

SIEMENS EDA

Calibre® OPCpro™ 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Unpublished work.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 Sparse Calibre OPC Tools | 19 |
| Calibre OPC Workflow | 19 |
| Key Concepts for Sparse OPC..... | 20 |
| Syntax Conventions..... | 21 |
| Running Calibre OPCpro, ORC, and PRINTimage | 23 |
| Batch Mode..... | 23 |
| Inside Calibre WORKbench or Calibre LITHOview | 24 |
| Chapter 2 | |
| Working With the Calibre Hierarchical Engine | 27 |
| Prerequisite Knowledge..... | 27 |
| Calibre nmDRC Hierarchical Engine | 29 |
| Inputs Into the Calibre nmDRC Hierarchical Engine..... | 29 |
| calibre | 30 |
| Calibre Results..... | 31 |
| Calibre Data..... | 33 |
| Layer Types | 33 |
| Edge and Polygon Data..... | 33 |
| Layer of Origin | 35 |
| Derived Layers and Rule Check Statements | 37 |
| Dimension Checks..... | 41 |
| Identify Appropriate Edge Pairs | 41 |
| Construct Measurement Regions | 44 |
| Return Intersection Edges | 46 |
| Chapter 3 | |
| Introduction to SVRF Rule Files..... | 47 |
| Rule File Requirements | 47 |
| Rule File Contents | 49 |
| Specification Statements..... | 49 |
| Layer Operations..... | 52 |
| Output Statements..... | 54 |
| Chapter 4 | |
| Executing the RET Batch Tools | 57 |
| OPC Configuration | 57 |
| Edge Movement Constraints | 60 |
| Width and Spacing Rules | 60 |
| Tagging..... | 64 |
| Required Input Files..... | 65 |
| Design Database | 65 |

| | |
|---|------------|
| SVRF Rule File | 65 |
| Litho Setup Files | 67 |
| Model Files and Directories | 69 |
| Output Files | 69 |
| Chapter 5 | |
| Key Concepts | 71 |
| Fragmentation Overview | 72 |
| Fragments and Edges | 72 |
| Effect of Fragments on OPC | 73 |
| Fragment Types and Related Objects | 74 |
| Fragmentation Setup | 77 |
| Fragmentation Constraint Parameters | 78 |
| Fragmentation Strategies | 80 |
| Global Fragmentation | 81 |
| Intrafeature Fragmentation | 81 |
| Interfeature Fragmentation | 86 |
| Maxedgelength Fragmentation | 89 |
| Island-Layer Fragmentation | 90 |
| Visible-Layer Fragmentation | 92 |
| Implicit Fragmentation | 92 |
| Summary of Global Fragmentation Methods | 93 |
| Global Fragmentation Issues - Hierarchical Interactions | 94 |
| Special Case Fragmentation (fragmentTag) | 94 |
| Multilayer Fragmentation (fragmentLayer) | 98 |
| Multilayer Fragmentation Issues - Managing Default Settings | 102 |
| Viewing Fragmentation | 103 |
| Tagging | 104 |
| Fragmentation Built-In Tags | 104 |
| Using Fragtype to Refine Fragmentation Settings | 105 |
| New Tags (newTag) | 107 |
| Methods to Control Edges With Tags | 108 |
| Debugging Tagging Scripts With Calibre WORKbench | 109 |
| Control Sites | 112 |
| Control Sites and EPE | 112 |
| Control Site Keywords | 113 |
| Adjusting Sites | 114 |
| Site Styles for Corners | 115 |
| Layer Usage | 120 |
| Layer Types | 121 |
| Additional Fragmentation | 124 |
| Controlling Correction | 125 |
| Layer Names and Numbers | 126 |
| Tiles and Tile Boundaries | 126 |
| Chapter 6 | |
| Sparse OPC Command Reference | 131 |
| SVRF Commands | 136 |

Table of Contents

| | |
|-------------------------------------|-----|
| LITHO FILE | 137 |
| Litho Setup File Format | 138 |
| LITHO OPC | 146 |
| LITHO ORC | 148 |
| LITHO PRINTIMAGE | 150 |
| Litho Setup File Keywords | 152 |
| background | 155 |
| coincidentLayer | 158 |
| concaveAdjDist | 160 |
| concavecorn | 161 |
| convexAdjDist | 163 |
| cornedge | 164 |
| cornerRadius | 167 |
| cornerSiteStyle | 169 |
| flaglayer | 174 |
| fragmentLayer | 176 |
| gridsize | 179 |
| gse | 180 |
| inline_optical1 | 181 |
| inline_optical2 | 182 |
| inline_optical3 | 183 |
| inline_resist | 184 |
| interfeatLayers | 186 |
| interfeature | 187 |
| intrafeature | 197 |
| islandFragment | 201 |
| iterations | 206 |
| layer | 207 |
| lineEndAdjDist | 214 |
| lineEndLength | 215 |
| maxedgelength | 216 |
| minedgelength | 221 |
| minfeature | 222 |
| minjog | 223 |
| modelpath | 224 |
| movelimit | 225 |
| MRC_RULE | 226 |
| opcIter | 232 |
| optical_model | 241 |
| opticalmodel | 243 |
| outlayer | 244 |
| processModel | 246 |
| progversion | 248 |
| resistpolyfile | 250 |
| secondOpticalmodel | 251 |
| seriftype | 252 |
| set_kernel_count | 253 |
| set_resist_model | 255 |
| siteinfo | 257 |

| | |
|--|-----|
| sse | 259 |
| stepsize | 260 |
| svrf_var | 261 |
| thirdOpticalmodel | 263 |
| tilemicrons | 265 |
| visibleLayers | 266 |
| Litho Variables | 267 |
| ALLOW_SMALL_MOVE | 273 |
| DISALLOW_SHALLOW_ANGLE | 274 |
| DONT_MOVE_ALL_ANGLE_EDGES | 276 |
| FRAG_BREAK_IN_HALF | 278 |
| GRIDS_FOR_ANGLE45 | 280 |
| INTENSITY_INTERP_CENTER | 281 |
| INTERACTION_DISTANCE | 282 |
| LITHO_ALLOW_MAP_BY_ORDER | 284 |
| LITHO_ASYM_SOURCE_CELL_CLONE | 286 |
| LITHO_AUTO_PUSH | 288 |
| LITHO_CLEAN_OPCT | 289 |
| LITHO_CONTEXT_RELAX_MRC | 290 |
| LITHO_DEANGLE | 292 |
| LITHO_HISTOGRAM_OUTPUT_FILE | 294 |
| LITHO_IMPL_FRAG_LEN | 295 |
| LITHO_MASK_PRIORITY_MOVEMENT | 296 |
| LITHO_MOVE_CONTEXT | 297 |
| LITHO_MOVEMENT_PRIORITY_TAG | 298 |
| LITHO_MRC_CLEANUP_LIMIT | 299 |
| LITHO_MRC_MODE | 300 |
| LITHO_PRECISE_PROMOTION | 301 |
| LITHO_PRINT_LEVEL | 302 |
| LITHO_PRIORITY_BASED_MOVEMENT | 304 |
| LITHO_PROMOTION_TILING | 306 |
| LITHO_REPORT_REMOTE_CPU_TIME | 308 |
| LITHO_SIDE_SEGMENT_FILTER | 309 |
| LITHO_SIMULATE_CONTEXT | 311 |
| LITHO_SIMULATE_CONTEXT_DISTANCE | 314 |
| LITHO_SIMULATE_CONTEXT_SITE_SHIFT | 315 |
| LITHO_SUPPRESS_INTERACTION_ERROR | 317 |
| LITHO_SUPPRESS_MAPNUMBER_ERROR | 318 |
| LITHO_TAG_TIMER | 319 |
| LITHO_TILE_SLIVER_SIZE | 320 |
| LITHO_TOLERATE_MINOR_SKEW | 322 |
| LITHO_WRITE_DEBUG_SVRF (shell variable only) | 323 |
| MASK_CHIP_CORNER | 327 |
| OPC_AUTO_STEPBY_THRESHOLD | 329 |
| OPC_CORR_SEARCH_DISTANCE | 330 |
| OPC_CROSS_MEEF_DEADBAND | 331 |
| OPC_DO_SKew_CHECK | 332 |
| OPC_ENCLOSURE_MIN_LENGTH | 333 |
| OPC_ENFORCE_FRAG_LAYER_ORDER | 334 |

Table of Contents

| | |
|---|-----|
| OPC_EPE_TOLERANCE_FRAC | 337 |
| OPC_FAST_MOVE | 338 |
| OPC_FEEDBACK | 339 |
| OPC_IGNORE_ENCLOSURE_AT_CORNER | 340 |
| OPC_LOW_MEEF_DEFAULT | 341 |
| OPC_LOW_MEEF_FREEZE_TAG | 342 |
| OPC_LOW_MEEF_LIMIT | 343 |
| OPC_LOW_MEEF_NEG_SCALE | 344 |
| OPC_LOW_MEEF_NEG_TAG | 345 |
| OPC_LOW_MEEF_POS_SCALE | 347 |
| OPC_LOW_MEEF_POS_TAG | 348 |
| OPC_MAX_ITER_MOVEMENT | 349 |
| OPC_MIN_ENCLOSURE_01 | 350 |
| OPC_MIN_ENCLOSURE_10 | 352 |
| OPC_MIN_SITE_ANGLE | 354 |
| OPC_NEAREST_ADJ | 355 |
| OPC_NEW_FRAG_ORDER | 356 |
| OPC_USE_LE_DEF | 357 |
| SEM_CONSIDER_SITE_OFFSET | 359 |
| SEM_DO_MORPH | 360 |
| SEM_MAX_NONPRINT_MOVE | 361 |
| SEM_MORPH_SIZE | 362 |
| SEM_PWL_JOT_SIZE | 363 |
| SEM_USE_PWL | 364 |
| SEM_USE_SUF_FRAG | 365 |
| SEM_USE_SUF_SITES | 366 |
| VERBOSITY | 367 |
| Tagging Commands | 368 |
| bindModelToTag | 373 |
| clearTagScript | 374 |
| dump | 375 |
| epeToleranceTag | 379 |
| fastIter | 382 |
| fragalign | 384 |
| fragcoinc | 388 |
| fragmentTag | 389 |
| fragmentTag ... interfeature | 395 |
| fragmentTag ... intersection | 397 |
| fragmentTag ... maxedgelength | 403 |
| fragmentTag ... projection | 404 |
| fragmentTag ... split | 406 |
| histogram | 407 |
| jogsmooth | 409 |
| MATRIX_OPC | 412 |
| newTag | 426 |
| newTag ... -how ANDTAGS | 429 |
| newTag ... -how COINCIDENTEDGE | 430 |
| newTag ... -how COINCIDENTINSIDEEDGE | 432 |
| newTag ... -how COINCIDENTOUTSIDEEDGE | 434 |

| | |
|--|-----|
| newTag ... -how DISPLACEMENT | 436 |
| newTag ... -how EDGE | 437 |
| newTag ... -how EPE | 446 |
| newTag ... -how EPESENSITIVITY | 449 |
| newTag ... -how EXTERNAL | 451 |
| newTag ... -how fragAfterEdge | 454 |
| newTag ... -how fragAfterFrag | 455 |
| newTag ... -how fragBeforeEdge | 456 |
| newTag ... -how fragBeforeFrag | 457 |
| newTag ... -how FRAGMENT | 458 |
| newTag ... -how fragNextToEdge | 462 |
| newTag ... -how fragNextToFrag | 463 |
| newTag ... -how HAS_SITE | 464 |
| newTag ... -how IMAGE | 465 |
| newTag ... -how INSIDEEDGE | 472 |
| newTag ... -how INTERNAL | 474 |
| newTag ... -how LAYERAND | 477 |
| newTag ... -how LAYERANDNOT | 479 |
| newTag ... -how loopWithFrag | 480 |
| newTag ... -how MRC | 482 |
| newTag ... -how NOTCOINCIDENTEDGE | 484 |
| newTag ... -how NOTCOINCIDENTINSIDEEDGE | 486 |
| newTag ... -how NOTCOINCIDENTOUTSIDEEDGE | 488 |
| newTag ... -how NOTINSIDEEDGE | 490 |
| newTag ... -how NOTOUTSIDEEDGE | 492 |
| newTag ... -how NOTTOUCHEDGE | 494 |
| newTag ... -how NOTTOUCHINSIDEEDGE | 496 |
| newTag ... -how NOTTOUCHOUTSIDEEDGE | 498 |
| newTag ... -how ORTAGS | 500 |
| newTag ... -how OUTSIDEEDGE | 501 |
| newTag ... -how SEQUENCE | 503 |
| newTag ... -how SUBTRACTTAGS | 506 |
| newTag ... -how TOUCHEDGE | 507 |
| newTag ... -how TOUCHINSIDEEDGE | 509 |
| newTag ... -how TOUCHOUTSIDEEDGE | 511 |
| opcTag | 513 |
| setMoveLimitForTag | 517 |
| siteControl DELETE | 518 |
| siteControl OFFSET_FROM | 519 |
| siteControl OFFSET_PROP | 523 |
| siteControl OFFSET_ROTATED | 525 |
| siteControl SHIFT_PROP | 527 |
| siteControl SHIFT_TOWARD | 529 |
| siteControl SMOOTH | 531 |
| sitemodify | 536 |
| tags2boxes | 539 |
| Pre-Defined Tags | 541 |

Table of Contents

| | |
|---|------------|
| Chapter 7 | |
| Model-Based OPC Examples..... | 551 |
| OPC Examples..... | 552 |
| Scenario 1: Basic OPC | 552 |
| Scenario 2: OPC With Sub-Resolution Assist Features (SRAFs) | 556 |
| Scenario 3: OPC With Printing Assist Features (PAFs)..... | 560 |
| Scenario 4: Etch Retargeting With Calibre TDopc and Calibre OPCpro..... | 565 |
| Scenario 5: OPCpro Concurrent Operations..... | 570 |
| Calibre OPCpro Example | 572 |
| Using Matrix OPC in Calibre OPCpro | 575 |
| Matrix OPC for Multiple Masks and Complex EPE | 575 |
| Methods of Matrix OPC Fragment Mapping | 577 |
| Matrix OPC Mapping Controls..... | 579 |
| Viewing Full-Chip Mappings With Visualize Matrix OPC | 580 |
| Matrix OPC Scenarios | 581 |
| Example 1: Multiple Tolerances..... | 581 |
| Example 2: Matrix OPC Followed by Calibre ORC | 582 |
| Example 3: Using Enclosure Checks With Matrix OPC | 585 |
| Example 4: Matrix OPC for Alternating Phase Shift Masks | 588 |
| Chapter 8 | |
| Calibre ORC Procedures and Examples | 595 |
| Options for Running ORC | 596 |
| Tagging Basics..... | 597 |
| Properties Tagging | 597 |
| Relationship Tagging | 598 |
| Topological Tagging..... | 599 |
| Dimension Check Tagging | 600 |
| EPE Tagging | 601 |
| EPE Sensitivity Tagging | 602 |
| Image Tagging | 603 |
| Displacement Tagging | 605 |
| Boolean Tagging..... | 605 |
| Has_Site Tagging | 605 |
| MR _C Tagging | 605 |
| Inspecting Tagged Data | 606 |
| Calibre Layer Output | 606 |
| Histogram Output | 611 |
| Control OPC With Tagging..... | 612 |
| Requirements for ORC in Standalone Mode | 614 |
| Requirements for ORC Within OPC Mode | 616 |
| Calibre ORC Example | 618 |
| Chapter 9 | |
| Calibre PRINTimage Procedures and Examples | 621 |
| SEM, Dense Aerial, and Dense Print Images | 622 |
| SEM Image Controls | 624 |
| Calibre PRINTimage Results | 626 |

| | |
|--|-----|
| To Run the Calibre PRINTimage Batch Tool | 627 |
| Calibre PRINTimage Example | 628 |

Chapter 10**Calibre OPCpro Best Practices **631****

| | |
|---|-----|
| Recommended Settings for Best Results | 631 |
| Recommended Settings for Best Run Time | 635 |
| Fragmentation Best Practices | 636 |
| Practices to Improve OPC Output Consistency | 638 |
| Fix Simulation Context | 638 |
| Fix Recipe Non-Convergence | 639 |
| Improve MRC Rules | 640 |
| Practices to Reduce the Impact on Quality of Jogs | 642 |
| Layer-Based Best Practices | 643 |

