)

Aligns fragmentation vertices on sraf layers for sraf_print_avoidance.

Usage

sraffragalign [*max_pair_dist dist*] [*unalignedTag outTagName*]

Arguments

- *max_pair_dist dist*

An optional argument that specifies that SRAF fragments that are more than *dist* apart are not paired. Unpaired fragments can not form a “cut”, and SRAFs containing unpaired fragments cannot be deleted. The default value is 2*Nyquist.

- *unalignedTag outTagName*

After the first pass is finished, some fragments may still not be aligned. Fragments that were not aligned can be output to the tag set *outTagName*. For a fragment to be in this tag set, it must be alignable 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.

Description

This command aligns tagged fragment break points on sraf layers so that facing fragment breaks project to each other. It is intended for use by [sraf_print_avoidance](#). This command is implicit, which means that it is executed automatically on every SRAF layer.

The command acquires the layer name(s) and default parameter settings internally, using the following methods:

- The layer name is taken from the name of the sraf layer specified to Calibre nmOPC,
- The minedgelength value from the value of [fragment_min](#) for the layer,
- The maximum polygon width defaults to 2*Nyquist value (this is the same value calculated for the default *max_pair_dist*).

Tag sets are automatically cleared when sraffragalign is called. Negative SRAFs are not affected.

Examples

```
. . .
layer e1_pbar sraf -0.9741 -0.0095
. . .
fragment_layer e1_pbar {
    fragment_min 0.03
    fragment_max 0.10
}
. . .
sraffragalign
. . .
```

In this example, only one layer will be processed, and its name, e1_pbar, will come from the layer definition. The fragment_min value will come from the fragment_layer block (0.03), with maxpolywidth derived from the optical model parameters.

This command is applied to all SRAF layers listed in the Calibre nmOPC layer block (for this example, there is only one).

sraf_print_avoidance

Calibre nmOPC Tcl Scripting Commands

Unit Conversions and Miscellaneous

Prevents selected Sub-Resolution Assist Feature (SRAF) from printing.

Usage

```
sraf_print_avoidance [-pattern n] [-layer layer] [-asraf_layer layer] [-frags tag] [-step value]
[-check_only] [-pw name [threshold] [neg_sraf_threshold]]...
[-max_pair_dist dist] [-allow_cut] [-disallow_cut] [-nonrectangle] [-rectangle] [-irregular]
[-minimize_jogs] [-out_printing name] [-out_requires_cut name] [-out_unsolved name]
[-min_area value] [-min_square_area value]
[-asraf_min_area value] [-asraf_min_square_area value]
[-separate_mrc] [-adjust_frags] [-freeze_unaligned]
[-xmeef] [-xmeef_distance value] [-xmeef_threshold threshold]
```

Arguments

- **-pattern *n***

An optional argument that specifies which pattern to process to support multi-patterning. If not specified, all patterns that have sraf or asraf layers will be processed. If a layer is specified with -layer or -asraf_layer, only the corresponding pattern is processed.

- **-layer *layer***

An optional argument that specifies the layer containing SRAFs that are not to be printed. This must be of layer type sraf. Fragment pairs will only be created on this layer. If -layer is omitted, Calibre nmOPC will default to the first layer found that is of type “sraf.”

Note

 Starting from the 2014.2 release, -layer is only used for backwards compatibility. By default, sraf_print_avoidance processes one layer of sraf or asraf type simultaneously. To turn off sraf layer processing, use -layer OMIT.

- **-asraf_layer *layer***

An optional argument to use when specifying different process window conditions for anti-SRAF and SRAF layers, or otherwise using non-default layer processing. Fragment pairs will only be created on this layer. By default, the first layer of type asraf found is simulated to check for potential hole creation.

- To associate an anti-SRAF layer with a process window condition, specify the layer name. Layers must be of type asraf or opc.
- To turn off anti-SRAF layer processing, use -asraf_layer OMIT.

- **-frags *tag***

An optional argument that specifies that only fragments belonging to this tag will be moved. This may contain fragments on the opc layer (negative SRAFs).

Note



The tag set specified for this argument and the layer must match.

The tag should be valid when passed to sraf_print_avoidance. Some commands such as sraffragalign may manipulate fragments belonging to the tag and without invalidating the tag. In such cases, regenerate the tags if necessary.

The negative SRAFs are sub-resolution holes in the OPC features. In order to specify negative SRAFs, you can find the fragments comprising the holes and label them using this tag.

- **-step *value***

An optional argument that specifies the step value to size down (per fragment) for printing fragments. The default is 1 dbu.

- **-check_only**

An optional argument that specifies that Calibre nmOPC not resize or move any SRAF fragments. Instead, Calibre nmOPC will detect fragments that are printing and output them if -out_printing is specified. It also detects and outputs (if -out_requires_cut is specified) those fragments that cannot be sized down by the step value because of MRC constraints (thus requiring a cut). It also detects and outputs (if -out_unsolved is specified) those fragments that are unsolved.

- **-pw *name* [*threshold*] [*neg_sraf_threshold*]**

An optional argument that specifies the process window condition to consider when evaluating printing status and the threshold to use for it. The *threshold* and *neg_sraf_threshold* values are read from the model if none is specified.

- For positive SRAFs, if any intensity sampling point from any site on a fragment has an intensity higher than this *threshold*, the fragment is considered as printing (and will be moved in to make the SRAF smaller).
- For negative SRAFs, if any intensity sampling point from any site on a fragment has an intensity lower than *neg_sraf_threshold* (which defaults to *threshold*, if not specified in the argument or model), the fragment is considered as printing (and will be moved out to make the negative SRAF smaller).

This argument can be specified multiple times.

If not specified, Calibre nmOPC defaults to the [pw_condition](#) (or conditions) marked with the “spa” keyword. If a “spa” pw_condition is specified in a -pw command, it is not handled by default (explicit specification takes precedence over implicit). A list of conditions may be specified using the “-pw” keyword, and any “spa” conditions not in the list will be added by default.

- **-max_pair_dist *dist***

An optional argument that specifies that SRAFs or anti-SRAFs fragments that are more than *dist* apart are not paired. Unpaired fragments can not form a “cut”, and SRAFs containing unpaired fragments cannot be deleted. The default value is 2.5*Nyquist.

- **-allow_cut**

An optional argument that specifies that cuts are allowed. This is the default behavior; disable with **-disallow_cut**.

When cuts are allowed, minimum-width SRAFs that print only in a small section of their length or anti-SRAFs that create holes have the problematic section removed. Cutting only happens if the fragmentation is aligned prior to calling this function, which requires opposing pairs of fragments to be within **-max_pair_dist** distance of each other.

Cutting involves moving the two opposing fragments to the middle of the line (thus forming a singularity and, in effect, making a piece of the SRAF disappear) and freezing them.

Note that cuts must not be manually unfrozen, otherwise OPC will move the opposing fragments to their minimum internal distance in order to avoid MRC violations.

- **-disallow_cut**

By default, an SRAF or anti-SRAF is automatically cut if it shrinks below MRC limits. Setting **-disallow_cut** instructs Calibre nmOPC to stop shrinking an SRAF if it reaches the MRC limit.

- **-nonrectangle | -rectangle**

By default, **sraf_print_avoidance** works upon rectangles aligned along the axes or at multiples of 45 degrees. This is also the behavior when **-nonrectangle** is specified. When **-rectangle** is specified, only Manhattan rectangles are considered.

Note



The **-rectangle** option may give inconsistent results in hierarchical runs when the SRAF crosses a hierarchical boundary.

- **-irregular**

By default, an irregular region on an SRAF is not corrected. An irregular region is a region that has a concave corner or is across from a concave corner. Other regions of the SRAF are still corrected; only fragments near the concave corner do not receive corrections. To correct these regions as well, specify the **-irregular** option to enable all regions of an SRAF to be corrected.

- **-minimize_jogs**

If this option is present, fragments that do not cause SRAF printing or anti-SRAF holes may be moved along with their neighbors that do to prevent jog formation. The default value is the option is not specified.

Note that in general, minimizing jogs does not ensure that an SRAF that crosses tile or cell boundaries will have no jogs.

- **-out_printing *name***

Optionally specifies an output tag set. This contains all fragments that this command detects as “printing.”

- **-out_requires_cut *name***

An optional argument that specifies an output tag set. The tag set contains all printing fragments that cannot move the desired step amount because of an MRC constraint, thus needing to be cut.

- **-out_unsolved *name***

Optionally specifies an output tag set. This contains all fragments that cannot be “solved.” For example, fragments can be left “unsolved” because:

- It must be cut, but the cut is disallowed
- It must be cut, but it has no pair
- It must be moved inside, but it is frozen by user

Note that if a fragment is not marked as unsolved, that does not mean it is “fixed” (for example, not printing).

- **-min_area *value***

If this option is specified, rectangular features having area less than the specified value are deleted by sraf_print_avoidance. If a long rectangular feature gets split into multiple parts during the print avoidance check, the limit is applied to each part separately. If there is a conflict between printability and the min area constraint, the latter is given priority. The -max_pair_dist option must be specified when using this option and should be set to a reasonable value such as the maximum possible width of the assist features.

Note that if a fragment is not marked as unsolved, that does not mean it is “fixed” (for example, not printing).

- **-min_square_area *value***

This option acts similarly to -min_area, but its limit is applicable only to the shapes that were square at the start of the print avoidance check. If the shape changes from a square to a rectangle during print avoidance, this limit will still apply. This limit is applied to square-shaped features in addition to -min_area limit (if present). There should not be any fragmentation of square shapes in order for -min_square_area to function properly. The -max_pair_dist option must be specified when using this option.

- **-asraf_min_area *value***

An optional keyword that is identical to the -min_area keyword, but only applies to an asraf layer.

- **-asraf_min_square_area *value***

An optional keyword that is identical to -min_square_area, but only applies to an asraf layer.

- **-separate_mrc**

An optional keyword that specifies that **-min_area** is applied only to SRAFs that are not square, while **-min_square_area** is applied to the square SRAFs. By default, the minimum area specifications are considered simultaneously for a square; a square SRAF must be larger than both the **-min_area** and **-min_square_area** values or it will be cut. If **-min_square_area** is less than **-min_area**, it will have no effect. In this case, it is helpful to have the two values considered separately.

Note

 For **-separate_mrc**, a square SRAF is not removed if it is larger than the **-max_pair_distance** value. If you have an issue with square SRAFs that are not being removed (even though their area is less than the **-min_square_area** value), the **-max_pair_dist** value should be checked.

- **-adjust_frags**

Fragmentation boundaries may be adjusted to fit printing or non-printing regions. Selecting this option allows the fragments to be fit around a printing or non-printing region in order to minimize the effects of **sraf_print_avoidance**.

- **-freeze_unaligned**

An optional argument that allows movement of aligned fragments only (to shrink or grow). Unaligned fragments are not moved.

- **-xmeef**

An optional argument that enables cross-intensity calculations. Cross-intensity calculations allow more than one fragment to be moved to prevent printing.

When **-xmeef** is specified, fragments in a regions $\pm 12 \times \text{Nyquist}$ around a printing location are measured to determine their influence on the printing location. SRAFs are considered for trimming in outside-to-inside order. Once enough fragments are identified to eliminate the printing, no further adjustments are considered. This allows an outer SRAF to be shortened or cut in order to prevent printing at an inner SRAF location.

Any of the three arguments **-xmeef**, **-xmeef_distance**, and **-xmeef_threshold** can be used to start cross-intensity calculations. They may be specified together or independently.

- **-xmeef_distance *value***

An optional argument that enables cross-intensity calculations, like **-xmeef**, except overriding the default distance of $12 \times \text{Nyquist}$ with *value* specified in user units.

The cross-intensity distance defines a square around a printing location in which fragments are considered. The value should be large enough to include the inside of the innermost SRAF to the outside of the outermost SRAF. (This allows a printing inner SRAF to shrink an outer SRAF if necessary.)

- **-xmeef_threshold *threshold***

An optional argument that enables cross-intensity calculations, like -xmeef, except overriding the default threshold of 0.00001 with *threshold*.

The threshold is the minimum intensity reduction to be achieved by moving or shrinking a fragment by one dbu. This controls whether a fragment gets adjusted. Lower values mean more fragments can be adjusted.

Description

This command is designed to detect and correct for any printing occurring around sub-resolution assist features (SRAFs) or holes around anti-SRAFs. (For the rest of the discussion, all references to SRAFs printing also refer to anti-SRAFs creating holes, except where explicitly noted.)

This command is executed iteratively using the [OPC_ITERATION](#) command. For each iteration, [sraf_print_avoidance](#) checks sites, compares to the specified threshold, and moves the SRAF fragments to prevent SRAF printing. This does not guarantee that incoming SRAFs that print will not print upon completion, but it will reduce the probability of any printing upon completion and will not result in MRC violations.

The [sraf_print_avoidance](#) command is fragment-based and requires some definitions before it can be executed. These include:

- The SRAF layer has to be defined as an sraf type layer in the [denseopc_options](#) section. This type marks layer as correctable but removes unnecessary EPE calculations.
- The anti-SRAF layer has to be defined as an asraf or opc type layer in the [denseopc_options](#) section.
- The SRAFs must be defined using [fragment_layer](#) statements. If no fragmentation was defined, each SRAF edge will be treated as a fragment.
- Evaluation sites must be placed on each fragment in order to define locations where SRAF intensity will be measured and compared against a user-defined threshold. This is done using the [sraf_sites_create](#) custom scripting command.
- Define SRAF-specific area rule checks for the SRAF layer using the “[mrc_rule area](#)” command.

[Figure 3-32](#) on page 87 illustrates an example setup file using [sraf_print_avoidance](#).

MRC rules will be obeyed by [sraf_print_avoidance](#) if specified for the sraf or asraf layer. If the corrected shape is in violation with an internal MRC rule, violating portions will be cut or the entire shape will be deleted if necessary.

To summarize how this command works, the [sraf_print_avoidance](#) command, when executed:

1. Finds all rectangle SRAFs on the input layer (and non-rectangle SRAFs, if allowed).

2. Looks at sites which are already present and have simulations on input. If no sites are present, then no action is taken.
3. If any site has intensity that exceeds the specified threshold, the fragment attached to the site is marked for movement.
4. If cutting is permitted (the default), sraf_print_avoidance determines if a cut is needed for all marked fragments (for example, if MRC limit already reached).
If cutting is not permitted (-disallow_cut), then sraf_print_avoidance marks the entire edge on which the fragment resides, and marks its paired fragments.
5. For anything else that is not already at the MRC limit, the command moves edge pairs by step movement amount, obeying MRC.

In practice, sraf_print_avoidance will try to make the fewest changes possible. It may do one of the following:

- Reduce the width of a rectangular SRAF by changing displacement of either edge in the long dimension, while obeying MRC.
- Eliminate a rectangular SRAF by forcing the displacement of a singularity.
- Adjust non-square SRAFs by reducing displacement of fragments near the printing area.
- Cut non-square SRAFs (if fragmentation is aligned) by forcing displacement of a singularity of a pair of fragments in specific regions.
- If -minimize_jogs is not present, aligned or non-aligned fragment pairs may move in order to thin out an SRAF in some spots but not in others.

There are a number of best practices and recommendations available for sraf_print_avoidance. Refer to “[SRAF Print Avoidance Best Practices](#)” on page 661 for information.

Examples

```
sraf_print_avoidance -layer assist \
    -pw pw_0 0.102 \
    -pw pw_1 0.102 \
    -pw pw_2 0.102 \
    -pw pw_3 0.102 \
    -step 0.002 \
    -out_printing prints \
    -out_unsolved unsolved \
    -irregular
```

sraf_sites_create

[Calibre nmOPC Tcl Scripting Commands](#)

[Unit Conversions and Miscellaneous](#)

Creates sites on an SRAF layer.

Usage

```
sraf_sites_create [-layer layer] [-tag tag] [-site_spacing value] [-extent_inward value]  
[-extent_outward value]
```

Arguments

- **-layer *layer***

An optional keyword that specifies a layer containing SRAFs that are not to be printed. The layer must be of type “sraf” or “asraf.” By default, Calibre nmOPC creates sites on the first sraf-type and the first asraf-type layers found.

- **-tag *tag***

If specified, only fragments belonging to the specified tag set have sites placed on them. The tag should be valid when passed to `sraf_sites_create`. The default is to place sites on all fragments on the layer.

- **-site_spacing *value***

An optional keyword that defines the spacing between sites along a fragment. The default value is the Calibre nmOPC Nyquist sampling distance (NSD). If specified, this value must be > 0 .

- **-extent_inward *value***

An optional keyword that defines the distance inward that a site is supposed to cover. This distance is measured from the edge of the SRAF towards the inside of the SRAF. The default value is half the distance across the SRAF, which means that the site extends to the middle of the SRAF. If the distance across the SRAF is larger than $3 \times \text{Nyquist}$, the extent_inward value is only $1 \times \text{Nyquist}$. (This prevents oversized sites from being created at line ends.) If specified, this value must be ≥ 0 .

- **-extent_outward *value***

An optional keyword that specifies the distance outward that a site is supposed to cover. This distance is measured from the edge of the SRAF away from the inside of the SRAF. The default value is twice the NSD. If specified, this value must be ≥ 0 .

Description

This command is used to create sites on layers of type sraf or asraf. (The term SRAF applies to both for the rest of the discussion.) It clears out any preexisting sites and places new sites on the SRAF layer’s fragments. This is intended to prepare for a later call to [`sraf_print_avoidance`](#). The default sites created by this command are farther apart and offset outward from the center of the

SRAF. This provides coverage to detect unwanted printing while reducing the overall computation cost versus normal nmOPC sites.

In previous versions of Calibre nmOPC, when implementing [sraf_print_avoidance](#), you would need to have code similar to the following:

```
NEWTAG all e1_pbar -out all_sraf
SITES_DELETE all_sraf
SITES_CREATE all_sraf -multi -numx 7 -spacing 0.008438
```

This has been entirely replaced by the following:

```
sraf_sites_create
```

TAG add | remove | ismember

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Adds, removes, or identifies members of a tag set.

Usage

TAG {add | remove | ismember} *tin id*

Arguments

- **add | remove | ismember**

Specifies whether you are adding a member to a tag set, removing a member, or identifying a member of a tag set.

- ***tin***

Specifies an input tag variable name.

- ***id***

Specifies a single ID value represented as a long integer.

Description

This command is used for individual element membership query and manipulation. You can add, remove, or determine the membership of tagged elements using this command.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

ismember returns 0 or 1 (boolean). Others have no return value.

Performance

O(1) constant access time.

Rating = fast. However, iterated calls would be slow.

Examples

```
if {$smoothed} {  
    # If the target is smoothed we always use average for every fragment  
    set epe $simvar(epeav) ; puts $dbg "smoothed target, using average"  
} elseif { [TAG ismember line_end $fragid] } {  
    # it is a line end, use the maximum EPE  
    set epe $simvar(epemax) ; puts $dbg "line end, using max"  
} elseif { [TAG ismember convex_end $fragid] } {  
    # it is a convex end, use the maximum EPE  
    set epe $simvar(epemax) ; puts $dbg "convex end, using max"  
} elseif { [TAG ismember convex $fragid] } {  
    # It's convex. Use maximum EPE  
    set epe $simvar(epemax) ; puts $dbg "convex, using max"  
} elseif { [TAG ismember space_end $fragid] } {  
    # It is a space end, use the minimum EPE  
    set epe $simvar(epemin) ; puts $dbg "space end, using min"  
} elseif { [TAG ismember concave_end $fragid] } {  
    # It is a concave end, use the minimum EPE  
    set epe $simvar(epemin) ; puts $dbg "concave end, using min"  
} elseif { [TAG ismember jog $fragid] } {  
    # It's got one concave and one convex corner. We'll use average  
    set epe $simvar(epeav) ; puts $dbg "jog, using average"  
} elseif { [TAG ismember concave $fragid] } {  
    # It's concave. Use minimum EPE  
    set epe $simvar(epemin) ; puts $dbg "concave, using min"  
} else {  
    # It doesn't touch a corner. Use average EPE  
    set epe $simvar(epeav) ; puts $dbg "not corner, using average"
```

TAG and | or | subtract

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Performs boolean operations on tag sets.

Usage

TAG {and | or | subtract} *t1* ... *tN* -out [-annotated] *tag*

Arguments

- **and | or | subtract**

A required argument that specifies the operation to be performed on the tag sets.

and — Performs a Boolean AND. Only fragments that appear in all tag sets are written to the output tag set.

or — Performs a Boolean OR. Fragments from all sets are collected in the output tag set.

subtract — Removes fragments from ***t1*** that appear in sets ***t2* ... *tN*** and writes the subset of ***t1*** to ***tag***.

- ***t1* ... *tN***

A required list of two or more tag set names.

- **-out *tag***

A required argument that specifies the output tag set name. If ***tag*** also appears in ***t1* ... *tN***, the operation results replace the initial contents.

- **-annotated**

An optional argument indicating to write fragment properties to ***tag***. “-out -annotated” 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).

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.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

None

Performance

O(N) where N is the sum of the sizes of all input sets.

Rating = fast

Examples

Example 1

This identifies fragments which are in both sets A and B, and overwrites set A.

```
TAG and A B -out A
```

Example 2

This combines annotated sets Width, width, and Tgt_width to W. A fragment that occurs in sets Width and Tgt_width is annotated with the Width property in W.

```
TAG or Width width Tgt_width -aout W
```

Example 3

This creates a subset D from set A by removing fragments that also occur in sets B and C.

```
TAG subtract A B C -out D
```

TAG fromlist

[Calibre nmOPC Tcl Scripting Commands](#)

[Tagging Controls](#)

Converts a fragment ID list to a tag set.

Usage

TAG fromlist *listin* -out *tag*

Arguments

- ***listin***
Specifies a Tcl list of single fragment IDs.
- **-out *tag***
Specifies an output tag set to contain members in the input list.

Description

This command converts a list of fragment IDs (long integer) into a tag set and can be used to verify that a tag set exists. NEWTAG commands that do not tag any fragments will not produce the tag set names given by -out. This can cause a runtime error if the nonexistent tag set is used in a subsequent operations.

Defining an empty list and then creating all tag sets from it in advance can prevent this kind of problem.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

None

Performance

O(N) where N is the size of the list.

Rating = fast

TAG size

Calibre nmOPC Tcl Scripting Commands

Tagging Controls

Returns tag set size.

Usage

TAG size *tin*

Arguments

- *tin*

Specifies a tag variable name.

Description

This command returns the size of a specified tag set.

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

Returns the size as an integer.

Performance

O(1) constant time

Rating = fast

TAG tolist

[Calibre nmOPC Tcl Scripting Commands](#)

[Tagging Controls](#)

Converts a tag set to a list of fragment IDs.

Usage

TAG tolist *tin* -out *list*

Arguments

- ***tin***
Specifies an input tag set to convert to a Tcl list fragment IDs.
- **-out *list***
Specifies the name of the list of fragment IDs to output.

Description

This command converts a tag set into a list of fragment IDs. This can be useful if you want to create an empty tag set (using an empty list as input).

Note

 This command can only be used within a [lapi_exec_tcl](#) script block.

Tcl Results

None

Performance

O(N) where N is the size of the set ***tin***.

Rating = fast

TAG_FRAG_SRC

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations, Tagging Controls

Finds all fragments tagged with a specific fragment_corner source number and places them in a tag set.

Usage

TAG_FRAG_SRC *layer source_number -out tag*

Arguments

- ***layer***

A required argument that specifies the name of a layer containing fragments.

- ***source_number***

A required argument used to identify the source of the fragmentation. The value must be an integer from 0 to 255, inclusive. 0 identifies fragments created by an operation that did not use a source identifier.

- ***-out tag***

A required argument that specifies a name for the variable to hold the output tag set.

Description

The optional TAG_FRAG_SRC command finds all fragments on a layer with the specified source number. The source number is set by the “source” argument in [fragment_corner](#) rules.

Examples

Example 1

This command creates a tag set named type7 that contains fragments on m3 marked with source number 7.

```
TAG_FRAG_SRC m3 7 -out type7
```

To mark fragments with a source number 7, the denseopc_options block should contain a fragment corner rule with “source 7”, like so:

```
fragment_corner convex convex length < 0.8 fragment 0.10 0.10 source 7
```

Example 2

To find all fragments that were not created by fragment_corner rules, add the source option to all of the fragment_corner rules and then look for fragments with a source number of 0.

```
TAG_FRAG_SRC m3 0 -out not_intrafeature
```

There is no requirement that each rule have a unique source value; for example, if you are looking for implicit (maximum length) fragmentation, all fragment_corner rules could use source 1.

TAG_MRC_SRC

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations, Tagging Controls

Identifies which MRC rule blocked a fragment when used in conjunction with MRC_RULE ... source.

Usage

TAG_MRC_SRC tag_in source_number {-out | -aout} tag

Arguments

- **tag_in**

A required argument that identifies the initial tag set of fragments.

- **source_number**

A required argument used to identify the source of the fragmentation. The value must be an integer from 1 to 1024, inclusive. Use 1 to return all blocked fragments.

When **source_number** is 1, only **-aout** is supported.

- **-out | -aout tag**

A required argument that specifies a name for the variable to hold the output tag set. The **-aout** form creates an annotated tag set with a property of **source_number**.

Description

The optional TAG_MRC_SRC command creates a tag set from fragments that were not able to be moved because of MRC_RULE constraints. The tag set can either be specific to a rule or include all blocked fragments.

For annotated tag sets, the source number of the blocking “use” clause is attached as a property. Fragments that are blocked by a rule that did not include “source number” have their property set to 1. Only the first rule that blocked the fragment is attached.

Examples

For all examples, assume the following MRC_RULE statements:

```
MRC_RULE internal tAll tAll {  
    projecting 0.014 use 0.043 opposite source 2  
    not_projecting use 0.041 euclidean source 3  
}  
  
MRC_RULE external tAll tAll {  
    projection 0.014 use 0.042 opposite source 4  
    not_projecting use 0.041 euclidean source 5  
}
```

Example 1

This command creates a tag set named mrc_rule_src_3 from tag set tAll. It contains fragments marked with source number 3.

```
TAG_MRC_SRC tAll 3 -out mrc_rule_src_3
```

The mrc_rule_src_3 tag set is not annotated, but the name makes clear which MRC_RULE is the origin of the block.

Example 2

To find all fragments that were blocked by an MRC_RULE constraint, add the source option to the rules and call TAG_MRC_SRC with “1”. If you output an annotated tag set, the blocking rule can be read as a property.

```
OPC_ITERATION 1
TAG_MRC_SRC tAll 1 -aout blockedBy
```

To see the values, use the property keyword when writing out the layers.

Related Topics

[MRC_RULE \(Custom Scripting Command\)](#)

[NEWTAG blocked](#)

[Tag Set Properties](#)

TARGET_CONTROL

Calibre nmOPC Tcl Scripting Commands

Target Modification

Changes the target value (ideal placement) of specified fragments.

Usage

```
TARGET_CONTROL tag {function_slot | -applyProperty |
[-incremental] -constant {val | property(annotated_tag)[:default(val)]} }
```

Arguments

- **tag**

A required argument that specifies a tag set name.

- **function_slot**

A numeric argument identifying the target function to use for shifting the fragment location. Target functions are assigned identifiers and equations using the command **TARGET_FUNCTION**, which must occur before TARGET_CONTROL.

This argument cannot be combined with -applyProperty or -constant.

- **-applyProperty**

A keyword indicating that the adjustment is available as an annotated property. In this case, the tag set must be created using -aout to attach a value to the fragment.

This argument cannot be combined with functions or -constant.

- [-incremental] **-constant** {val | property(annotated_tag)[:default(val)]}

A required argument specifying to apply a constant shift.

-incremental — An optional keyword that specifies that the retargeting value is to be added to the existing retargeting value.

val | **property(annotated_tag)[:default(val)]** — A required argument that can be specified in one of two ways.

val is a number in user units. The value is applied to all fragments in **tag**.

property(annotated_tag)[:default(val)] reads the value to apply from an annotated tag set. If **tag** is annotated it can be used here, but this is not required. Use **:default(val)** to provide a shift value for fragments in **tag** but not in **annotated_tag**. Without **:default**, fragments that do not have an annotation do not receive an updated shift value.

For simple individual target controls, -constant does not have a limit on the number of target controls. The **adjust_target_control_sites** command can specify how dense sites are regenerated with -constant in cases that use **algorithm** 2 or PV Band PWOPC.

This argument cannot be combined with functions or -applyProperty.

Description

TARGET_CONTROL maps fragments in a tag set to a new location. This can be done in one of three ways:

- Using a target function.
- Passing the adjustment values as an annotated tag set.
- Shifting all fragments the same amount.

All subsequent DENSE_EPE and OPC_ITERATION commands measure against the new location instead of the normal or retargeted target.