Glossary**Index****Third-Party Information**

List of Figures

| | |
|--|-----|
| Figure 1-1. Complete Process Flow on the Calibre Platform | 20 |
| Figure 2-1. How Polygon Data Divides Space | 34 |
| Figure 2-2. How Edge Data Divides Space | 35 |
| Figure 2-3. Reference Data Dependence on Layer of Origin | 36 |
| Figure 2-4. Effect of Reference Data on Dimension Checks..... | 37 |
| Figure 2-5. Edge Pairs Measured by the Internal Operation | 42 |
| Figure 2-6. Edge Pairs Measured by the External Operation | 42 |
| Figure 2-7. Edge Pairings Measured by the Enclosure Operation..... | 43 |
| Figure 2-8. Measuring “Appropriate” Angles for Edge Pairings | 43 |
| Figure 2-9. Measurement Regions for Edge A | 44 |
| Figure 2-10. Creating Measurement Regions | 46 |
| Figure 5-1. Edge Illustration | 72 |
| Figure 5-2. Edge Fragment Illustration | 73 |
| Figure 5-3. Effect of High Versus Low Fragmentation..... | 74 |
| Figure 5-4. Setup File Example (Fragmentation) | 77 |
| Figure 5-5. Intrafeature Fragmentation | 82 |
| Figure 5-6. cornedge and concavecorn Example..... | 83 |
| Figure 5-7. non90, Line-End Adjacent, and Space-End Adjacent Fragmentation | 84 |
| Figure 5-8. Priority Example | 85 |
| Figure 5-9. Interfeature Fragmentation Example..... | 86 |
| Figure 5-10. interfeature Example | 88 |
| Figure 5-11. Asymmetrical Interfeature Fragmentation | 89 |
| Figure 5-12. Maxedgelenlength Fragmentation | 89 |
| Figure 5-13. Island-Layer Fragmentation | 90 |
| Figure 5-14. islandFragment Example | 91 |
| Figure 5-15. Hierarchical Promotion Issues | 94 |
| Figure 5-16. Adding Additional Fragmentation | 95 |
| Figure 5-17. Removing Fragmentation | 96 |
| Figure 5-18. Unfragmenting With Re-fragmentation | 97 |
| Figure 5-19. Applying Additional Interfeature Fragmentation | 97 |
| Figure 5-20. Changing the Coincident Layer to Affect Fragmentation..... | 102 |
| Figure 5-21. Viewing Fragmentation in the Calibre WORKbench Application | 103 |
| Figure 5-22. Fragtype Built-In Tagging Commands | 105 |
| Figure 5-23. Tag Lines From Setup File | 108 |
| Figure 5-24. Control Sites on Fragmented Polygon | 112 |
| Figure 5-25. Example Control Site | 114 |
| Figure 5-26. Site Placement With the siteinfo Keyword..... | 115 |
| Figure 5-27. Corner Site Styles | 115 |
| Figure 5-28. Site Styles and EPE..... | 116 |
| Figure 5-29. SITES_ON_EDGE Specified With Distance | 117 |

| | |
|---|-----|
| Figure 5-30. SITES_ON_ARC Specified With Distance | 117 |
| Figure 5-31. Customizing Sites for Special Corners Example | 118 |
| Figure 5-32. Layer Keyword Arguments..... | 120 |
| Figure 5-33. Using External Layers for ORC | 123 |
| Figure 5-34. False Edges | 125 |
| Figure 5-35. Tile Boundaries Example | 127 |
| Figure 5-36. 4 x 4 Tiled Areas..... | 128 |
| Figure 5-37. Next Level of Tiling | 129 |
| Figure 5-38. Top Level of Tiling..... | 129 |
| Figure 6-1. Changing the Coincident Layer to Affect Fragmentation..... | 159 |
| Figure 6-2. Intrafeature Fragmentation With concavecorn | 162 |
| Figure 6-3. Adjacency Requirements | 163 |
| Figure 6-4. Intrafeature Fragmentation Example (cornedge) | 165 |
| Figure 6-5. Stubby Feature Fragmentation | 166 |
| Figure 6-6. Corner Radius | 168 |
| Figure 6-7. Corner Site Styles | 169 |
| Figure 6-8. SITES_ON_EDGE Specified With Distance | 171 |
| Figure 6-9. SITES_ON_ARC Specified With Distance | 171 |
| Figure 6-10. Interfeature Shielding Example | 188 |
| Figure 6-11. Using -distancePriority With Interfeature Fragmentation | 188 |
| Figure 6-12. Example -ripplestyle | 189 |
| Figure 6-13. Differences Between -ripplestyle 0 and -ripplestyle 1 Settings | 191 |
| Figure 6-14. -split Example | 192 |
| Figure 6-15. -shift Example | 193 |
| Figure 6-16. Using Shield With Interfeature Fragmentation..... | 194 |
| Figure 6-17. -skipCorners Option | 195 |
| Figure 6-18. Typical Fragmentation Caused by a Scattering Bar | 196 |
| Figure 6-19. Symmetric Fragmentation Caused by a Scattering Bar and -sym | 196 |
| Figure 6-20. Intrafeature Fragmentation Example..... | 199 |
| Figure 6-21. Without -dontcross Option | 202 |
| Figure 6-22. With -dontcross Option | 203 |
| Figure 6-23. Island Layer Fragmentation Example | 204 |
| Figure 6-24. Island Fragmentation Without an Offset | 205 |
| Figure 6-25. Island Fragmentation With an Offset | 205 |
| Figure 6-26. Line End Adjacent Distance | 214 |
| Figure 6-27. Comparing MAXEDGELENGTH -split 0 and 1 | 217 |
| Figure 6-28. Maxedgelength Fragmentation | 218 |
| Figure 6-29. Minimum Jog Size | 223 |
| Figure 6-30. The Four Metrics..... | 227 |
| Figure 6-31. Tile Boundaries..... | 244 |
| Figure 6-32. (a) Full Serifs (b) Hammerheads..... | 252 |
| Figure 6-33. Site Placement With the siteinfo Keyword..... | 258 |
| Figure 6-34. Effect of FRAG_BREAK_IN_HALF..... | 279 |
| Figure 6-35. INTERACTION_DISTANCE | 282 |
| Figure 6-36. Improving Accuracy by Increasing INTERACTION_DISTANCE | 283 |

List of Figures

| | |
|--|-----|
| Figure 6-37. De-angling a Skew Fragment With Stairsteps | 292 |
| Figure 6-38. CORNERDIST Using LITHO_MRC_MODE 1 | 300 |
| Figure 6-39. LITHO_PRIORITY_BASED_MOVEMENT Effects | 305 |
| Figure 6-40. Better MRC Violation Clean-Up | 305 |
| Figure 6-41. LITHO_SIDE_SEGMENT_FILTER Example | 310 |
| Figure 6-42. LITHO_SIMULATE_CONTEXT Example. | 312 |
| Figure 6-43. Long Fragment Near False Edge With Context | 315 |
| Figure 6-44. Bad OPC Results Near Long Fragment | 316 |
| Figure 6-45. Long Fragment Site and Context Shifted | 316 |
| Figure 6-46. Better OPC Results With Site Shifting | 316 |
| Figure 6-47. Identifying Regions to Use LITHO_TILE_SLIVER_SIZE | 321 |
| Figure 6-48. Effects of MASK_CHIP_CORNER | 327 |
| Figure 6-49. Effects of Using MASK_CHIP_CORNER | 328 |
| Figure 6-50. Minimum Fragment Length for Enclosure Check | 333 |
| Figure 6-51. Turning Off Enclosure Checking for Corner Fragments | 340 |
| Figure 6-52. Enclosure Check Between Mask 0 and Mask 1 | 350 |
| Figure 6-53. Enclosure Check Between Mask 1 and Mask 0 | 352 |
| Figure 6-54. OPC_USE_LE_DEF Effects. | 357 |
| Figure 6-55. Piecewise Linear vs. Rectilinear SEM | 364 |
| Figure 6-56. Tolerance Windows | 379 |
| Figure 6-57. A Layout With and Without fragalign | 386 |
| Figure 6-58. Re-Fragmenting a Line End to Create a Hammerhead | 394 |
| Figure 6-59. Without the -dontcross Option | 398 |
| Figure 6-60. With the -dontcross Option | 398 |
| Figure 6-61. FragmentTag ... Intersection Without an Offset | 401 |
| Figure 6-62. FragmentTag ... Intersection With an Offset | 402 |
| Figure 6-63. Effects of Jog Smoothing and Jog Alignment | 410 |
| Figure 6-64. How Distance is Measured Using MIDPOINT | 413 |
| Figure 6-65. How Distance is Measured Using CLOSEPOINT | 414 |
| Figure 6-66. Projecting | 415 |
| Figure 6-67. OPC Corrections Related to Parameter Space | 419 |
| Figure 6-68. Standard OPC Versus Matrix OPC | 424 |
| Figure 6-69. Coincident Edges | 430 |
| Figure 6-70. Coincident Inside Edge | 432 |
| Figure 6-71. Coincident Outside Edge | 434 |
| Figure 6-72. Edges and Logical Edges | 440 |
| Figure 6-73. A Logical Edge | 442 |
| Figure 6-74. WITHJOG1 Tagging | 443 |
| Figure 6-75. WITHJOG Tagging | 444 |
| Figure 6-76. WITHJOG and WITHJOG1 Tagging | 444 |
| Figure 6-77. Adjacent Jogs | 445 |
| Figure 6-78. Image Gradient Tagging | 469 |
| Figure 6-79. INSIDEEDGE | 472 |
| Figure 6-80. LAYERAND (Island Fragmentation Off) | 477 |
| Figure 6-81. Positive and Negative Loops. | 481 |

| | |
|---|-----|
| Figure 6-82. Not Coincident Edges | 484 |
| Figure 6-83. Not Coincident Inside Edge | 486 |
| Figure 6-84. Not Coincident Outside Edge | 488 |
| Figure 6-85. NOTINSIDEEDGE | 490 |
| Figure 6-86. NOTOUTSIDEEDGE | 492 |
| Figure 6-87. NOTTOUCHEDGE | 494 |
| Figure 6-88. NOTTOUCHINSIDEEDGE | 497 |
| Figure 6-89. NOTTOUCHOUTSIDEEDGE | 499 |
| Figure 6-90. OUTSIDEEDGE | 501 |
| Figure 6-91. Sequence Tagging Example | 504 |
| Figure 6-92. Sequence Tagging With OUTPUT_ALL | 505 |
| Figure 6-93. TOUCHEDGE | 508 |
| Figure 6-94. TOUCHINSIDEEDGE | 510 |
| Figure 6-95. TOUCHOUTSIDEEDGE | 512 |
| Figure 6-96. Positioning the Site on a Fragment | 520 |
| Figure 6-97. Positioning the Site Center | 520 |
| Figure 6-98. Orienting the Site | 521 |
| Figure 6-99. Geometry and Sites With and Without siteControl smooth | 534 |
| Figure 6-100. Convolution of Mask Shape With Gaussians for 3 Sigma Values | 535 |
| Figure 6-101. Defining Gaussian Kernel Radius | 535 |
| Figure 6-102. Convex Corner Fragments | 542 |
| Figure 6-103. tConcaveConcave | 545 |
| Figure 6-104. tConcaveConvex | 546 |
| Figure 6-105. tConcaveCorner | 546 |
| Figure 6-106. tConcaveStepAdjacent | 546 |
| Figure 6-107. tConvexConvex | 547 |
| Figure 6-108. tConvexCorner | 547 |
| Figure 6-109. tConvexStepAdjacent | 548 |
| Figure 6-110. tStraight | 549 |
| Figure 7-1. Before and After OPC | 552 |
| Figure 7-2. Fragmentation | 555 |
| Figure 7-3. Simulation | 555 |
| Figure 7-4. Correction | 556 |
| Figure 7-5. Final Results | 556 |
| Figure 7-6. Layer Data for OPC With SRAFs | 557 |
| Figure 7-7. Fragmentation for Design With SRAFs | 559 |
| Figure 7-8. OPC Results With SRAFs | 560 |
| Figure 7-9. Layer Data for OPC With PAFs | 561 |
| Figure 7-10. Fragmentation for Design With PAFs | 564 |
| Figure 7-11. OPC With PAFs | 564 |
| Figure 7-12. Simulation Based on PAFs and SRAFs | 565 |
| Figure 7-13. Etch Retargeting Flow | 565 |
| Figure 7-14. Etch Bias Output | 570 |
| Figure 7-15. Log File Analysis of Concurrency | 571 |
| Figure 7-16. Layer Decomposition for Alt PSM | 575 |

List of Figures

| | |
|---|-----|
| Figure 7-17. OPC on Multimask Layers | 576 |
| Figure 7-18. Moving One Edge Can Affect Multiple EPEs | 576 |
| Figure 7-19. One-to-One Mapping With Euclidean Metric. | 577 |
| Figure 7-20. Many-to-One Mapping With Euclidean Metric | 578 |
| Figure 7-21. One-to-Many Mapping With Euclidean Metric | 578 |
| Figure 7-22. The Basic Matrix OPC Flow. | 579 |
| Figure 7-23. Graphic Description of Projection | 579 |
| Figure 7-24. Multiple Tolerance Windows | 582 |
| Figure 7-25. Overlay of Standard OPC, Matrix OPC, Target Layer. | 583 |
| Figure 7-26. ORC Tagging Results From Example 2 | 584 |
| Figure 7-27. Histogram of Slope (Intensity per Micron). | 585 |
| Figure 7-28. Enclosure Check With Multiple Masks | 586 |
| Figure 7-29. Affected Edges With OPC_IGNORE_ENCLOSURE_AT_CORNER | 587 |
| Figure 7-30. Test Layout With Phase 0 and 180 Separations | 588 |
| Figure 7-31. Matrix OPC Tcl Script Mappings | 593 |
| Figure 7-32. OPC Results | 593 |
| Figure 8-1. Sequence Tagging (Basic) | 599 |
| Figure 8-2. Sequence Tagging With OUTPUT_ALL | 599 |
| Figure 8-3. EPE Sensitivity | 603 |
| Figure 8-4. Controlling the Size of Highlight Boxes. | 607 |
| Figure 8-5. MAP Using Name Qualifiers | 610 |
| Figure 8-6. Using RVE to Simplify Results Viewing | 611 |
| Figure 8-7. Moving Sites To Fit a Contour | 613 |
| Figure 8-8. Processing Order in Standalone Mode | 615 |
| Figure 8-9. Processing Order Using ORC With OPC | 617 |
| Figure 9-1. Corner Rounding. | 621 |
| Figure 9-2. Piecewise Linear Versus Rectilinear SEM | 625 |
| Figure 9-3. Calibre PRINTImage Results | 627 |
| Figure 9-4. PRINTImage Result and Original Layer. | 629 |
| Figure 9-5. PRINTImage Result and Original Enlarged | 629 |
| Figure 10-1. Example of Small Step Interfering in OPC Movements. | 640 |
| Figure 10-2. Using Length-Based MRC Rules | 641 |
| Figure 10-3. Pullback Due to Excess MRC Restriction | 641 |
| Figure 10-4. MRC Metrics Can Cause Inconsistency | 642 |
| Figure 10-5. Optimized Site Placement Around a Jog | 643 |
| Figure 10-6. I-Shaped OPC Output | 645 |

List of Tables

| | |
|--|-----|
| Table 1-1. Syntax Conventions | 21 |
| Table 3-1. Layout Statement Summary | 50 |
| Table 3-2. Unit Statement Summary | 50 |
| Table 3-3. Flag Statement Summary | 51 |
| Table 3-4. DRC Statement Summary | 51 |
| Table 3-5. The Layer Specification Statement | 51 |
| Table 3-6. Commonly Used Layer Selector Operations | 53 |
| Table 3-7. Commonly Used Layer Constructor Operations | 53 |
| Table 3-8. The Layer Creation Statement | 54 |
| Table 3-9. Output Statement | 55 |
| Table 4-1. Priorities for Width and Spacing Rules | 62 |
| Table 5-1. Fragment Types and Related Objects | 74 |
| Table 5-2. Fragmentation Constraint Parameters Summary | 78 |
| Table 5-3. Intrafeature Fragmentation Summary | 82 |
| Table 5-4. Interfeature Fragmentation Summary | 86 |
| Table 5-5. Maxedgelength Fragmentation Summary | 89 |
| Table 5-6. Island-Layer Fragmentation Summary | 90 |
| Table 5-7. Visible-Layer Fragmentation Summary | 92 |
| Table 5-8. Fragmentation Types Quick Summary | 93 |
| Table 5-9. Keywords Allowed in a fragmentLayer Block | 99 |
| Table 5-10. Default Values for a fragmentLayer Block | 102 |
| Table 5-11. Global Built-In Tags | 104 |
| Table 5-12. Fragtype Tagging Commands | 104 |
| Table 5-13. Keywords Used to Control Simulation Sites | 113 |
| Table 5-14. How RET Layer Types Affect OPC | 121 |
| Table 6-1. SVRF Command Summary | 136 |
| Table 6-2. Summary of Litho Setup File Keywords and Commands | 152 |
| Table 6-3. Supported Fragmentation Keywords in fragmentLayer Block | 176 |
| Table 6-4. layer Keyword frag bit Functionality | 207 |
| Table 6-5. Valid layer_types | 208 |
| Table 6-6. Layer Properties Matrix | 210 |
| Table 6-7. layer Keyword transmission Argument Settings | 211 |
| Table 6-8. How MAXEDGELENGTH Splits Long Fragments (-split=0) | 216 |
| Table 6-9. How MAXEDGELENGTH Splits Long Fragments (-split=1) | 216 |
| Table 6-10. minedgelength and maxedgelength Scale and Measurement Unit | 218 |
| Table 6-11. Results With -split = 0 | 218 |
| Table 6-12. Results With -split = 1 | 219 |
| Table 6-13. Litho Setup Variable Summary | 267 |
| Table 6-14. Auto Stepby Movements | 329 |
| Table 6-15. Allowed Fragmentation in a fragmentLayer Command | 334 |

| | |
|---|-----|
| Table 6-16. Verbosity Levels | 367 |
| Table 6-17. Tagging Commands Summary | 368 |
| Table 6-18. Setup File Command Execution — Case 1 | 392 |
| Table 6-19. Setup File Command Execution — Case 2 | 393 |
| Table 6-20. Command Execution Comparisons | 393 |
| Table 6-21. Starting Values for max_displacement | 410 |
| Table 6-22. Potential Mappings | 415 |
| Table 6-23. Distances | 415 |
| Table 6-24. Discounting Distances for Parallel Projecting Edges | 420 |
| Table 6-25. Primary and Secondary Distance Comparisons | 421 |
| Table 6-26. Edge Movements | 422 |
| Table 6-27. Constraints Without Lower Bounds | 437 |
| Table 6-28. LENGTH Constraints | 439 |
| Table 7-1. Layer Decomposition for Multimask Layers | 575 |
| Table 7-2. Full-Chip Layers Created With Visualize Matrix OPC | 580 |
| Table 8-1. Comparing ORC Standalone to OPCpro | 596 |
| Table 8-2. Built-In Tags | 598 |
| Table 8-3. Arguments to tags2boxes, by Mode | 607 |
| Table 8-4. Layer Usage With the ORC Batch Tool | 615 |
| Table 8-5. Layer Usage With the ORC Batch Tool Usage | 618 |
| Table 9-1. Comparison of Image Types | 623 |
| Table 9-2. Calibre PRINTimage Environment Variables | 626 |
| Table 10-1. Recommended Settings for Best OPC Results | 631 |
| Table 10-2. Recommended Settings for Best OPCpro Run Time | 635 |
| Table 10-3. Layer Fragmentation Recommendations | 637 |

Chapter 1

Introduction to Sparse Calibre OPC Tools

The Calibre® OPCpro™ tool performs optical and process correction (OPC) on chip layouts to compensate for lithography distortions inherent in the manufacturing process. It supports a mix of rule and simulation-based methods, and simulates corrections using optical and process models. By performing OPC, you can print smaller features more reliably and improve yield.

The corrections are performed using a shape-based or sparse simulation engine to calculate the wafer image contour. This technique is sometimes referred to as “sparse OPC”. This is in contrast to the pixel-based simulation sites used in the Calibre® nmOPC™ tool, also known as “dense OPC”.

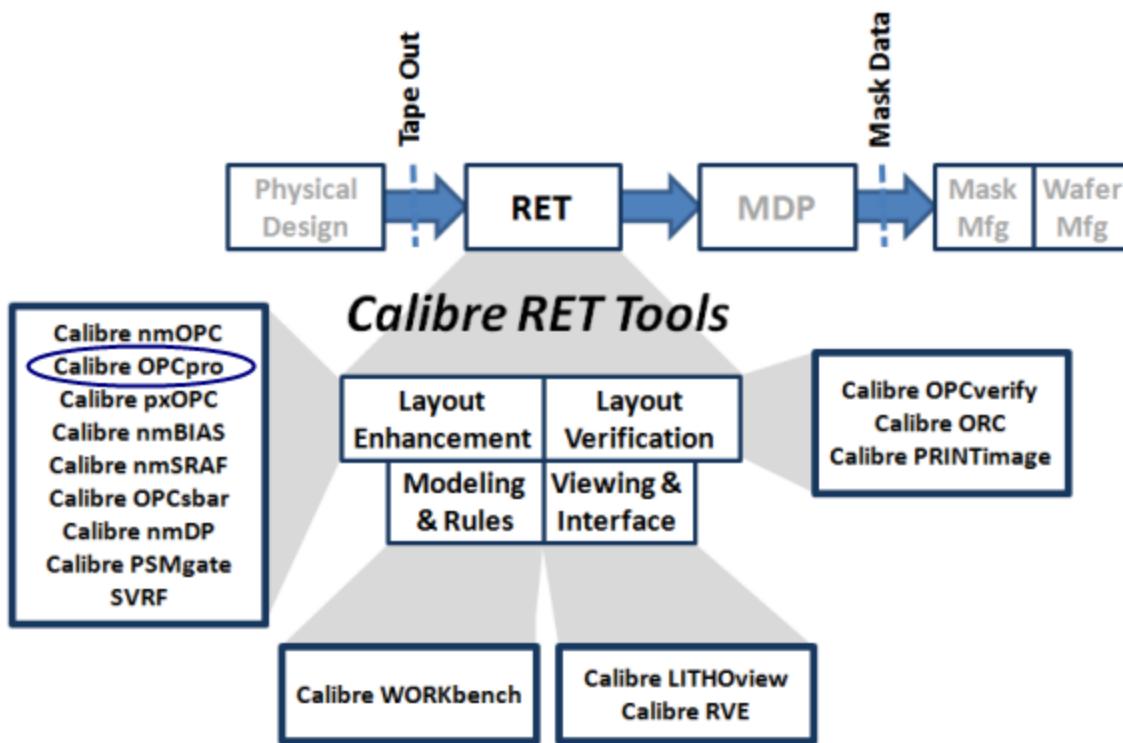
OPCpro uses the SVRF and TVF controls and command language formats common to all Calibre tools. This manual describes the key concepts and techniques for Calibre OPCpro and its companion tools Calibre® ORC™ (used for categorizing polygon edges and optical rule checking) and Calibre® PRINTimage™ (used for predicting how features print on a wafer), as well as the SVRF and TVF controls.

| | |
|--|-----------|
| Calibre OPC Workflow | 19 |
| Key Concepts for Sparse OPC | 20 |
| Syntax Conventions | 21 |
| Running Calibre OPCpro, ORC, and PRINTimage | 23 |
| Batch Mode..... | 23 |
| Inside Calibre WORKbench or Calibre LITHOview | 24 |

Calibre OPC Workflow

Both Calibre OPCpro and Calibre nmOPC are fully integrated with the complete set of Calibre RET tools. Calibre OPCpro, Calibre ORC, Calibre PRINTimage and Calibre® OPCVerify™ provide sparse OPC and OPC verification capabilities in an integrated flow. Together with Calibre® nmDRC™, Calibre® OPCsbar™, and Calibre® FRACTURE, a complete hierarchical flow can be built on the Calibre platform.

Figure 1-1. Complete Process Flow on the Calibre Platform



Calibre OPCpro and Calibre nmOPC are similar products. The appropriate one to use depends on critical dimensions and pattern density. Generally, Calibre OPCpro is preferred for processes at or above 65 nm and for half-pitch processes above 45 nm.

Key Concepts for Sparse OPC

These are the key concepts you need to understand to use the OPC and RET tools effectively:

- **Edge** — An edge is a part of a polygon, extending from corner to corner. Parts of the interface also use the term loosely to refer to a fragment of an edge.
- **EPE** — Edge placement error, defined as how far the simulated printed shape (or contour) is from the drawn, or intended, shape. The goal of OPC is to minimize EPE.
- **Fragmentation** — The process of breaking an edge into edge segments, called fragments, using OPCpro setup commands.
- **Site** — A site, also called a control or measurement site, is a line of points along which EPE is measured. Each movable edge fragment has a site cutting through it. Ideally sites are perpendicular to the expected printed contour.
- **Type of layer** — Layer types are very important in OPC. There are two categories: DRC types (original, derived, polygon, edge, and error) and OPC types. The DRC types are covered extensively in “[Layers](#)” in the *Calibre Verification User’s Manual*. The OPC types are

- **opc** — Intended shape
- **correction** — Shifting fragments
- **external** — Final placed fragments
- **visible** — Shapes that appear on the final mask but should not be corrected, such as scattering bars
- **asraf** — Shapes that appear on the final mask that are subject to MRC checks, such as anti-scattering bars
- **island** — Shapes used for fragmentation and tagging controls
- **hidden** — All layers in the layout file that are not involved in OPC

The OPC layer types are described in more detail in “[Layer Types](#)” on page 121.

Related Topics

[Fragmentation Overview](#)

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. |
| ' ' | Single quotation marks enclose metacharacters that are to be entered literally. Do not include single quotation marks when entering braces or brackets in a command. |

Table 1-1. Syntax Conventions (cont.)

| Convention | Description |
|------------|---|
| 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:

```
MRC_RULE {EXTERNAL | INTERNAL} [name1 name2] {OFF |
    {' USE d1 {metric | d2}
        [{LENGTH m1 USE d1 {metric | d2}}...]...]
    '}}
```

Running Calibre OPCpro, ORC, and PRINTimage

The Calibre RET toolset can be run either in batch mode (that is, by executing a script at the shell prompt) or from within Calibre® WORKbench™, a graphical interface.

For both modes, you must set the CALIBRE_HOME environment variable before starting. See “[Setting the CALIBRE_HOME Environment Variable](#)” in the *Calibre Administrator’s Guide* for details.

Calibre OPCpro, Calibre ORC, and Calibre PRINTimage must be run with the Calibre hierarchical engine. They do not support the Calibre® FullScale™ platform.

| | |
|--|-----------|
| Batch Mode..... | 23 |
| Inside Calibre WORKbench or Calibre LITHOview | 24 |

Batch Mode

Runs for large sections of layout often use batch mode. Running OPC on full chip layouts can take hours.

Prerequisites

- The appropriate licenses installed. For batch mode, this includes Calibre nmDRC-H as well as Calibre OPCpro. For details on license substitution and scaling, see “[Licensing: Calibre RET Products](#)” in the *Calibre Administrator’s Guide*.
- An SVRF rule file containing at least one LITHO OPC statement. See “[Introduction to SVRF Rule Files](#)” on page 47.
- A litho setup file containing process settings, fragmentation commands, and other rules. The litho setup file may be part of the SVRF rule file. Litho setup files are described in “[Litho Setup File Format](#)” on page 138.
- An input design containing at least one drawn layer.

Procedure

At the command line, run Calibre using DRC syntax. The SVRF rule file contains the call to run OPC, ORC, or PRINTimage.

Results

The Calibre run starts writing the transcript to the terminal window. The first line is typically the Calibre version. The command line prompt appears again when the run is complete.

Examples

The typical command line is similar to the following:

```
calibre -drc -hier -turbo rules.svrf
```

“[Calibre nmDRC Hierarchical Engine](#)” on page 29 explains the syntax.

Inside Calibre WORKbench or Calibre LITHOview

Use the Calibre WORKbench viewer for exploring different correction schemes, viewing contours, and creating basic litho setup files via mini-OPC.

Prerequisites

- All prerequisites described in “[Batch Mode](#)” on page 23.
- A Calibre WORKbench or Calibre® LITHOview™ license. Refer to the [Calibre Administrator’s Guide](#) for license information.

Procedure

1. Start the viewer and load the design.

For Calibre WORKbench:

```
calibrewb design.oas
```

For Calibre LITHOview:

```
calibrelv design.oas
```

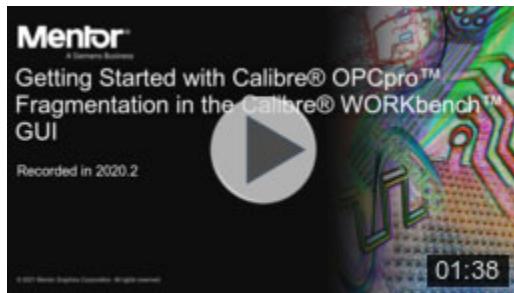
For additional invocation settings, see the [Calibre WORKbench User’s and Reference Manual](#).

2. Zoom in to some portion of the layout by holding down the right mouse button and dragging it diagonally down and to the right.
3. In the viewer menu, select **Litho > RET Flow Tool**.
The RET Flow Tool (RFT) window appears.
4. In the RFT window, select **File > Extract from SVRF**.
5. Browse to your SVRF rule file and click **Extract**.
6. In the RFT tool bar, click the **Options** button under the **OPC** button.
An area below the tool bar expands to show possible outputs.
7. Select Fragment Markers.
8. Click the **OPC** button.

This runs the OPC commands in the extracted litho setup file on only the selected region.

9. To see the control sites, in the layer palette select a layer with a name that begins “FRG_” and then click the **Frag** button.
10. To display information about the fragment, in the main viewer window click the **FInfo** button. The OPCpro Fragment Information dialog box displays the position and properties of the fragment.

The following video shows Steps 6 through 10, though it uses a different method for loading the setup file.



Related Topics

[Using the Calibre RET Flow Tool \[Calibre WORKbench: RET Flow Tool User's Manual\]](#)

Chapter 2

Working With the Calibre Hierarchical Engine

This manual describes the operations of Calibre® OPCpro™, Calibre® ORC™, and Calibre® PRINTimage™.

The following related applications are described in further detail in other documents:

- Calibre® WORKbench™ in the *Calibre WORKbench User's and Reference Manual*
- Calibre® LITHOview™ in the *Calibre WORKbench User's and Reference Manual*
- Calibre® nmDRC™ in the *Calibre Verification User's Manual*

| | |
|--|-----------|
| Prerequisite Knowledge | 27 |
| Calibre nmDRC Hierarchical Engine | 29 |
| Inputs Into the Calibre nmDRC Hierarchical Engine..... | 29 |
| calibre | 30 |
| Calibre Results | 31 |
| Calibre Data | 33 |
| Layer Types | 33 |
| Edge and Polygon Data..... | 33 |
| Layer of Origin | 35 |
| Derived Layers and Rule Check Statements | 37 |
| Dimension Checks | 41 |
| Identify Appropriate Edge Pairs | 41 |
| Construct Measurement Regions | 44 |
| Return Intersection Edges | 46 |

Prerequisite Knowledge

In our experience with dozens of semiconductor manufacturers, one of the most critical factors for successful and rapid RET deployment is the formation of a cross-functional team.

At least one person on the cross-functional team should be knowledgeable in each of the following areas:

- IC physical layers and their interaction
- Impact of RET compliant rules on design

- Limitations or constraints of the GDSII format
- Design hierarchy
- Basic CAD technology
- CAD scripting languages
- Design rule concepts
- Linux[®]¹ workstation environment
- Basic lithographic principles
- Mask format constraints, including address units and grids, and how they relate to design grids
- Impact of mask inspection constraints, mask manufacturing constraints, and mask format constraints on RET recipes

1. Linux[®] is a registered trademark of Linus Torvalds in the U.S. and other countries.

Calibre nmDRC Hierarchical Engine

The Calibre nmDRC hierarchical engine is the foundation for the Calibre RET solutions. This engine is a powerful layout design verification and correction tool you use for manipulating layer data, and performing design and manufacturability rule checks quickly and efficiently. The Calibre nmDRC hierarchical engine is also the starting point for the Calibre RET batch processes, coordinating data exchange between layout files and the tools.

This section describes the inputs and gives the detailed syntax for the calibre command.

| | |
|---|----|
| Inputs Into the Calibre nmDRC Hierarchical Engine | 29 |
| calibre | 30 |

Inputs Into the Calibre nmDRC Hierarchical Engine

You begin with the Calibre nmDRC hierarchical engine, no matter which Calibre RET solution you use. There are two types of input you must supply when you invoke the Calibre nmDRC hierarchical engine.

- One or more layout databases

These are GDS databases containing the design layer or layers on which you perform OPC.

- One or more Standard Verification Rule Format (SVRF) rule files

These are ASCII files containing legal SVRF statements. The statements control the following aspects:

- design environment
- loading of data from the GDS database(s)
- layer preprocessing (for example, sizing)
- design and manufacturability checks
- invoking batch RET processes
- rule-based OPC
- writing the final results to a results database

This section introduces the basic concepts you must understand when using Calibre functionality. For more information on SVRF rule files, refer to “[Introduction to SVRF Rule Files](#),” and to the [Standard Verification Rule Format \(SVRF\) Manual](#). For more information on the Calibre nmDRC hierarchical engine, refer to the [Calibre Verification User’s Manual](#).

calibre

Starts a Calibre batch run.

Usage

```
calibre -drc [ -hier ] [ -turbo [ n ] ]
[ -turbo_litho [ N ] ] [ -turbo_all ] rules
```

Arguments

- **-drc**
A required switch that specifies a DRC run.
- **-hier**
An optional switch that specifies a hierarchical DRC run (nmDRC-H).
- **-turbo [n]**
An optional argument that instructs Calibre nmDRC-H to use multithreaded parallel processing. The value of *n* specifies the number of CPUs to use. If you do not specify *n*, Calibre nmDRC-H uses the maximum number of CPUs available for which you have licenses. In general, you should avoid specifying *n*.