This command has the following effects:

- TARGET_CONTROL overrides “retarget_layer” only for specified fragments.
- Subsequent OPC_ITERATIONS, DENSE_EPE, and DENSE_CD commands consider TARGET_CONTROL when measuring EPE and CD.
- FRAGMENT delete and FRAGMENT split invalidate all previously performed TARGET_CONTROL settings. They reset back to retarget_layer (if defined) or the original target.

Sites (including default sites) must be deleted and then manually re-created for involved tag sets before OPC correction and site-related tagging unless [algorithm](#) 0 or 1 is explicitly set or the setup file does not include PV Band PWOPC.

Tcl Results

None

Performance

O(N) where N is the size of the input set.

Rating = medium

Examples

Example 1

The following example assumes line_end1, line_end2, and sram_end have been created.

The TARGET_CONTROL command is used to redefine targets for those feature types, then OPC_ITERATION is used to do 4 iterations using new target.

```
TARGET_FUNCTION 1 quadratic -0.01 -0.1 0
TARGET_FUNCTION 2 quadratic -0.01 -0.01 -0.005
TARGET_CONTROL line_end1 1
TARGET_CONTROL line_end2 1
TARGET_CONTROL sram_end 2
OPC_ITERATION 4
```

Example 2

Using the -constant option makes it unnecessary to define TARGET_FUNCTION explicitly for constant shifts. Typically, TARGET_CONTROL required TARGET_FUNCTION to be defined in advance, as in the following example:

```
TARGET_FUNCTION 1 constant -0.01
TARGET_CONTROL line_end1 1
```

With the -constant option, these statements can be combined:

```
TARGET_CONTROL line_end1 -constant -0.01
```

Example 3

This example demonstrates using annotated tag sets for TARGET_CONTROL. Horizontal fragments receive a shift of 0.01 and vertical fragments -0.01. Fragments not at 0 or 90 degrees are not updated.

```
NEWTAG horizontal all_frags -out horiz_frags
NEWTAG vertical all_frags -out vert_frags

NEWTAG expression all_frags {
    horiz_frags?0.010:(vert_frags?-0.010:0.0)} -aout targetWeight

TARGET_CONTROL all_frags -constant property(targetWeight)
```

Related Topics

[retarget_layer](#)

[TARGET_FUNCTION](#)

target_curve

Calibre nmOPC Tcl Scripting Commands

Target Modification

Creates a curve from the target based on the settings and moves sites to the new curve.

Usage

```
target_curve [pattern num] [-moved] criticalDistance um
  [cornerRadius val] [cornerRadiusConcave val] [cornerRadiusConvex val]
  [stepRadiusConcave value stepRadiusConvex value] [minStepTanDist val]
  [jogRampConcave value jogRampConvex value] [colinearAngleTolerance degrees]
  [linearRatio {true | false}] [lineEndRatio val] [spaceEndRatio val]
  [lineEndWidth value] [spaceEndWidth value] [maxJog val] [jogMode {1 | 2}]
  [preferMid {true | false} [midSpanExt value] [maxMidSpan value]]
  [roundCorner {true | false} [roundedCornerRadius value]]
  [{emulate | spline} [concave_adjacent val] [convex_adjacent val]]
  [-tag name override...]
  [-outLayer name]
```

Arguments

- pattern *num*

An optional argument used for multi-patterning. Set *num* to limit the site move to the sites of a particular pattern. The *num* value must match that of a layer statement.

- -moved

An optional argument that causes the curve to be generated using the current position of the opc layer's fragments. This can be useful for sequential retargeting. By default, sites are moved on to the generated curve. This can be disabled with “[adjust_moved_target_sites no](#)”.

The -moved argument cannot be used if pattern is set to a target layer.

- **criticalDistance *um***

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

- cornerRadius *um*

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`

An optional argument that specifies the radius of curvature for concave corners. The radius of curvature used to estimate the achievable target. 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`

An optional argument that specifies the radius of curvature for convex corners. The radius of curvature is used to estimate the achievable target. 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.

- `stepRadiusConcave value stepRadiusConvex value`

An optional pair of arguments that allow setting a smaller value than `cornerRadius` for S-steps. (An S-step is a concave corner and a convex corner separated by a single fragment.) You cannot use `stepRadiusConcave` and `stepRadiusConvex` separately.

The `stepRadius` arguments are ignored for overlapped S-steps when `roundCorner` true is set.

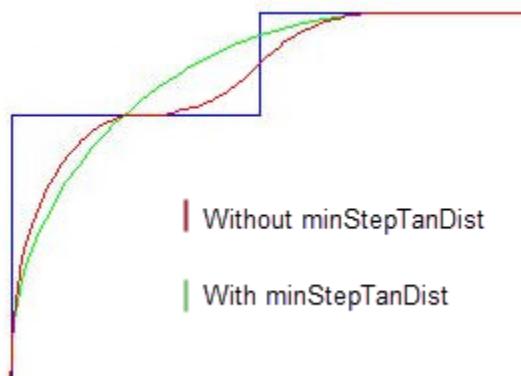
The setting for both arguments must be in the range defined by the minimum and maximum of `cornerRadius`, `cornerRadiusConcave`, and `cornerRadiusConvex`. If they are set to a value outside the range, the run issues a warning and uses the nearest end of the range.

- `minStepTanDist val`

An optional argument that sets a minimum tangent distance for both legs of a step shape. If the tangent distance on either side is not met, the step is smoothed. In effect, it defines the minimum curvature.

The default value is 0, no required minimum distance.

Figure 5-17. Effect of minStepTanDist



- jogRampConcave *value* jogRampConvex *value*

An optional pair of arguments that allow setting different values for the distance between the tangent point and the jog corner for concave and convex jog corners. (Whether an edge segment is a fragment or a jog is controlled by maxJog.) You cannot use jogRampConcave and jogRampConvex separately.

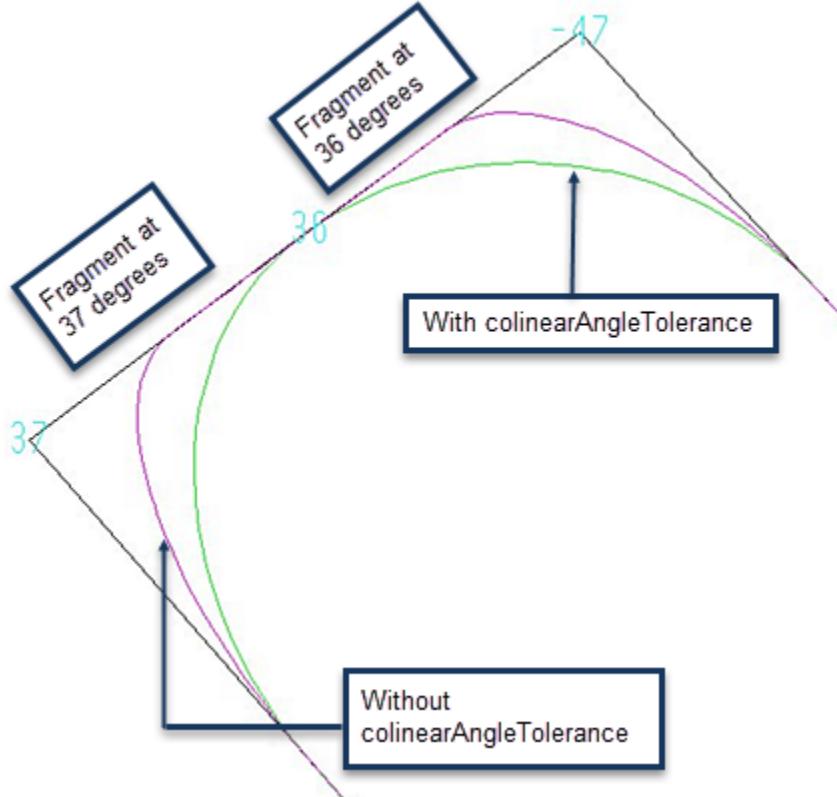
The setting for both arguments must be in the range defined by the minimum and maximum of cornerRadius, cornerRadiusConcave, and cornerRadiusConvex. If they are set to a value outside the range, the run issues a warning and uses the nearest end of the range.

- colinearAngleTolerance *degrees*

An optional argument 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*

An optional argument that 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.

- **spaceEndRatio *val***

An optional argument 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.

- **lineEndWidth *value***

An optional argument that specifies 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***

An optional argument that specifies 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 $2 * \text{cornerRadiusConcave}$.

- **maxJog *val***

An optional argument that sets 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 the value of jog_freeze (if specified) or approximately $0.292893 * \text{radius}$, where radius is the minimum of cornerRadiusConcave, cornerRadiusConvex, and cornerRadius.

- **jogMode {1 | 2}**

An optional argument that switches modes of jog handling.

1 — Keep tangent points on the first fragment of an edge. If there is a jog on the line-end near the corner, tangent points will be before the jog. This is the default.

2 — Tangent points on the first fragment of an edge are treated the same as other tangent points and appear a standard distance from the corner. This makes the curve less sensitive to small jogs.

- **preferMid {true | false}**

An optional argument that, if set to true, 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 is may be a more appropriate fit than curves that are typically used. The default value for this parameter is false.

- **midSpanExt *value***

An optional argument that 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 (*value* 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 *value***

An optional argument that 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).

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

An optional argument that causes Calibre nmOPC 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.

- **{emulate | spline} [concave_adjacent *val*] [convex_adjacent *val*]**

An optional argument set to control site placement based on offset.

emulate — When shifting sites, use a continuous angle-weighted distance function.
There is no limitation to degree changes.

spline — When shifting sites, use a continuous angle-weighted distance function but do not shift too far from the original OPC edges.

Both concave_adjacent and convex_adjacent require that emulate or spline be specified. See “[retarget_layer](#)” on page 334 for a detailed discussion of emulate and spline.

concave_adjacent *val* — An optional argument to emulate or spline that causes fragments wholly within *val* from a concave corner to be treated as concave adjacent. Concave adjacent fragments have their sites shifted based on minimum offset.

convex_adjacent *val* — An optional argument to emulate or spline that causes fragments wholly within *val* from a convex corner to be treated as convex adjacent. Convex adjacent fragments have their sites shifted based on maximum offset.

The default distance for both concave_adjacent and convex_adjacent is 5*Nyquist.

- **-tag *name overrides...***

An optional argument set that allows you to specify different overrides for a particular tag set. Up to 16 different tags can be specified. Any number of overrides can be specified.

The arguments that can be specified within a -tag set are limited to the following:

- cornerRadiusConcave and cornerRadiusConvex
- colinearAngleTolerance
- jogMode
- jogRampConcave and jogRampConvex

- linearRatio
- lineEndRatio
- lineEndWidth
- maxMidSpan
- midSpanNext
- preferMid
- roundCorner
- roundedCornerRadius
- spaceEndRatio
- spaceEndWidth
- stepRadiusConcave and stepRadiusConvex
- -outLayer *name*
 - An optional argument that writes the curve to a layer named *name* in the layout output.

Description

This optional command is similar to setlayer curve_target, but uses the whole curve to determine site placement. For setlayer curve_target if the distance between target and OPC layers is large enough, corners may not receive enough sites for good results. The target_curve command, in contrast, can redistribute or even add EPE sites. Additionally, setlayer curve_target runs before any OPC iterations whereas target_curve can generate curves during iterations, including the results from TARGET_CONTROL, and output a retarget layer or move EPE sites without creating a layer for the curve.

The target_curve command moves EPE sites. This can cause paired sites to become misaligned. If you are performing checks that rely on site pairs, regenerate them after using target_curve.

This command cannot be used with retarget_layer.

Examples

The following example creates a smoothed target layer for a layer with a minimum feature size of 40 nm:

```
target_curve criticalDistance 0.04 cornerRadius 0.05 \
    lineEndRatio 0.6 -tag tag_1 cornerRadiusConvex 0.08 \
    -tag tag_2 cornerRadiusConvex 0.04
```

Related Topics

[retarget_layer](#)

TARGET_FUNCTION

[Calibre nmOPC Tcl Scripting Commands](#)

[Target Modification](#)

Associates a target function with a function slot.

Usage

TARGET_FUNCTION *slot type a [b [c]]*

Arguments

- ***slot***
Specifies a target function slot. Values of 1 through 255 are valid.
- ***type***
Specifies the target function type, which describes how the new target is to be placed. This is to specify manual retarget functions that can be set on individual fragments for custom retargeting. This may be one of the following functions:
 - constant — The entire fragment gets retargeted by the specified amount. A constant target function will output a box.
 - linear — One end of the fragment receives a smaller amount of retargeting than the other end. All points in-between are retargeted by the amount being linearly interpolated between end points of the fragment. This will output a trapezoid or two triangles.
 - quadratic — Same as linear, but interpolation between the end points is according to quadratic equation. This may output up to three triangles or trapezoids.
 - constant_retarget — Same as constant, but the interpolation is done on the retarget_layer instead.
 - linear_retarget — Same as linear, but the interpolation is done on the retarget_layer instead.
 - quadratic_retarget — Same as quadratic, but the interpolation is done on the retarget_layer instead.
- ***a***
Specifies displacement (for constant) or the first point displacement (for linear and quadratic).
- ***b***
Optionally specifies the second point displacement (for linear and quadratic). This is invalid for constant.
- ***c***
Optionally specifies the middle point displacement (for quadratic). Otherwise, it is invalid.

Description

This command defines a target function and binds it to a target function slot to modify the OPC target.

Note

 For PV Band PWOPC, sites (including default sites) must be deleted and then manually re-created for involved tag sets before OPC correction and site-related tagging.

Tcl Results

None

Performance

O(1)

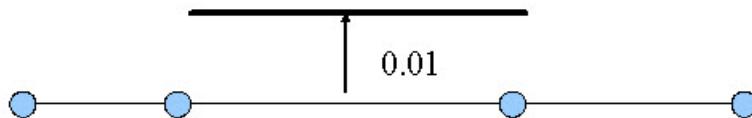
Rating = fast

Examples

The following example uses the constant parameter.

```
TARGET_FUNCTION 1 constant 0.01
TARGET_CONTROL t1 1
```

Figure 5-19. TARGET_FUNCTION Example 1



The following example uses the linear parameter.

```
TARGET_FUNCTION 2 linear 0 0.01
TARGET_CONTROL t1 2
```

Figure 5-20. TARGET_FUNCTION Example 2

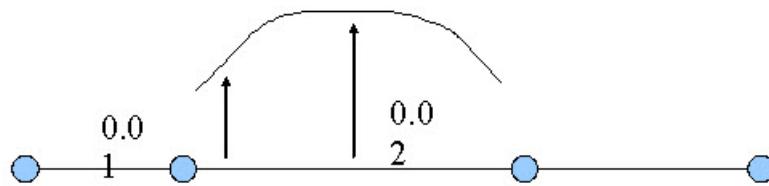


The following example uses the quadratic parameter.

```
TARGET_FUNCTION 3 quadratic 0.01 0.01 0.02
TARGET_CONTROL t1 3
```

In this example, the 1st and 2nd values are displacements at endpoints, and the 3rd value is displacement in middle.

Figure 5-21. TARGET_FUNCTION Example 3



TDBIAS

Calibre nmOPC Tcl Scripting Commands

Fragment-Based Operations

Sets a rule-based fragment displacement.

Usage

TDBIAS *layer* [-moved] [-force] *rules*

Arguments

- ***layer***
Specifies the layer to bias.
- **-moved**
An optional argument that measures the moved edges while performing distance measurements. The movement in this case is relative to the moved position.
- **-force**
An optional flag that forces edge movement without checking constraints.
- ***rules***
Specifies TDBIAS rules. The format matches the format used for the OPCBIAS SVRF command.

Description

This command sets the displacement (which requires a call to FRAGMENT_MOVE unless the -force flag is used). The TDBIAS command will move a maximum amount which is governed by the amount specified by the [max_iter_movement](#) command, unless the -force option is used.

The rules are input as a single text block with the following syntax:

```
rules = rule*
rule = SPACE constr [WIDTH constr] [WIDTH2 constr] MOVE distance
```

Tcl Results

None

Performance

Rating = slow

Examples

```
set bias1 {  
    SPACE >= 0.090 < 0.125    WIDTH >= 0.085 < 0.100    MOVE 0  
    SPACE >= 0.125 < 0.150    WIDTH >= 0.085 < 0.100    MOVE 0.003  
    SPACE >= 0.150 < 0.200    WIDTH >= 0.085 < 0.100    MOVE 0.01  
    SPACE >= 0.200 < 0.350    WIDTH >= 0.085 < 0.100    MOVE 0.02  
}  
TDBIAS MYLAYER $bias1
```

UU2DBU

Calibre nmOPC Tcl Scripting Commands

Unit Conversions and Miscellaneous

Converts user units into database units.

Usage

UU2DBU *user_units*

Arguments

- ***user_units***

Specifies the user units (typically in microns) to be converted.

Description

This command converts the user-supplied user units (usually microns) into database units which is returned in the Tcl result.

Tcl Results

The converted number of database units as an integer.

Performance

O(1) constant time

Rating = fast

Examples

The following example writes out that a micron is 1000 dbu:

```
puts "1 micron = [UU2DBU 1] dbu"  
--> 1 micron = 1000 dbu
```


Chapter 6

Calibre nmOPC Best Practices

There are a number best practices available when using Calibre nmOPC to correct distortions on a mask input layer introduced by optical and resist processes.

Recommended Settings for Best OPC Results	614
Recommended Settings for Best Runtime	615
Impact Reduction on Quality of Jogs.....	617
EPE Measurement Method Reset	617
Small Jog Movement Prevention	618
Process Window OPC Improvement	619
Using PV Band PWOPC	621
Sites for Minimum CD Control	623
Output Violation Tags	624
Optimize the Number of Process Window Iterations	624
Optimize the Number of PW Conditions	626
Optimize Simulation Time	626
Additional Recommendations for Process Window OPC	626
Enclosure of Contacts and Vias	627
Process Window Violations	627
Optimize Results From setlayer curve.....	629
Retarget Layers and setlayer curve	632
Setup File Best Practices	634
Recommended Settings for tilemicrons per Node	634
General Setup File Recommendations	635
MRC Rule Best Practices	636
Fragmentation Best Practices.....	638
Long Edge Fragmentation	638
Asymmetric Intrafeature Fragmentation for Long Edges.....	639
Fragmentation Consistency and Tile Boundaries	641
Asymmetric Intrafeature Fragmentation on the Short Edge	643
Asymmetric Interfeature Fragmentation on the Short Edge	645
Effect of Jogs on Fragmentation and OPC Consistency	647
Layer-Based Best Practices.....	649
POLY/ACTIVE Layers	649
Contact Layers	649
OPC Output Consistency	650
Practices to Improve VEB Results	653

Use Automatic Fragmentation	653
Set the Image Grid to 10 nm	655
Perform Convergence Analysis to Find Optimal Iterations	655
Use a Hint Offset	656
Set Final feedback to -1	657
Use Resist Constraints When Needed	657
Use Corner Chipping	658
VEB and notchfill	660
SRAF Print Avoidance Best Practices	661
Starting Recommendations for SRAFs	661
Negative SRAF Handling	662
Iterations for sraf_print_avoidance	664
SRAF Fragmentation	664
SRAFs and MRC Rules	664
Wafer-Level Data and Printing Threshold	665
SRAFs and Process Window OPC	666
Simplifying a Calibre mpOPC Setup File	667

Recommended Settings for Best OPC Results

There are a number of recommended settings for Calibre nmOPC setup file commands for better performance and results.

Commands marked with “Calibre OPCverify” indicate that these are Calibre OPCverify commands, documented in the *Calibre OPCverify User’s and Reference Manual*.

Table 6-1. Recommended Settings for Best OPC Results

Command	Default Setting	Recommended Setting	Notes
algorithm	2	2	Improves output consistency and convergence.
opc_grid_multiplier	on	on	Improves output consistency.
mrc_mode	2	2	Improves output quality.
fragment_max	Nyquist*4	0.1 to 0.4 um	Smaller values can lead to faster OPC.
jog_ignore_metric	0	tech_node/4 (where <i>tech_node</i> is the technology node size)	Improves output quality.
jog_freeze	0	0.005 to 0.01	Improves output quality.
mask_sample_grid (Calibre OPCverify)	rsm	rsm	Improves output consistency and performance.

Table 6-1. Recommended Settings for Best OPC Results (cont.)

Command	Default Setting	Recommended Setting	Notes
<code>simulation_consistency</code>	0	2 (DUV) euv (EUV)	Improves output consistency with faster simulator settings.
<code>step_size</code>	1 dbu	1 dbu	Improves output quality. Value should be based on CD/EPE tolerance specification and MEEF.
<code>tilemicrons</code> (Calibre OPCverify)	100 um	Use “adjust” and set size based on technology node. See Table 6-6 in “Recommended Settings for tilemicrons per Node” on page 634 .	Improves performance. Smaller tiles lead to better scaling, but consume more memory on the primary host.
<code>feedback</code>	-0.4	Adjust per iteration	Smaller for high MEEF processes, early iterations use smaller values.
<code>mrc_rule</code> (metric)	Euclidean	euclidean “metric”	Improves output consistency.
<code>mrc_rule</code>	none	\leq <code>fragment_min</code>	Improves quality and prevents overconstrained results.

Recommended Settings for Best Runtime

The recommended setup file parameter settings described in the following table are important for creating the fastest possible nmOPC setup file runtime.

Commands marked with “Calibre OPCverify” indicate that these are the Calibre OPCverify commands, documented in the [Calibre OPCverify User’s and Reference Manual](#).

Table 6-2. Recommended Settings for Best Runtime

Command	Recommended Setting	Notes
<code>max_iterations</code>	Depends on correction scenario	It is often possible to reduce the number of iterations with careful optimization of feedback.

Table 6-2. Recommended Settings for Best Runtime (cont.)

Command	Recommended Setting	Notes
tilemicrons (Calibre OPCverify)	Use “adjust” and set size based on technology node. See “ Recommended Settings for tilemicrons per Node ” on page 634.	Smaller tiles lead to better scaling, but consume more memory on the primary host.
mask_sample_grid (Calibre OPCverify)	rsm (default)	Improves consistency.
fragment_max	0.1 to 0.4 um	Larger values mean larger interaction distance which translates to less hierarchy in the layout.
max_opc_move	optimize/minimize	Reduce to smallest value which produces quality results.
clone_transformed_cells	Depends on illumination	Disable if your illumination source is symmetrical.
progress_meter (Calibre OPCverify)	on (for predictability)	Trade-off between 1% performance loss and predictability.

Impact Reduction on Quality of Jogs

There are several different ways that jog quality can affect OPC results.

EPE Measurement Method Reset	617
Small Jog Movement Prevention	618

EPE Measurement Method Reset

As jogs introduce corner points (both concave and convex), it changes the way EPEs are measured.

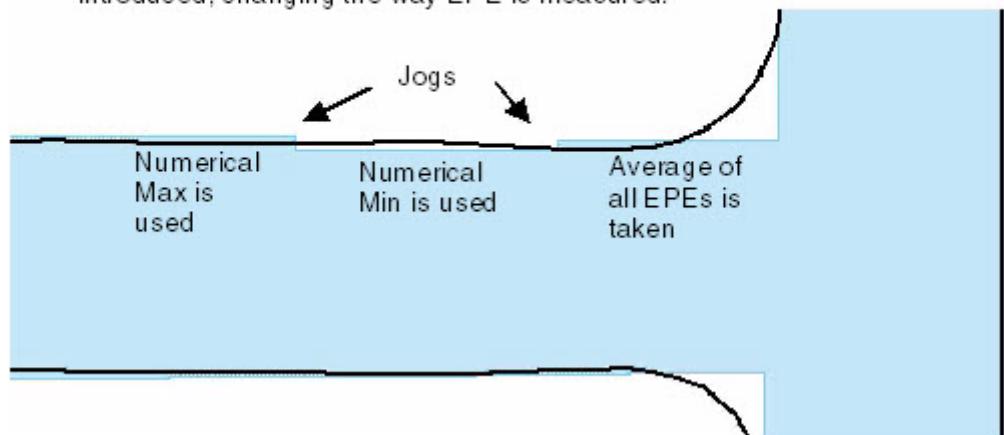
The following is a quick review of how EPE is measured on different types of fragments:

- For convex corners, the numerical maximum is used, typically the point farthest from the corner.
- For concave corners, the numerical minimum is used, typically the point farthest from the corner.
- At line ends, the numerical maximum is used, typically the center point.
- For non-corner edges, the average of all the EPEs are taken.

Thus, if jogs are present, the method by which EPE is measured changes from average (an average of all EPEs taken for a fragment) to either minimum (numerical minimum) or maximum (the numerical maximum), as illustrated in [Figure 6-1](#).

Figure 6-1. Jogs and EPE Measurement Method

This should ordinarily be a straight fragment where the average of all EPEs are taken as a measurement. However, since jogs are present, small corners are introduced, changing the way EPE is measured.

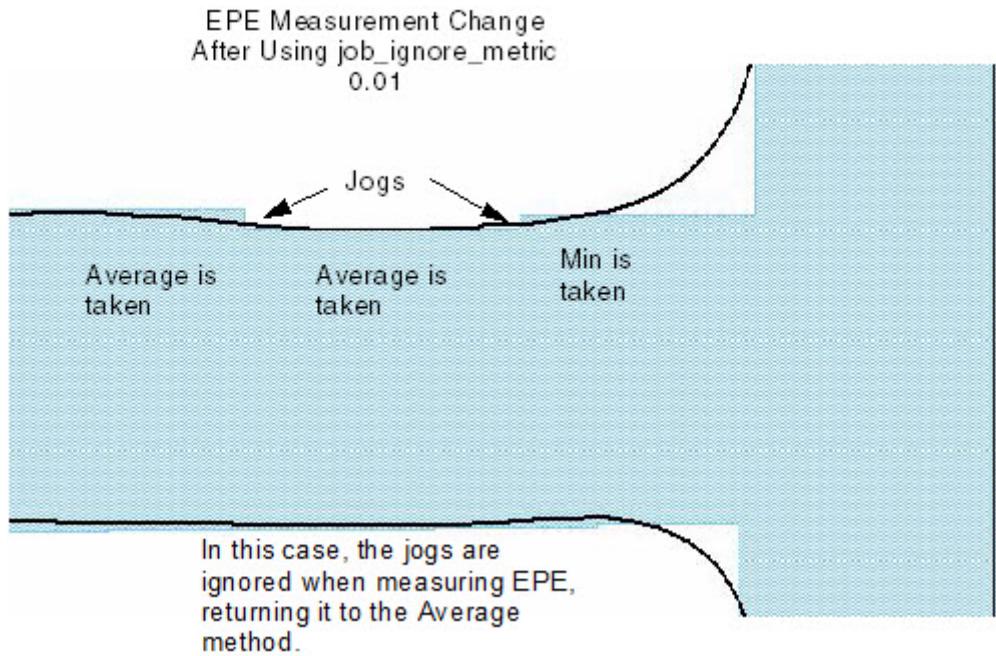


If a pinch or bridge issue is found that is nearby a jog, using the command [jog_ignore_metric](#) will ignore the jogs when setting the EPE measurement metric.

Tip

i Set `jog_ignore_metric` to $tech_node/4$ (where `tech_node` is the technology node size).

Figure 6-2. Resetting EPE Measurement Method for Jogs



Small Jog Movement Prevention

Jogs can be simulated and moved just like any other edge. However, their small size typically means that controlling the EPE on the jog edge is unimportant. Furthermore, moving the jog edges can cause the more critical neighbor edges to be blocked by MRC constraints.

To prevent small jogs from moving during OPC, use the `jog_freeze` command.

Tip

i Set `jog_freeze` to about 0.005 to 0.01 um, or Nyquist/3. You can get the Nyquist value from the AERIAL INTENSITY SAMPLING GRID entry from the output log file transcript, which is usually about 0.02-0.025 um.

Process Window OPC Improvement

The Calibre nmOPC Process Window OPC (PWOPC) capability is available to improve the litho process window. Process Window OPC is used to reduce the risk of critical failures due to a number of process variations.

The variations include:

- Focus variation
- Exposure dose variation
- Reticle CD error
- Alignment error

Critical failures to be compensated through target modification are:

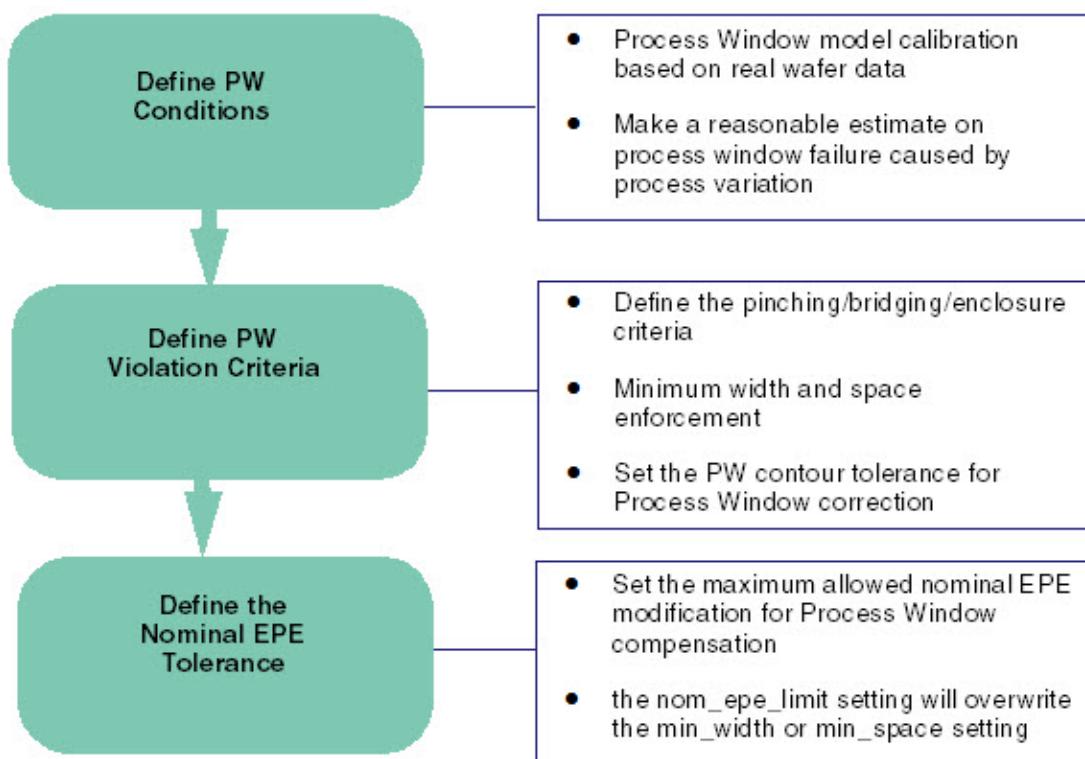
- Bridging
- Pinching
- Overlapping margins between layers

You should use PWOPC if all of the following conditions apply:

- Nominal OPC meets nominal specification but fail PW criteria
- Dealing with critical layers of advanced technology nodes
 - 45 nm node Poly and M1 layers
 - 32 nm node and below critical layers
- Nominal OPC gives good nominal results
- A good process window model is available
- Process conditions, violation criteria as well as nominal EPE tolerance information is available

When implementing PWOPC, there are a number of important steps to follow to ensure best results. These steps are summarized in [Figure 6-3](#).

Figure 6-3. Important Steps of PWOPC Implementation



The `pw_condition` command is used to create alternative process window conditions (a process window is the tolerable change in dose, focus, and mask sizing until the design fails to yield on silicon) compared with a nominal condition. When applying process windows to the OPC process, there are several methods and recommendations available to obtain better results.

It is recommended that you use two extreme process conditions without the nominal condition for PWOPC. The two process window conditions selected should cover the pinching and bridging seen from the real wafer process.

A process window condition can be a combination of dose, focus, and mask sizing variation, as shown in this example:

```
pw_condition PW1 mask_size 0.001 optical defocus02 dose 0.9 ...
```

When implementing process window conditions, always keep the runtime impact in mind.

- 0% added runtime for additional dose only variation
- ~30-40% added runtime for additional focus condition
- ~80% added runtime for each added mask sizing condition

See also the following topics:

Using PV Band PWOPC	621
Sites for Minimum CD Control	623
Output Violation Tags.....	624
Optimize the Number of Process Window Iterations.....	624
Optimize the Number of PW Conditions.....	626
Optimize Simulation Time	626
Additional Recommendations for Process Window OPC	626
Enclosure of Contacts and Vias	627
Process Window Violations.....	627

Using PV Band PWOPC

When using the `pw_condition` command to implement process window-based OPC (or PWOPC), use PV Band PWOPC. This method uses physical verification bands that are simulated using different process window conditions. This method has several key advantages.

- It is tag-based, giving more flexible control.
- Minimum CD can be specified in addition to EPE-based tolerance control.
- Sites are automatically generated.

Tip

 It is also recommended that you use `pw_condition` for 32 nm nodes and below critical layers, and likely M1 and Poly at 45 nm as well.

Simulating PV bands under different process window conditions are done inside sites. This means that sites are required in order to perform PV Band PWOPC. The site-based simulation makes the CD width/space cost function possible using this method.

Sites are automatically generated using PV Band PWOPC, and tag-based operations are supported for further control.

Prerequisites

- Process window conditions for simulation

Procedure

1. Define PWOPC (you must use the `pvband` option in the `pw_condition` command):

```
pw_condition PW1    optical OPTIC_PW dose 1.03  pvband
pw_condition PW2    optical OPTIC_PW dose 0.97  pvband
```

- Run PWOPC on a selected number of OPC iterations (using the [OPC_ITERATION](#) command). For example:

```
OPC_ITERATION 5 -PW \
    nominal_epe_limit all frags -0.003 0.003 \
    min_width all frags 0.034 \
    min_space all frags 0.037 \
    exclude tag
```

For OPC_ITERATION:

- The nominal_epe_limit for fragments belonging to this tag will be considered, but nmOPC will try to prevent the nominal EPE from exceeding the *tol_inner tol_outer* tolerance.
- The min_width option will try to ensure that the contour width across process window conditions for this tag set will be at least *width*.
- The min_space_tag option will try to ensure that the contour spacing across process window conditions for this tag will be at least *space*.
- The exclude option for fragments belonging to this tag will be ignored; only nominal EPE will be used.

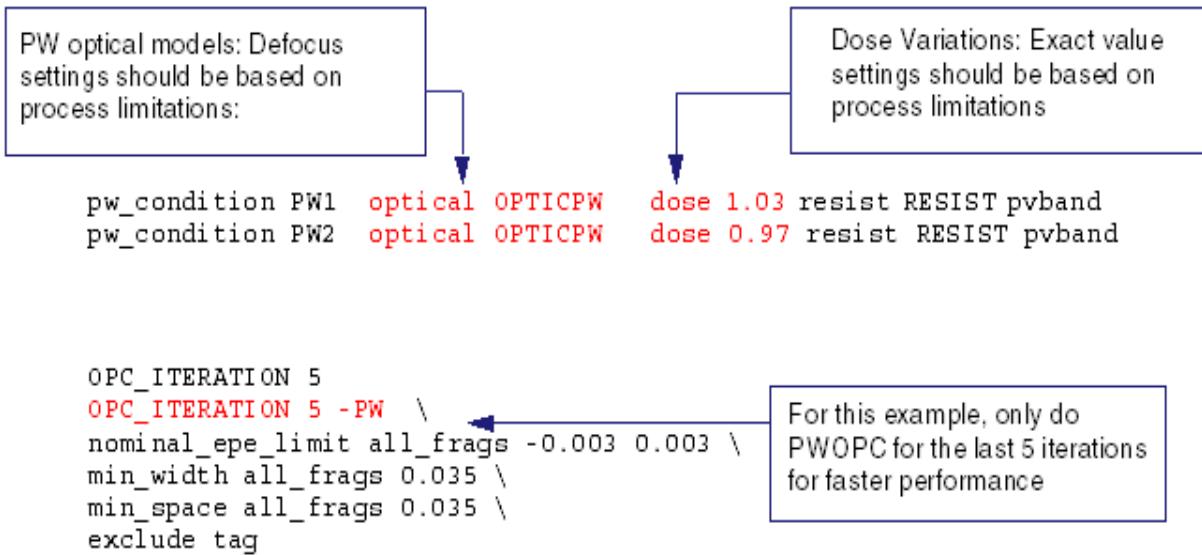
Results

Using PV Band PWOPC should result in more flexible control over adjusting process window conditions to get the most accurate EPE calculations, and will automatically generate sites.

Examples

[Figure 6-4](#) summarizes several best practices for PV band-based PWOPC.

Figure 6-4. PV Band PWOPC Example



Related Topics

[pw_condition](#)

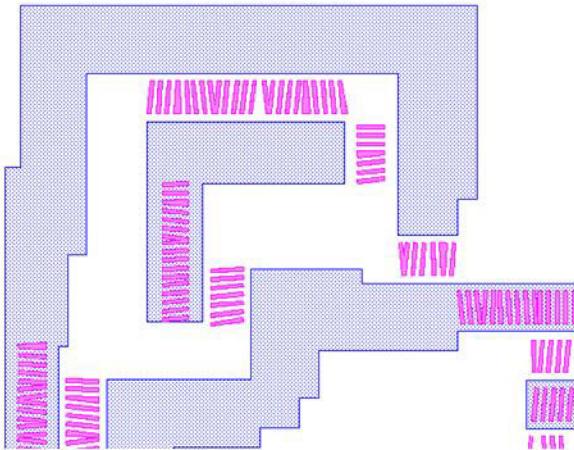
[OPC_ITERATION](#)

Sites for Minimum CD Control

For minimum CD control, a pair of sites are created for minimum CD evaluation. CD evaluation sites are only created on the location where the target CD (not contour CD) is within the range of minimum CD +/- Nyquist value.

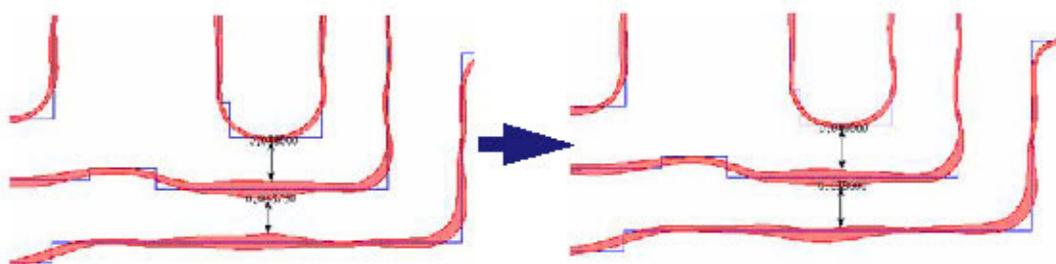
[Figure 6-5](#) illustrates an example, where the pink regions indicate the sites.

Figure 6-5. Sites for Minimum CD Control



The goal of PWOPC on these locations is to make the image meet the min_width or min_space specification. [Figure 6-6](#) shows an example of the effects.

Figure 6-6. Example of PWOPC With Minimum CD



PV Band PWOPC without min CD control
min Width = 33.5 nm

PV Band PWOPC:
min_width = 0.040 min_space = 0.040
final min Width = 39 nm
without nominal_epe_limit

The following are constraints that might affect minimum CD enforcement in PV Band PWOPC:

- No pair of sites found for CD evaluation.
- The layout target CD is out of the range of minimum CD +/- Nyquist value.
- The OPC movement restricted by nominal_epe_limit constraints.
- MRC_RULE (Custom Scripting Command) restrictions.
- Conflicts between min_width and min_space enforcement.

Output Violation Tags

One recommendation is to output violation tags.

Use the [NEWTAG sites](#) and [OUTPUT_SHAPE fragment](#) commands to output violation tags, as shown in the following examples:

```
OPC_ITERATION -PW min_width all_frags 0.034 min_space \
    all_frags 0.037 -set_epe
NEWTAG sites all_frags -wafer min_width < 0.033 -out tag3
OUTPUT_SHAPE fragment tag3 tag3 0.002 0.002

NEWTAG sites all_frags -wafer min_space < 0.035 -out tag4
OUTPUT_SHAPE fragment tag4 tag4 0.002 0.002
```

Optimize the Number of Process Window Iterations

The number of process window iterations has the largest impact on the PV Band PWOPC runtime. In general, the initial recommendation is to use a 50/50 ratio between non-process window and process window iterations settings, ranging down to a ratio of 3/12 (no-PW/PW) for good quality of results.

A typical setup for the PV Band PWOPC is as follows:

```
image nominal . . .
pw_conditions cond_1 . . . pvbnd
pw_conditions cond_2 . . . pvbnd

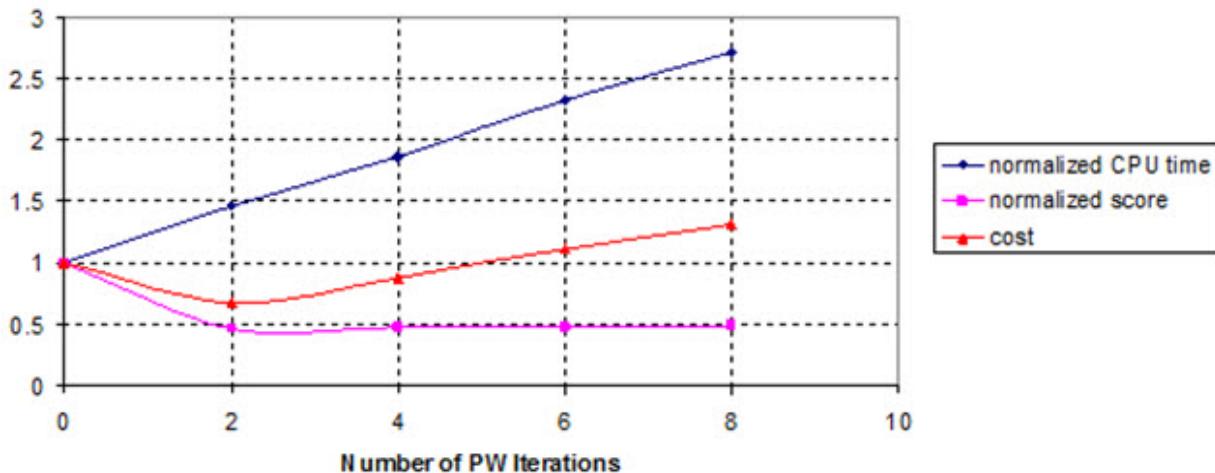
OPC_ITERATION 5
OPC_ITERATION 4 -PW exclude gates min_space connects 0.036 \
    min_width connects 0.038
```

This example shows five nominal OPC iterations, switching to four process window iterations (using OPC_ITERATION) during which no PWOPC is done for the tag set called “gates”. Space and width 36 nm/38 nm limits are also applied to the “connects” tag set.

In order to get the best choice, you can experiment where the number of PW iterations will be as a variable with one result being run time and the other a number of Calibre OPCverify bridge, pinch, or contact errors (or some combination of them).

Figure 6-7 shows a plot that summarizes one example:

Figure 6-7. Optimizing PW Iterations
Impact of the number of PW iterations



In this figure:

- Both CPU time and score are normalized to the no PW iterations run.
- Score is a cost function representing the Calibre OPCverify flag counts.
- Cost is the product of score and CPU time.
- High run time is undesirable.
- A lower score means better quality of results.

The cost function minimum at two process window iterations shows the optimal number of iterations. Before this, the quality was poor and after this run time was too high. The essentially flat score confirms there are no improvements in quality of results with the additional iterations.

Note

 Increasing the number of process window iterations beyond the optimal is one option to account for the unknown. For example, the limited calibration of the test case might not include all possible geometries, requiring a larger process window iteration number for good quality of results.

Optimize the Number of PW Conditions

The second most important factor that has a direct impact on PWOPC speed is the number of PW conditions. The more conditions that are specified, the slower the PWOPC run. Although some of the calculations in PV Band PWOPC case are concurrent, the impact of adding several conditions is significant.

Use the following rules to minimize the runtime:

- Specify only necessary process window conditions in the process window statements. There is no need to add a nominal condition; PV Band PWOPC extracts the nominal EPE from the Calibre nmOPC image command.
- Run Calibre OPCverify on a medium size test case and determine the two most extreme conditions that are responsible for a majority of the pinch and bridge flags. Once found, use these conditions in the PV Band PWOPC setup. Typically these conditions are:
 - Bridging: nominal focus, positive dose, positive mask size
 - Pinching: negative or positive focus, negative dose, negative mask size

Optimize Simulation Time

In order to make general improvements to the run time of the PV Band PWOPC, try switching to an alternative simulator settings simulation_consistency keyword. This makes use of several enhancements to the simulation engine.

The general recommendations are:

- Use Calibre version release 2012.2 and later as the simulation engine makes faster intensity interpolations.
- Use simulation_consistency 2.

Additional Recommendations for Process Window OPC

There are several more recommendations that can affect the performance of process window-based OPC.

- When setting dose variations using the dose parameter in `pw_condition`, exact value settings should be based on fabrication process limitations.
- When creating nominal and process_window optical models using the optical parameter in `pw_condition`, defocus settings should also be based on fabrication process limitations.

- For PV Band PWOPC, sites (including default sites) must be deleted and then manually re-created for involved tag sets before OPC correction and site-related tagging. As default sites are created before execution of tag scripts in PV Band PWOPC, the OPC process will not initially recognize target-based controls such as TARGET_CONTROL and TARGET_FUNCTION (both are tag script commands). This practice allows for these target controls to function correctly.

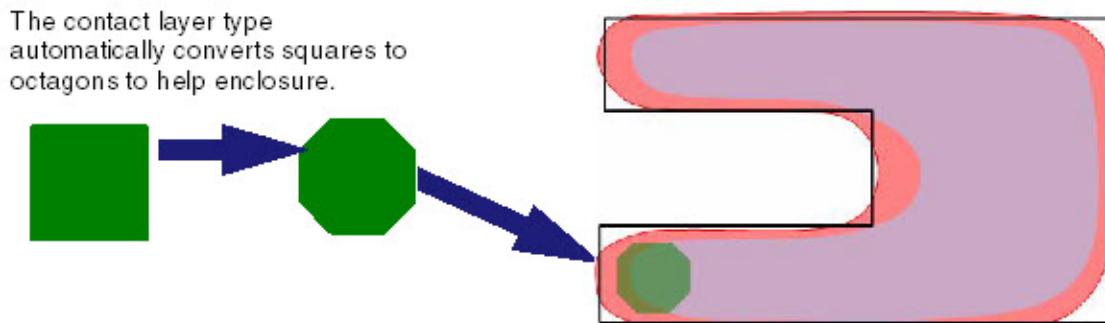
Enclosure of Contacts and Vias

For applications where enclosure of another layer (such as M1 enclosure of contacts and vias) is critical, use the wafer_enclose command. For example:

```
layer CA contact clear
wafer_enclose CA by -0.005
wafer_enclose CA by -0.005
```

The layer type “contact” converts the square shapes to octagons (the corners are chopped) to help fit the contact in the enclosure (see [Figure 6-8](#)).

Figure 6-8. Enclosure of Contacts and Vias



In PV Band PWOPC, default sites are created before the execution of tag scripts. This order means that default sites do not see target controls, because they rely on tag script and the dynamic logic of tagging. Refer to the [wafer_enclose](#) command for information on wafer enclosure.

Process Window Violations

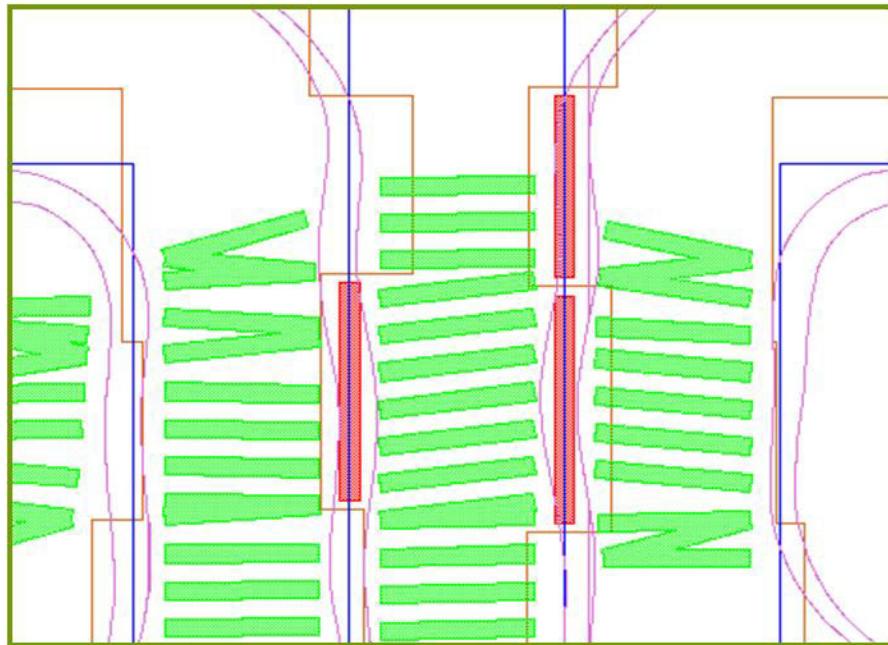
If you encounter an OPC violation for your process window, check for the following:

- Are there any MRC rule restrictions?
- Is the OPC movement restricted by nominal_EPE_limit constraints?
- Is the target edge outside the range of minimum CD constraint +/- Nyquist?
- Are there conflicts between min_width and min_space enforcement?

PV band mode provides flexibility to output violation tags as shown in Figure 6-9. Use NEWTAG sites and OUTPUT_SHAPE fragment to output the violation tags (shown in red). For example:

```
OPC_ITERATION -PW min_width all_frags 0.033 min_space all_frags 0.035 \
    -set_epe
NEWTAG sites all_frags -wafer min_width < 0.033 -out tag3
OUTPUT_SHAPE fragment tag3 tag3 0.002 0.002
NEWTAG sites all_frags -wafer min_space < 0.035 -out tag4
OUTPUT_SHAPE fragment tag4 tag4 0.002 0.002
```

Figure 6-9. Violation Tags



Related Topics

[NEWTAG sites](#)

[OUTPUT_SHAPE fragment](#)

Optimize Results From setlayer curve

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 and jogs. The setlayer curve command implements a B-spline, and control points are derived from existing polygon vertices.

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 6-3 lists the recommended general setlayer curve parameter settings to get the best overall curvature fit.

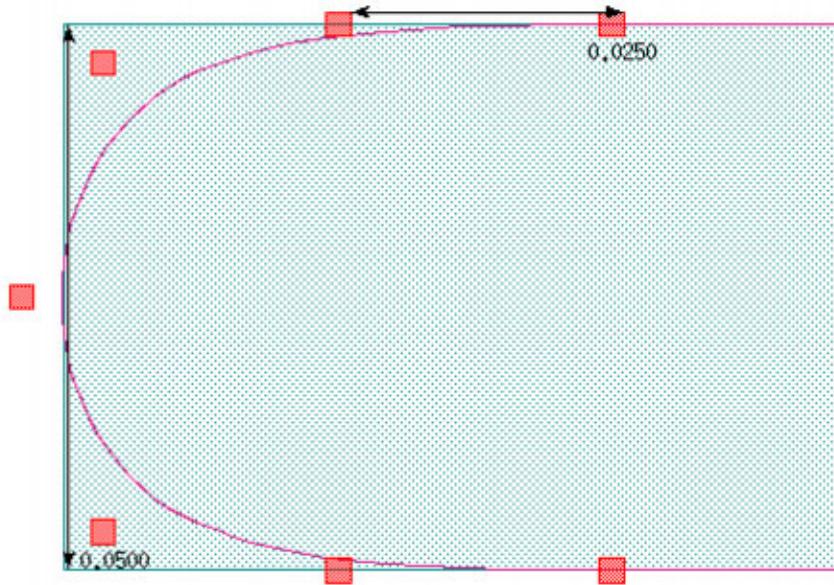
Table 6-3. General Recommended Settings for setlayer curve

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 minimum CD	Specifies the distance between additional control points inserted to the left and right of an original polygon point. Longer distances increase rounding.
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 or 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.
scale1	2.0	If a line end or space end is smaller than $cpdist * scale1$, control points on adjacent edges are put at a $cpdist * scale2$ distance, instead of the $cpdist_value$. See Figure 6-10 for an example of line_end and space_end scaling.

Table 6-3. General Recommended Settings for setlayer curve (cont.)

Parameter	Recommended Setting	Notes
scale2	0.5	Any line/space end that has width < scale1 * cpdist has a control point inserted on an adjacent edge at a distance of scale2 * width. See Figure 6-10 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 6-10. Example of Line/Space End Scaling



[Table 6-4](#) lists the recommended setlayer curve parameter settings for better handling of line ends and space ends.

Table 6-4. 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.

Table 6-4. Line/Space End-Specific Recommended Settings for setlayer curve (cont.)

Parameter	Recommended Setting	Notes
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.
length	[1.0 to 1.5]*min CD	Specifies the fragment length of the line end or space-end edge. See Figure 6-11 .

Figure 6-11. Length and Width Representation for setlayer curve



[Table 6-5](#) lists the recommended parameter settings to reduce jogs in the rendered curve.

Table 6-5. 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*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	0.000	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.
length	roughly 0.6*min CD	Specifies the fragment length of the feature.

Note



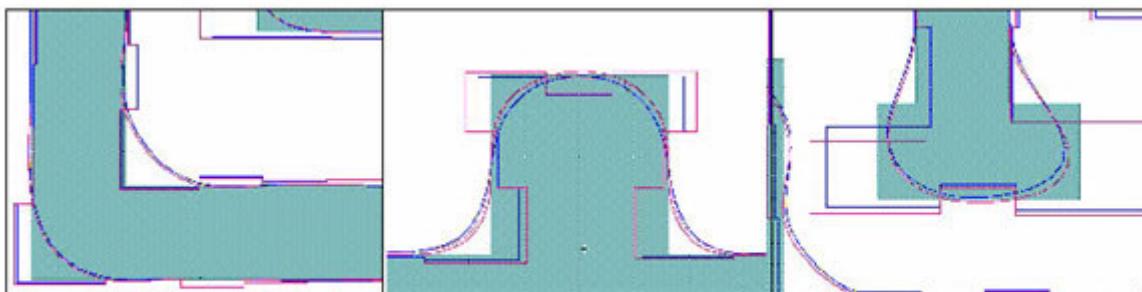
A concave/convex-specific setting is generally not needed unless a different curvature is desired.

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 OPC recipes.

Figure 6-12 shows the effects of using `retarget_layer emulate`.