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

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

You can specify the -turbo and the -turbo_litho options concurrently in a single command line and the respective arguments can vary between the two options.
- **-turbo_all**
An optional argument used in conjunction with -turbo or -turbo_litho. When the number of licenses specified in -turbo and -turbo_litho are not available, the Calibre nmDRC-H hierarchical engine normally runs with fewer threads than requested. This command line option causes the Calibre hierarchical engine to exit if it cannot get the number of licenses specified.

- **rules**

A required user-supplied argument specifying the name of the SVRF rule file that contains the LITHO operation that executes the batch tool.

Calibre Results

The Calibre nmDRC hierarchical engine saves results and status information from each run in the formats summarized in this section.

Results Database

This database stores the results of the Calibre run. The Calibre nmDRC hierarchical engine writes results to the database file you specify using the DRC RESULTS DATABASE statement in your SVRF rule file. If this file is a GDS database, which is typical of OPC runs, then the Calibre nmDRC hierarchical engine writes the data to layer 0.

Using the DRC CHECK MAP statement, you can assign different types of data to different layers. You can also write the results to different database files. For more information, refer to “[DRC Statements](#)” on page 51 and “[Output Statements](#)” on page 54.

Calibre Transcript

The Calibre nmDRC hierarchical engine automatically generates a text transcript at run time and sends the transcript to stdout, which is typically the shell window from which you invoked the application. During the Calibre run, the transcript displays event logs, warning messages, and summary information. This transcript documents tasks the application performs: compiling the rule file, loading in the design data, and operating on the data.

When you invoke any of the model-based batch tools, the transcript also records the full setup file you have referenced, and the optical and resist models the application used for simulation. You should save the transcript for each run by redirecting the output to a file, which you can do when you invoke the Calibre nmDRC hierarchical engine, by using the following pipe-and-tee syntax:

```
calibre -drc -hier -turbo opc.drc | tee opc.log
```

DRC Results Summary

When you specify a pathname using the DRC Summary Report SVRF statement, the Calibre nmDRC hierarchical engine generates a results summary text file and saves it to this file. The file contains heading information, describing the particulars of the run, run-time warnings list, statistics for the layers before and after processing, and a run-time summary.

Related Topics

[DRC Check Map \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Results Database \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DRC Summary Report \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Calibre Data

Raw data for Calibre nmDRC hierarchical engine consists of polygons located on layers the IC designer has created. This section discusses the data the Calibre nmDRC hierarchical engine recognizes and creates, and the impact of layer types on the operations the hierarchical engine performs.

| | |
|-----------------------------|----|
| Layer Types | 33 |
| Edge and Polygon Data | 33 |
| Layer of Origin..... | 35 |

Layer Types

The Calibre nmDRC hierarchical engine distinguishes between four types of layers.

- **Original layers** (or drawn layers) are layers representing original layout data. In the GDS file, you reference original layers by number. In a rule file, you assign a layer name to each original layer you use and then reference it by its name.
- **Derived polygon layers** are layers containing polygons the Calibre nmDRC hierarchical engine generates from layer operations (for example, Boolean functions, area functions, and polygon-directed dimension check operations).
- **Derived edge layers** are layers containing edges or sub-edges of polygons the Calibre nmDRC hierarchical engine generates from layer operations (for example, edge operations and edge-directed dimension check operations).
- **Derived error layers** are layers representing clusters of one, two, three, or four edge segments. Each cluster is the result of an error-directed dimensional check operation. You use these layers strictly for reporting errors between the edges within each cluster. For more information on error layers, refer to the *Calibre Verification User's Manual*.

Edge and Polygon Data

Each Calibre SVRF operation operates on polygon data or edge data or both. When classifying and manipulating data with SVRF, you must pay close attention to the type of data each operation returns: edge layers or polygon layers.

Polygon Data

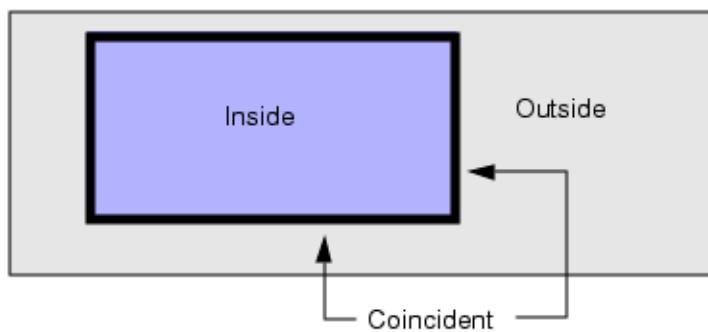
Polygon data consists of whole polygons. In most cases, the Calibre nmDRC hierarchical engine automatically merges the polygon data during any polygon operation. Specifically, if any polygons on a single layer overlap or share an edge, then the Calibre nmDRC hierarchical engine merges these polygons into a single polygon. Normally, this merged data is a more accurate depiction of the true mask than unmerged data.

Each polygon divides space into following three categories:

- inside
- outside
- coincident

[Figure 2-1](#) illustrates this division. The black rectangle represents the polygon itself and the portion of the space coincident with it. The blue area shows the portion of space inside, and the gray area shows the space outside.

Figure 2-1. How Polygon Data Divides Space



Edge Data

Edge data consists of individual polygon edges, or portions of edges. These edges contain both *geometric data* and *polygon reference data*.

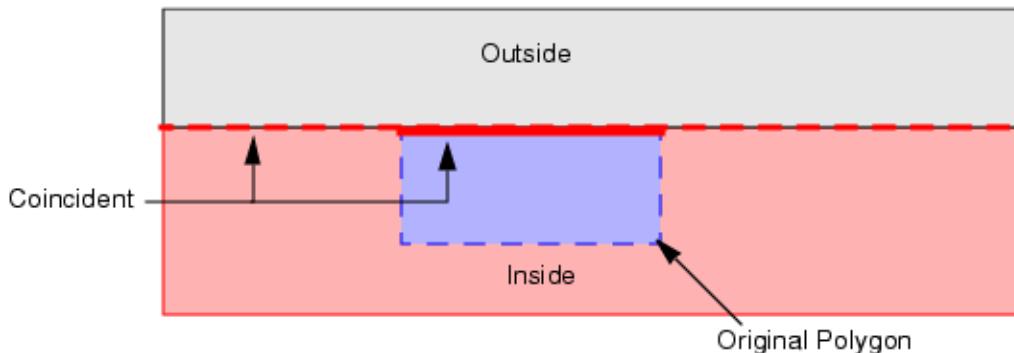
- Geometric data refers to the x and y coordinates of the edge.
- Polygon reference data refers to a record of the polygon an edge belongs, and its orientation, specifically, which direction is inside and which is outside.

Edge data divides space into the following three categories:

- **Inside** — All points on the side of the line inside of the original polygon are on the inside of the edge.
- **Outside** — All points on the side of the line outside of the original polygon are on the outside of the edge.
- **Coincident** — All points collinear with the line are coincident with the edge.

[Figure 2-2](#) illustrates this division.

Figure 2-2. How Edge Data Divides Space



In [Figure 2-2](#), the rectangle drawn with a thin dotted line is included to show how the orientation of the edge relates to the original polygon. In this case, the inside of the polygon is a subset of the inside of the edge.

[Figure 2-2](#) also represents the edge with solid red line, and a dashed red line drawn through the edge and extending beyond it in both directions. The red lines represent the portion of space coincident with the edge. The light red area shows the portion of space inside, and the gray area shows the space outside.

Layer of Origin

Polygon reference data associates each edge with a specific polygon; consequently, you must pay close attention to the layer of origin—specifically, where the data comes from.

In [Example 2-1](#), it might appear the following layer definitions produce the same data in the layer X:

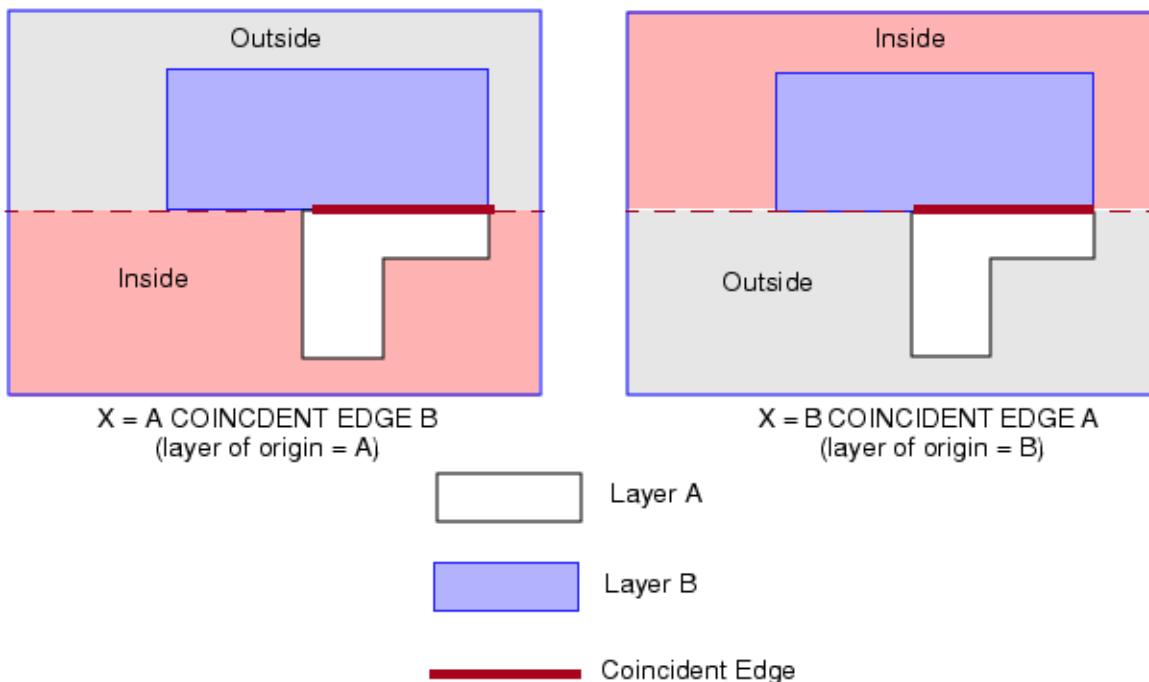
Example 2-1. Layer Definition Example

```
X = A COINCIDENT EDGE B          // Edges on layer A coincident
                           // with edges on layer B.
X = B COINCIDENT EDGE A          // Edges on layer B coincident
                           // with edges on layer A.
```

The polygon reference data the application stores with the results, however, is different depending on the layer of origin. While both operations seemingly generate the same geometric data, one contains edges the application selects off layer A, and the other contains edges the application selects off layer B.

[Figure 2-3](#) depicts this difference in selection results: the inside and outside are different depending on the layer of origin.

Figure 2-3. Reference Data Dependence on Layer of Origin



Dimension Checks, which measure distances in a specific direction relative to the edge, provide a good example of just how important the layer of origin is. [Example 2-2](#) shows the Internal operation in the sequence A has an entirely different effect than the Internal operation in the sequence B, which follows it.

Example 2-2. Dimension Checks and Layer of Origin

Sequence A:

```
X = metal coincident edge poly // Edges on layer metal that are
                                // coincident with edges on layer poly.
internal X metal < 3          // Find internally facing edges closer
                                // than 3 microns.
```

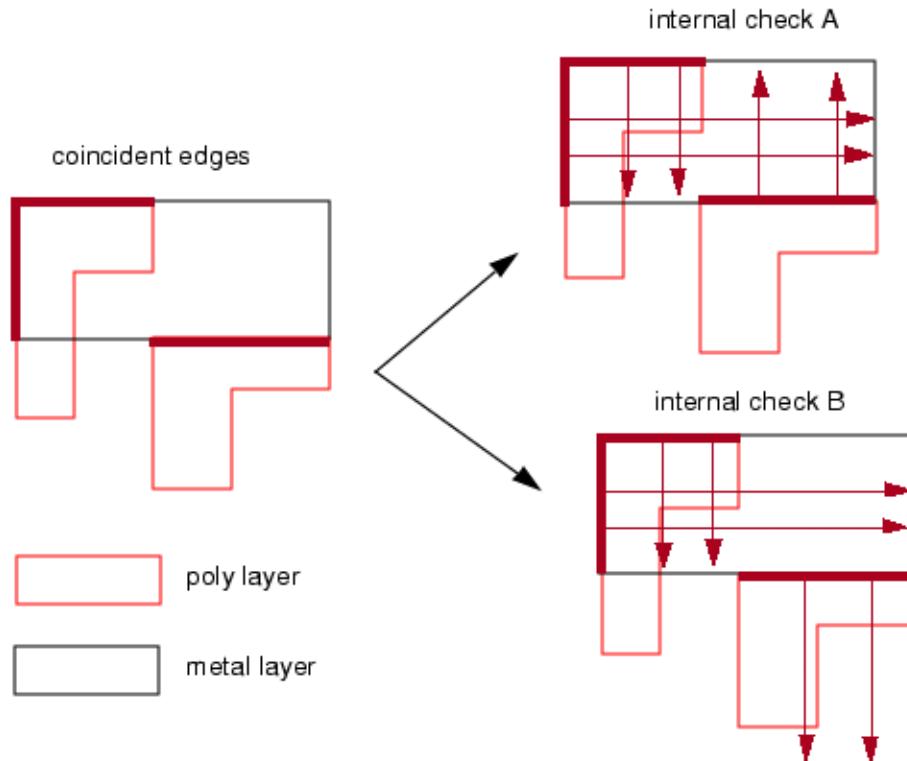
Sequence B:

```
X = poly coincident edge metal // Edges on layer poly that are
                                // coincident with edges on layer metal.
internal X metal < 3          // Find internally facing edges closer
                                // than 3 microns.
```

In sequence A, layer X contains geometric data carrying polygon reference data from layer metal; in sequence B, this data carries polygon reference data from layer poly.

Figure 2-4 shows how this results in the internal check, which “looks” inside from edge X, looking in opposite directions for the lower coincident edge.

Figure 2-4. Effect of Reference Data on Dimension Checks



Derived Layers and Rule Check Statements

The result of any SVRF operation is a collection of data. The Calibre nmDRC hierarchical engine can store collections of data in three main types.

- **Derived error layers** — These are named layers containing error data. Because error data differs from standard geometric data, you must have no references to error layers in any subsequent operation.
- **Derived edge or derived polygon layers** — These are named layers containing geometric data you can reference in subsequent operations. These layers contain *intermediate data* for input into other layer operations. The data is composed of either edges or polygons. The Calibre nmDRC hierarchical engine omits derived layers from the results database unless you explicitly instruct the tool otherwise. The term derived layer refers exclusively to a *derived edge* or a *derived polygon* layer.

Note

 The LITHO operation exclusively accepts derived polygon layers and original design layers for input.

Derived Layers

You create a derived layer using a layer definition statement in an SVRF rule file using the following syntax:

```
layer_name = layer_operation
```

where *layer_name* assigns a name to the derived layer and *layer_operation* generates the contents of the layer. See “[Layer Operations](#)” on page 52 for more information.

Derived layers can be either global or local.

- *Global layers* are any derived layers you create outside a rule check statement. You use global derived layers for storing data you may need for reference multiple times within a single SVRF rule file. The names of global derived layers must be unique.
- *Local layers* are any derived layers you create inside [Rule Check Statements](#). You use local derived layers for intermediate steps when isolating or creating the data the rule check statement writes to the database. You must reference local derived layers from within that rule check statement. The names of local derived layers must be unique within the rule check statement in which you create them.

Rule Check Statements

The rule check statements are named output statements specifying the writing of layer operations results into the results database by the Calibre nmDRC hierarchical engine. You create rule check statements using the following syntax:

```
name {layer_operation | layer_definition_statement
...
layer_operation | layer_definition_statement}
```

Note

 For rule check statements, you must include the braces ({}).

You define the data resulting from a rule check statement using a series of layer operations and layer definition statements within the braces portion of a rule check statement. When you use these, the layer definition statements create local derived layers. The local derived layers are intermediate data. Subsequently, the layer operations generate output and write this output directly to the results database.

The following rule check statement shows how you might combine layer operations and layer definition statements.

```
metal_end_cap {  
  
    x = CONVEX EDGE m1 == 2  
    // Derive layer x from original layer.  
  
    y = LENGTH x == 3  
    // Derive layer y from layer x.  
  
    z = EXPAND EDGE y OUTSIDE BY 1  
    // Derive layer z from layer y.  
  
    m1 OR z  
    // Layer operation -- merge layers  
    // and store results in the database.  
}
```

When a rule check statement contains multiple layer operations, the Calibre nmDRC hierarchical engine merges the data resulting from the layer operations into the final results. For example, the following two rule checks statements output the same results:

```
metal_end_cap1 {  
  
    INTERNAL metal < 3  
    // Layer operation stores short edges in the database.  
  
    long_met = metal LENGTH > 5  
    // Creates derived layer of long edges.  
  
    INTERNAL metal long_met < 4  
    // Layer operation -- adds edges close to long edges to the database.  
}  
  
metal_end_cap2 {  
  
    M1 = INTERNAL [metal] < 3  
    // Creates derived layer of short edges.  
  
    long_met = metal LENGTH > 5  
    // Creates derived layer of long edges.  
  
    M2 = INTERNAL metal [long_met] < 4  
    // Creates derived layer of edges close to long edges.  
  
    M1 OR M2  
    // Layer operation -- stores edges that are either  
    // short, or close to long edges, in the results database.
```

Rule check names must be unique within a single rule file. If a rule file includes other rule files, then rule check names in the files you include must also be unique.