Figure 6-12. 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_retarget_region \
    -out frag_ignore_retarget
TARGET_FUNCTION 1 constant 0.0
TARGET_CONTROL frag_ignore_retarget 1
```

Setup File Best Practices

There are a number of recommended settings for setup file commands.

The Calibre nmOPC setup file can be considered essentially two parts:

- The [LITHO DENSEOPC](#) section contains layer info, simulation settings, models, miscellaneous optional setlayer commands, and so on.
- The [denseopc_options](#) command block contains options and commands specific to nmOPC (MRC, tagging, fragmentation, feedback, and so on).

For more information on the Calibre nmOPC setup file, refer to the “[Configuring Calibre nmOPC](#)” chapter.

Recommended Settings for tilemicrons per Node.	634
General Setup File Recommendations.....	635

Recommended Settings for tilemicrons per Node

The selection of tile size is critical for obtaining optimum turn-around-time in full chip OPC runs. If the tile size is too large, the remote memory consumption (in MTflex runs) can be so high that the remote CPU could go into swap (with large impact on run time). If the tile size is too small, more memory on the primary host is consumed, with a similar negative impact on performance.

The best settings specify the tile size according to the technology node, with smaller tile sizes chosen for each smaller technology node. This Moore's Law scaling of the tile size selects tile sizes which contain roughly an equivalent amount of data per tile.

The current recommendations for optimum tile size settings are shown below, and are designed to keep peak memory consumption at less than 75%. It is recommended that you set the Calibre OPCverify command [tilemicrons](#) to the following values (based on node) listed in [Table 6-6](#) (this is intended for Calibre nmOPC only).

Table 6-6. Hardware Requirements and tilemicrons Settings (Calibre nmOPC)

Technology Node	Hardware Requirement (with Hyperthreading)	Hardware Requirement (Without Hyperthreading)	tilemicrons Setting (um)
45-40 nm	4 GB/core	2 GB/core	45 to 55
32-28 nm	6 GB/core	3 GB/core	35 to 40 (30 for M1)
22-20 nm	8 GB/core	4 GB/core	25 to 30 (20 for M1)
14 nm	8 GB/core	4 GB/core	20
10 nm	TBD	TBD	15

For double-exposure cases, use the smaller values.

General Setup File Recommendations

There are a number of recommendations for the Calibre nmOPC setup file to obtain better results.

- [progress_meter](#): This Calibre OPCverify command allows you to monitor the progress of the run. It does come with a small (1%) performance penalty. Leave the command enabled to improve runtime predictability.
- Consistency Settings: Use the settings described in the section “[OPC Output Consistency](#)” on page 650 to improve the output consistency of nmOPC.
- [fragment_max](#): Set from 0.1 to 0.4 um to minimize runtime. The interaction distance used to control promotion (flattening) during OPC must be greater than the fragment_max value and the optical diameter.
- [mrc_mode](#) 2: The mrc_mode command uses the best movement code for high-quality MRC controls (fewer MRC errors after OPC). This is also the default setting.
- [jog_ignore_metric](#) and [jog_freeze](#): Set these commands to small values (0.01 um to 0.02 um) to control edge movement and EPE measurement on jogs and jog adjacent fragments.
- [feedback](#) or [FEEDBACK](#): Use lower values during initial and final iterations. For example:

```
feedback -0.3 -0.4 -0.5 -.5 -0.5 -0.5 -0.5 -0.3
```

Contact and via layers require very low values (-0.1 to -0.25 typically).

- By default, global fragmentation settings for [fragment_max](#) and [fragment_min](#) are calculated automatically by Calibre nmOPC. However, if you are using custom fragmentation ([fragment_layer](#)), then these global settings are not passed to the [fragment_layer](#) blocks. You must explicitly set fragment_max and fragment_min both inside and outside the block. For example:

```
denseopc_options opc_opt {
    ...
    image optical duv193_nom dose 1 resist r.mod
    ...
    fragment_min 0.025
    fragment_max 0.2
    fragment_layer m.opc.t_fill {
        fragment_min 0.025
        fragment_inter -interdistance 0.5 -num 0 -internalonly
        fragment_corner convex 0.05 0.05 0.05 0.00 ...
        fragment_max 0.2
    }
    ...
}
```

Whenever you use `fragment_layer`, you are creating custom fragmentation. Calibre nmOPC ignores global `fragment_min` and `fragment_max` values within the `fragment_layer` blocks. Calibre nmOPC also generates an error if these values are not explicitly defined.

- **`step_size`**: This command is used to control the resolution of the OPC correction. All simulated EPE values will be converted into a mask displacement in increments of `step_size`. Smaller values result in higher correction accuracy (but they may have an impact on performance). The smallest allowed value is 0.025 nm.

The ultimate limit of OPC correction accuracy can be estimated as:

$$\text{Correction Accuracy} \approx \text{step_size} * \text{mask error enhancement factor (MEEF)} .$$

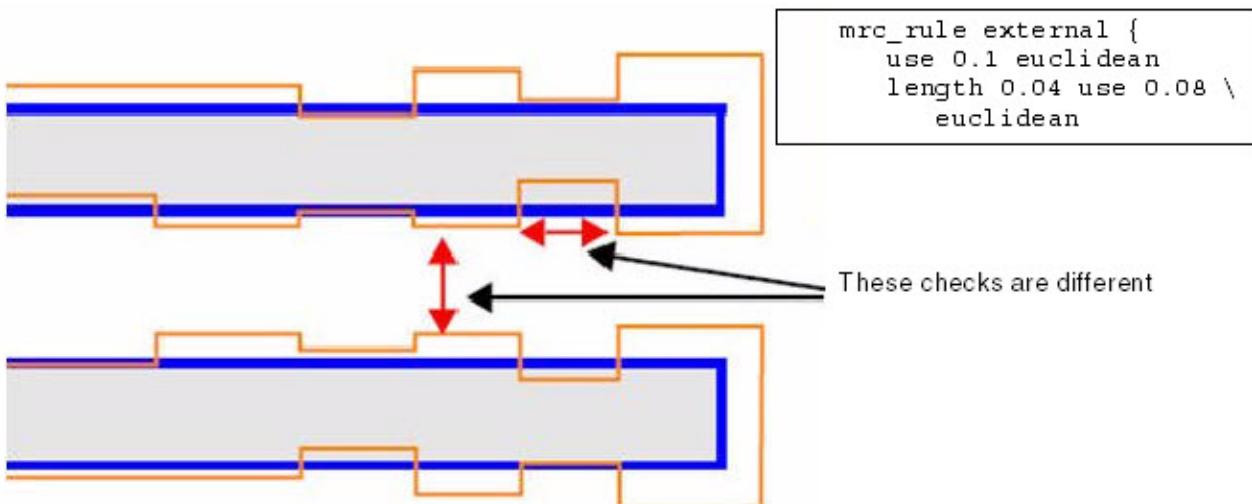
MRC Rule Best Practices

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 following recommendations apply to using `mrc_rule` to set constraints:

- MRC constraints should always be set to less than `fragment_min`. Failure to do so will overconstrain the correction.
- The input layout must not contain polygons which are already in violation of MRC rules.
- For `mrc_rule`, use length-based MRC rule specifications to prevent notches and corners from limiting edge movement, as shown in [Figure 6-13](#).
- When specifying MRC constraints for nub or jog edges, be sure the “length” criteria in the MRC rules matches the length definition of the nubs or jogs. For example, if the jog length definition is 5 nm, the “length” criteria should also be 5 nm so that the subcheck can be applied properly at the nub or jog edges.

Figure 6-13. Using Length-Based MRC Rules



Fragmentation Best Practices

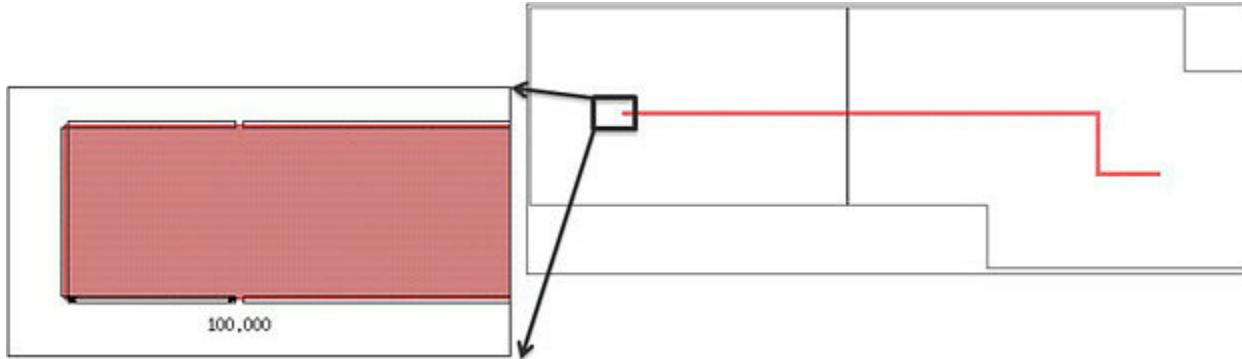
Fragmentation has a large impact on the consistency of corrections performed during an OPC run. There are a number of best practices that can be used to improving fragmentation for better consistency.

Long Edge Fragmentation	638
Asymmetric Intrafeature Fragmentation for Long Edges.	639
Fragmentation Consistency and Tile Boundaries.	641
Asymmetric Intrafeature Fragmentation on the Short Edge	643
Asymmetric Interfeature Fragmentation on the Short Edge	645
Effect of Jogs on Fragmentation and OPC Consistency	647

Long Edge Fragmentation

In the case illustrated in the following figure, the 15 micron long edge trapped between the convex and concave corners is crossing the tile boundary.

Figure 6-14. Long Edge Fragmentation Example 1



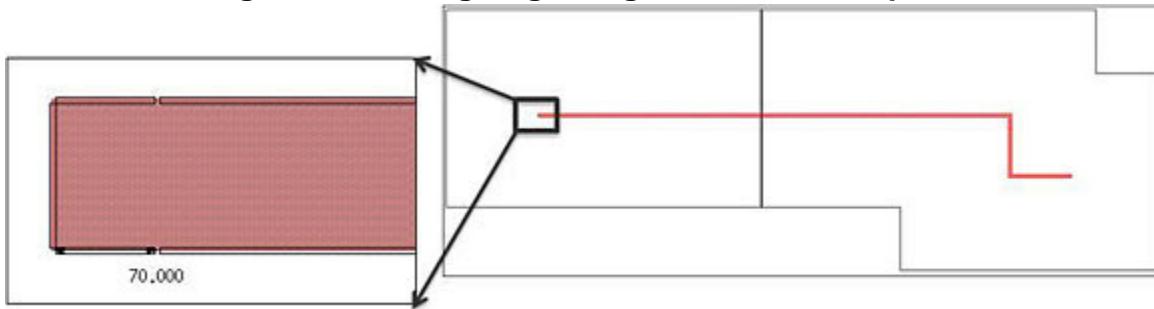
The fragmentation rules are as follows:

```
fragment_layer bias.in {  
    fragment_min 0.050  
    fragment_corner convex convex length >= 0.300 0.100 # RULE1  
    fragment_corner convex concave length >= 0.300 0.050 # RULE2  
    fragment_max 0.4  
}
```

The edge is intended to be fragmented according to RULE2, creating a 50 nm fragment at the convex corner. Actually, the edge is fragmented according to the RULE1 since the line is cut at the tile boundary and the edge is now classified as a long edge trapped between two convex corners.

The syntax for this example generates a warning message. To avoid the warning, specify the upper limit for the length constraint in the `fragment_corner corner_type` statements. The upper limit must be smaller than or equal to the interaction distance.

Figure 6-15. Long Edge Fragmentation Example 2



Unbound constraints can be only used for the `fragment_corner corner_type` types of statements (RULE3)

```
fragment_layer bias.in {
    fragment_min 0.050
    fragment_corner convex convex length >= 0.300 < 1.0 0.100 # RULE1
    fragment_corner convex concave length >= 0.300 < 1.0 0.050 # RULE2
    fragment_corner convex           length >= 1.0            0.070 # RULE3
    fragment_max 0.4
}
```

Asymmetric Intrafeature Fragmentation for Long Edges

Optimizing fragmentation for long edges reduces simulation run time. In the case of asymmetric intrafeature fragmentation for long edges, there are methods of getting better consistency.

For this example, the following fragmentation is defined:

```
fragment_min 0.030
fragment_corner convex convex 0.050 0.050 0.050 0.050
fragment_max 0.1
```

In the case where the edge is long enough to accommodate the specified fragmentation rule, the fragmentation starts from the corners and creates long fragments of 50 nm. Since only four are specified, after creating the fourth fragment from each corner, the simulation stops and verifies that the remaining fragment is larger than `fragment_min` using the following rules:

- If remaining length < `fragment_min`, the edge is not left and the closest fragments are merged together forming a fragment of `remaining_length+2*fragment_min`.

- If `fragment_min` <= remaining length < $2 * \text{fragment_max}$, a single remaining fragment is generated.
- If $2 * \text{fragment_max}$ <= remaining length, then the max edge length fragmentation is applied and breaks the remaining edge into smaller fragments.

In the last case, tile boundaries may create fragments of different length when maximum edge length fragmentation occurs on identical polygons with identical edges but are placed in a different orientation or location. To avoid tile boundary-related issues, the following measures should be taken:

- The number of specified ripples for fragmentation should be as high as possible without exceeding the interaction distance.
- The `breakinhalf` keyword should be added at the end of the `fragment_corner` command to ensure equal length fragments are generated when possible.

[Figure 6-16](#) and [Figure 6-17](#) illustrate the effect of ripples for `fragment_corner` definitions (four ripples and five ripples, respectively). The red square markers indicate the fragmentation differences. Note that there are fewer fragmentation differences for the `fragment_corner` defined with five ripples.

Figure 6-16. Effect of `fragment_corner` With Four Ripples

```
fragment_min 0.037
fragment_max 0.074
fragment_corner convex fragment 0.037 0.037 0.037 0.037
fragment_corner concave fragment 0.037 0.037 0.037 0.037
```



Figure 6-17. Effect of fragment_corner With Five Ripples

```
fragment_min 0.037
fragment_max 0.074
fragment_corner convex fragment 0.037 0.037 0.037 0.037 0.037 breakinhalf
fragment_corner concave fragment 0.037 0.037 0.037 0.037 0.037 breakinhalf
```



Fragmentation Consistency and Tile Boundaries

To manage fragmentation consistency around tile boundaries there are a number of best practices to be applied to fragment_inter statements.

The recommendations include the following

- The number of ripples specified for fragmentation should be as high as possible without exceeding the interaction distance
- The -interdistance value should be made as large as possible.
- Use fragment_inter -ripplestyle 2.

These recommendations are illustrated in [Figure 6-18](#) and [Figure 6-19](#).

Figure 6-18. Effect of Smaller -interdistance Values at Tile Boundaries

With smaller -interdistance values, fragmentation may become inconsistent if the tile boundary crosses the edge.

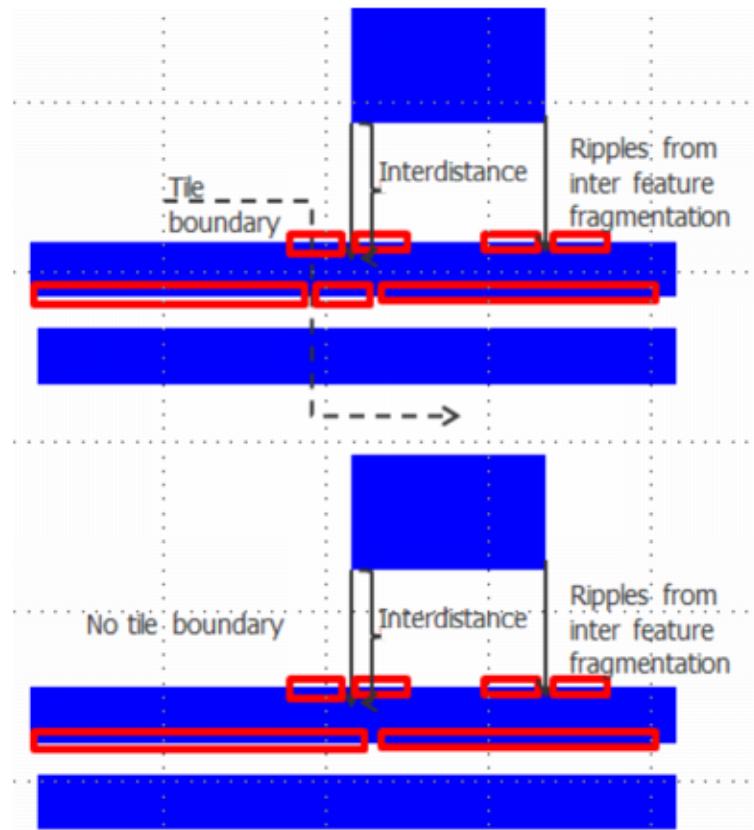
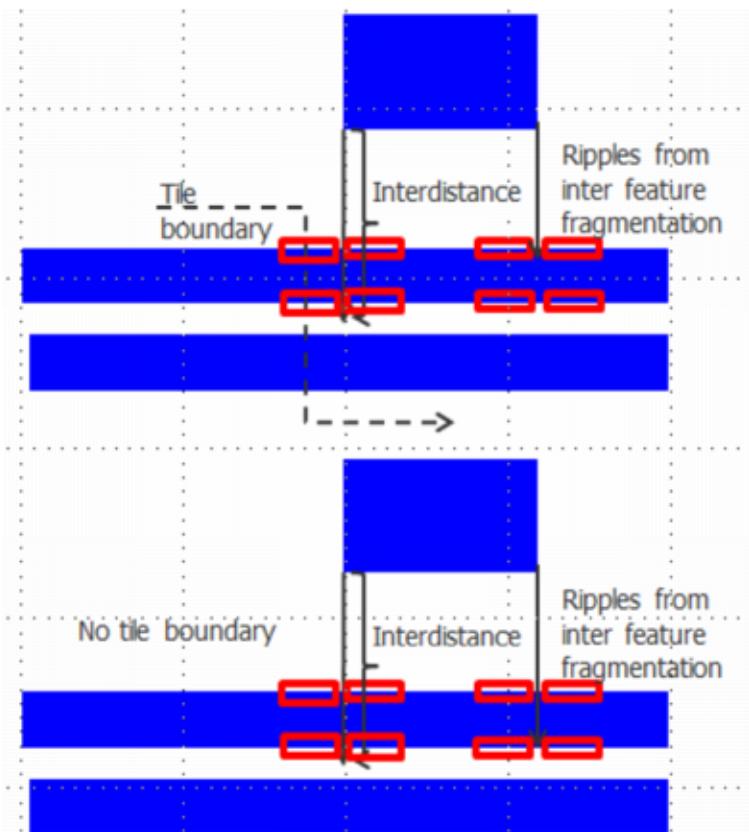


Figure 6-19. Effect of Larger -interdistance Values at Tile Boundaries

With larger -interdistance values, fragmentation may become more consistent since there is no implicit fragmentation in effect.



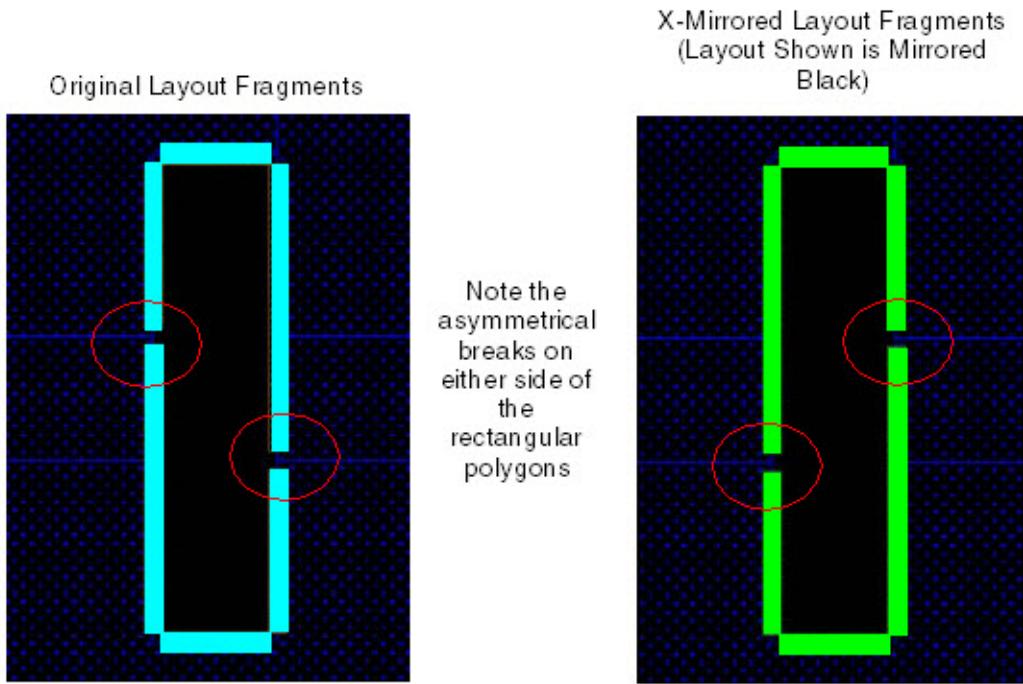
Asymmetric Intrafeature Fragmentation on the Short Edge

Optimizing fragmentation for short edges improves simulation results. For asymmetric intrafeature fragmentation for short edges, there are methods of getting better consistency.

In the following example, the original layout was X-mirrored and fragmented. The asymmetry in fragmentation is apparent, despite the design symmetry. The fragmentation rules are as follows:

```
fragment_min 0.037
fragment_corner convex fragment 0.037 0.037 0.037 0.037
fragment_corner concave fragment 0.037 0.037 0.037 0.037
```

The asymmetric fragmentation results are illustrated in [Figure 6-20](#).

Figure 6-20. Asymmetric Fragmentation for Short Edges

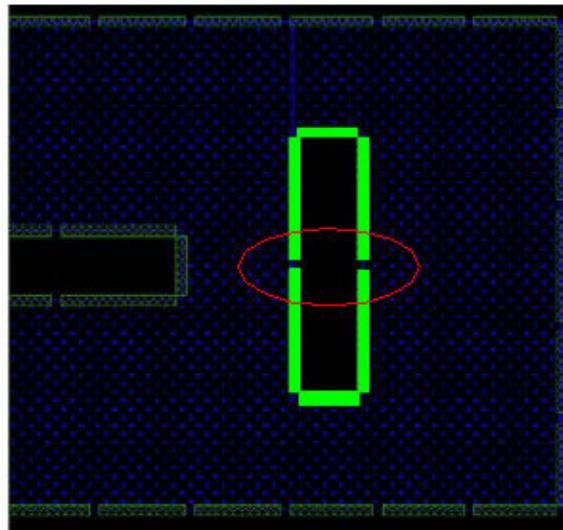
The reason why the fragmentation is asymmetric in this case is that, with the rules specified in the prior example, it was impossible to create an additional fragment without violating the minimum edge length constraint. Consequently, either a break on the top occurs, or a break on the bottom occurs. Since the intrafeature algorithm picks one corner and starts fragmentation from that corner, the asymmetric results cannot be avoided in this case.

To solve this problem, use the `fragment_corner` “breakinhalf” keyword to allow the intrafeature fragmentation to break the edges without violating the minimum edge length constraint. The following is an example of using “breakinhalf” option in `fragment_corner` keyword.

```
fragment_min 0.037
fragment_corner convex fragment 0.037 0.037 0.037 0.037 breakinhalf
fragment_corner concave fragment 0.037 0.037 0.037 0.037 breakinhalf
```

The symmetric fragmentation results are shown in [Figure 6-21](#).

Figure 6-21. Symmetrical Fragmentation

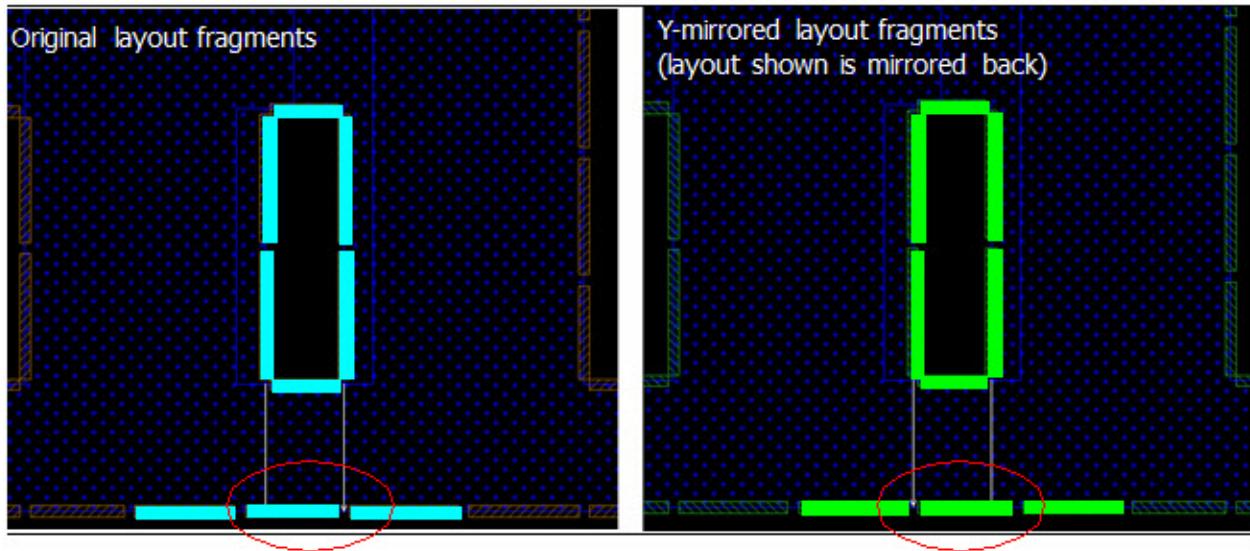


Asymmetric Interfeature Fragmentation on the Short Edge

The fragment_inter command can introduce asymmetric results under some conditions.

In the following example, the interfeature fragmentation attempts to fragment the bottom edge at two points. It is unable to do so because two breaks would violate the minimum edge rule. Interfeature fragmentation can choose only one edge to fragment, but not both.

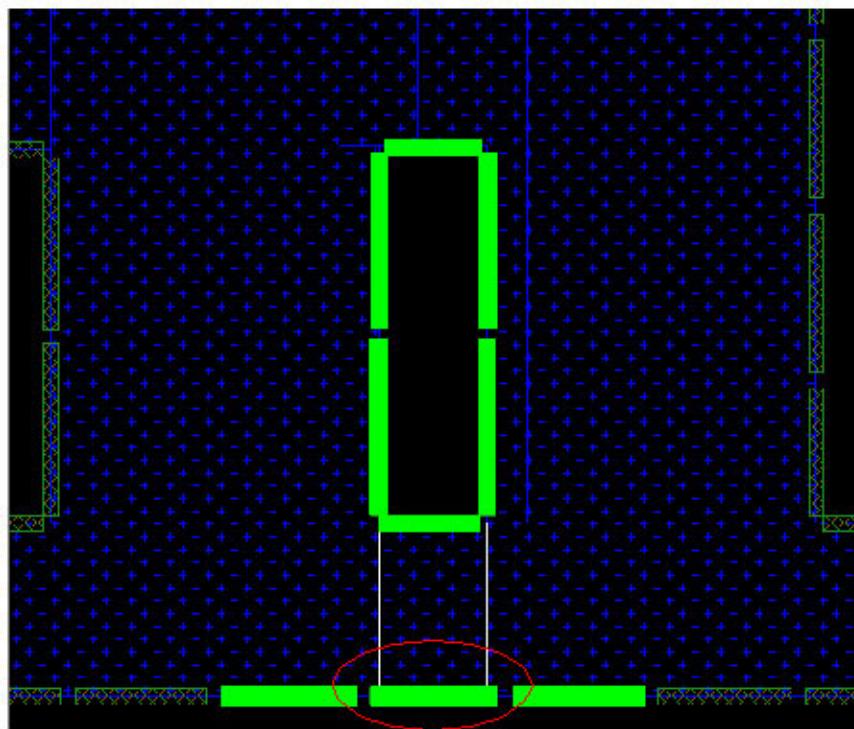
```
fragment_min 0.037
fragment_inter -ripples 1 -ripplelen 0.037 -interdistance 0.185 \
    -ripplestyle 1
```

Figure 6-22. Asymmetric Interfeature Fragmentation

The `fragment_inter -sym` option prevents interfeature asymmetric fragmentation. When using `-sym`, the interfeature fragmentation points are pushed outward slightly to accommodate one `minedgelength` fragment on the bottom edge. This option is only useful when polygons smaller than `minedgelength` create fragments using interfeature fragmentation. If you are not certain that `-sym` is effective for your case, do not use it.

```
fragment_min 0.037
fragment_inter -ripples 1 -ripplelen 0.037 -interdistance 0.185 \
    -ripplestyle 1 -sym
```

Figure 6-23. Symmetric Interfeature Fragmentation



Effect of Jogs on Fragmentation and OPC Consistency

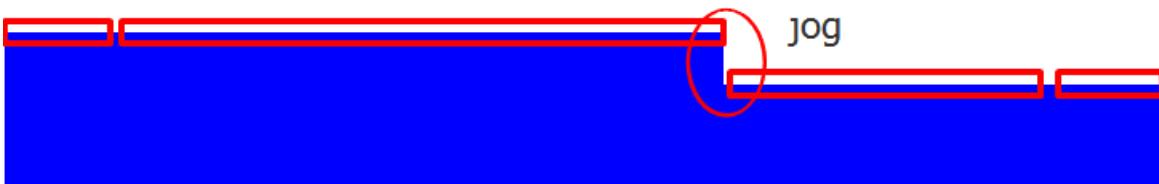
Jogs impact the fragmentation and correction consistency.

In the following example, because the function “`fragment_flaglayer opc_target intrafeature_smalljogs disable`” is specified, the jog corner does not use proper fragmentation:

```
fragment_flaglayer opc_target intrafeature_smalljogs disable
fragment_min 0.037
fragment_corner convex fragment 0.037
fragment_corner concave fragment 0.037
fragment_max 0.75
```

This `fragment_flaglayer` statement was designed for very long edges with very small jogs in the middle of the long edges. This function disables the fragmentation close to the corners so in this case it leads to worse OPC output consistency. This is illustrated in [Figure 6-24](#).

Figure 6-24. Jog Impact on Fragmentation

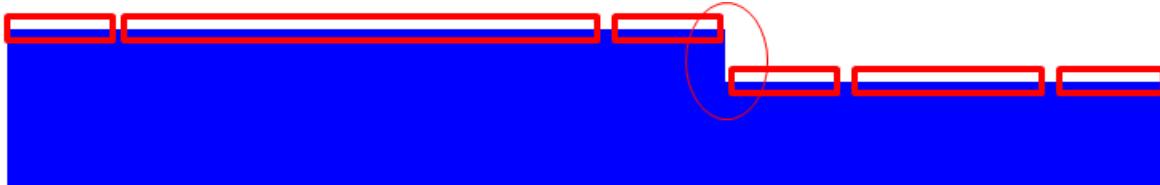


In the following example, the fragment_flaglayer statement is removed from the script.