Controlling Data Destinations

When the Calibre nmDRC hierarchical engine writes rule check output directly to a results database in GDS format, it merges the data onto layer 0. In most cases, you could find storing

each type of data on a different layer more useful. Using the DRC Check Map statement, you can map the results of a rule check statement to an additional layer, and if necessary, specify a data type. You can specify this additional layer in the same database or in a different database.

The following is the basic syntax for using the DRC Check Map statement in this manner:

```
DRC CHECK MAP RuleCheck_name layer [ datatype ] [ filename ]
```

The DRC CHECK MAP statement provides additional functionality. For more information, refer to “[DRC Check Map](#)” in the *Standard Verification Rule Format (SVRF) Manual* or to the “[Rule Check Statements](#)” section in the *Calibre Verification User’s Manual*.

Storing Derived Layers With Rule Check Statements

Calibre derived layers are intermediate data internal to the Calibre nmDRC hierarchical engine. You can save global derived layer data for use with other applications by writing the data to the results database using a rule check statement and the Copy operation.

```
rule_check_name {COPY derived_layer_name}
```

Note

 Because derived layers are data while rule check statements are output statements, you can create rule check statements with the same name as a derived layer.

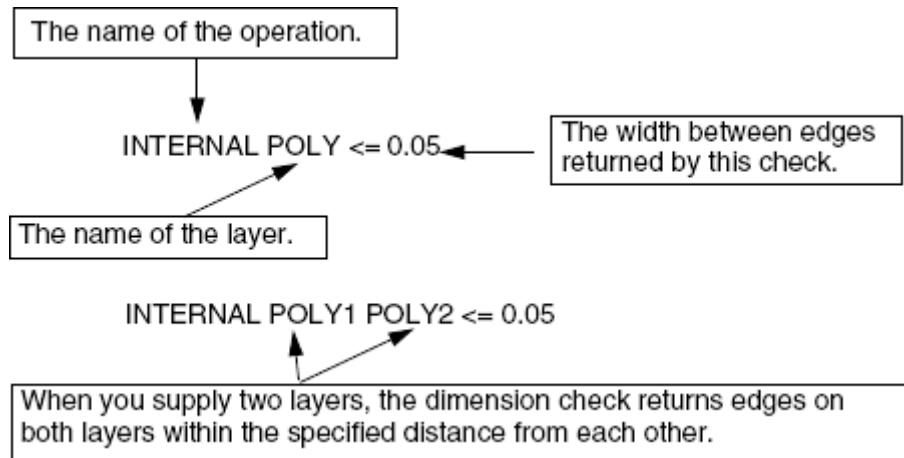
Dimension Checks

The SVRF dimension check operations measure distances between opposing edges. Because dimension checks provide you with a means for classifying edges using key properties (for example, width and spacing), they are the foundation for RET-related processing.

The Calibre nmDRC hierarchical engine provides the following three dimension checks:

- **Internal** defines a minimum separation between the interior sides of two edges.
- **External** defines a minimum separation between the exterior sides of two edges.
- **Enclosure** defines a minimum separation between the interior side of an edge on one layer and the exterior side of an edge on another layer. This results in returning edges on one layer enclosed by polygons on the other layer.

The following example shows two simple dimension checks. Using additional arguments, you can fine-tune the dimension check. For specific information on dimension check arguments, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).



The Calibre nmDRC hierarchical engine performs dimension checks using the following three-step process:

| Step | Description |
|------|---|
| 1 | Identify Appropriate Edge Pairs |
| 2 | Construct Measurement Regions |
| 3 | Return Intersection Edges |

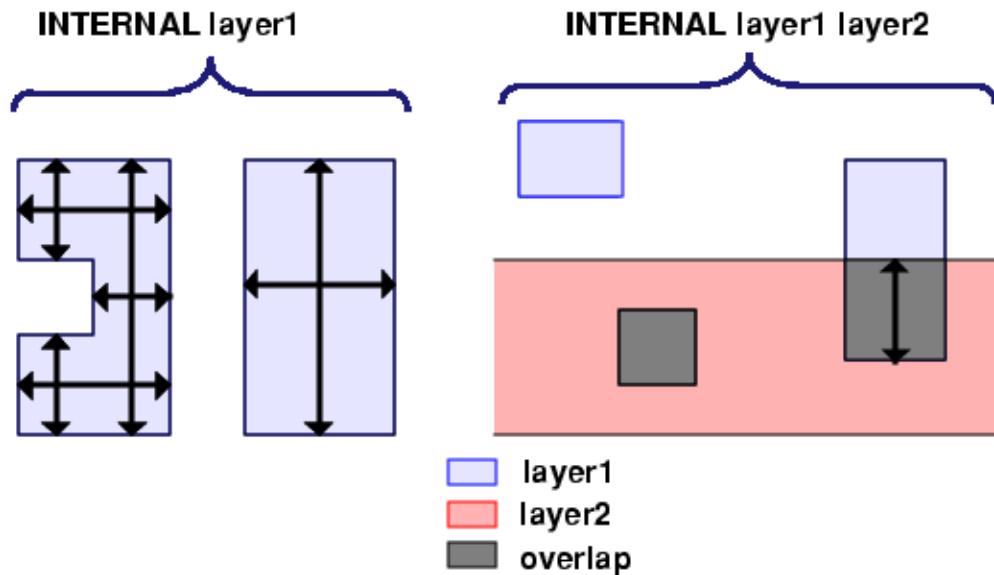
Identify Appropriate Edge Pairs

The Calibre nmDRC hierarchical engine begins the measurement process by evaluating the edges on the layer (or pair of layers) for “appropriate” edge pairs, specifically, any pair of edges

meeting the operation's criteria. The first criterion relates to the orientation of the edges. Each dimension check measures from a specific side of the first edge to a specific side of the second edge:

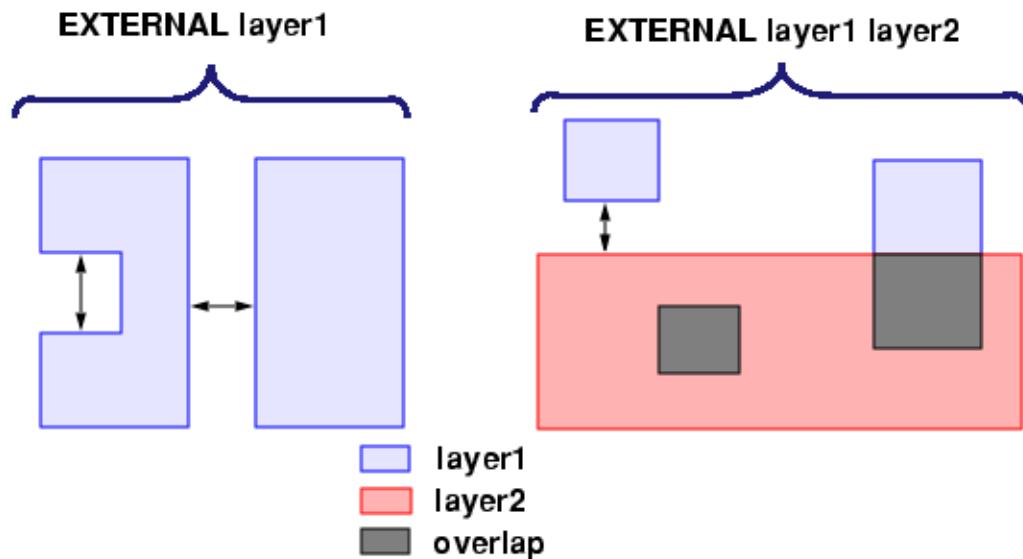
- For the Internal dimension check, the application measures from the inside of one edge to the inside of another edge.

Figure 2-5. Edge Pairs Measured by the Internal Operation



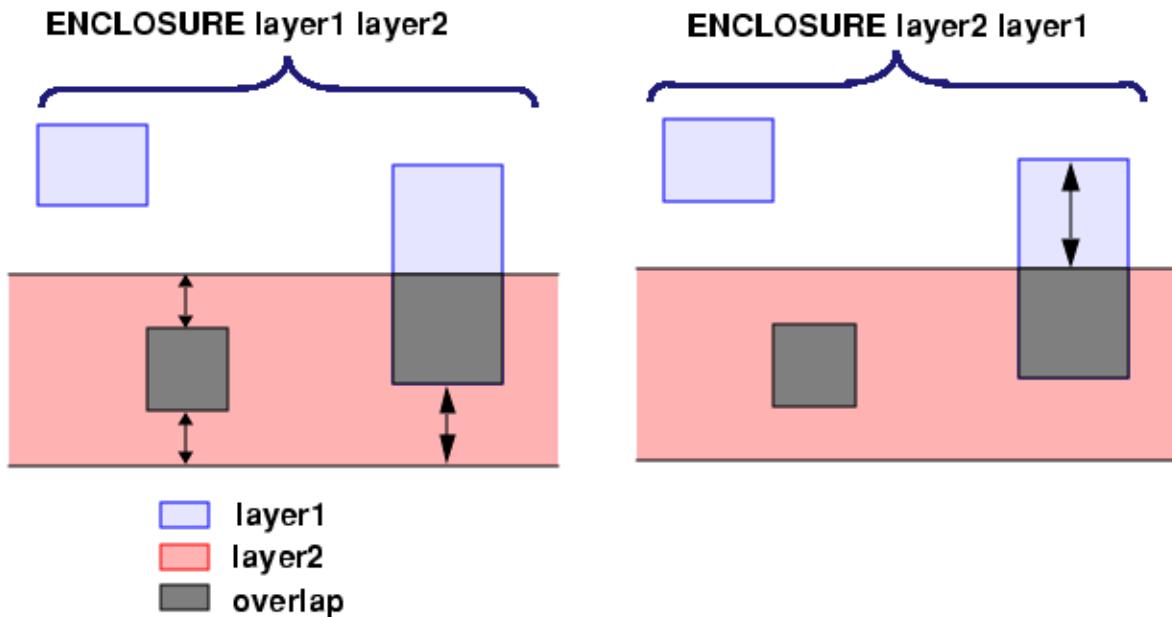
- For the External dimension check, the application measures from the outside of one edge to the outside of another edge.

Figure 2-6. Edge Pairs Measured by the External Operation



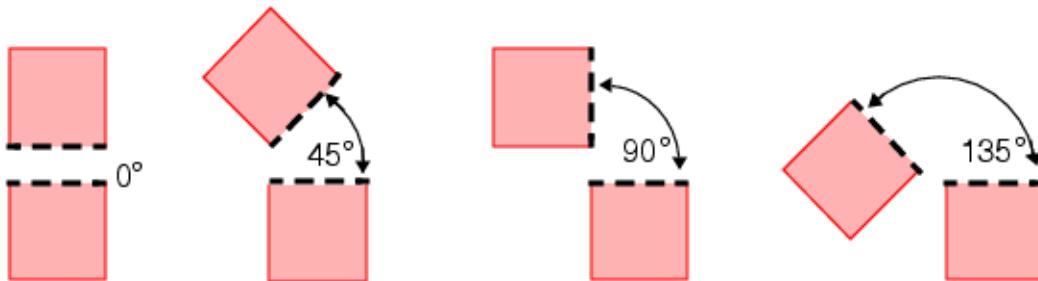
- For the Enclosure dimension check, the application measures between the outside side of an edge on the first input layer to the inside of an edge B on the second input layer.

Figure 2-7. Edge Pairings Measured by the Enclosure Operation



The second criterion for “appropriate” edge pairs relates to the angle between the specified sides of the edges. [Figure 2-5](#), [Figure 2-6](#), and [Figure 2-7](#) show edge pairings measuring the distance between parallel edges (angle = 0). By default, the application considers a pair of edges “appropriate” if the angle between the two edges ≥ 0 and < 90 . For overriding this default behavior, you can use additional arguments with dimension checks; however, the angle must be less than 180 degrees.

Figure 2-8. Measuring “Appropriate” Angles for Edge Pairings



For the EXTERNAL dimension check, the appropriate angle is between the outsides of the dashed edges.

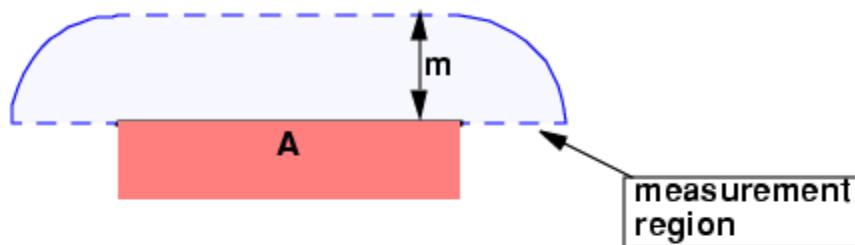
Construct Measurement Regions

Once the Calibre nmDRC hierarchical engine has identified pairs of edges meeting the criteria, it creates measurement regions for each of the edges in a pair. The application measures the separation between edges using these measurement regions. Each measurement region extends in the direction of the dimension check, either outside or inside. It contains any point in this direction so the distance from the edge satisfies the constraint for the operation.

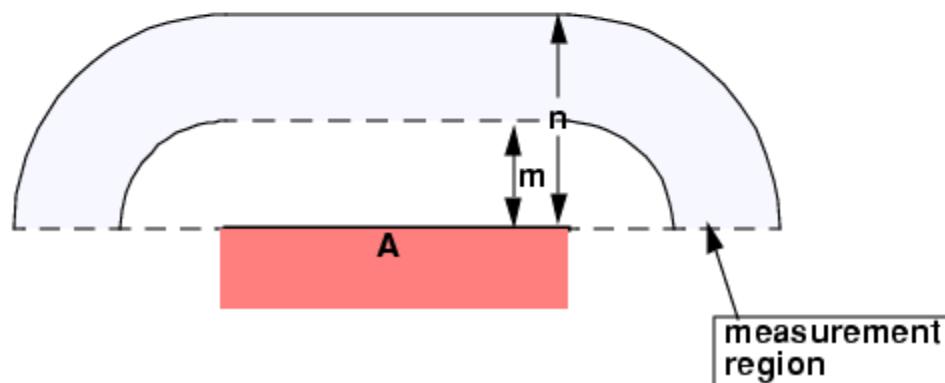
Figure 2-9 shows two regions outside of edge A. The first region assumes the constraint $x < m$, and the second region assumes the constraint $m < x \leq n$. By definition, the measurement region omits the points on the line containing edge A.

Figure 2-9. Measurement Regions for Edge A

rule1 { external layer1 < m }



rule2 { external layer1 > m < n }



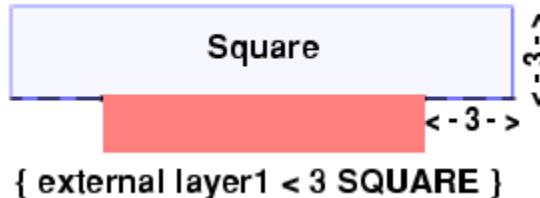
Measurement Region Metrics

Figure 2-9 depicts the measurement regions with a rounded shape. You control the shape of the measurement regions by specifying the metric, or measurement style. The Calibre nmDRC hierarchical engine supports the following five metrics:

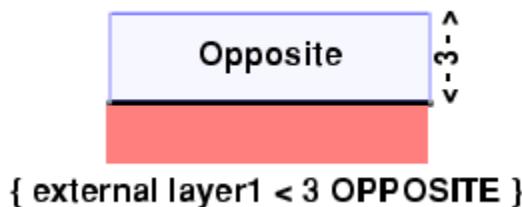
- **Euclidean.** The Euclidean metric forms a region with rounded boundaries extending past the corners of the selected edges. This is the default metric.



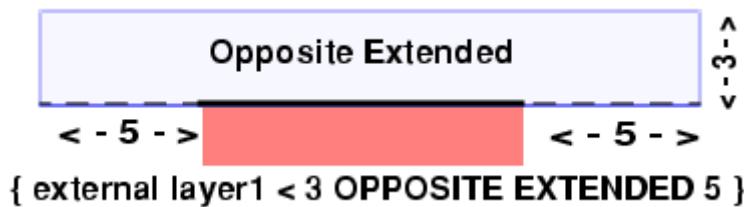
- **Square.** The square metric forms a region with right-angle boundaries extending past the corners of the selected edges.