```
jog_freeze 0.015  
fragment_min 0.037  
fragment_corner convex fragment 0.037  
fragment_corner concave fragment 0.037  
fragment_max 0.75
```

This code modification leads to the proper fragmentation around the jog corners. The [jog_freeze](#) keyword is added to prevent the jog movement during OPC. This setup provides better fragmentation and OPC consistency. The results are illustrated in [Figure 6-25](#).

Figure 6-25. Fragmentation Corrected for Jog Corners



Increasing [fragment_max](#) and the number of specified ripples for the [fragment_corner](#) statement beyond the optical interaction distance impacts Calibre nmOPC performance and increases the run time.

Layer-Based Best Practices

Different layer types can be optimized to produce better results. There are several practices and recommendations available for these layers.

POLY/ACTIVE Layers.....	649
Contact Layers.....	649

POLY/ACTIVE Layers

When building a POLY or ACTIVE layer in an OPC recipe, there are several issues to consider.

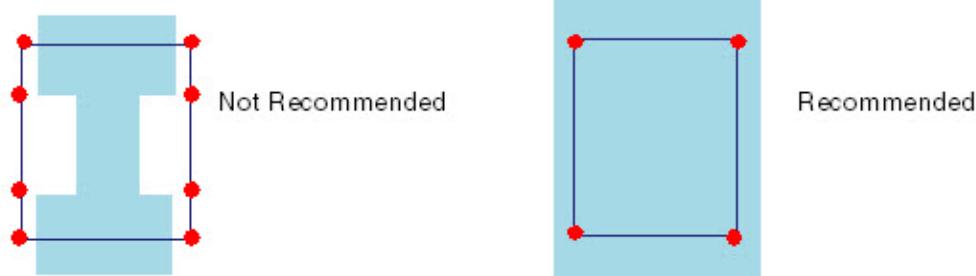
- POLY and ACTIVE are device layers and are very critical.
- The primary objective for this layer type is to control CD variation in gate area (ADLV), which has a direct impact on device electrical performance.
- Design shapes are more complex (for example, L, T, and U shapes).
- Manual fragmentation is most likely needed (see “[Custom Fragmentation](#)” on page 63).
- Application of target smoothing and retargeting is highly recommended (see “[Smooth Biased Target Layers](#)” on page 80 and “[The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC](#)” on page 671 for more information).
- POLY is usually harder to optimize than ACTIVE due to smaller dimensions (POLY width in gate area = channel length).

Contact Layers

When building a contact layer in an OPC recipe, there are several issues to consider.

- The primary objective for this layer type is to maintain a minimum coverage area between contact and metal and device levels (poly and active).
- For 45nm nodes and beyond, have only one fragment/contact edge and try to avoid I-shape OPC output (as shown in [Figure 6-26](#)).

Figure 6-26. I-Shape OPC Output



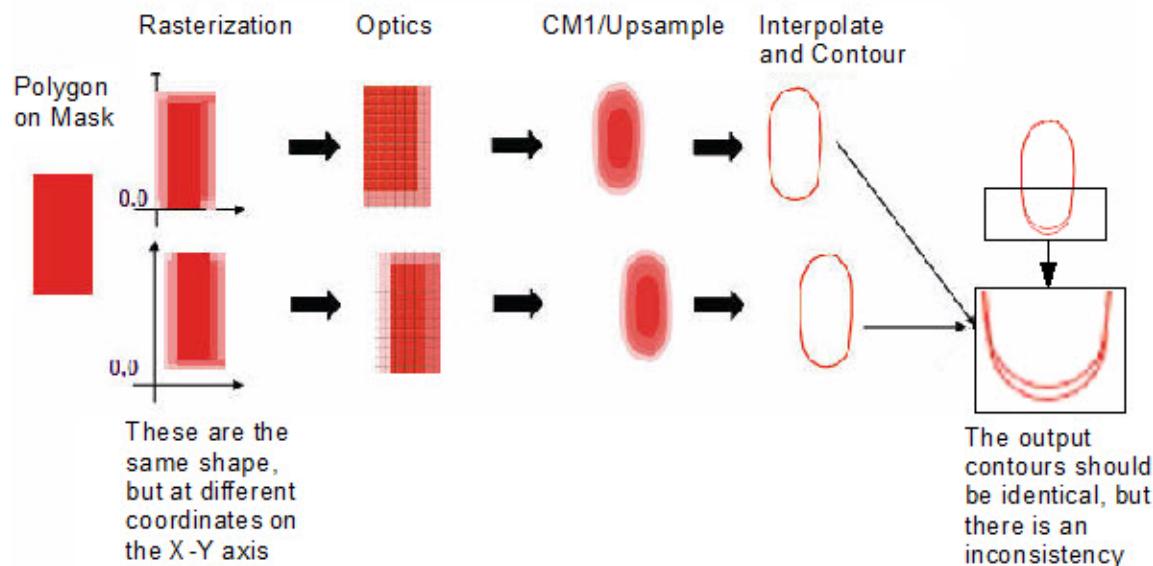
- Lower [feedback](#) is often required.
- Use [max_opc_move](#) to “clip” the maximum allowed per-iteration movement, and possibly reduce oscillations from one iteration to the next.

OPC Output Consistency

There can be occasions where inconsistencies in Calibre nmOPC output occur.

Inconsistencies typically arise when the simulated shape is shifted or rotated relative to the grid origin ([Figure 6-27](#) shows an example when the shape is shifted).

Figure 6-27. Dense Simulation Inconsistency Illustration



This situation can arise due to a number of different factors:

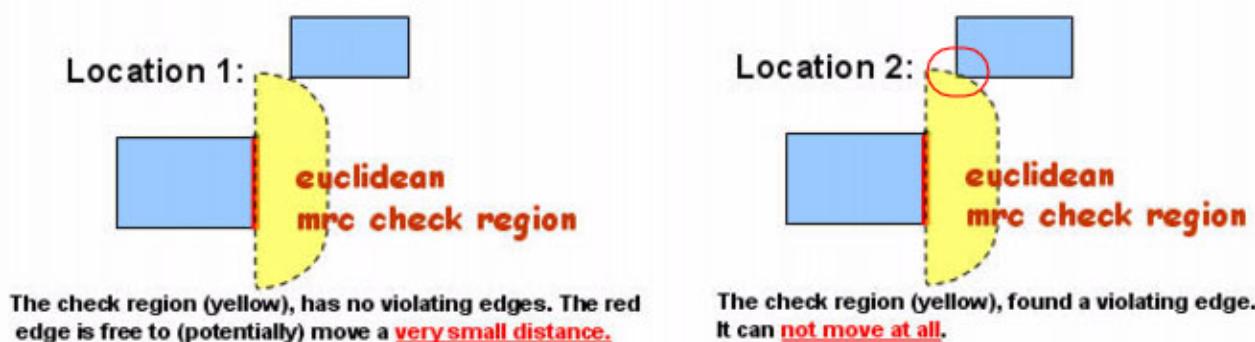
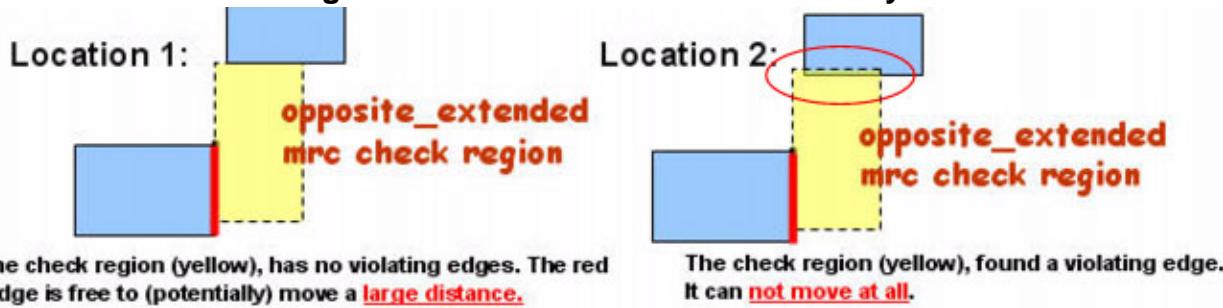
- Issues with rasterization
 - “Aliasing” error introduced by gray-scaling technique.
- CM1 issues
 - Upsampling from Nyquist to the image grid, in combination with the neutralization operation, can lead to inconsistencies. Refer to “[Creating an Initial CM1 File](#)” in the *Calibre WORKbench User’s and Reference Manual* for complete information on neutralization and image grid.
- Contouring issues
 - Non-variance introduced by drawing edges through pixels.
 - Edge endpoints must be drawn on pixel borders.

- Interpolation issues
 - Interpolation from [imagegrid](#) (Calibre OPCVerify) to final contour grid has a small error.
- nmOPC algorithm issues
 - Iterative nature of OPC can cause small errors to accumulate with each iteration.

Other issues can worsen consistency issues. For example:

- Large [feedback](#) values can also make your consistency worse.
- Use of large (or default) [max_iter_movement](#) in high-MEEF cases such as the contact layer can amplify simulation inconsistency.
- Some [mrc_rule](#) measurement metrics can result in inconsistent post-OPC mask shapes. For instance, the *opposite_extended* metric can cause inconsistencies (such as the use of non-euclidean MRC metrics), as illustrated in [Figure 6-28](#).

Figure 6-28. MRC-Related Inconsistency



To improve output consistency, there are a number of different methods available to address these issues:

- Enable [opc_grid_multiplier](#) to refine the correction grid (set it to “on”). This can help reduce inconsistencies resulting from small differences that accumulate into large ones over many iterations.
- Use a small [max_iter_movement](#) of 5-10 nm.

- Enable [simulation_consistency](#), instructing Calibre OPCVerify to perform several tasks to improve output consistency.
- Use SMOOTH_NEUTRALIZATION in CM1. It is recommended that you use SMOOTH_NEUTRALIZATION 2 and SMOOTH GRADIENT 2 to improve simulation consistency.
- For MRC-related inconsistencies, as the opposite and opposite extended metrics are far more inconsistent than the euclidean metric, use the euclidean metric instead.
- Make sure the rsm option is enabled (it is the default) for the Calibre OPCVerify command [mask_sample_grid](#). This uses a mask rasterization method that is more accurate than uniform rasterization.

Practices to Improve VEB Results

There are several different ways to improve the results of your etch-based OPC runs.

Use Automatic Fragmentation	653
Set the Image Grid to 10 nm.....	655
Perform Convergence Analysis to Find Optimal Iterations	655
Use a Hint Offset	656
Set Final feedback to -1.....	657
Use Resist Constraints When Needed	657
Use Corner Chipping.....	658
VEB and notchfill.....	660

Use Automatic Fragmentation

When setting up the VEB setup file, keep it simple and use automatic fragmentation as a starting point.

Automatic fragmentation uses set default values for the length of different fragments depending on the fragment type such as line end, convex corner, concave corner, and so on. Automatic fragmentation requires no other adjustments initially, and it is the default behavior for Calibre nmOPC if no fragmentation commands are supplied.

With automatic fragmentation, you set a value for `veb_pseudo_nyquist` (this is equivalent to the Nyquist value calculated from optical parameters in Calibre nmOPC). A recommended value is $((0.7 \text{ to } 0.8) * \text{minimum feature size})$. For example, for a 32 nm technology node, a Nyquist value of 0.025 is a good estimate. [Figure 6-29](#) and [Figure 6-30](#) show bad fragmentation results from incorrect usage of manual fragmentation and improvements by using automatic fragmentation.

Figure 6-29. Incorrect Manual Fragmentation Example

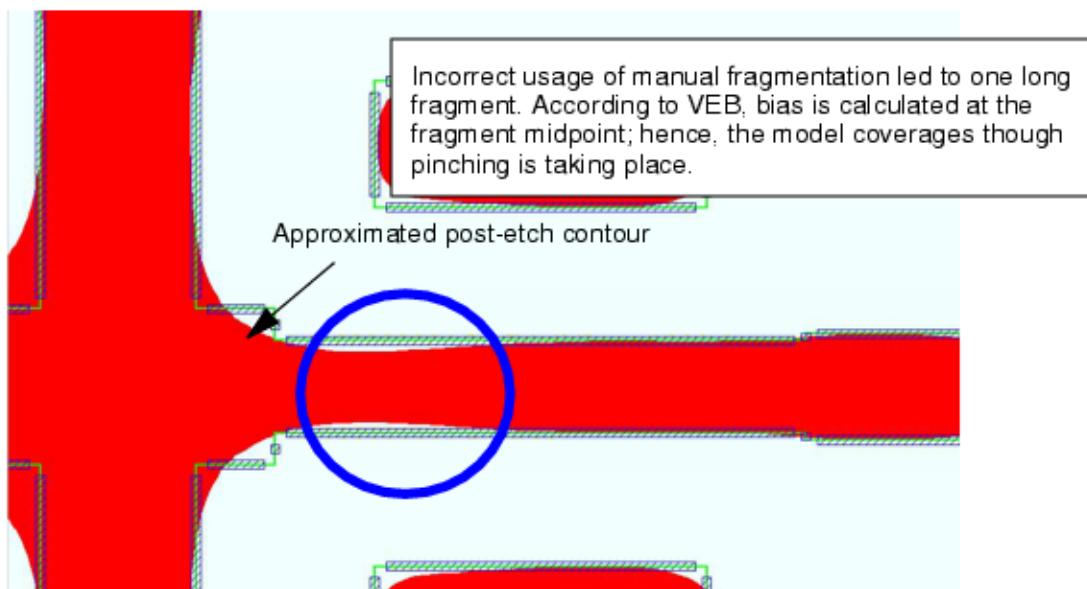
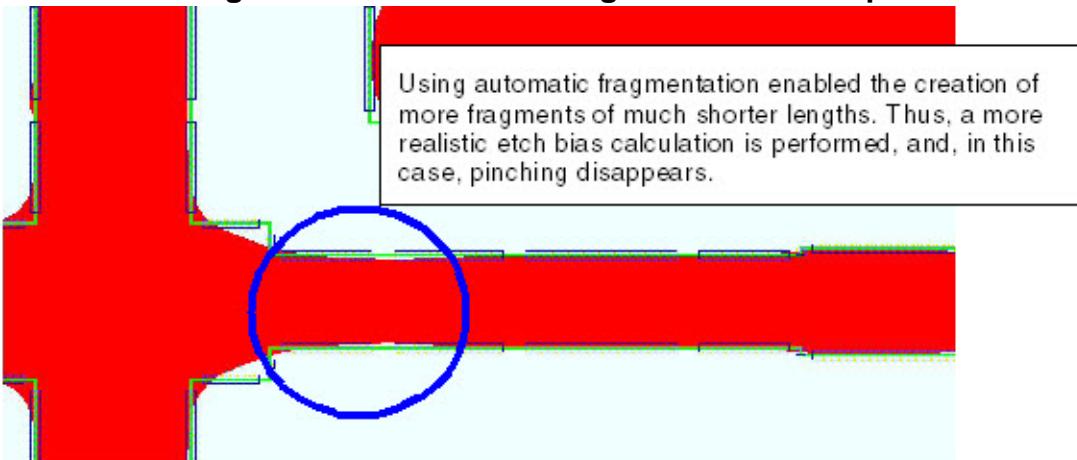


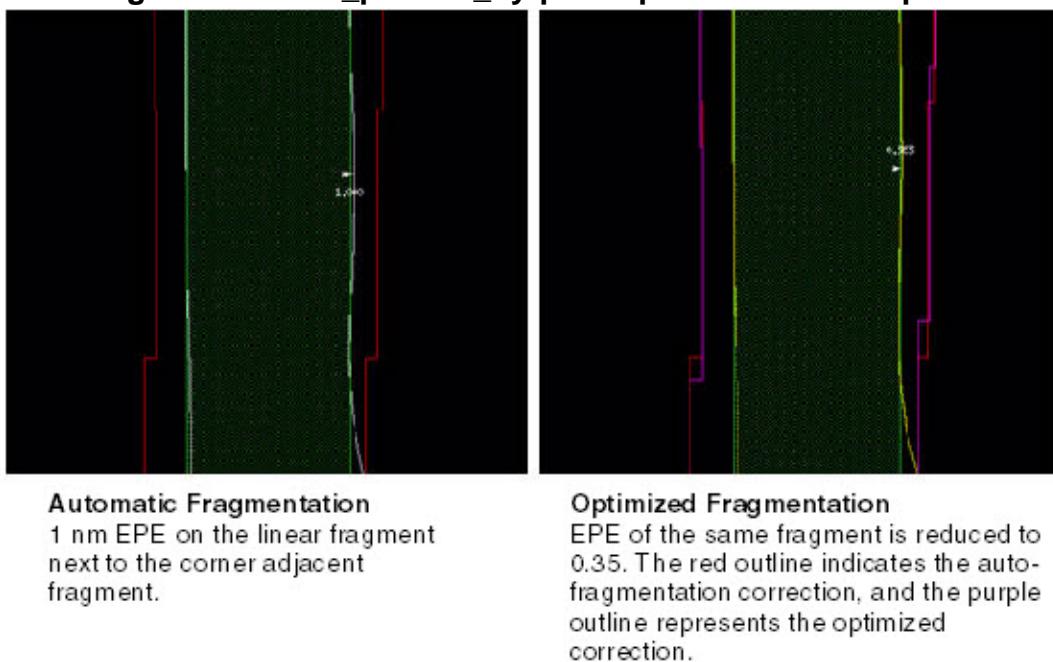
Figure 6-30. Automatic Fragmentation Example



Note

Though the veb_pseudo_nyquist-based fragmentation might achieve a desired answer, some level of optimization may be required. [Figure 6-31](#) illustrates one such occurrence.

Figure 6-31. veb_pseudo_nyquist Optimization Example



Set the Image Grid to 10 nm

Use the `etch_imagegrid` command to set the etch image grid to 10 nm.

For example:

```
etch_imagegrid 0.01
```

See the [etch_imagegrid](#) command for further information.

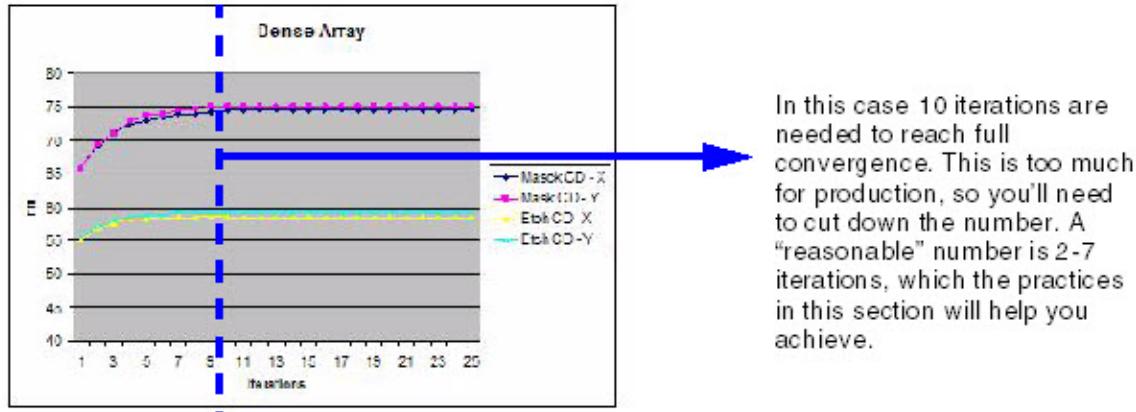
Perform Convergence Analysis to Find Optimal Iterations

When determining how many OPC iterations must be performed, you must have enough iterations for the recipe to converge. This essentially means that the output layer no longer changes with increasing iterations. You can perform a simple convergence analysis by plotting the output resist target plus the etch contour (`veb_simulate`) versus iterations.

For example, you can output the post-etch contour using `veb_simulate` on a smoothed resist target, then output the CD measured data to a spreadsheet and plot the results versus iterations, as shown in [Figure 6-32](#).

A recommended “good” range for your final iterations number is two to seven iterations. You can set this by using either `max_iterations`, or `OPC_ITERATION` if you are the Tcl-based custom scripting.

Figure 6-32. Convergence Analysis



Use a Hint Offset

You can use the Tcl-based custom scripting capability of Calibre nmOPC to perform a number of different optimizations.

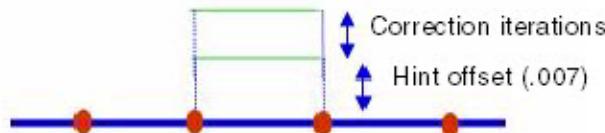
- Tagging certain layout features based on their dimensions, topology and so on.
- Speed up convergence by exercising special control on tagged edges/fragments feedback and displacement.
- Modify fragmentation for selected tags.

A very common usage of scripting would be creating a “hint offset” to allow fragments to converge in a smaller number of iterations. The basic syntax would be as follows:

```
NEWTAG all etch_target -out tag_name
FRAGMENT_SET_DISPLACEMENT tag_name offset -force
OPC_ITERATION integer
```

Figure 6-33 shows how the hint offset works on a fragment.

Figure 6-33. Hint Offset



For example, if convergence requires more than the recommended two to seven iterations, you can use a hint offset approach through scripting as follows:

```
NEWTAG all      var_bias_etch_target -out all frags
FRAGMENT_SET_DISPLACEMENT all frags 0.007 -force
OPC_ITERATION 4
```

Set Final feedback to -1

The feedback command defines the default feedback values to be used per iteration. Edge movement is set by ($EPE * feedback$). If there is no valid EPE, edge movement is set to a default of Nyquist/3.

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.

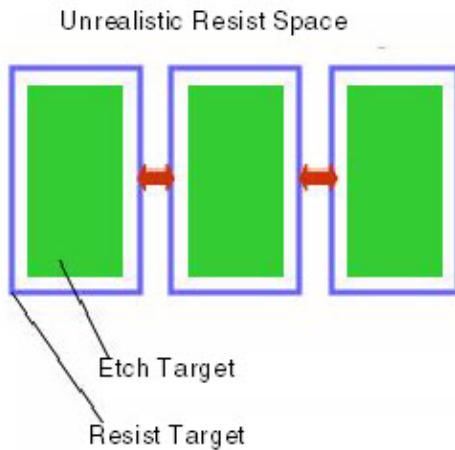
In case a special feedback sequence is required, it is recommended that you end the feedback sequence with -1. For example:

```
feedback -0.4 -0.8 -0.8 -0.3 -1
```

Use Resist Constraints When Needed

In some situations, reaching good convergence means using an “unrealistic” resist target space/width. An “unrealistic” target in this case means that the output from the VEB correction step (the resist target) is either too narrow or very closely spaced and would typically fail during lithography. For example, you can have an etch bias from a VEB model that is very large, leading to a resist target that has a very narrow process window.

Figure 6-34. Unrealistic Target Example

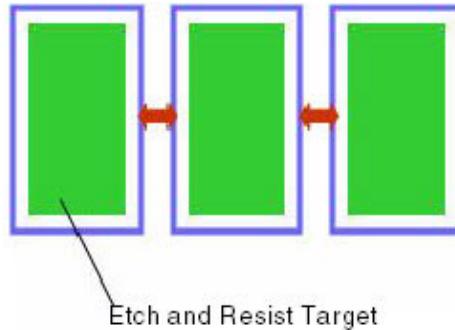


In this case, you can apply resist constraints by using the mrc_rule keyword in the VEB denseopc_options block, the same command as used by the optical and resist version of the setup file. For example:

```
...  
mrc_rule external etch_target {  
    use 0.030 euclidean  
}  
...
```

This example specifies the minimum distance to allow for the rule and applies a Euclidean metric. [Figure 6-35](#) shows how this applies to the “unrealistic” target from [Figure 6-34](#).

Figure 6-35. mrc_rule Adjustment

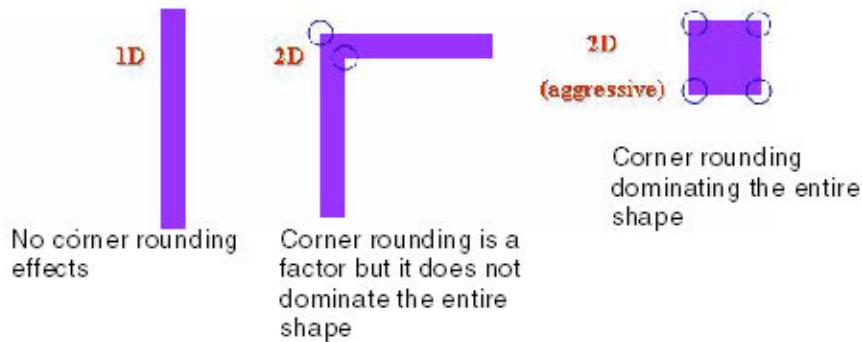


Use Corner Chipping

When corner rounding occurs, it tends to favor optical characteristics and not etch characteristics.

[Figure 6-36](#) shows the effects of corner rounding for different shapes.

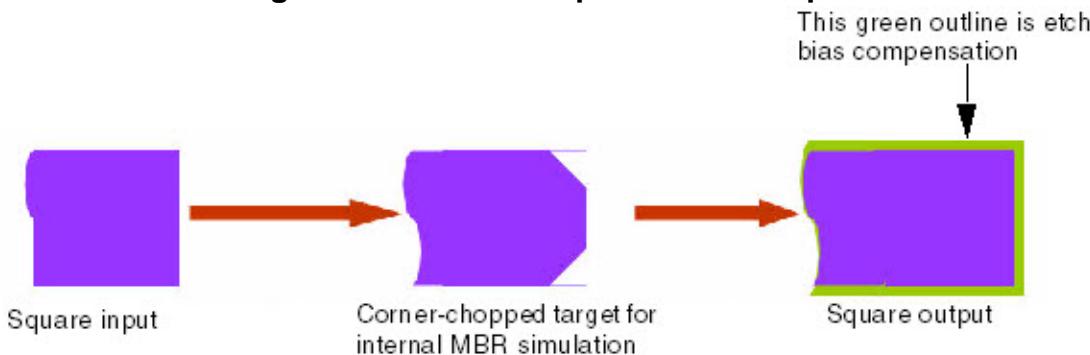
Figure 6-36. Corner Rounding Effects



The Optical and CM1 models can deal with corners by using grid-based simulations and contouring, but VEB cannot deal with corners as it does not have contouring and edge-based models. To resolve this, for pure 2D structures such as contacts, use the cornerchop method

rather than a regular square target to achieve better accuracy, and use the [mask_chip_corner convex_val concave_val](#) command to make VEB simulations over octagonal shapes (octagons for contacts and chopped line ends in other cases). Unlike the cornerchop command, mask_chip_corner clips corners only during the simulation step at each VEB Model-Based Retargeting iteration, leaving the processed layout with 90-degree corners (as illustrated in Figure 6-37).

Figure 6-37. mask_chip_corner Example



Tip

i Set *convex_val* and *concave_val* to around 1.3 * Nyquist.

You can use this adjusted layer in the following circumstances:

- As an OPC layer following the Calibre nmOPC run
- As an OPC layer and generate a smoothed target (using [setlayer curve](#)) based on that layer

If you are using mask_chip_corner with OPC layer and SRAF layers to perform a Calibre OPCverify or Print Image contour simulation, you must use the Calibre OPCverify cornerchop command on both layers.

For example, in a Calibre nmOPC setup file containing:

```
mask_chip_corner 0.004 0.004
```

You should use the following to generate a chopped mask layer for the simulation to perform correctly:

```
setlayer opced_chop_lyr = cornerchop opced 0.004 0.004
setlayer sraf_chop_lyr = cornerchop sraf 0.004 0.004
```

Related Topics

[cornerchop \[Calibre OPCverify User's and Reference Manual\]](#)

VEB and notchfill

The notchfill operation should not be done after setlayer veb_retargt. Using notchfill inside the VEB setup file after the VEB retargeting process can introduce potential MRC violations after OPC.

SRAF Print Avoidance Best Practices

The `sraf_print_avoidance` command is designed to detect and correct for any printing occurring around sub-resolution assist features (SRAFs).

This command examines simulations and performs a single SRAF fragment adjustment iteration. The following is a list of best practices that can potentially improve the results of the `sraf_print_avoidance` operation:

Starting Recommendations for SRAFs	661
Negative SRAF Handling	662
Iterations for <code>sraf_print_avoidance</code>	664
SRAF Fragmentation	664
SRAFs and MRC Rules.....	664
Wafer-Level Data and Printing Threshold	665
SRAFs and Process Window OPC	666

Starting Recommendations for SRAFs

The most important recommendation is to treat the `sraf` layer with the attention to detail as you would with the OPC layer.

Fine-tuning the recipe includes the following factors:

- MRC-clean input SRAF
- Proper and complete fragmentation
- Correct MRC specifications
- Sites and spacing
- Minimum length, width and area of final SRAFs
- Process window conditions
- Printing Threshold
- Step size

The following basic practices should be followed for any `sraf_print_avoidance` implementation:

- For better simulation time, use the same optical model for `sraf_print_avoidance` as you do for your nominal OPC process window condition.
- All layers containing SRAFs should be defined as an `sraf` or `asraf` layer, otherwise fragments on the layer will not be moved.

The following code example illustrates these recommendations:

```
denseopc_options DOPC {
    ...
    background clear
    layer TARGET      opc      atten 0.06
    layer MBSRAF     sraf     atten 0.06
    ...

    pw_condition spa optical OPT dose 1.15 resist RESIST pband spa
    ...

    fragment_layer CONTACT_VSB_PRE {
        fragment_min 0.03
        fragment_max 0.12
    }
    ...

    mrc_rule area sraf_layer {
        use area1
        square use area2
    }
    ...

    sraf_sites_create

    #OPC iteration
    OPC_ITERATION 5

    #OPC and SRAF print avoidance
    For {set i 0} {$i < 5} {incr i} {
        OPC_ITERATION 1
        sraf_print_avoidance -step 0.001
    }

    OPC_ITERATION 3
}
```

Negative SRAF Handling

The asraf layer specifies layers used for negative SRAFs and allows modification of them by SRAF print avoidance.

Prerequisites for proper formation of negative SRAFs include:

- Negative SRAFs should not be subtracted from the target layer prior to loading it into Calibre nmOPC. The subtraction is done automatically in the simulator.
- Definition of the asraf layer polarity is redundant.
- Unlike in previous flows, the negative SRAF layer should not be defined as a no_opc_region.

- A process window condition for negative SRAFs must be separate from the positive SRAF conditions and must be designated using the “spa” keyword.
- You must subtract the output ASRAFs after correction is done to obtain a valid mask

The following code example using these recommendations:

```
denseopc_options DOPC {
    ...
    background clear
    layer TARGET opc atten 0.06
    layer MBSRAF sraf atten 0.06
    layer SRAF_NEG asraf
    ...
    pw_condition SPA_pos optical OPT dose 1.15 resist RESIST pvbnd spa
    pw_condition SPA_neg optical_pw OPT dose 1.15 resist RESIST pvbnd spa
    ...
    fragment_layer MBSRAF {
        fragment_min 0.03
        ...
        fragment_max 0.12
    }
    fragment_layer SRAF_NEG {
        fragment_min 0.03
        ...
        fragment_max 0.12
    }
    ...
    mrc_rule area MBSRAF {
        use <areal>
        square use <area2>
    }
    ...

    sraf_sites_create
    #OPC_iteration

    OPC_ITERATION 5
    #OPC and SRAF print avoidance
    For {set i 0} {$i < 5} {incr i} {
        OPC_ITERATION 1
        sraf_print_avoidance -step 0.001 -pw SPA_pos <t_pos> \
            -pw SPA_neg <t_neg>
    }
    OPC_ITERATION 3
}
```

If you have both positive and negative SRAFs using the same printing condition, following syntax is allowed:

```
sraf_print_avoidance -step 0.001 -pw SPA_pos_neg <t_pos> <t_neg>
```

Iterations for sraf_print_avoidance

In general, a careful study of EPE must be done to ensure proper SRAF printing elimination as well as good convergence of the main features.

As a general rule, a 2:1:1 ratio of OPC/[sraf_print_avoidance](#)/OPC iterations is recommended. The maximum number of iterations and correction steps are related, as well as the maximum SRAF size and MRC constraint. This relationship can be summarized in the following equation:

$$\max_spa_iter = \text{int}\left(\frac{\max SRAFsize - internalMRCconstraint}{2 * step}\right) + 1$$

where **max_spa_iter** is the maximum sraf_print_avoidance iterations and **step** is the movement per iteration. The overall idea is to make sure that the SRAF fragment has enough freedom from constraints to move (and disappear entirely if necessary).

SRAF Fragmentation

Careful practices applied to fragmentation of SRAFs improve SRAF results.

- SRAFs require simple fragmentation that does not violate MRC constraints and ensures proper site placement for robust printing detection.
- Set fragment_min to be the minimum allowed post-SRAF print avoidance SRAF length.
- If the sraf layer consists of squares only, there is no need for fragmentation, as each edge will be treated as a fragment.