- **Opposite.** The opposite metric forms a region with right-angle boundaries extending up to the corners of the selected edges.



- **Opposite Extended.** The opposite extended metric forms a region with right-angle boundaries extending past the corners of the selected edges by a value you specify.



- **Opposite Symmetric.** The opposite symmetric metric uses the opposite metric with postprocessing for use with non-parallel edges. For more information on this metric, refer to the *Calibre Verification User's Manual*.

Return Intersection Edges

The final step in performing the dimension check returns those portions of the edges intersecting the measurement region for the opposing edge. Because the measurement region for edge A omits the line containing edge A, dimension checks omit any points of intersection, if they exist.

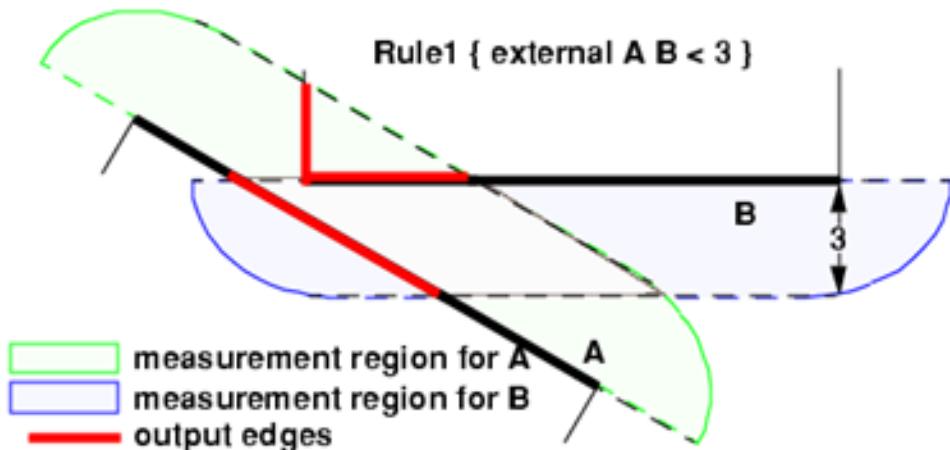
Example: Returning Intersecting Edges

Assume the measurement is from the outside of edge A to the outside of edge B, and the operation returns any edges closer than 3 microns. Using the default metric, which is Euclidean (refer to “[Measurement Region Metrics](#)”), the operation begins processing by constructing two regions. One region consists of any points in the half-plane on the outside of edge A within 3 microns of edge A; a similar region consists of any such points around edge B.

The output is an edge pair consisting of the following:

- the portion of edge A intersecting the measurement region for edge B
- the portion of edge B intersecting the measurement region for edge A

Figure 2-10. Creating Measurement Regions



Chapter 3

Introduction to SVRF Rule Files

A Standard Verification Rule Format (SVRF) rule file is an ASCII text file containing legal SVRF statements and operations. It contains the statements and operations the Calibre application uses for reading in GDS data, processing the layer data, and applying an RET solution.

You can create a new SVRF rule file using any of the following methods:

- Create a new rule file from scratch using a text editor.
- Modify an existing rule file using the Calibre® Interactive™ interface and save it.
For more information, refer to the [Calibre Interactive User's Manual](#).
- Copy an existing rule file and modify it.

SVRF rule file elements are either *operations* or *statements*. Operations control data manipulation, and statements prepare the environment in which the operations work.

Note

 This chapter describes a basic SVRF rule file. For more detailed information on specific statements and operations, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).

| | |
|-------------------------------------|-----------|
| Rule File Requirements | 47 |
| Rule File Contents | 49 |
| Specification Statements | 49 |
| Layer Operations | 52 |
| Output Statements | 54 |

Rule File Requirements

The syntax requirements for SVRF rule files are summarized in the sections following.

Order

Before processing, the Calibre applications compile the SVRF rule file. Except for variable declarations and conditional statements you nest in the rule file, you can write statements and operations in any order. For clarity, the examples in this manual organize the operations according to function and the expected order of processing. This enhances readability and maintainability of the SVRF rule file.

Within individual SVRF operations, keyword (operation or statement parameters) order is important if changing the order results in ambiguity. For example, the following four operations are equivalent:

```
INTERNAL METAL < 3  
INTERNAL < 3 METAL  
METAL INTERNAL < 3  
< 3 METAL INTERNAL
```

SVRF Case Sensitivity

SVRF keywords and names *are not* case-sensitive.

Pathnames are case-sensitive. You should enclose pathnames in single or double quotation marks.

Reserved Words

SVRF keywords are reserved words. Do not use them as names of variables or layers. For a complete list of reserved words, refer to the “[Keyword - Name Conflicts](#)” section of the *Standard Verification Rule Format (SVRF) Manual*.

Abbreviations

When you can abbreviate an SVRF keyword, its entry in the *Standard Verification Rule Format (SVRF) Manual* shows the keyword with the portion you can abbreviate in uppercase letters.

Comments

SVRF supports the following three types of comments:

- **In-line comments** begin with slashes (//), can begin anywhere on the input line, and end at the end of the line. The Calibre nmDRC hierarchical engine ignores in-line comments.
- **Block comments** begin with /* and end with */. They may begin anywhere on a line and span multiple lines. The Calibre nmDRC hierarchical engine ignores block comments.
- **DRC rule check comments** are *inside* a rule check statement, begin with @, can begin anywhere on the input line, and end at the end of the line. You can include DRC rule check comments in rule check output, so you can view them in the Calibre® RVE™ results viewer.

Rule File Contents

This section summarizes the parts of an SVRF rule file. The contents of a basic SVRF rule file contain statements that specify all the settings for Calibre, operate on the layers, and write out the results.

| | |
|---------------------------------------|-----------|
| Specification Statements | 49 |
| Layer Operations..... | 52 |
| Output Statements..... | 54 |

Specification Statements

Specification statements in a rule file identify the layout data for subsequent processing, provide basic information concerning how the tool will process the operations, and specify the output (results) location.

The following sample code shows typical specification statements from an SVRF rule file:

```
LAYOUT SYSTEM GDSII
LAYOUT PATH "gdsout/demo_ab.gds"
LAYOUT PRIMARY "demo"

PRECISION 1000
RESOLUTION 1

FLAG SKEW YES
FLAG ACUTE YES
FLAG OFFGRID YES

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "gdsout/ab_sense.gds" GDSII PSEUDO
DRC SUMMARY REPORT "reports/demo_ab.rep"
DRC MAXIMUM VERTEX 199
```

Layout Statements

Layout statements define the database or multiple databases that the Calibre nmDRC hierarchical engine operates on. If you specify multiple database paths, the Calibre nmDRC hierarchical engine merges these databases before processing. The SVRF operations in the rule file operate on the identical data.

[Table 3-1](#) lists the mandatory layout statements, described in further detail in the *Standard Verification Rule Format (SVRF) Manual*.

Table 3-1. Layout Statement Summary

| Statement | Description |
|----------------|---|
| Layout Path | Specifies the pathname for the layout design database. When you supply multiple Layout Path statements, then the first database must contain the top cell you specify using the Layout Primary statement. You must specify this in most situations. |
| Layout Primary | Identifies the top level or cell on which to operate. You must include this statement. |
| Layout System | Identifies the database format of the layout design database. You must include this statement. |

In addition to these required statements, you can obtain optimal performance from the Calibre Hierarchical Engine by supplying an optional statement ([Layout Base Layer](#) or [Layout Top Layer](#)) and identify the layout base layers. For more information, refer to “[Optimal Performance Using LAYOUT BASE LAYER](#)” in the *Calibre Post-Tapeout Flow User’s Manual*.

Unit Statements

Unit statements provide database unit precision and resolution information. [Table 3-2](#) lists the more common unit statements:

Table 3-2. Unit Statement Summary

| Statement | Description |
|-------------|--|
| Unit Length | Specifies the user unit for length. When you supply a number, it specifies the user unit of length in terms of meters. When you supply a factor, it specifies the user unit in terms of another unit (for example, mil, mm, cm, or inch). The default is 1e-6, or 1 micron. |
| Precision | Defines the ratio of database units to user units. The default is 1000. Given the default unit length of 1 micron and the default precision of 1000, the default database unit is a nanometer. |
| Resolution | Defines the layout grid step size. When you supply a single value, then the grid step size is the same in both the x and y direction. The default value is 1. |

Flag Statements

Flag statements control how the Calibre nmDRC hierarchical engine responds with respect to certain types of errors. [Table 3-3](#) lists the more common flag statements.

Table 3-3. Flag Statement Summary

| Statement | Description |
|--------------|---|
| Flag Skew | The application issues a warning when it encounters a skew edge. |
| Flag Acute | The application issues a warning when it encounters an acute angle. |
| Flag Offgrid | The application issues a warning when it encounters an off-grid vertex. |

DRC Statements

DRC statements specify how the Calibre nmDRC hierarchical engine treats the results of the tool run. [Table 3-4](#) lists the more common DRC statements. For more information on results, refer to “[Calibre Results](#)” on page 31.

Table 3-4. DRC Statement Summary

| Statement | Description |
|----------------------|---|
| DRC Maximum Results | Specifies the number of results to report. The default value is 1000. For OPC, you must set this statement to ALL, and thereby ensure the application saves any corrected geometry. |
| DRC Results Database | Specifies the full pathname to the primary results database, and its format. The optional keyword PSEUDO stipulates the results database should include the additional levels of hierarchy the application creates. |
| DRC Summary Report | Specifies the full pathname for the summary report file. Additional optional arguments let you control how this file the Calibre nmDRC hierarchical engine saves the results. |

Layer Specification Statements

Layer specification statements make data in the design database available to the Calibre nmDRC hierarchical engine. During any run, the application exclusively uses the layers you specify with these statements.

Table 3-5. The Layer Specification Statement

| Statement | Description |
|-----------|---|
| Layer | Declares a layer in the design database by number and assigns it a name by which the application references it. The layer name is a alphanumeric string you define. It cannot be an SVRF reserved word, unless it is in quotation marks. Calibre names must be unique within the SVRF rule file. You can assign the original layer any number of names you want. Nevertheless, you can only assign the Calibre name to one original layer. |

Table 3-5. The Layer Specification Statement (cont.)

| Statement | Description |
|-----------|---|
| Layer Map | Creates a new “original” layer by mapping data of the specified type(s) on the specified GDS layer(s) to a new GDS layer. |

Layer Operations

Layer operations and layer creation statements isolate specific layer data, modify this data, and generate new data.

The following sequence of layer statements and operations illustrate using layer operations for finding and modifying data:

```
// Layer Specifications
LAYER ACTIVE 2
LAYER POLY 4

//Layer Operations in Layer Creation Statements
LE = CONVEX EDGE POLY WITH LENGTH == 0.18
    ANGLE1 == 90 LENGTH1 > 0.18 ANGLE2 == 90 LENGTH2 > 0.18

LE_ISLAND = EXPAND EDGE LE OUTSIDE BY 0.01

INSIDE_CORNi = EXT [POLY] <= 0.13 ABUT ==90 INTERSECTING ONLY

INSIDE_CORN = (LENGTH POLY >= .39) COINCIDENT EDGE INSIDE_CORNi

INSIDE_CORN_ISL = EXPAND EDGE INSIDE_CORN INSIDE BY 0.01
```

Using layer operations, you can manipulate layer data. Some layer operations operate on a single layer, some on a pair of layers. When deciding which operation you should use, think in terms of what layer the operation operates on, and what the operation does with that data.

Each operation operates on either polygons or edges. The resulting data consists of one of these types of data. Because of this, you must pass the correct type of data to the operation.

Note

 To the Calibre nmDRC hierarchical engine, edge data is more than just isolated line segments. Each edge maintains a record of the polygon from which it was derived, and which side is interior and which is exterior.

Each operation can do one of two things: layer selection or layer construction. [Layer Selectors](#) select existing data from the input layer or layers you specify. [Layer Constructors](#) create new polygon or edge data by modifying existing data.

Layer Selectors

Layer selectors are operations that *select existing data* meeting the criteria you specify. These operations can select data (some select polygons, others select edges) from one layer or from any of the layers you specify. Most operations are layer selectors.

Table 3-6. Commonly Used Layer Selector Operations

| Operation | Description |
|------------------------|--|
| Copy | Copies (selects) the data on the specified layer. |
| Inside | Selects polygons on layer1 lying inside polygons on layer2. |
| Outside | Selects polygons on layer1 lying outside polygons on layer2. |
| Length | Selects edges on the specified layer satisfying the length constraint. |
| Internal ¹ | Defines a required separation between the interior sides of two edges. |
| External ¹ | Defines a required separation between the exterior sides of two edges. |
| Enclosure ¹ | Defines a required separation between the interior side of an edge on one layer and the exterior side of an edge on another layer. |
| Convex Edge | Selects edges based on the specified constraints, which can include the value of the angle at one or both ends of the edge, the length of the edge, or the length of abutting edges. |
| Coincident Edge | Selects layer1 edges or edge segments coincident with layer2 edges. |

1. Internal, External, and Enclosure are dimension checks you can use for layer selection or layer creation.

For a complete list of layer operations, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).

Layer Constructors

Layer constructors *create new data* by modifying existing data. Some construct polygon data from existing polygons. Others create polygon data from existing edges. For a complete list of operations, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).

Table 3-7. Commonly Used Layer Constructor Operations

| Operation | Description |
|-----------|---|
| And | Creates new polygon data from polygon regions common to one or more polygons on layer1, and one or more polygons on layer2. |
| Or | Creates new polygon data by combining polygons on layer1 and polygons on layer2, merging overlapping polygons into larger polygons. |

Table 3-7. Commonly Used Layer Constructor Operations (cont.)

| Operation | Description |
|-------------|--|
| Expand Edge | Expands edges into rectangles. The expansion parameters define how the application expands the edge. |
| Size | Expands or shrinks polygons according to the sizing parameters. The sizing parameters define the factor by which the application sizes the polygons. They can also define bounding areas and specify how the application treats the resulting polygons (for example, merge polygons, output overlaps only, or truncate). |
| Grow | Allows outside expansion of objects on the input layer in the direction of the x-axis, y-axis, or both. |
| Shrink | Specifies inside contraction of objects on the input layer in the direction of the x-axis, y-axis, or both. |
| Deangle | Replaces skew edges with orthogonal or 45-degree edges. |

Layer Creation Statements

Layer creation statements use *derived layers* for storing the results from a layer operation. You can reference these derived layers in other layer operations. For a complete discussion, refer to “[Derived Layers and Rule Check Statements](#)” on page 37.

Table 3-8. The Layer Creation Statement

| Statement | Description |
|--|---|
| Layer definition statement of the form <i>layer_name</i> = <i>layer_operation</i> | Creates a derived layer with the specified name. The contents of this layer are the results of the operation. |

Layer definition statements are described in “[Layer Definitions](#)” in the *Calibre Verification User’s Manual*.

Output Statements

You write data to the results database using the rule check statement. By default, the Calibre nmDRC hierarchical engine writes data resulting from a rule check statement to layer 0 of the primary results database; you define this database using the DRC Results Database statement. The DRC Check Map statement lets you direct rule check results to a specific layer or database file.

Table 3-9. Output Statement

| Statement | Description |
|--|---|
| Rule check statement of the form <i>name {layer_operation layer_definition}</i> ... [<i>layer_operation layer_definition</i>] {} | The application outputs the results of the specified layer operations to the results database. |
| DRC Check Map | The application outputs the results of the specified rule check statement on the specified layer in the results database. You can also output the results of the rule check statement to a database other than the primary results database using this statement. When using this functionality, you must specify the pathname and format of the database. |

Related Topics

[Rule Check Statements \[Calibre Verification User's Manual\]](#)

Chapter 4

Executing the RET Batch Tools

The Calibre RET toolset includes the following model-based batch tools:

- **Calibre OPCpro batch tool** performs hierarchical or flat automated OPC operations during the design rule checking (DRC) stage.
- **Calibre ORC batch tool** performs full-chip optical and process rule checking (ORC), identifying the polygon edges in a layout that do not print within a user-specified tolerance.
- **Calibre PRINTimage batch tool** performs optical and process simulations to calculate the shapes of a full-chip layer when it prints on the wafer

This chapter explains how to set up these batch tools. It also describes the files you must specify when running the tools and the output the tools generate.

| | |
|---------------------------------------|-----------|
| OPC Configuration | 57 |
| Edge Movement Constraints..... | 60 |
| Width and Spacing Rules | 60 |
| Tagging | 64 |
| Required Input Files | 65 |
| Design Database | 65 |
| SVRF Rule File | 65 |
| Litho Setup Files | 67 |
| Model Files and Directories | 69 |
| Output Files | 69 |

OPC Configuration

You control Calibre OPCpro, Calibre ORC, and Calibre PRINTimage operation by properly configuring the data and the application prior to running. The three tools have some difference in how the common options should be configured.

The options are read from a litho setup file. The litho setup file can be either inside the SVRF file or a separate file.

Note

 Do not modify the litho setup file or the models it includes during a run.

Configuration involves some or all of the following tasks:

- **Selecting the proper optical and resist models.**

The optical and resist models you select must match the equipment and processes used by the fab. If you are in doubt as to which models to use, please consult either your project manager or the fab itself.

- **Controlling how the batch tool calculates EPEs.**

EPE is the difference between the drawn edge and the simulated printed edge. You control how the EPE is calculated by defining the placement, type, and length of the controls sites that make the measurement. See “[Control Sites](#)” on page 112 for more details.

- **Defining how large an EPE must be before it requires correction.**

Some variance from the drawn polygon is inevitable. You define how much is allowable on a global basis with [OPC_EPE_TOLERANCE_FRAC](#) or for specific groups of fragments with [epeToleranceTag](#). The default tolerance, which depends on [stepsize](#), is equivalent to a tolerance of 3 nm; any fragment with a larger EPE may be moved.

- **Defining the smallest allowable edge movement.**

There are many options that control how the tools move edges. Some are generic and apply globally, and others restrict movement of sets of fragments. See “[Edge Movement Constraints](#)” for the most common options, including specifying width and spacing rules.

- **Passing all necessary layer data to the tool.**

The minimum layout data you need to run OPC is the layer containing the image to print on the wafer. You can improve results by providing additional layers that provide simulation data, assist with fragmentation, and control correction. See “[Layer Usage](#)” on page 120 for more details.

- **Fragmenting the polygon’s edges into the optimal arrangement.**

The Calibre OPCpro batch tool corrects mask geometries by moving individual edge fragments. Defining how polygon edges are broken into fragments is one of the most critical aspects of data preparation. You must consider the type of chip, turn around time, fab and mask making facility requirements, and design requirements. The primary trade-off is precision versus processing time. Refer to “[Fragmentation Strategies](#)” on page 80 for a full description.

- **Tagging edge fragments to control OPC according to edge type and need for correction.**

Tagging allows you to select fragments by various properties and operate on them as a set. The section “[Tagging](#)” provides more details.

- **Controlling how the batch tool cleans up after making corrections.**

You control how the application cleans up hierarchical gap and sliver features (outward and inward notches) during post-OPC processing using the [LITHO_CLEAN_OPCT](#) setup variable. You specify the filter used to identify gaps and slivers using one of the following methods:

- Specify the height in terms of *nsteps*. The application fills in notches with height $f \leq N$ stepsize and length of strictly less than the minimum edge length.
- Specify the height and length constraints in microns. The application fills in notches up to *h* in microns which are shorter than *m* microns in length.

Related Topics

[Litho Setup Files](#)

Edge Movement Constraints

With the Calibre OPCpro batch tool, you define both generic and specific constraints on how edges are moved to correct EPEs. *Generic* constraints restrict movement of all edges on the layer being corrected. *Specific* constraints restrict movement of individual edges or sets of edges.

You define generic constraints through keywords and variables in the setup file. You define specific constraints through tagging commands that both identify edges and also describe how the edges are to be moved. This section describes the generic constraints, some of which are defined by requirements of your mask shop or fab. For information on tags and tagging commands, which you use to define specific constraints, refer to “[Tagging](#)” on page 64.

Some constraints control how the tool moves edges. They include:

- [movelimit](#) — The maximum outward or inward distance any edge can move. By default, this variable is set to the greater of 0.08 and 0.5*[minfeature](#).
- [OPC_MAX_ITER_MOVEMENT](#) — The maximum inward or outward distance an edge can move during a single iteration. By default, this is set to $(\lambda/NA)/8$. Set this setup variable when working with low-contrast sub-130 nm processes to improve OPC stability and control convergence.
- [setMoveLimitForTag](#) — Inward and outward limits defined on a tag-by-tag basis. These limits override the limits set with movelimit.
- [stepsize](#) — The size of the grid to which corrections are snapped.
- [OPC_FEEDBACK](#) — The feedback factor used to calculate the edge movement, which is EPE * [OPC_FEEDBACK](#).

Edge movement is also constrained by the minimum width and spacing rules, which are described in the following sections.

| | |
|--------------------------------------|-----------|
| Width and Spacing Rules | 60 |
| Tagging | 64 |

Width and Spacing Rules

A typical set of constraints are external or internal spacing and width limits. Width and spacing rules define the minimum spacing and minimum width allowed for output shapes. Fragments are not moved if doing so would cause them or their two neighbors to violate these rules. If a violation exists before performing OPC, a fragment is not moved unless movement decreases the violation.

It is important to note that checking occurs between nearby output shapes, which may be on the same polygon or may be on different polygons.

There are two different methods you can use to define width and spacing rules:

- Method 1: OPC_MIN_INTERNAL, OPC_MIN_EXTERNAL, and opcMinCheckTag
- Method 2: MRC_RULE

Method 1: OPC_MIN_INTERNAL, OPC_MIN_EXTERNAL, and opcMinCheckTag

Note

 As of version 2015.4, this method is deprecated. Use MRC_RULE (method 2) instead.

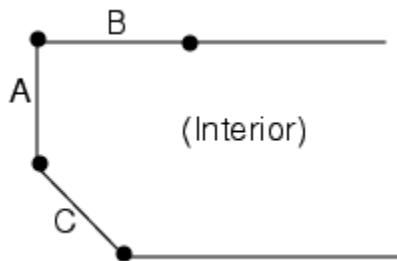
How the Checks Work. The Calibre OPCpro batch tool checks for spacing or width violations by creating keep-out measurement regions. During OPC, edges are not moved if moving them would bring them inside the measurement region for a nearby, non-perpendicular edge or would bring its two neighbors inside the measurement region for any nearby edge.

The keep-out regions are defined by the minimum spacing or width distance and how the distance is measured. The OPC_MIN_INTERNAL and OPC_MIN_EXTERNAL variables and the opcMinCheckTag command define these metrics.

The following examples illustrate how the edge and its neighbors are checked.

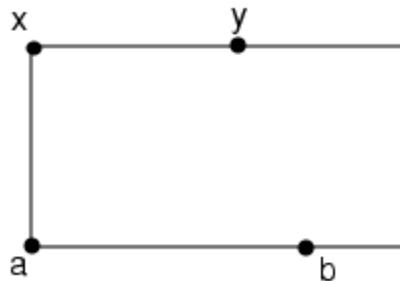
- **Example 1** — When moving a fragment or edge would result in its two neighbors violating OPC_MIN_INTERNAL or OPC_MIN_EXTERNAL, the fragment is not allowed to move.

In the illustration that follows, where edge A moving outwards would result in B and C having an OPC_MIN_INTERNAL violation, A is no longer allowed to move.



- **Example 2** — When moving an edge with neighbor fragments within CORNERDIST for either OPC_MIN_EXTERNAL or OPC_MIN_INTERNAL would cause the edge to become shorter than the space distance for OPC_MIN_EXTERNAL or OPC_MIN_INTERNAL, the edge is not allowed to move.

For example, in the following illustration, if xy is less than CORNERDIST, then ax can only move as long as xy remains greater than or equal to the OPC_MIN_EXTERNAL space or OPC_MIN_INTERNAL space.



The OPC_MIN_INTERNAL and OPC_MIN_EXTERNAL variables set up global constraints. The opcMinCheckTag command sets up constraints specific to groups of fragments.

Prioritizing Checks. When you define different width or spacing rules for different types of fragments, you create a situation in which multiple rules can apply to the same fragment. If a fragment falls under multiple rules, the rule with the highest priority is used. Most of the rules have predefined priorities; however, you define the priority for rules you create using opcMinCheckTag.

Table 4-1. Priorities for Width and Spacing Rules

| Rule | Priority |
|------------------|-----------------------|
| OPC_MIN_EXTERNAL | -1 |
| OPC_MIN_INTERNAL | -1 |
| opcMinCheckTag | 0 - 99 (user defined) |

Using CORNERDIST. OPC_MIN_EXTERNAL and OPC_MIN_INTERNAL also define whether or not the application enforces the minimum width and spacing rules with respect to nearby edges on the same polygon. The CORNERDIST keyword for these variables defines how far a fragment must be from a corner in order to be considered by the check, and is expressed in microns. The default is infinity, which means nearby edges can move during OPC, even if they create notches or jogs that violate the width or spacing rules.

The CORNERDIST option applies only to fragments that meet these conditions:

- They are on the same polygon as the fragment generating the measurement region.
- Before OPC, they are not collinear with the fragment generating the check region.
- They are inside the check region either before or after OPC.
- The distance to the edge containing the fragment generating the check region is greater than CORNERDIST.

There are further exceptions to the CORNERDIST option. When LITHO_MRC_MODE is 1 (the default value), a CORNERDIST violation is excluded if:

- The edges causing the error are separated by less than the CORNERDIST value.
- The fragments that they are derived from are part of the same polygon loop.
- The fragments that they are derived from are not facing each other.

Method 2: MRC_RULE

The second method for defining space and width rules is to use the MRC_RULE command. [MRC_RULE](#) allows you to set up external and internal constraints for each tag. The MRC_RULE keyword provides the following advantages over OPC_MIN_EXTERNAL or OPC_MIN_INTERNAL:

- You can filter edges based on length.
- You can filter edges based on projection.
- You can specify limits 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_RULE.
- Tags on the same mask can have the check between them disabled by setting MRC_RULE ... OFF.

The [LITHO_PROMOTION_TILING](#) environment variable must be explicitly set to 1 in order to use the MRC_RULE syntax, as shown in the following example:

```
#----- Arbitrary Commands Can Follow This Line -----  
sse LITHO_PROMOTION_TILING 1  
  
MRC_RULE EXTERNAL poly poly {  
    USE 0.160 EUCLIDEAN  
    LENGTH 0.01 USE 0.030 OPPOSITE  
    LENGTH 0.1 USE 0.050 EUCLIDEAN  
    PROJECTING  
    USE 0.160 EUCLIDEAN  
    LENGTH 0.02 USE 0.050 EUCLIDEAN  
}
```

See “[MRC_RULE](#)” on page 226 for details on syntax and for examples of common usage such as notches and feature-to-feature checks.

Tagging

When performing OPC with the Calibre OPCpro batch tool, you use tagging commands to control correction of individual fragments or sets of fragments.

Some ways you might use tagging commands would be:

- Identify fragments that are or are not subject to OPC.
- Identify fragments requiring rule-based OPC.
- Limit or otherwise control the movement for specific fragments.
- Highlight key fragments for inspection after processing, using the tags2boxes tagging command.
- Define different width and spacing rules for different types of fragments.

With tagging commands you can both classify fragments and define how the variously classified fragments should be handled.

Related Topics

[Tagging](#)

[Tagging Commands](#)

[Control OPC With Tagging](#)

Required Input Files

The following files are required to run the Calibre Sparse OPC tools. Each is described in the subsequent sections. You can supply the setup parameters as a separate litho setup file, or within the SVRF rule file in the form of setup parameter blocks.

| | |
|--|-----------|
| Design Database | 65 |
| SVRF Rule File. | 65 |
| Litho Setup Files | 67 |
| Model Files and Directories | 69 |

Design Database

Typically, the design database for model-based RET is a GDS or OASIS[®]¹-formatted layout database. The Calibre RET batch tools operate on one mask at a time. However, the Calibre nmDRC hierarchical engine operates equally efficiently whether you supply the entire design database or a smaller database containing copies of only those layers used to create a single mask. Because of this, when you invoke any of the RET batch tools, you must explicitly pass the necessary layers to the tool by including them in the layer list for the LITHO operation.

SVRF Rule File

An SVRF rule file is a text file containing SVRF operations used to perform design rule checks, manipulate data, and invoke the Calibre RET batch tools. SVRF rule files must follow the syntax and conventions for Standard Verification Rule Format (SVRF).

Refer to “[Introduction to SVRF Rule Files](#)” on page 47 for more information.

LITHO Operation in an SVRF Rule File

You execute the RET batch tools by including the LITHO operation in the SVRF file. Technically, the LITHO operation is a layer constructor. That is, it creates new data based on existing data. As with all operations, the output must be stored as either a derived layer or a rule check statement in order for it to persist.

You should only store the data as a rule check statement if you do not plan to do any further processing on the data, as shown in the following example.

```
POLY_ORC {LITHO ORC FILE setup/orc_poly.in POLY}
```

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.

If you intend to use the output from one LITHO operation as input to another operation, you must store the data as a derived layer. In this case, you can Copy the derived layer to a rule check statement and output it as well. The following example shows how you can do this.

```
POLY_ISL_OP = LITHO OPC
FILE setup/isl_frag.in POLY ISL          // output as layer
POST_IMAGE { LITHO PRINTIMAGE
    FILE "./setup/image_poly_post.in" POLY POLY_ISL_OP }
POLY_ISL_OP {COPY POLY_ISL_OP}           // Rule check given same name as
                                         //derived layer
```

To save the results of the LITHO operation (that is, the results generated by any of the RET batch tools), you must specify a layer number for that data using the DRC Check Map statement. This statement instructs the Calibre nmDRC hierarchical engine to output the specified rule check statement on the specified layer in the results database. The following is an example of a simple rule file you can use for OPC on a POLY layer.

Example SVRF Rule File

```
//This rule file covers how to execute basic OPC on a layout with
// a diffusion layer and a poly layer.
LAYOUT SYSTEM OASIS
LAYOUT PATH "gds/my_opc.oas"
LAYOUT PRIMARY "*"
LAYER DIFF 2
LAYER POLY 4

PRECISION 1000
RESOLUTION 1

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "gdsout/opc_result.oas" OASIS PSEUDO
DRC SUMMARY REPORT "reports/opc_result.rep"

POLY {copy POLY}
//Copies the original POLY layer for a Pre-OPC/Post-OPC comparison.

OPC_poly {LITHO OPC file "setup/opc_setup.in" POLY}
// The SVRF call to LITHO to carryout OPC on your POLY layer.

DRC CHECK MAP POLY 4
//Outputs the original POLY layer to the Results Database. It will appear
//in opc_result.oas as layer 4.

DRC CHECK MAP OPC_poly 104
//Outputs the OPC corrected layer to the Results Database. It will appear
//in opc_result.oas as layer 104.
```

For information on [Layer Constructors](#) and [Output Statements](#), refer to “[Introduction to SVRF Rule Files](#)” on page 47. For additional information about the SVRF language, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).

Litho Setup Files

The Calibre OPCpro, Calibre ORC, and Calibre PRINTimage batch tools are controlled through simulation parameters and settings that are stored as setup files. A setup file may be either a separate file or part of a LITHO statement. By convention, the name of a setup file includes the suffix *.in* such as *my_opc.in*.

Setup files are described in more detail in “[LITHO FILE](#)” on page 137.

You can also create the setup file using a text editor. The exact requirements of this file are described in “[Litho Setup File Format](#)”. See “[Litho Setup File Keywords](#)” on page 152, “[Litho Variables](#)” on page 267, and “[Tagging Commands](#)” on page 368 for detailed descriptions of the setup contents.

You can test and fine-tune any litho setup file using the Litho File Tool or the RET Flow Tool. The **OPC** button lets you perform OPC on a small portion of the design and see the results in the display window.

Note

 If the setup file is a separate file, it must remain the same throughout the Calibre run. In the best case, changed files result in syntax errors; in the worst case, different calculations occur than expected. If you want to modify a single setup file for a series of runs, set Calibre to save a copy when the run starts using code like the following:

```
LITHO FILE setup_block [ /*  
    INCLUDE litho.in  
 */ ]
```

Example Setup File

```
----- opc_setup.in Setup File Begins Here -----  
# This sample setup file can be generated in the Setup File Wizard in  
# WORKbench. This sample targets the 180 micron technology node. It  
# governs all the OPC, ORC, and PRINTimage simulations.  
  
# ----- Simulation models -----  
# Simulation Models - Use this section to define simulation model  
# information such as:  
# - The directories to search for the models  
# - Optical model directory  
# - Process model name  
  
modelpath ./models  
opticalmodel model_directory_name  
resistpolyfile test.mod  
  
# ----- OPC algorithm -----
```

```
# OPC Algorithm - Use this section to define OPC parameters such as:  
# - Number of iterations for OPC  
# - The input GDS grid size  
# - The edge movement size, or stepsize  
# - Site information (location points for intensity calculations)  
# - Corner site styles (placement of corner sites)  
  
iterations 8  
tilemicrons 100  
stepsize 0.001  
gridsize 0.001  
siteinfo RESIST  
cornerSiteStyle SITES_ON_ARC  
lineEndAdjDist 0.190000  
convexAdjDist 0.140000  
concaveAdjDist 0.140000  
  
# ----- Fragmentation -----  
# Fragmentation - Use this section to define fragmentation parameters such  
# as:  
# - Fragmentation types such as interfeature, island-layer, visible-layer  
# - Features to fragment  
# - Fragmentation limits such as max or min edge length, corners, jog size  
  
minfeature 0.180000  
minedgeLength 0.060000  
maxedgeLength 50  
cornedge 0.130000  
concavcorn 0.130000  
interfeature -interdistance 0.60000 -ripplelen 0.130000 -num 0 \  
    -distancePriority 1 -ripplestyle 1  
seriftype 0  
minjog 0.060000  
lineEndLength 0.320000  
  
# ----- Layer Info -----  
# Layer Info - Use this section to define layer information such as:  
# - Layer definitions  
# - Background definitions  
  
background clear  
  
# Layers  
layer 2 DIFF 17 0 hidden dark  
layer 4 POLY 1 0 opc dark
```

```
--- Arbitrary Commands Can Follow This Line. Don't delete this line! -----
# Arbitrary Commands Area - Use this area to add additional setup features
# such as:
# - Setup environment variables
# - Custom Tcl scripts
# - fragmentLayer blocks (command blocks used to override
# global or default settings for specified layers)

sse LITHO_PROMOTION_TILING 1

MRC_RULE EXTERNAL {USE 0.16}
MRC_RULE INTERNAL {USE 0.13}
```

Model Files and Directories

The Calibre RET tools use models to simulate the effects of the various processes involved in transferring an image onto a wafer. They support two types of models: optical and resist.