```
fragment_layer sraf_layer {  
    fragment_min 0.030  
    fragment_inter -interdistance 0.21 -ripples 2 \  
        -ripplelen 0.030 -externalOnly -distancePriority 1  
    fragment_corner 0.030 0.030 0.030 0.030 breakinhalf  
    fragment_max 0.080  
    ...
```

- Fragment alignment is performed automatically on all layers of type sraf.
- Site placement is done by using the [sraf_sites_create](#) command. No arguments are necessary as defaults are determined internally by the command.

SRAFs and MRC Rules

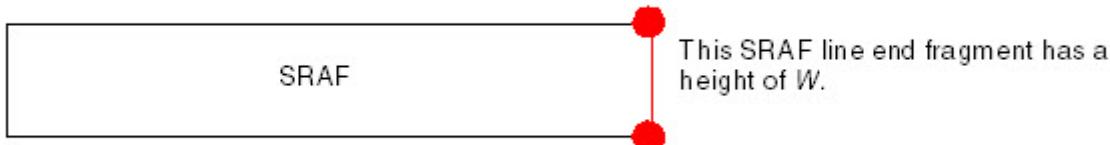
There are a number of issues to consider when using SRAFs and MRC rules.

- Fragmentation should be large enough that the “mrc_rule internal” for the SRAF multiplied by the minimum fragment length is a legal shape for the mask shop.

- Use length rules for long and short SRAFs:

```
mrc_rule internal sraf_layer {
    projecting use minWidth_from_mbSRAF opposite
    length W use value_equal_to_fragment_min
    not_projecting use minWidth_or_smaller euclidean
    length W use value_equal_to_fragment_min
}
```

Figure 6-38. SRAF Line End Fragment Example



- Use “mrc_rule area” rules to enforce SRAFs larger than the minimum allowed area in the Calibre nmOPC options. For example:

```
mrc_rule area sraf_layer {
    use minWidth*minLength_area_or_smaller
    square use min_square_area_or_smaller
}
```

Refer to the [Calibre nmSRAF User’s and Reference Manual](#) for information on mbSRAF.

Wafer-Level Data and Printing Threshold

In order to use sraf_print_avoidance correctly, you must have wafer-level data specifying at which condition SRAF printing is occurring.

There is no need to directly specify a printing threshold for `sraf_print_avoidance`. By default, it is read out of the CM1 model used by `sraf_print_avoidance` and the indicated process window conditions.

The choice of printing condition should be defined based on your wafer-level data. In cases when the printing condition is unknown, the best “guess” for the printing threshold is one of the following:

- The THRESHOLD parameter out of the CM1 .mod file.
- In cases of an optical model only, the constant threshold used for contour generation.
- An over or under dose process window condition for dark or bright field masks, respectively.

- A slightly decreased or increased threshold (1 to 3%) for dark or bright field masks, respectively.
 - If a nominal value is used, `sraf_print_avoidance` might not detect printing reported by Calibre OPCverify.
 - Slight differences in simulation results (`sraf_print_avoidance` versus Calibre OPCverify) are possible due to noise or inaccuracy of simulations with different grid origins.
 - Increases or decreases in threshold need to be adjusted as a point of balance between excessive print reduction versus process window benefits.

For example, if you have the following pre-conditions:

- Dark field and PTD process
- Printability check condition in Calibre OPCverify: +10% overdose
- Initial recipe setup without wafer data

You can use the following example dose and threshold options to select the correct process window condition to predict printing in `sraf_print_avoidance`.

- Dose +10%, 1.000 x Threshold

Or:

- Dose +10%, 0.909 x Threshold (to account for simulation and grid origin inconsistencies).

SRAFs and Process Window OPC

Use the `pw_condition` command to define process windows at which SRAF printing occurs.

However, there can be cases where the printing condition is such that it is undesirable to use it for `pw_condition` in Process Window OPC (PWOPC).

```
pw_condition nominal mask_size 0 optical opticsfile dose 1.000 pband
pw_condition sraf_model mask_size 0 optical opticsfile_SrafModel \
    dose 1.000 pband spa
```

The “spa” parameter indicates that the condition is active for `sraf_print_avoidance` only. This ensures that the process window conditions for `sraf_print_avoidance` are only evaluated during their specific iterations and ensure better quality of the solution along with improved processing speed.

The following is an example of implementing the “spa” keyword with PW iterations:

```
OPC_ITERATION 3
```

```

for{set i 1} {$i <= 3} {incr i} {
    OPC_ITERATION 1 -PW \
    nominal_epe_limit All_Tags -0.002 0.002 \
    min_space All_Tags 0.036 \
    min_width All_Tags 0.036
        sraf_print_avoidance -step 0.001
}

OPC_ITERATION 3 -PW \
nominal_epe_limit All_Tags -0.002 0.002 \
min_space All_Tags 0.036 \
min_width All_Tags 0.036

```

The sraf_print_avoidance threshold should be adjusted down for positive resist tone models.

```
SPA_threshold = (0.97 ÷ 0.99) * CM1_threshold
```

For negative tone (NTD) resist models, it must be adjusted up:

```
SPA_threshold = (1.01 ÷ 1.03) * CM1_threshold
```

Simplifying a Calibre mpOPC Setup File

Calibre mpOPC is used for double and multi-pattern mask correction. This is implemented in a standard Calibre nmOPC setup file using the mpopc command. Double- and triple-patterning significantly increases the complexity of the setup file because you are required to define fragmentation and MRC constraints for every pattern. Often, the fragmentation and constraints are the same for every pattern, but they are still explicitly defined for each combination of patterns.

You can simplify the rule file by creating Tcl command blocks that only need to be defined once, then reused for each pattern combination. The basic structure is as follows:

```

set fragmentation_block_name {...}
set mrc_block_name {...}

fragment_layer layer_name "{ [subst -nocommands $block_name] }"
mrc_rule_type layer_name [layer_name] "{ [subst -nocommands $block_name] }"

```

To pass a layer name into a block, use the following Tcl construct with \$layer_name_variable in the same block:

```
set layer_name_variable layer_name
```

The following example assumes double-patterning with two OPC patterns and two SRAF patterns. This example setup illustrates how to simplify a rule deck by defining fragmentation and MRC rules once and reuse them as many times as needed.

```
layer opc_layer_p0 opc pattern 0
layer opc_layer_p1 opc pattern 1
layer sraf_layer_p0 sraf pattern 0
layer sraf_layer_p1 sraf pattern 1
...
#####
# Fragmentation setup
# Here, the fragmentation only needs to be defined once, then re-used by
# all pattern combinations
#####

...
set opc_frag_rules {
...
}

set sraf_frag_rules {
...
}

fragment_layer opc_layer_p0 "[subst -nocommands $ opc_frag_rules]"
fragment_layer opc_layer_p1 "[subst -nocommands $ opc_frag_rules]"

fragment_layer sraf_layer_p0 "[subst -nocommands $ sraf_frag_rules]"
fragment_layer sraf_layer_p1 "[subst -nocommands $ sraf_frag_rules]"

#####
# MRC rule setup
# These constraints will be reused by all pattern combinations
#####

set mrc_ext_opc_rules {
...
}
set mrc_int_opc_rules {
...
}
set mrc_opc_sraf_rules {
use ...
}
set mrc_sraf_int_rules {
...
}
set area_rules {
...
}

mrc_rule internal opc_layer_p0 "[subst -nocommands $mrc_opc_ext_rules]"
mrc_rule external opc_layer_p0 "[subst -nocommands $mrc_opc_int_rules]"

mrc_rule internal opc_layer_p1 "[subst -nocommands $mrc_opc_ext_rules]"
mrc_rule external opc_layer_p1 "[subst -nocommands $mrc_opc_int_rules]"
```

```
mrc_rule external opc_layer_p0 sraf_layer_p0 \
  "{ [subst -nocommands $mrc_opc_sraf_rules] }"
mrc_rule external opc_layer_p1 sraf_layer_p1 \
  "{ [subst -nocommands $mrc_opc_sraf_rules] }"

mrc_rule internal sraf_e1 "{ [subst -nocommands $mrc_sraf_int_rules] }"
mrc_rule internal sraf_e2 "{ [subst -nocommands $mrc_sraf_int_rules] }"
mrc_rule area sraf_e1 "{ [subst -nocommands $area_rules] }"
mrc_rule area sraf_e2 "{ [subst -nocommands $area_rules] }"
```


Appendix A

The Calibre VEB Model-Based Retargeting Flow in Calibre nmOPC

Variable Etch Bias (VEB) is a process that allows you to separately account for etch effects. This methodology differs from previous OPC practices, where modeling effects including optical, resist, and etch are calculated at the end, using only the post-etch target data.

This VEB model-based retargeting process enables you to perform the following:

- Separate the modeling of etch bias effects from optical and resist effects.
- Separate the correction of etch effects and correction of optical and resist effects.
- Derive an etch-corrected design based on the VEB model. It uses fragmentation and edge movement calculations, similar to making OPC corrections to a layout.

VEB is equivalent to performing model-based biasing with Calibre nmBIAS, but it uses the Calibre nmOPC license. Biasing is a separate operation from OPC and does not use the optical or resist models.

A list of VEB commands is available at “[Variable Etch Biasing \(VEB\) Commands](#)” on page 96.

What is VEB Retargeting?	671
VEB Requirements	672
Add Retargeting to Your Process	672
Stage 1 - Calibrating the Optical, CM1 Resist, and VEB Models	673
Stage 2 - Retargeting the Design	674
Stage 3 - Using the Retargeted Design	676
Stage 4 - Verifying the OPC Output in Calibre OPCverify	678
Frequently Asked Questions	679

What is VEB Retargeting?

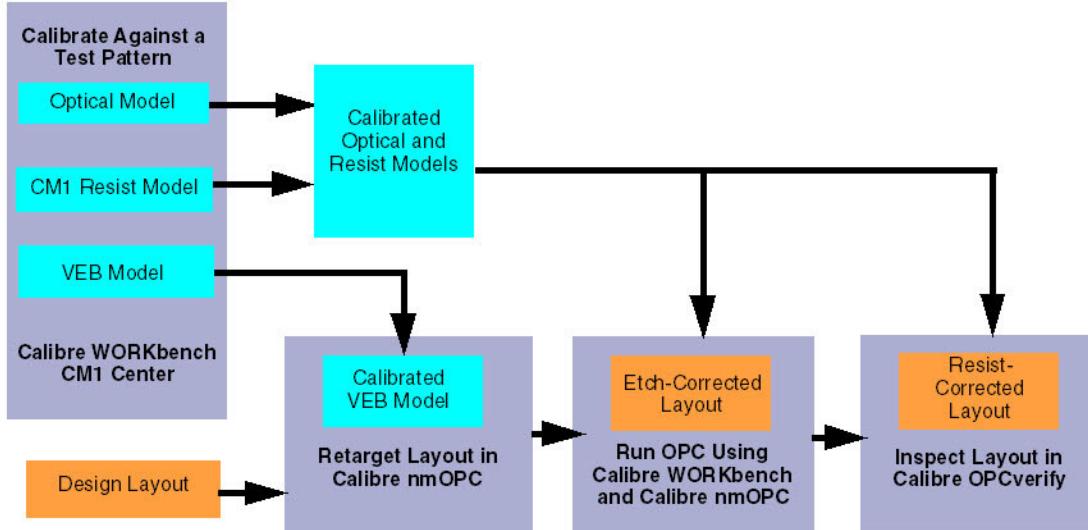
Using VEB to separate out the post-etch data and using it to retarget the design has a number of key benefits.

- Reduces the run time of etch bias correction, because the corrections are applied to the fragment edges on the original target rather than applied to the resist contours.

- Allows you greater control over your OPC corrections by separating the resist and etch effect requirements.
- Allows you to verify post-OPC corrections at the resist level against the resist target, enabling detection of problems in the resist phase.

Figure A-1 illustrates the VEB retargeting flow.

Figure A-1. VEB Retargeting Flow



VEB Requirements

VEB has a number of requirements in order to perform Calibre nmOPC.

The following are the requirements:

- Calibre WORKbench 2007.2 or later
- Calibre nmOPC 2007.4 or later
- Calibre OPCverify 2007.2 or later
- Measurement information from a sample file spreadsheet (post-resist development phase)
- Measurement information from the same sample spreadsheet (post-etch phase)
- The layout you want to use the models on

Add Retargeting to Your Process

VEB model-based retargeting processes produce etch-corrected simulation results.

Table A-1. The VEB Model-based Retargeting Process

Stage	Description
1	Calibrate the Optical, CM1 resist, and VEB models on a test pattern.
2	Using the VEB model, retarget the original design file in Calibre nmOPC to generate an etch-corrected design file.
3	Using the etch-corrected design file, run your simulations through Calibre nmOPC to generate an OPC-corrected file.
4	Using the OPC-corrected file from Calibre nmOPC, inspect the design file in Calibre OPCverify.

An alternative method available is to directly correct etch effects using process windows conditions and a litho model that includes etch. This method is described in “[Direct Etch Effect Correction](#)” on page 71.

Stage 1 - Calibrating the Optical, CM1 Resist, and VEB Models

In the initial stage, calibrate your models using a test pattern instead of the layout design.

Tip  The procedures in this chapter are described in detail in the [Calibre WORKbench User's and Reference Manual](#).

Prerequisites

- You must have separate optical, CM1, and VEB models based off of a test pattern that has been run through your process.
- A gauge object file created from the resist measurements
- A second gauge object file created from the combined resist and etch measurements

To create gauge objects, you can use the Calibre WORKbench CM1 Center. You may have to add the etch measurements to a resist spreadsheet by hand.

Procedure

1. Load the gauge files into Calibre WORKbench.
2. Using CM1 Center:
 - a. Create an optimized optical model.
 - b. Save the optical model file. Turn the Optical model controls to (T)ry.
 - c. Create an optimized CM1 resist model.

- d. Save the CM1 resist model. Turn the CM1 controls to (T)ry.
 - e. Create an optimized VEB model.
 - f. Save the VEB model.
3. You should attempt to optimize the CM1 and VEB models as closely as possible to your process.

Results

You should now have separate etch and resist models. Proceed to “[Stage 2 - Retargeting the Design](#)” on page 674.

Stage 2 - Retargeting the Design

After separating the etch and resist models, use the etch model to retarget your design file. In this case, the Calibre nmOPC veb_retarget operation is used to set the guidelines for fragment movement.

Prerequisites

- Your design file containing the layout to be retargeted
- The etch (VEB) model you created in “[Stage 1 - Calibrating the Optical, CM1 Resist, and VEB Models](#)”

Procedure

1. Using a text editor, open a Calibre nmOPC setup file with the following items:

Table A-2. Calibre nmOPC and OPCVerify Keyword Requirements for VEB Retargeting

Keyword	Arguments	Reason
modelpath	<i>path_to_your_model_files</i>	Calibre nmOPC uses this keyword to locate your model files.
etch_model_load	<i>label [path/]VEB_model_filename</i>	Loads the etch model file into Calibre nmOPC and assigns it the given label for later reference.
background	[clear dark atten <i>value</i>]	Sets the background properties of the design.
layer	[visible hidden etch_underlying]	Sets the layer properties of the drawn layer in the design to be retargeted. Only one layer can be retargeted.

This is not a restrictive list; other commands may also be present.

2. Add the retargeting command, `setlayer veb_rettarget`.
3. Save the Calibre nmOPC setup file.
4. Using a text editor and a standard SVRF rule file template, add a call to Calibre nmOPC (a LITHO DENSEOPC command) specifying the revised setup file. The LITHO DENSEOPC command returns the retargeting layer.

```
derived_resist_target = LITHO DENSEOPC FILE "setup_file_name.in"  
original_etch_target MAP ret_output
```

5. Place the new layer in the layout using the DRC CHECK MAP command:

```
derived_resist_target {COPY derived_resist_target}  
DRC CHECK MAP derived_resist_target layer_number
```

Tip

 The SVRF rule file should write its output to a different file than the input file, as you may need to re-run your retargeting command if you are not satisfied with the results.

6. Save the file and run the SVRF rule file in a shell window:

```
calibre -drc -hier rule_file
```

Calibre runs and places the retargeted results on a new layer in the output file.

Results

Two files are created for use with the Calibre nmOPC tool:

- A Calibre nmOPC setup file
- An SVRF rule file containing information about your design file

Proceed to “[Stage 3 - Using the Retargeted Design](#)” on page 676.

Examples

This example VEB setup file is called *cmdfile.in*. This is separate from a Calibre nmOPC setup file used for optical and resist process correction.

```
modelpath ~/calibre/models
# note that the optical and resist models are not loaded
# because they are not required for retargeting
etch_model_load veb1 etch_veb1.mod
background atten 0.064
layer RETARGET_ME visible clear

denseopc_options OPTS {
    version 1
    etch_model veb1
    background atten 0.064
    layer RETARGET_ME opc clear
    max_iterations 3
    feedback -1 -1 -1
    veb_pseudo_nyquist 0.06
}

setlayer ret_output = veb_rettarget RETARGET_ME MAP RETARGET_ME \
OPTIONS OPTS
```

Example Calibre SVRF Rule File

This example assumes that the SVRF rule file contains definitions for the layers in the design, and includes the following lines:

```
retargeting = LITHO DENSEOPC FILE "cmdfile.in" RETARGET_ME MAP ret_output
retargeting {COPY retargeting} DRC CHECK MAP retargeting 49
```

Stage 3 - Using the Retargeted Design

Now that you have a modified design that has been retargeted to account for etch effects, proceed with Calibre nmOPC using only the optical and resist model, since the etch effects have already been corrected.

Prerequisites

- The optical and CM1 resist models (the VEB etch model is no longer required)
- The modified design file resulting from the “[Stage 2 - Retargeting the Design](#)” procedure
- Calibre WORKbench

Procedure

1. In Calibre WORKbench, open the retargeted design file (**File > Open Layout Files**).
2. Open the RET Flow Tool (**Litho > RET Flow Tool**) and add a new nmOPC session (**File > Add New Session**).
3. Configure the RET Flow Tool as described in the [*Calibre WORKbench: RET Flow Tool User's Manual*](#). When setting up the derived layers:
 - Use the original drawn layer for fragmentation measurements

- Use the retargeted layer for simulations
4. Save the session file (**File > Save Session File**).
 5. You are now able to run the following simulation tools:
 - Mini OPC (OPC and ILT)
 - Aerial image maps
 - Dense print images
 - Cutlines
 - Fragmentation and Fragmentation information
 - EPE
 6. Continue to refine your settings and run simulations until you are satisfied with the output. Increasing the Start Layer field for Output Layer(s) Options box in the Calibre nmOPC session (click **Options** beneath the **OPC** button) increments the resulting output layer, allowing you to compare one simulation to the next.
 7. Save the modified design file (with its new derived layer from Calibre nmOPC) to a new filename.

Examples

The following fragment of code is a suggested use case for Calibre nmOPC and retargeted input files. It uses the original (unmodified design) as the fragmentation measurement layer, and the retargeted layer as the Calibre nmOPC movement layer.

```
LITHO FILE dense_opc.in [
    modelpath ./models
    optical_model_load optical_mod optical
    resist_model_load resist_mod cm1.mod
    background clear
# The next two layers are as passed in through the LITHO DENSEOPC call
    layer original_etch_target hidden atten 0.06
    layer derived_resist_target visible atten 0.06

    setlayer smoothed_resist_target = curve derived_resist_target \
        order 6 cpdist 0.70 pts_per_cp 4 maxdist 0.050 midpt 3 \
        scale1 1.25 scale2 0.7 endpt_offset -0.004 jog_tol 0.001 \
        jog_adj 0.06

# The following inset dense_opc_options perform the operation
denseopc_options opts {
    layer original_etch_target opc atten 0.06
    layer smoothed_resist_target hidden atten 0.06
    retarget_layer derived_resist_target
    ...
}
setlayer M1_OPc = denseopc original_etch_target derived_resist_target \
    MAP original_etch_target OPTIONS opt
```

Stage 4 - Verifying the OPC Output in Calibre OPCverify

Load the results of the Calibre nmOPC simulations back into Calibre OPCverify to test the corrected design file.

Prerequisites

- The optical, CM1 resist, and VEB etch models from the procedure “[Stage 1 - Calibrating the Optical, CM1 Resist, and VEB Models](#)” on page 673.
- The modified design file from the procedure “[Stage 3 - Using the Retargeted Design](#)” on page 676. This design file should contain both the retargeted layer and the Calibre nmOPC-revised layer.

Procedure

Tip

 Siemens EDA recommends performing only full-chip verification at resist level.

1. Create a Calibre OPCverify setup file. It should contain the following items:
 - Layer statements to import each of the layers (original, retargeted, nmOPC-revised).
 - Image statements to generate simulated contours for the following:
 - The full set of models
 - The optical and resist models only
 - Any Calibre OPCverify design checks you want to test your design for.

Tip

 Because you have all of the layers (original etch target and derived resist target) available to you, you can run simulations and rule checks on the post-resist and post-etch designs, and compare them to find design problems.

Currently, there is a noticeable runtime performance penalty when using VEB models (especially for models using the direct visible kernel during image generation). VEB models should only be used for small critical areas of the layout. The performance penalty is small when used in the pre-OPC model-based retargeting flow, but becomes large when used with Calibre nmOPC or Calibre OPCverify. For this reason, use of visible kernels is specifically not recommended for full-chip use other than with the pre-OPC model-based retargeting flow.

2. Create an SVRF rule file containing a call to the Calibre OPCverify setup file.
3. Run the SVRF rule file.

Examples

```
modelpath ./models
    optical_model_load opt_mod optical
    resist_model_load resist_mod cm1.mod
// etch model will only be used for post-etch
    etch_model_load etch_mod ./models/veb.mod

layer M1_OPc visible clear
background atten 0.06

setlayer M1_CONTOUR_ETCH = image optical opt_mod dose 1.0 \
    resist_model resist_mod etch_model etch_mod
setlayer M1_CONTOUR_RESIST = image optical opt_mod dose 1.0 \
    resist_model resist_mod
```

Frequently Asked Questions

There are a number of commonly-asked questions for the VEB process.

Q: Why do I need to create separate optical, resist, and etch models?

A: Each model type is a separate stage in the modeling process. Isolating the etch model is a requirement for the retargeting command.

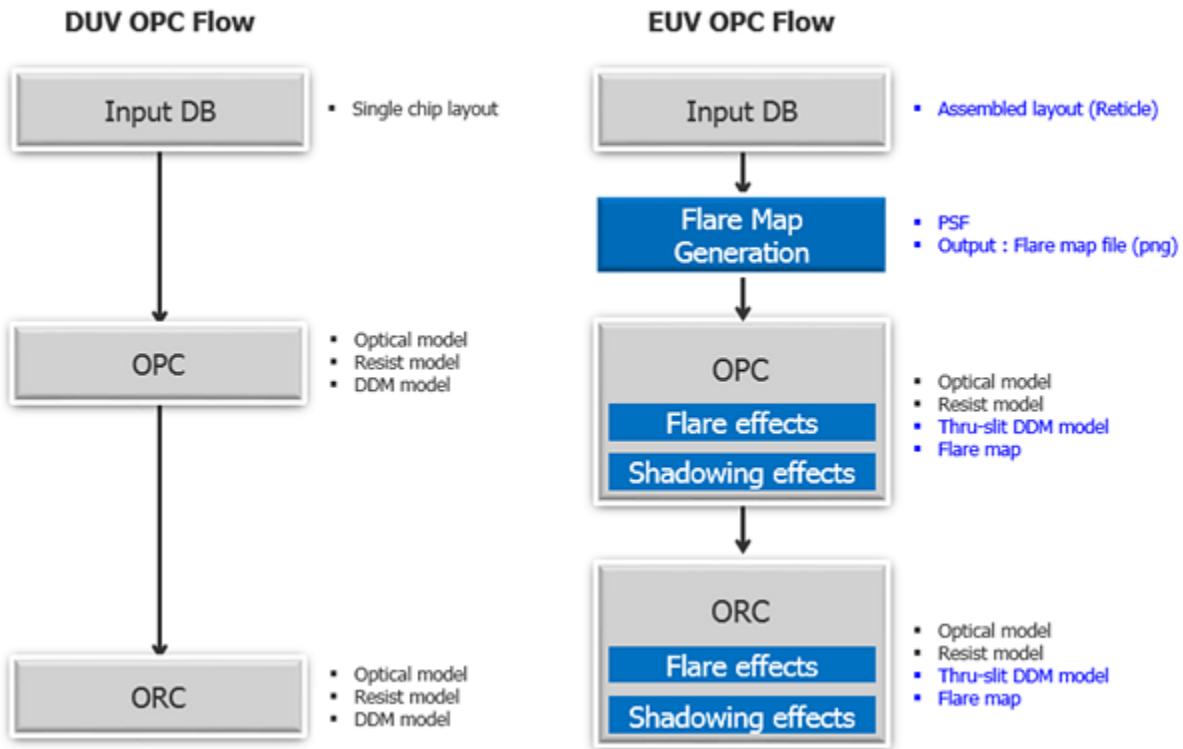
Q: Do I need to run the Calibre OPCpro tool afterwards on the retargeted design?

A: You would actually run Calibre nmOPC instead, since the Calibre WORKbench CM1 Center resist model cannot be used with Calibre OPCpro. You can run the Calibre nmOPC tool in batch mode, or by invoking the Calibre WORKbench viewer.

Appendix B

Calibre OPC EUV Flows

Extreme ultraviolet (EUV) processes require some additional keywords be added to the setup files before you can use the EUV models in simulations. In addition, while EUV models can be used in a typical post-tapeout flow with only the keyword changes, there are some additional types of runs you can do to improve overall runtime and file size.



A significant problem for EUV OPC flows is that flare is a long-range effect. Layouts are hierarchical, but unlike in DUV, the contents of the cells do not receive the same exposure. Similarly, different slit locations have different shadow bias effects. The overall result is that unless the full design can be analyzed in the context of the EUV optical model, the design must be flattened.

In Calibre nmOPC, short-range flare and long-range flare are handled separately. Short range flare is simulated on a fine grid (~5 nm) inside Calibre nmOPC. It is used to directly adjust the aerial image output. Long range flare intensity uses the biased design file and accompanying long range flare data to calculate additional effects.

A list of EUV-specific commands can be found at “[EUV Commands](#)” on page 94.

Comparison of EUV Flows	682
Generating a Flare Map	684
Automatically Analyzing the Hierarchy	685
Adding Hierarchy to the OPC Recipe	686
Reading Reticle Information Files	688
EUV Information in the Transcript.....	690

Comparison of EUV Flows

Many types of chip designs are being produced on EUV processes. Some of these can still preserve hierarchy.

For all EUV OPC flows, you need an EUV litho model containing

- an EUV optical model
- a DDM library
- a CM1 resist file
- a flare model file
- (optional) a shadow bias model
- (optional) a black border model

If you do not have an EUV litho model, follow the process outlined in “[Basic Flow for Creating an EUV Model](#)” in the *Calibre WORKbench User’s and Reference Manual*.

General EUV OPC Flow

The most basic OPC flow for EUV processes runs flat. It has the best accuracy for all types of design, but may take longer to run.

1. Generate a PNG flare map using RET FLARE_CONVOLVE, as described in “[Generating a Flare Map](#)” on page 684.
2. In a separate Calibre nmOPC run, run LITHO EUV DENSEOPC with the litho model and PNG.

Differences From DUV Setup

Some key differences from a non-EUV setup include the following:

- Set “processing_mode flat” or use the SVRF statement [RET GLOBAL LITHOFLAT](#), documented in the *Calibre Post-Tapeout Flow User’s Manual*.

- The best practices for simulation settings are different:
 - simulation_consistency_euv
 - mask_sample_grid rsm
 - optical_transform_size 512
- The litho setup file may contain euv_slit_x_center (for global litho models) and euv_field_center (if the litho model includes black border effects).

Hierarchical EUV Flow

The hierarchical flow approximates flare and DDM effects within a small tolerance, just large enough to preserve some design hierarchy. It adds an additional Calibre run to precompute the areas of the layout that can be treated as identical cells.

1. Generate a PNG flare map using RET FLARE_CONVOLVE, as described in “[Generating a Flare Map](#)” on page 684.
2. Identify hierarchy using the PNG flare map, the litho model, and RET OPTIMIZE_EUV_HDB. This produces a file, *optimize_euv_hdb.out*. See “[Automatically Analyzing the Hierarchy](#)” on page 685.
3. Run LITHO EUV DENSEOPC in hierarchical mode after including *optimize_euv_hdb.out*. Instructions for including the file are in “[Adding Hierarchy to the OPC Recipe](#)” on page 686.

Unlike the general EUV OPC flow, for hierarchical EUV you must have a global litho model.

All three stages are conducted in separate runs.

Chiplet Reuse Flow

The chiplet reuse flow combines pattern recognition and flare tolerance to improve performance. It does not require a separate run to analyze hierarchy, but you must identify the placements of the chiplets (large blocks). It is intended for reticles containing a single repeated design. All chiplets must be placed in the same orientation (no rotations or mirroring).

Make these modifications to an EUV flow:

- Identify separate flare maps for each placement using the flare_map_shift and name options of [flare_longrange](#). The flare_map_shift value will be used instead of any euv_slit_x_center. (The setup file must still contain an “anonymous” flare_longrange statement with no name or shift.)
- Specify how much two flare maps can differ and still allow OPC to be reused between chiplets with [chiplet_placement_flare_tolerance](#).

- For each chiplet, separately specify a `setlayer denseopc` command with `CHIPLET_PLACEMENT`. The index values start at 1 and should be numbered sequentially. The `setlayer denseopc` statements use the `flare_longrange` statements in the same order.

For each tile, the chiplet reuse algorithm performs the following sequence:

1. Runs Calibre nmOPC on the first chiplet.
2. For the next chiplet, adjusts the flare map using the next `flare_longrange` specification.
3. Compares the pixels of the flare map to the pixels of the previous flare map.
 - If all pixels are within the tolerance specified by `chiplet_placement_flare_tolerance`, the previous OPC results are copied to this placement.
 - If pixels are not within the tolerance, Calibre nmOPC runs on the placement.
4. Repeats Steps 2 and 3 for subsequent placements, except that “previous flare map” and “previous OPC results” now includes all previously computed flare maps and placements.

Generating a Flare Map

The flare map is a PNG file that when used in conjunction with an EUV litho model specifies the flare as a function of layout position. EUV flare effects are approximated by a convolution with the mask instead of intensity.

Restrictions and Limitations

- Each flare map is specific to a layout. Do not attempt to use a flare map on a layout other than the one it was created with.

Prerequisites

- A flare model file, preferably inside a litho model.
- A design.

Procedure

1. Set up an SVRF rule file for the design file.
2. Add the `RET FLARE_CONVOLVE` command, specifying the layer that represents the full contents of the mask that will be used in manufacturing.
3. Add a DRC CHECK MAP statement for the layer that `RET FLARE_CONVOLVE` generates. The layer itself is not used, but only operations whose output contributes to the final results are run.

4. Run the rule file as normal:

```
calibre -drc -hier -turbo rules.svrf
```

Results

A PNG file with the name specified in the pngout argument is created. This is the flare map.

Examples

The following represent a minimal SVRF file for running RET FLARE_CONVOLVE:

Example B-1. SVRF for Generating a Flare Map

```
LAYOUT PATH input.oas
LAYOUT SYSTEM oasis
LAYOUT PRIMARY '*'
PRECISION 4000

LAYOUT ULTRA FLEX YES

DRC RESULTS DATABASE dummy_flare_map.oas OASIS PSEUDO

LAYER m1_drawn 65001
LAYER boundary 65002

dummy_layer = RET FLARE_CONVOLVE m1_drawn INSIDE OF LAYER boundary
FILE /* 
    flare_model ./models/flare.fmf
    pngout ./models/input-oas-flare.png
    WRAP
*/
dummy_check { COPY dummy_layer } DRC CHECK MAP dummy_check 1
```

Related Topics

[RET FLARE_CONVOLVE](#)

Automatically Analyzing the Hierarchy

To determine where hierarchy can be preserved, the layout must be combined with the DDM information and flare map. This step must be run before OPC.

Restrictions and Limitations

- Hierarchy preservation is not guaranteed. For best results, the layout should be highly repetitive.

Prerequisites

- A global litho model for the EUV process.
- The flare map produced in “[Generating a Flare Map](#)” on page 684.

Procedure

1. Set up an SVRF rule file for the design file.
2. Add these lines to the SVRF file:

```
LAYOUT CELL OVERSIZE AUTOLITHO

LITHO FILE optimize /*  
    png ./models/input-oas-flare.png  
    global_lithomodel_path ./models/throughslit/Lithomodel  
    euv_slit_x_center 0  
*/]

dummy = RET OPTIMIZE_EUV_HDB dummy_layer MAP "DUMMY" FILE optimize
```

Where the lines use green, supply the correct values for your setup.

3. Run the rule file:

```
calibre -drc -hier -turbo rules.svrf
```

Results

After the run completes, you have an encrypted file, *optimize_euv_hdb.out*. Instructions for using this file are given in “[Adding Hierarchy to the OPC Recipe](#)” on page 686.

The transcript also reports the results of the analysis with one of these two messages:

- Based on analysis of the HDB and flare, the SVRF variable EUV_RECOMMEND_FLAT will be set to FALSE, causing OPC to run in HIERARCHICAL (not FLAT) mode.
- Based on analysis of the HDB and flare, the SVRF variable EUV_RECOMMEND_FLAT will be set to FALSE, causing OPC to run in FLAT (not HIERARCHICAL) mode.

Related Topics

[LAYOUT CELL OVERSIZE AUTOLITHO \[Calibre Post-Tapeout Flow User's Manual\]](#)

[EUV Through-Slit Litho Model Extension \[Calibre OPCVerify User's and Reference Manual\]](#)

Adding Hierarchy to the OPC Recipe

The *optimize_euv_hdb.out* file contains many of the changes you would otherwise need to add manually to an SVRF file.

Restrictions and Limitations

- Runtime with the hierarchical flow might be longer in general designs than a flat run. Generally more hierarchical layouts such as multicore and SRAM chips improve.

Prerequisites

- The *optimize_euv_hdb.out* file generated in “[Automatically Analyzing the Hierarchy](#)” on page 685.
- The EUV litho model.
- The SVRF and litho setup files for a standard EUV run.

Procedure

1. In the litho setup file called by the LITHO EUV DENSEOPC statement, add the following lines:

```
svrf_var_import EUV_RECOMMEND_FLAT
if {$EUV_RECOMMEND_FLAT > 0} {
    processing_mode flat
}
if {$EUV_RECOMMEND_FLAT == 0} {
    processing_mode hierarchical
}
```

These lines reference a variable exported by the *optimize_euv_hdb.out* file that indicates whether, given the flare and input layout, the run should be run hierarchically or flat. The setting is reported in the transcript as “VARIABLE EUV_RECOMMEND_FLAT 0” or “VARIABLE EUV_RECOMMEND_FLAT 1”

2. In the main SVRF file, add these lines:

```
include optimize_euv_hdb.out

LAYOUT TURBO FLEX YES

LAYOUT CELL OVERSIZE AUTOLITHO
```

The include line includes the optimized information. Besides the EUV_RECOMMEND_FLAT variable, it also contains additional SVRF rules that normally you would have to add by hand.

3. Run the rule file as normal using the -hier and -turbo switches. The additions in the file will automatically switch to flat mode if necessary.

```
calibre -drc -hier -turbo rules.svrf
```

Results

Calibre nmOPC is run on the layout in hierarchical mode if the particulars of the setup allow. This can save substantial time and result in smaller OASIS files post-OPC.

Related Topics

[svrf_var_import](#)

[LAYOUT CELL OVERSIZE AUTOLITHO \[Calibre Post-Tapeout Flow User's Manual\]](#)

Reading Reticle Information Files

If you have process information in a structured file, you can create a custom parser to pass the information to the litho setup file instead of manually creating individual versions. The parsed values can then be read with TVF.

Prerequisites

- Compile-time TVF
- Structured text file that contains the information

Procedure

1. Create a script that extracts the relevant information from the reticle information file and stores it in an environment variable.

In [Example B-2](#), the environment variable is rinfoLine. The example shown stores only a single value, but you could create an array of keywords and values and pass them using the same method.

2. Place the script in a TVF rule file.

[Example B-3](#) shows the relevant lines only. In a typical TVF rule file, there are also lines for creating the required SVRF statements such as LAYOUT PATH.

3. In the TVF rule file, add a call to the script supplying the filename of the reticle information file.

For example,

```
Retinfo "/home/EUV_DA/95Job_Deck/reticle_info.txt"
```

4. Add other required statements as needed and save the file with the suffix “.tvf.”

Results

When you run the TVF file, it reads the information file to supply the values to the indicated keywords. TVF files can be run using the same syntax as SVRF files:

```
calibre -drc -hier -turbo rules.tvf |& tee tvf.log
```

Examples

This is the code used in the procedure. Note that the Tcl procedure *must* be in the TVF file for the information it adds to “env” to be available to the rest of the program.

Example B-2. Tcl Procedure to Store Information in Variable

```
Proc Retinfo {input_information_file_name} {
    # Sync local variable env to global namespace.
    global env

    # Open file
    if [catch {open $input_reticle_info_file_name "r"} FIID ] {
        puts "fail to open file $in_f"
        exit
    }

    set i 0
    set my_value ""
    while {[gets $FIID line] >= 0} {
        if {$i == 0} {
            lappend my_value "0"
        } else {
            # Place custom parser here
            # Use the setup file keywords as the array index
        }
        incr i
    }
    puts $my_value
    set env(rinfoLine) $my_value
}
```

Example B-3. TVF Lines to Use Parser Procedure

```
#! tvf

namespace import tvf::*
...
# Call the Tcl procedure.
Retinfo "input_information_file_name"
...
VERBATIM {
LITHO FILE nmopc_setup /*

# [lindex ...] is specific to the way the parser constructed rinfoLine.
# If the parser added only a single type of information, you can access
# it like the euv_slit_x_center line below.
    euv_slit_x_center [lindex $env(rinfoLine) 1]
...
*/】
} //END VERBATIM
```

Related Topics

[Custom Tcl-Based Script Creation](#)

[Tcl Verification Format \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

EUV Information in the Transcript

When you use EUV models, the transcript includes some additional information that can help you diagnose setup problems.

When you use global litho models, the parser checks slit positions. Currently the maximum spread across all slits of a single source is 26 mm.

Messages like these indicate that a global litho model is not set up correctly:

```
...
//-- WARN: Slit position 79362.4 um is outside expected range of +/- 14000 um
//-- WARN: Slit position 48028.7 um is outside expected range of +/- 14000 um
//-- WARN: Slit position -58346.2 um is outside expected range of +/- 14000 um
...
//-- WARN: Outermost EUV slits, centered at x = -120.315968 mm and 109.965310 mm,
are 230.281278 mm apart. This exceeds the fixed maximum expected distance of
26.000000 mm. Check global litho model x values for accidental overscaling.
```

The transcript also includes information about the positions of the outermost slits when reporting the coordinates to be used for interpolating DDM libraries, as in this example:

```
//-- INFO: Global Lithomodel ct98c_thruslit_correct will use interpolated
DDM libraries across slit
//-- INFO: Outermost EUV slits, centered at x = -12.031597 mm and 10.996531 mm,
are 23.028128 mm apart.
```

Appendix C

Calibre mlOPC

The Calibre® mlOPC license unlocks the machine learning API for use with Calibre nmOPC runs. The machine learning API uses models generated from trained neural nets to reduce OPC runtime with no loss of accuracy.

Calibre mlOPC functionality enhances Calibre nmOPC performance rather than replacing traditional dense OPC. The machine learning models provide the initial displacement, which Calibre nmOPC optimizes.

The machine learning model uses a type of architecture referred to as “artificial neural network,” or ANN. ANNs use hidden layers to calculate outputs based on multiple parameters. The Calibre mlOPC model uses fragment parameters and returns an output layer.

Calibre mlOPC Requirements	691
Collecting Model Data.....	692
Using the Trained Model.....	694

Calibre mlOPC Requirements

Calibre mlOPC requires a product license and certain files, as well as Calibre version 2018.4 or newer.

Licensing

License requirements and considerations are described in “[Calibre mlOPC](#)” in the *Calibre Administrator’s Guide*.

Required Files

To capture initial data for the machine learning model, you need the following:

- Production-level Calibre nmOPC setup that uses a litho model instead of separately loading each model. Do not use a rule file or setup that is still being developed.
- Layout with representative patterns that cover acceptable behavior for a layer. There should be a variety of patterns in the layout and a retargeting layer if necessary. (That is, the retargeting should not be done by the Calibre nmOPC setup as part of the run.)

The layout should include SRAFs.

- Model template in *.mod* format that identifies the data to collect. Ask your Siemens representative for a “seed model” reflecting the most recent best practices.

To train the model, you need the following:

- Verification script to mark residual EPEs or other lithographic issues that detract from overall quality.
- Calibration file and calibrated model created during data capture.
- Configuration file.
- Training scripts. These are undergoing active enhancement; contact your Siemens representative to train the model.

To use the trained model with layouts that have not undergone OPC, you need the following:

- Calibre mlOPC license.
- The same original production-level Calibre nmOPC setup that was used for data capture.
- The trained machine learning model.

Calibre mlOPC is not compatible with Calibre LPE or EUV_MODEL_REDUCTION (the EUV advanced hierarchical flow).

Collecting Model Data

The first step is to populate a model template with data from a known-good rule file and layout.

Prerequisites

- Production-level Calibre nmOPC rule file.
- Layout with representative arrangements of main features and SRAFs.
- Model template (*.mod* file).

See “[Calibre mlOPC Requirements](#)” on page 691 for more information on these.

Procedure

1. If necessary, modify the production rule file to run retargeting separately from OPC, and save the layout with the retargeted layer.
2. In the SVRF file, change “LITHO DENSEOPC” to “LITHO ML DENSEOPC”.
3. In the litho setup file called by the LITHO ML DENSEOPC statement, add the model template as follows:

```
ml_featurevector_load nmod phv.mod
```

where “nmod” is the reference used by the rest of the commands for the model, and *phv.mod* is the filename of the model template.

4. In the denseopc_options block, identify the section that performs the OPC. Add ML_VECTOR_CAPTURE_INIT and ML_VECTOR_CAPTURE_END to capture information about the fragments of interest.
Any fragment modification such as splitting or aligning should be done before ML_VECTOR_CAPTURE_INIT. Small frozen jogs should not be included in the tag set.
5. If the setup file uses PW OPC (a -PW keyword in OPC_ITERATION), be sure each PW OPC iteration is followed by a PW_RETARGET statement so that the retargeting is captured.
6. Save the modified rule file and run it with your usual invocation.

Results

After the run, the following files are available:

- *phv0.mod*

This is the calibrated model file. (The model still has to be trained.) Do not modify it by hand; treat it as any other model file.

- *calibration_vectors.csv* or name specified in ML_VECTOR_CAPTURE_END

This is the calibration file, also called the normalized file. It contains feature vectors used in model training.

If you later determine that the training data needs additional features, you can add them to the calibration and model files by adding -old and -oldmodel to ML_VECTOR_CAPTURE_END and re-running with a different layout.

Examples

In the following example, the modified parts are shown in bold.

```
modelpath /shared/models:models:.  
ml_featurevector_load nmod phv.mod          # Step 3  
  
layer for_opc  
  
denseopc_options opc_opts {  
    version 1  
    layer for_opc    opc mask_layer 0  
    layer opc.curve hidden  
    ...  
    retarget_layer opc.curve  
    image lmod          # Loads optical and resist models  
    ...
```

```
fragment_layer for_opc {
    ...
}

mrc_rule internal for_opc {
    ...
}

...
sraf_sites_create
NEWTAG all for_opc -out allFrags
NEWTAG edge allFrags len >=0.1 <0.14 corner1 convex corner2 convex \
        -out fat_le
NEWTAG external allFrags fat_le opposite range >=0.05 <0.1 \
        -out fat_le_opposite
SITES_DELETE fat_le_opposite

ML_VECTOR_CAPTURE_INIT allFrags nmod duv # Step 4

NEWTAG edge allFrags...
...
FEEDBACK -keep line_end_adj -0.2

OPC_ITERATION 4
OPC_ITERATION 3 -PW ...
sraf_print_avoidance ...
OPC_ITERATION 5 -PW...
PW_RETARGET
OUTPUT_RETARGET for_opc -out pw_retarget

ML_VECTOR_CAPTURE_END allFrags           # Step 4
}

setlayer opc_out = denseopc ... map for_opc OPTIONS opc_opts
```

Related Topics

[ml_featurevector_load](#)
[ML_VECTOR_CAPTURE_INIT](#)
[ML_VECTOR_CAPTURE_END](#)

Using the Trained Model

The trained machine learning model replaces many OPC iterations by providing an initial fragment displacement. This is then tuned with a few standard OPC iterations.

Restrictions and Limitations

- Machine learning models should only be used for the technology process that they were trained on.

- The RET Flow Tool in Calibre WORKbench is not supported. Run the modified rule file from the command line in batch mode.

Prerequisites

- Trained machine learning model files, generally named *frozen_model.pb* and *phv0.mod*.
- The same production-level Calibre nmOPC rule file that was used in “[Collecting Model Data](#)” on page 692.

Procedure

1. In the SVRF file, change “LITHO DENSEOPC” to “LITHO ML DENSEOPC”.
2. In the litho setup file, add the machine learning model files using *ml_model_load* and *ml_featurevector_load*. For example:

```
LITHO FILE opc.in [ /*  
    modelpath ./  
    ml_model_load      nmod frozen_model.pb  
    ml_feature_vector_load nmod phv0.mod
```

It is important that both commands give the same name to the model; in this case, “nmod.”

If the rule file uses PW OPC and PW_RETARGET, there are models for both displacement and retargeting. In that case, the lines might look like this:

```
LITHO FILE opc.in [ /*  
    modelpath ./  
    ml_model_load      nmod_disp disp_dir/frozen_model.pb  
    ml_model_load      nmod_rtrg rtrg_dir/frozen_model.pb  
    ml_feature_vector_load nmod_disp phv0.mod  
    ml_feature_vector_load nmod_rtrg phv0.mod
```

The same feature vector file is used by both the *nmod_disp* and *nmod_rtrg* models.

3. Find where the OPC iterations are called and add the *ML_OPC* command just before it. This should also be after all fragmentation changes, such as splitting or deleting.

```
ML_OPC all_frags nmod
```

The example command applies the model “nmod” to all fragments.

OPC iterations are performed by *OPC_ITERATION*, *OPC_XMEEF_ITERATION*, or *FRAGMENT_MOVE*.

4. If the models include both displacement and retargeting, also add a *PW_RETARGET* statement after the *ML_OPC* statement.
5. Reduce the number of OPC iterations by the standard command to 2 or 3. Keep the *OPC_ITERATION* or *OPC_XMEEF_ITERATION -PW* iterations if applicable.

6. For those fragments being placed by the machine learning model, remove the DISPLACEMENT, FRAGMENT_SET_DISPLACEMENT, TDBIAS, and FEEDBACK commands.
7. (Optional) To see the output at different stages, use [OUTPUT_OPCT](#).
8. Save the files and run from the command line.

Calibre nmOPC Glossary

b-spline

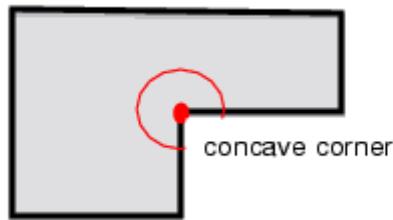
Also known as basis splines, a spline function with minimal support with respect to a given degree, smoothness, and domain partition. Spline functions of a given degree, smoothness and domain partition can be represented as a linear combination of B-splines of that same degree and smoothness, and over that same partition.

Compact Model 1 (CM1)

A dense modeling methodology that is more accurate than site-based modeling. CM1 takes advantage of the data available for resist simulation with a dense simulation engine.

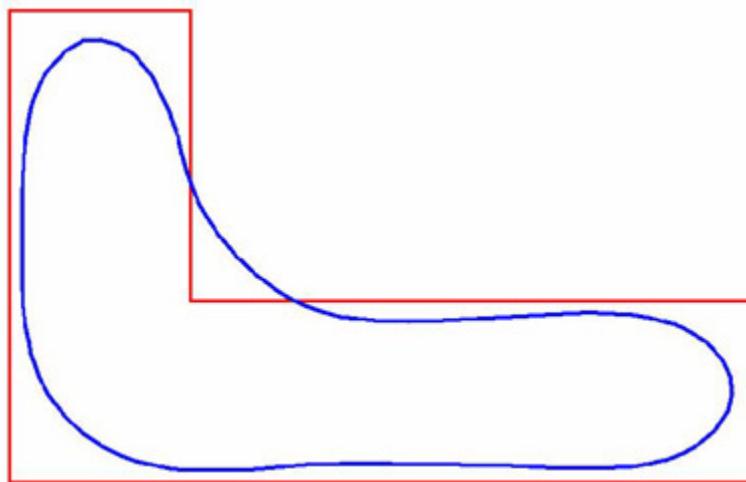
concave corner

A corner whose interior angle is greater than 180 degrees.



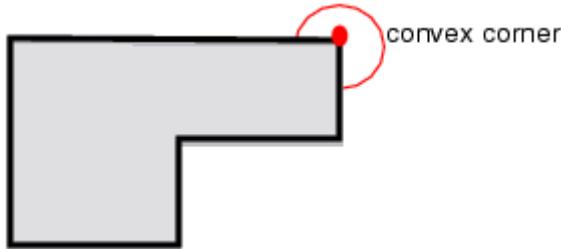
contour

An approximation of a dense simulated image at the point of resolution where an image can be clearly formed based on the resist and optical models. EPE is measured from the target layer to the contour image. Multiple contours may be generated for process window constraint checking.



convex corner

A corner whose exterior angle is greater than 180 degrees.



correction layer

A layer whose edges will be moved during OPC to compensate for optical and processing errors.

cost function

An optimizing function that attempts to maximize or minimize certain effects by choosing the values of different variables to garner the best possible results.

critical dimension (CD)

The size of a feature to be printed that is important to dimensional control.

cutline

A line, usually orthogonal to the fragment used to measure EPE.

dense simulation

A simulation that samples data at grid points on a pixel-based grid that is global for each cell. Grid points are evenly spaced throughout the entire design. The simulation data is calculated at every grid point on the design. This allows you to predict image distortions introduced by the exposure, resist, and etch processes. For Calibre nmOPC, calculations only depend on the area and simulation grid.

derived layer

A new layer that is created in a Calibre application by modifying another layer or layers in some way.

edge

A line segment between two adjacent corners of a polygon.

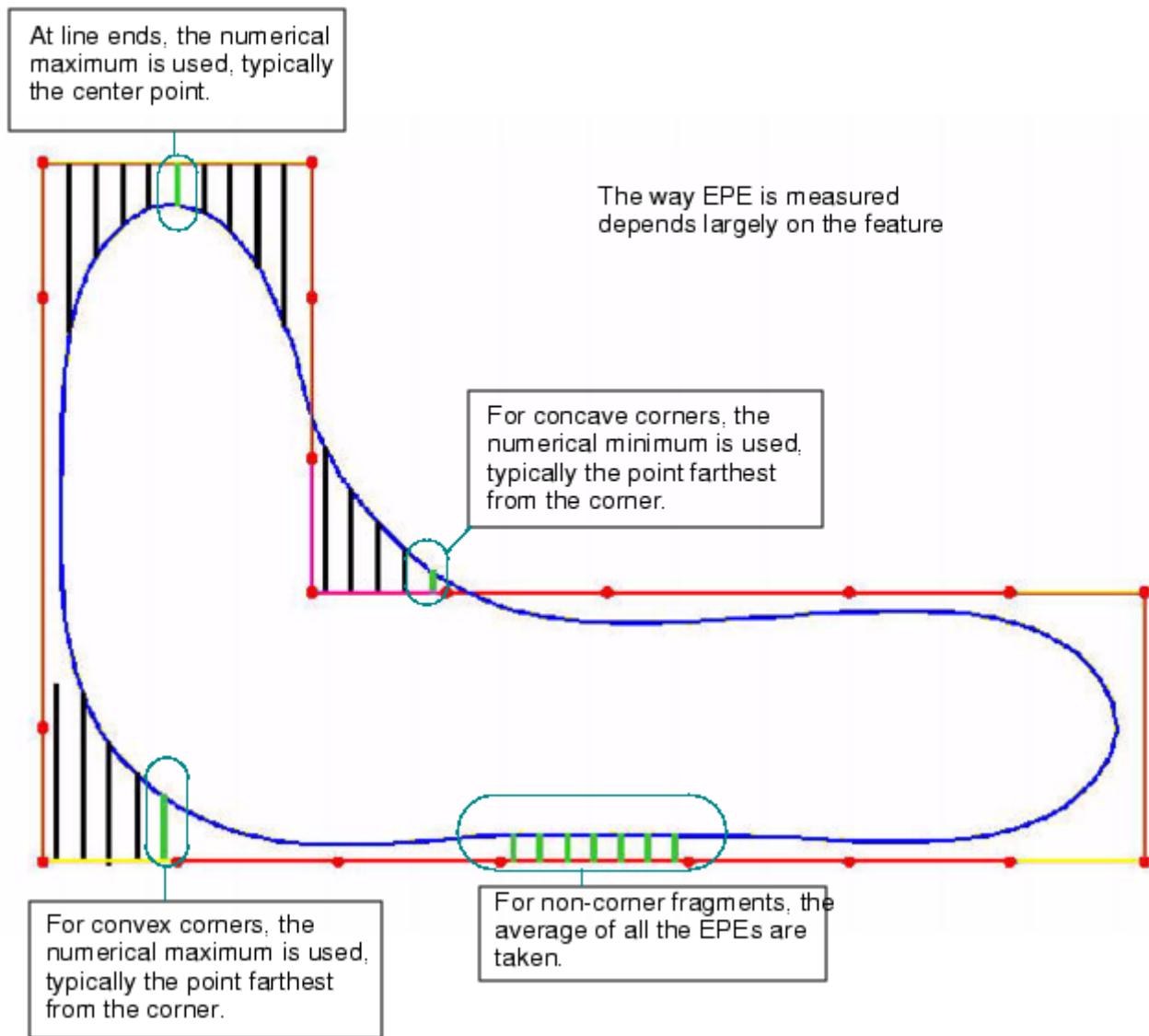


edge fragment

Any edge or part of an edge in a layout polygon. Note that this is a generic term. “Edge fragment” is synonymous with “fragment.”

edge placement error (EPE)

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 on a dense simulation is different than on a sparse simulation, and varies according to the feature.



EPE

See edge placement error.

EPE-based fragment movement

A correction technique that uses the simulated EPE values to calculate how far an edge will move. Fragments are moved based on the following formula:

$$\text{EPE}_{\text{effective}} * \text{feedback}$$

where:

feedback is a factor that can be adjusted for every fragment during each iteration.

external layer

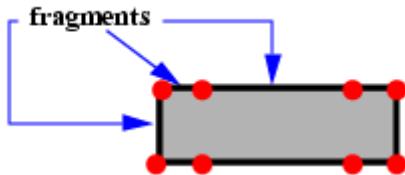
A layer that contains a corrected mask design. Its presence signals the LITHO command that the opc layer's geometries represent the target design rather than the mask design itself.

fast Fourier transform (FFT)

A mathematical algorithm used to transform the spatial domain representation of the layout into frequency domain and, in the case of Calibre nmOPC, is used to translate layout polygons to contours on a pixel-based simulation grid. FFTs are also known as simulation frames.

fragment

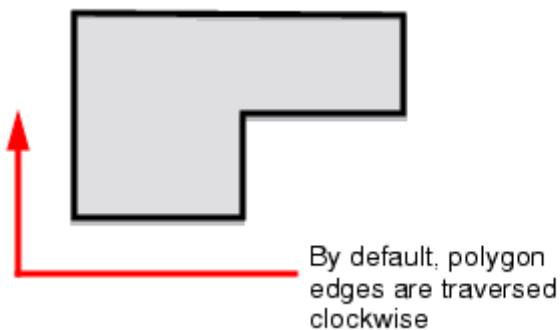
A line segment between two fragmentation vertices (points where the line segments break). A fragment, which may also be referred to as an *edge fragment*, can be an entire edge if the two vertices are both corners, or a part of an edge if at least one vertex is not.



fragment orientation

Fragment orientation refers to orienting one fragment in relation to adjacent fragments. For the terms “before” and “after” to have meaning with respect to adjacent fragments, they must be defined with respect to an agreed upon fragment orientation. Within the Calibre RET toolset, all edges and fragments are oriented in a direction such that the interior is always to the right of the edge. To get to the fragment “after” a given edge move along the edge such that the interior is to

the right. “Before” is defined as the opposite of after, and “next to” is defined as both before and after.



fragmentation

The process of breaking up a layout polygon’s edges into smaller segments called fragments. Fragmentation is important because when Calibre nmOPC modifies a layout to correct for inaccuracies introduced during the photolithographic process, it operates on individual fragments rather than entire edges. To make corrections, the tool moves each fragment as necessary to generate a mask that will result in the optimal image. Thus, the level of fragmentation you define can affect the resulting OPC.

fragmentation parameters

Parameters in a setup file that affect the way fragmentation vertices are created on a layer.

fragmentation unit (frgu)

The basic unit of length for fragmenting polygon edges into fragments. For Calibre nmOPC, the fragmentation unit is a scaled Nyquist value. Any reference to “fragmentation unit” refers to:

$$\text{fragmentation unit (frgu)} = \text{scale} * \text{Nyquist}$$

where the *scale* is a user-defined value, by default 1.0. The term “frgu” is equivalent to “fragmentation base value”, as reported by the nmOPC and nmbIAS run log files.

Note

 There can sometimes be a difference between a calculated value of frgu based on scale * Nyquist versus the value reported in the transcript under AERIAL INTENSITY SAMPLING GRID. This is because the actual Nyquist value is discretized to the nearest dbu, then multiplied by scale, then rounded to the nearest step size to get the fragmentation unit.

gauge object file

A file containing data structures representing gauges that is created by the CM1 Center from sample file spreadsheets.

hidden layer

A layer whose data does not affect the outcome of a LITHO tool run. However, the OPC process outputs data to hidden layers in the form of boxes that mark EPEs.

hierarchy boundary

The boundary of a cell in a hierarchical design. The hierarchy boundary encloses a portion of a design that is manipulated as a single unit in some level of the hierarchy.

interaction distance

How far Calibre nmOPC checks for geometry. The value is based on optical diameter, but certain commands may increase it. The value used is reported in the transcript in the SE: table.

interfeature fragmentation

A type of fragmentation that evaluates each polygon in terms of proximity to nearby polygons and adds vertices to edges that are within a specified distance from a corner on another figure.

intrafeature fragmentation

A type of fragmentation in which each polygon on the layout is evaluated independently of other polygons and adds vertices at set distances from the figure's corners.

island layer

A layer that defines areas on the opc or correction layer that are of special interest or require special treatment.

island-layer fragmentation

A type of fragmentation that inserts vertices where the edges on an opc layer or correction layer intersect edges or vertices on an island layer.

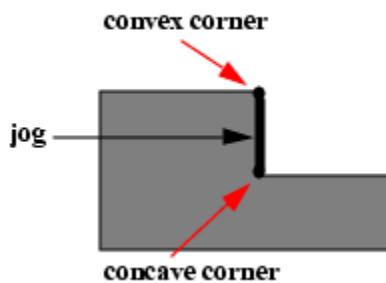
jog

Note

 The definition of jog varies according to the OPC type. The definition that follows applies only to dense OPC.

An entire fragment that satisfies all of the following criteria:

- one convex corner
- one concave corner
- length $\leq 1.5 * \text{fragmentation unit (frgu)}$



layer type

An argument to the layer keyword in the setup file. The layer type indicates how the information on a layer is used by the LITHO toolset.

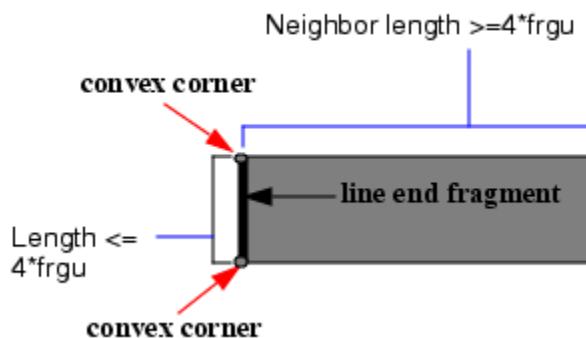
line end fragment

Note

 The definition of line end fragment varies according to the OPC type. The definition that follows applies only to dense OPC.

An entire edge that satisfies all of the following criteria:

- length constraint of $4*f_{rgu}$
- edges with length $\leq 4*f_{rgu}$
- 2 convex corners
- A neighbor fragment length $\geq 4*f_{rgu}$



mask error enhancement factor (MEEF)

The change in the resist critical dimension (CD) per unit change in the corresponding mask CD where the mask CD is in the wafer dimension. When performing linear imaging near the resolution limit, small errors in the mask dimension can translate to large errors in the final resist CD. The amplification of these errors is characterized by MEEF.

mask rule constraint (MRC)

A behavioral limitation observed by Calibre nmOPC that limits fragment movement during the OPC process based on internal and external check operations.

multithreading (MT)

A type of high-speed processing in which interdependent parallel operations are performed using a computer with multiple CPUs. Calibre applications use fine-grained cell-based MT.

Nyquist sampling distance (NSD)

The largest sampling distance of aerial intensity that does not introduce aliasing errors. It is calculated as follows:

$$\text{Nyquist} = f(0.25 * \lambda / (\text{NA} * (1 + \sigma)))$$

Where f is a function that slightly increases the frequency passband, which serves to slightly reduce the Nyquist sampling rate.

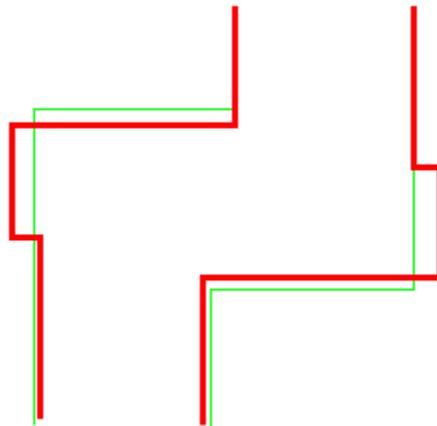
The dense simulations performed by Calibre nmOPC use a grid size equal to the Nyquist sampling distance.

opc layer

The target layer for most LITHO operations. This layer contains the actual geometry that you intend to transfer onto silicon.

optical and process correction (OPC)

A process performed on a layout to compensate for distortions introduced during the lithography process used to print on a wafer. The output is a layer (or set of layers) containing a modified version of the original data enhanced with features such as serifs, biasing, and line end hammer heads. The following figure illustrates a corrected section of an IC layout showing the original layer in green and the corrected layer in red.



The effects of the lithography process are simulated (using optical and resist model information) in order to verify whether modifications to the design will produce the desired corrections.

optically visible

Data that appears on a mask and interacts with other geometries to affect the final printed design.

optical model

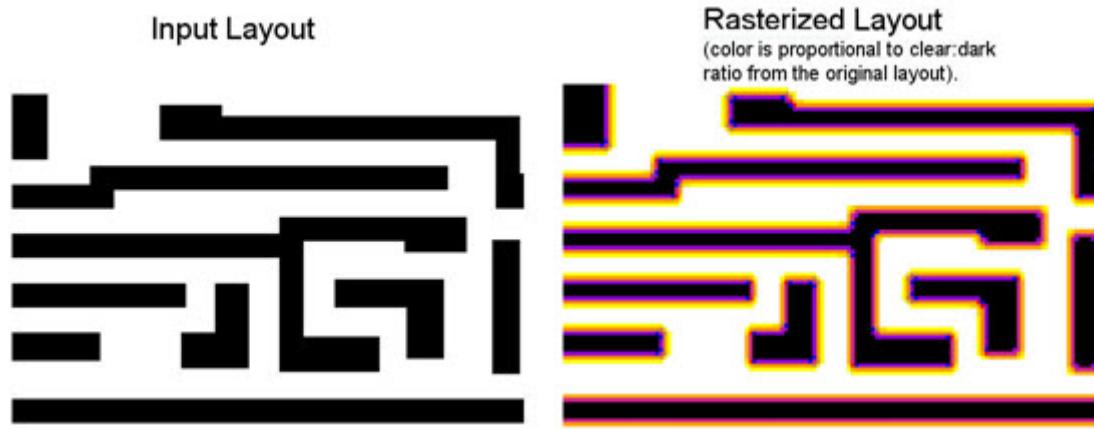
A set of parameters that describes the focus behavior of the lens, light source, and stepper, used to simulate the effects of the lithography process on a mask.

process window

The tolerable change in dose and focus until the design fails to yield on silicon. A process window is made by plotting contours that correspond to various specification limits, as a function of exposure and focus.

rasterization

A process that converts polygon data on the database unit grid into a grey scale representation on the mask sample grid. This is how the dense simulation engine begins processing shapes on the layout into simulated contours on the pixel grid.



resist model

A set of parameters that predicts the effects of a specific resist/etch process based on experimental or simulated measurements. A resist model is used to simulate the effects of the lithography process on a mask. Calibre nmOPC only supports the CM1 resist model.

retarget

The technique of using a rounded-corner version of the drawn polygons for computing EPE when performing OPC. The rounded version is sometimes also called a curved or smoothed target.

sample file spreadsheet

A data file that contains measurement information recorded on a per-test structure basis.

session file

In the context of the Calibre WORKbench simulation and modeling tool, a session file records the positions and values of switches in the RET Flow Tool. A session file can be used to create a setup file.

setup file

A text file containing the keywords, variables, and commands needed to configure a Calibre nmOPC run.

A setup file contains the following types of controls:

- Model definitions

-
- Fragmentation settings
 - Input layer definitions
 - Pre-OPC preparation
 - OPC algorithm
 - Post-OPC verification

site

Note

 The definition of site varies according to the OPC type. The definition that follows applies only to dense OPC.

A measurement object similar to a cutline that simulates image intensity along a straight line. By default, sites are perpendicular to the fragment or curve they are measuring and extend on either side. The type of site determines other characteristics, such as how far it extends.

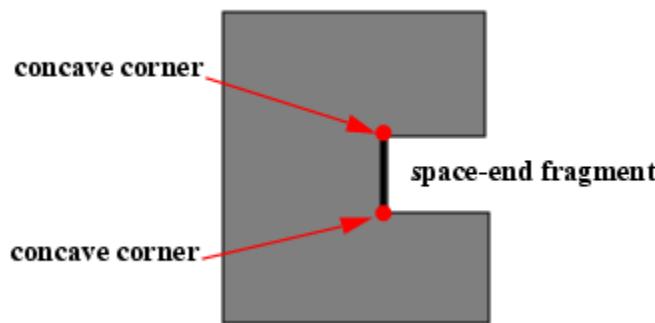
space-end fragment

Note

 The definition of space-end fragment varies according to the OPC type. The definition that follows applies only to dense OPC.

An entire edge that satisfies all of the following criteria:

- length constraint of $4*f_{rgu}$
- edges with length $\leq 4*f_{rgu}$
- 2 concave corners
- A neighbor fragment length $\geq 4*f_{rgu}$



spline

A piecewise parametric polynomial curve that can be used for smoothing of either one-dimensional or multidimensional data. Spline functions for interpolation are normally

determined as the minimizers of suitable measures of roughness subject to the interpolation constraints.

tagging

The process of defining named subsets of edge fragments so you can control how the different types of edge fragments are treated. Tagging algorithms can be created using the Tcl-based Calibre nmOPC customization script.

tile

A rectangular area of a design that is processed as a unit. Large designs or cells can be divided into tiles to improve processing speed.

tile boundaries

The boundary of a tile, where one tile abuts another tile.

variable etch bias (VEB)

An additional modeling step that separately accounts for etch effects. VEB performs the following:

- Separates out the modeling of etch bias effects from optical and resist effects.
- Separates out the correction of etch effects and correction of optical and resist effects.
- Derives an etch-corrected design based on the VEB model, replacing the rules-based OPC step. It uses fragmentation and edge movement calculations, similar to making OPC corrections to a layout.

vertex

A point added to an edge or fragment to divide that edge or fragment into smaller fragments. Fragmentation vertices are system-generated at run time.

Index

— Symbols —

[]²⁸
{}²⁸
|²⁸

— A —

Accuracy, [636](#)
active_layer, [179](#)
adjust_target_control_sites, [183](#)
algorithm, [184](#)
Angle constraints, [479](#), [498](#)
Angle snapping, [273](#)
Annotate fragments, [485](#), [507](#), [521](#)
Attenuated masks
 background, [188](#)
 transmission values, [289](#)

— B —

background, [187](#)
Best practices, [635](#)
Bias limit, [532](#)
Bold words, [28](#)
B-splines, [697](#)

— C —

Calibre mpOPC, [23](#)
Calibre nmOPC
 in VEB retargeting flow, [676](#)
Calibre OPCVerify
 in VEB retargeting, [675](#)
 VEB retargeting verification, [678](#)
CD measurements, [470](#)
check_movement, [190](#)
chiplet_placement_flare_tolerance, [191](#)
CLASSIFY_FRAGMENTS, [100](#)
clone_transformed_cells, [119](#)
Cloning cells, [120](#)
CM1 Center, [673](#)
Concave corners, [697](#)
Configuration

ddm_model_load command, [121](#)
etch_model_load command, [125](#)
flare_model_load command, [130](#)
modelpath command, [141](#)
optical_model_load command, [142](#)
resist_model_load command, [145](#)

Consistency, [424](#)
Contact auto-rounding, [57](#)
Contact layer type, [57](#)
Contact layers, [649](#)
Contours, [697](#)
Control sites, [565](#)
controller, [340](#)
Controlling OPC
 concurrency, [232](#), [262](#)
 maximum fragment length, [253](#)
 maximum movement, [244](#)
 minimum fragment length, [257](#)

Convergence, [199](#), [200](#)
Convex corners, [698](#)
Copy fragmentation, [260](#)
Corner chopping, [658](#)
Corner fragmentation, [63](#), [213](#)
corner_control, [62](#), [192](#)
Corners
 concave, [697](#)
 convex, [303](#), [698](#)
Correction layers, [56](#), [286](#), [698](#)
Cost function, [698](#)
Courier font, [28](#)
Critical dimension, [698](#)
Cross MEEF (XMEEF), [544](#)
Custom fragmentation, [63](#)
Cutline, [698](#)

— D —

Dark
 background, [187](#)
 features, [133](#)
DBU2UU, [397](#)

ddm_model_load setup command, 121
Dense aerial images
 grid points, 34
Dense simulation, 698
DENSE_CD, 398
DENSE_EPE, 400
DENSE_SIMULATE, 402
denseopc_options, 122, 382
Derived layers, 698
DFM properties, 333
Direct etch effect correction, 71
Direction, 700
Displacement, 530
DISPLACEMENT command, 193, 404
displacement_limit_mode, 193
Do Not OPC region, 79
Dose, 133
Double patterning, 509
Double pipes, 28
Double-patterning example, 667
Double-patterning mask correction, 23
DSA models, 361

— E —

Edge placement errors
 calculating, 56, 286
 definition of, 699
 measurement, 35
 oscillations, 195
Edges
 definition, 698
 fragments, 699
effective_epe_regression, 195
Enclosed by, 354
Enclosure, 351, 627
enhance_corner_to_corner_site_pairing, 196
EPE *see* Edge placement errors
epe_spacing, 197
Errors
 minedgelength violation, 210, 268
 nlitho model, 139
Etch bias, 671
etch_model, 198
etch_model_load setup command, 125
EUV, 109
EUV commands, 113, 126, 130, 132, 171

chiplet_placement_flare_tolerance, 191
EUV_RECOMMEND_FLAT, 686
External layers, 700

— F —

Factor
 used with background attenuation, 187
Fastest runtime, 615
FEEDBACK, 201, 413
feedback, 199
feedback_balance, 200
feedback_mode, 201
Files
 lithomodel, 359
 sample spreadsheet, 705
 session, 705
 setup, 31, 705
 SVRF, 32, 34
Flare maps, 684
flare_longrange command, 126
flare_map_shift, 129
flare_model, 203
flare_model_load setup command, 130
fragmaxedge, 416
FRAGMENT delete, 418
FRAGMENT end, 419
FRAGMENT modify, 420
FRAGMENT split, 422
fragment_coincident, 204, 260
FRAGMENT_CONFORM, 424
fragment_corner, 207
FRAGMENT_EPE_MEASURE_METHOD, 426
fragment_flaglayer, 217
FRAGMENT_GET, 427
FRAGMENT_GET_DISPLACEMENT, 429
fragment_grid, 218
fragment_inter, 219
fragment_interlayers, 232, 234
fragment_island, 236
FRAGMENT_ITERATOR first, 431
FRAGMENT_ITERATOR next, 431
FRAGMENT_ITERATOR release, 431
fragment_layer, 242
fragment_layer_order, 248
fragment_marker, 252

fragment_max, 253, 265
FRAGMENT_MAX_DISPLACEMENT, 433
fragment_min, 257
fragment_minjog, 259
FRAGMENT_MOVE, 435
fragment_nearly_coincident, 260
fragment_nop, 262
fragment_ripple, 266
FRAGMENT_SET_DISPLACEMENT, 437
fragment_visible, 271
Fragmentation
 concave corner, 213
 copying, 260
 custom fragmentation, 63
 definition, 701
 global vs custom, 635
 interfeature, 227, 238, 702
 intrafeature, 702
 island layer, 702
 maximum fragment length, 253, 265
 minimum fragment length, 257
 parameters, 701
 priority, 222
 snapping, 218
 vertices, 707
Fragmentation unit (fgru), 701
Fragments
 adjacent space ends, 209, 212, 267
 annotating, 485
 definition, 700
 edge, 699
 line end, 703
 promotion, 206
 ripple, 227
 sequential, 512
 space end, 706
fragtypetag, 439
Frames, 700
freeze_skew_edges, 272
FullScale, 41

— G —
Gates, 331
Gauges, 701
Generate
 layer based image, 31, 84

GET_FRAGMENT_EPE_INFO, 441
global_lithomodel_path, 132
grids_for_angle_45_snap, 273

— H —
Heavy font, 28
Hidden layers, 56, 701
Hierarchical EUV flow, 685, 686
Hierarchy boundaries, 702
Hint offset, 656

— I —
Illumination types, 133
image, 274
Image constraints
 offsets, 567
 wafer_enclose, 351
 wafer_enclosedby, 354
 wafer_exclude, 357
Implicit fragmentation, 206, 265
Interfeature fragmentation, 63, 64, 222, 702
 island crossing, 238
Intrafeature
 split, 210, 268
Intrafeature fragmentation, 207, 266, 702
Island layer, 57, 702
Island layer fragmentation, 64, 236, 702
Italic font, 28
Iterations, 529

— J —
jog_freeze, 279, 618
jog_ignore_metric, 280, 618
Jogs
 definition, 702
 reducing impact on quality, 617

— K —
Keep out zones, 357

— L —
lapi_exec_tcl, 282
layer
 denseopc_options, 285
 LITHO DENSEOPC, 133
Layers, 52
 contact, 57

correction, 56, 698
derived, 698
hidden, 56
island layer, 57
names, 133
OPC, 56
property, 333, 507
target, 57, 287
types, 133, 285
visible, 57

Limits
on bias, 532
on OPC movement, 244

Line end fragments, 703
line_end_fragment, 292
lint, 293
lintmodel, 135
LITHO CELLARRAY, 100
LITHO CL DENSEOPC, 103
LITHO DENSEOPC, 98, 99, 102
LITHO EUV DENSEOPC, 105

Litho model
format, 359
overview, 48

Lithomodel, 49, 359
Long range flare, 684

M

Manufacturability checks, 311
Mask Error Enhancement Factor (MEEF), 543, 703
Mask sizing, 275
mask_chip_corner, 297
Masks
background illumination, 188
Matrix OPC, 543
Matrix ret argeting, 537
max_iter_movement, 301
max_iterations, 300, 534
max_opc_move, 302
Maxedgelenlength fragmentation, 64, 253
MEASURE, 442
MEASURE_PER_FRAGMENT, 444
MEASURE_PER_FRAGMENT_QUERY_COUNT, 447
MEASURE_PER_FRAGMENT_QUERY_FRAGMENT, 448
MEASURE_PER_FRAGMENT_QUERY_N_EXT, 450
MEEF *see* Mask Error Enhancement Factor
Memory layouts, 100
min_dog_ear_length, 303
Minimum keyword, 28
modelpath setup command, 141

Models
calibrating, 673
machine learning, 139
optical, 704
resist, 705
mpOPC, 509
mpopc, 304
MRC, 460, 703
mrc_mode, 310
mrc_rule, 311, 463
best practices, 636
mrc_rule_area, 318
Multilayer fragmentation, 64, 242
Multi-patterning conditions, 326
Multi-patterning example, 667
Multi-patterning mask correction, 23

Multiple masks
defining background, 188
defining mask number and dose, 133
minimum spacing, 532

Multithreaded processing, 703

N

NEWTAG 45_angle, 520, 528
NEWTAG all, 466
NEWTAG all_angle, 520, 528
NEWTAG blocked, 469
NEWTAG cd, 470
NEWTAG complete, 472
NEWTAG corner, 474
NEWTAG displacement, 477
NEWTAG edge, 479
NEWTAG enclose, 490
NEWTAG enclosing, 490
NEWTAG epe, 483
NEWTAG expression, 485
NEWTAG external, 490

NEWTAG facing_pattern, 496
NEWTAG fragment, 498
NEWTAG horizontal, 520, 528
NEWTAG internal, 490
NEWTAG line_end, 503
NEWTAG mrcViolation, 504
NEWTAG neighbor, 505
NEWTAG property, 507
NEWTAG sequence, 512
NEWTAG space_end, 503
NEWTAG topological, 518
NEWTAG vertical, 520, 528
Next, 700
NMBIAS_MEASURE_TAG, 521
no_opc_region, 320
nominal_epe_limit, 321, 324
Nyquist sampling distance, 704

— O —

one_time_site_pairing, 322
OPC definition, 704
OPC layers, 56
 definition, 704
 function, 286
opc_grid_multiplier, 323
OPC_ITERATION, 529
OPC_XMEEF_ITERATION, 541
OPCverify
 generating a layer-based image, 31, 84
OPCverify examples
 VEB retargeting, 676
Optical models, 704
optical_model_load setup command, 142
Optically visible data, 57, 287, 704
optimize_euv_hdb.out, 686
Orientation, 700
Output consistency, 650
Output sites, 571
OUTPUT_IMAGE, 549
OUTPUT_MEASUREMENT_LOCATIONS,
 550
OUTPUT_RETARGET, 553
OUTPUT_SHAPE fragment, 554
OUTPUT_TARGET_CONTROL, 558
output_text_filepath, 143

— P —

Parentheses, 28
Pattern matching, 424
Pipes, 28
png setup file command, 144
POLY and ACTIVE layers, 649
Pre-bias, 334
Preventing small jog movement, 618
Previous, 700
Printing SRAF avoidance, 661
Process window, 67, 705
Process Window OPC (PWOPC), 68, 537
Protect edge, 303
PSM, 286
pto switch, 41
PV Band PWOPC, 70, 338, 534, 535, 621
PV band tolerance, 531
pw_condition, 325
PW_RETARGET, 559
pwopc_mode, 332

— Q —

Quotation marks, 28

— R —

Rasterization, 705
Read external file, 688
read_layer_properties, 333
Resetting EPE for jogs, 617
Resist constraints, 658
Resist models, 705
resist_model_load setup command, 145
Resolution, 636
RET FLARE_CONVOLVE, 109, 684
RET OPTIMIZE_EUV_HDB, 113, 685
Retarget (definition), 705
Retarget in PWOPC, 537
Retarget with VEB, 671
retarget_layer, 334
retargeting_options, 338
Ripple fragments
 controlling, 227, 266
 interfeature fragmentation, 227

— S —

SADP, 509

-
- Scale
 fragment lengths, 62
- scale, 339
- Scripting block, 382
- SET_EPE, 560
- SET_MEDIAN_EPE, 564
- setlayer curve
 recommended settings, 629
 retargeting layers, 633
- setlayer curve_target, 152
- setlayer denseopc, 160
- setlayer veb_rettarget, 167
- Setup file
 best practices, 634
 get values, 688
- Share SVRF variables, 172, 371
- Shielding
 in fragmenting features, 227
- simulation_consistency, 171
- Sites, 381
 customizing, 565, 584
 viewing, 571
- SITES_CREATE, 565
- SITES_DELETE, 570
- SITES_DUMP, 571
- SITES_SHIFT, 573
- Slanted words, 28
- SPA, 318, 341
- spa_promotion, 340
- Space end fragments, 706
- Splines, 706
- Split
 intrafeature, 210, 268
- Square parentheses, 28
- SRAF printing, 329
- sraf_print_avoidance
 iterations for, 664
 MRC, 318
 printing threshold, 665
 process window condition, 327
 syntax, 577
- sraf_promotion_distance, 341
- sraf_sites_create, 584
- sraffragalign, 575
- step_size, 342
- SVRF commands, 113
- svrf_var_import, 172
- T —
- TAG add, 586
- TAG and, 588
- TAG fromlist, 590
- TAG ismember, 586
- TAG or, 588
- TAG remove, 586
- Tag set calculations, 485, 521
- Tag set variables, 373
- TAG size, 591
- TAG subtract, 588
- TAG tolist, 592
- Tagging, 707
- Target layers, 57, 287
- TARGET_CONTROL, 597
- TARGET_FUNCTION, 606
- Tcl scripting, 85, 373, 382
- TDBIAS, 609
- tilemicrons
 recommended settings, 634
- Tiles, 265, 707
- Toploss, 327
- topo_model, 343, 344
- Transmission Value
 related to background, 189
 related to layer, 133
- Troubleshooting, 635
- TVF examples, 688
- U —
- Underlined words, 28
- UU2DBU, 611
- V —
- Variable etch bias (VEB), 707
- VEB retargeting
 in Calibre nmOPC, 676
 in Calibre OPCverify, 675
 in CM1 Center, 673
 requirements, 672
 verification in Calibre OPCverify, 678
- veb_corner_site_placement, 345
- veb_evalpts_per_fragment, 347

verb_pseudo_nyquist, [349](#)

version, [350](#)

Visible layers, [57](#)

— W —

wafer_enclose, [78, 351](#)

wafer_enclosedby, [78, 354](#)

wafer_exclude, [357](#)

— X —

XMEEF, [544](#)

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.