- **Optical Models** describe the aerial image formation.
- **Resist Models** describe the development of the aerial image onto the wafer.

You can create optical and resist models with Calibre WORKbench. Each optical model is stored as a *directory* of files. Each resist model is stored as a *file* whose name includes the suffix *.mod* by convention (for example, *ctr0.3.mod*). For model descriptions and development methods, refer to the “[VT5 Resist Model Creation](#)” and “[Optical Model Creation](#)” sections in the *Calibre WORKbench User’s and Reference Manual*.

For a setup file to load correctly, all model files referenced in the file must either be supplied in the setup file or reside in one of the directories beneath the directory specified by the [modelpath](#) keyword.

Output Files

Because the RET batch tools are invoked from within the Calibre nmDRC hierarchical engine, they return the results of their processing directly to the hierarchical engine in the form of derived layers or rule check statements—they do not generate any output files themselves.

To view the results, you must write the data to the results database. You do this using SVRF output statements. For information on this topic, refer to “[Calibre Results](#)” on page 31 and “[Working With the Calibre Hierarchical Engine](#)” on page 27.

Related Topics

[Derived Layers and Rule Check Statements](#)

[Calibre Results](#)

Chapter 5

Key Concepts

Getting good results with OPCpro requires that you configure the tools properly before running the applications. Some of this configuration involves controlling how and where the tools perform the simulations, and some involves controlling what the batch tools do with the simulation results. Because configuration varies according to the tool you are running, you should consult the tool-specific chapters for discussions of how to use the various configuration parameters and commands to optimize results. Before you begin, it is helpful to understand some of the basic configuration concepts.

- **Fragmentation** — The process of breaking up a layout polygon’s edges into smaller segments called “fragments.”
- **Tagging** — The process of defining named subsets of edges or edge fragments so you can control treatment of the different types of edge fragments.
- **Control sites** — The locations at which the tools calculate simulation data.
- **Tile boundaries** — The locations at which hierarchical run mode clips regions of a layer for processing in separate threads.

| | |
|--|------------|
| Fragmentation Overview | 72 |
| Fragmentation Strategies | 80 |
| Viewing Fragmentation | 103 |
| Tagging | 104 |
| Control Sites | 112 |
| Layer Usage | 120 |
| Tiles and Tile Boundaries | 126 |

Fragmentation Overview

Fragments are important for correcting designs. A fragment can be an entire edge or part of an edge. Most fragments have a control site that measures edge placement errors (EPEs). Calibre OPCpro corrects EPEs by moving individual fragments.

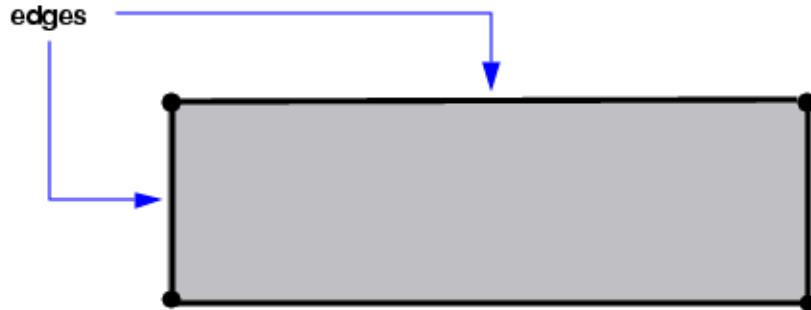
| | |
|--|-----------|
| Fragments and Edges | 72 |
| Effect of Fragments on OPC..... | 73 |
| Fragment Types and Related Objects | 74 |
| Fragmentation Setup..... | 77 |
| Fragmentation Constraint Parameters | 78 |

Fragments and Edges

A fragment is an edge or part of an edge.

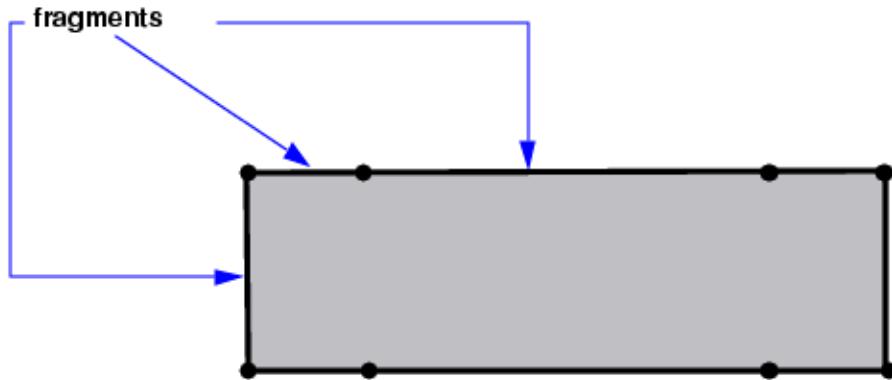
The term *edge* refers to a line segment between two corners. Figure 5-1 presents a simplified representation of two edges on a polygon.

Figure 5-1. Edge Illustration



A *fragment* can be an entire edge or part of an edge in a layout polygon (see Figure 5-2). Fragments are also called edge segments, especially within the SVRF documentation.

Figure 5-2. Edge Fragment Illustration

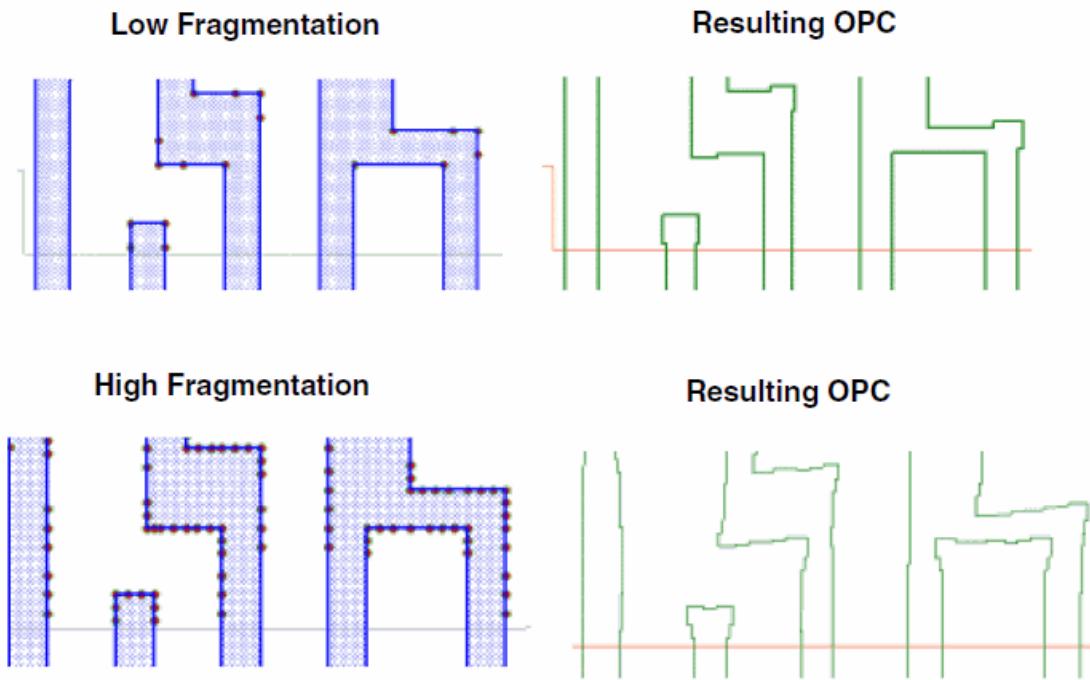


Effect of Fragments on OPC

Fragments are important for correcting designs. They represent the smallest unit that can be moved. Fragments include a control site at which edge placement errors (EPEs) are reported.

- The Calibre OPCpro batch tool corrects EPEs by moving individual fragments. Because of this, it only corrects the design *where fragments exist*.
- The RET batch tools only collect simulation data at control sites. Because all control sites are created on fragments (one control site per fragment), aerial image simulation can only report EPEs (the difference between a drawn edge and the simulated edge) and image intensity for those areas of the design *where fragments exist*.

When the Calibre OPCpro batch tool 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 Calibre OPCpro batch tool moves each fragment as necessary to generate a mask that produces the optimal image. The level of fragmentation you define can affect the resulting OPC. [Figure 5-3](#) compares the effect of high fragmentation versus low fragmentation.

Figure 5-3. Effect of High Versus Low Fragmentation

Fragment Types and Related Objects

The Calibre RET model-based tools recognize the following types of fragments: *edge fragments*, *line-end fragments*, *space-end fragments*, and *jogs*. There are also several objects relating to *fragments*, including *control sites*, *control points*, *convex* and *concave corners*, and *features*. This section illustrates these terms.

Table 5-1. Fragment Types and Related Objects

| Fragment Type or Object | Description |
|-------------------------|---|
| Edge | A line segment between two corners.  |

Table 5-1. Fragment Types and Related Objects (cont.)

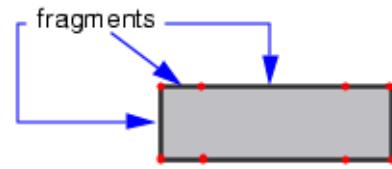
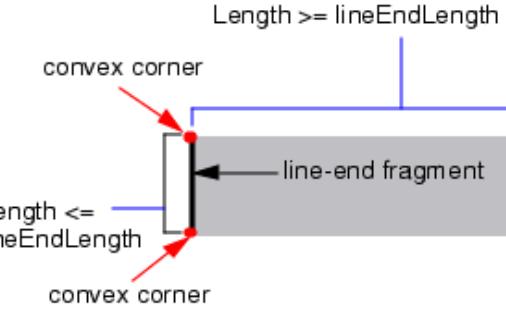
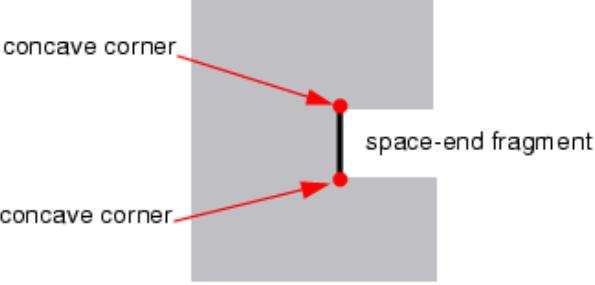
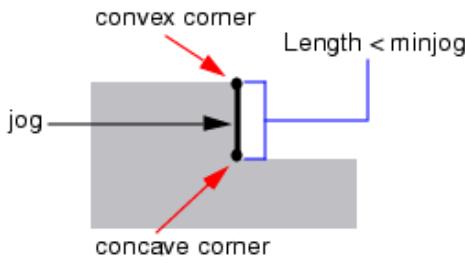
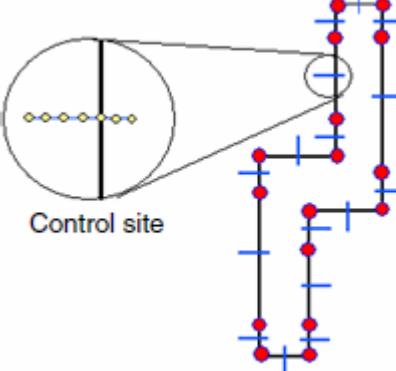
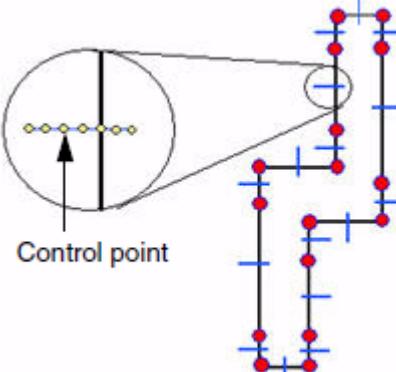
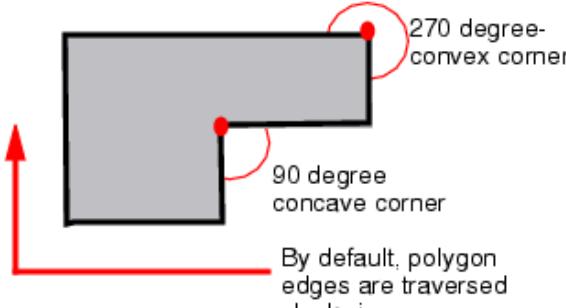
| Fragment Type or Object | Description |
|-------------------------|---|
| Fragment | <p>A line segment between two vertices. Fragments, which are also called “edge fragments,” can be an entire edge (if the two vertices are both corners) or a part of an edge (if at least one vertex is not a corner).</p>  |
| Line-end fragment | <p>An entire edge that satisfies all of the following criteria:</p> <ul style="list-style-type: none"> Length is less than or equal to the <code>lineEndLength</code>. Both endpoints are convex corners. Adjacent edges are longer than or equal to <code>lineEndLength</code>.  |
| Space-end fragment | <p>An entire edge that satisfies all of the following criteria:</p> <ul style="list-style-type: none"> Length is less than or equal to <code>minfeature</code>. Both endpoints are concave corners.  |
| Jog | <p>A fragment that satisfies all of the following criteria:</p> <ul style="list-style-type: none"> One endpoint is a concave corner; the other is a convex corner. Length is less than <code>minjog</code>.  |

Table 5-1. Fragment Types and Related Objects (cont.)

| Fragment Type or Object | Description |
|---------------------------------|---|
| Control site | <p>Locations on the opc layer where simulation data is evaluated for edge placement errors. The RET tools create one control site on every fragment except for jog fragments.</p>  |
| Control point | <p>Locations on a control site where the RET tools calculate the image intensities. Control points are numbered from the lightest area to darkest area, beginning with point 0. With clear background setting features, control points are numbered from the exterior of a figure inwards.</p>  |
| Feature | <p>A non-collinear corner. Features are typically corners of nearby polygons. However, they can also be corners on the same polygon as the edge being fragmented.</p> |
| Convex corner Concave corner | <ul style="list-style-type: none"> A convex corner is where two vertices form an exterior (outward) angle. A concave corner is where two vertices form an interior (inward) angle. <p>Generally, polygon edges are traversed clockwise.</p>  <p>By default, polygon edges are traversed clockwise</p> |

Fragmentation Setup

Fragmentation is the process of adding vertices to break up a layout polygon's edge (or part of an edge) into smaller segments. Instructions in the setup file let you control the number and location of vertices, which define the lengths and positions of the resulting fragments. These instructions take the form of variables, commands, and keywords.

Figure 5-4. Setup File Example (Fragmentation)

```
# ----- Simulation models -----
modelpath ./models
opticalmodel mymodel.opt
resistpolyfile mymodel.mod

# ----- OPC algorithm -----
iterations 4
tilemicrons 160.000000
stepsize 0.001
gridsize 0.001
siteinfo RESIST
cornerSiteStyle SITES_ON_ARC
lineEndAdjDist 0.190000
convexAdjDist 0.140000
concaveAdjDist 0.140000

# ----- Fragmentation -----
# 1. Define a Fragmentation Scheme
# Define constraints and global fragmentation schemes in
# the "Fragmentation" section of the setup file. The types of
# fragmentation schemes you can implement are described in
# "Fragmentation Strategies" on page 80
#
minfeature 0.180000
minedgelength 0.130000
maxedgelength 1000.000000
cornedge 0.130000
concavecorn 0.130000
interfeature -interdistance 0.290000 -ripplelen 0.130000 -num 0
seriftype 0
minjog 0.120000
lineEndLength 0.320000

# ----- Layer info -----
# 2. Create Input Layers
# In the "Layer info" section, use the layer keyword to define
# input layers for OPC runs.
#
background clear
# Layers
layer 2 DIFF 17 0 hidden dark
layer 4 POLY 17 0 opc dark
layer 5 L237 17 0 hidden dark
layer 104 HI_OPCTM 49 0 hidden dark
layer 105 DEF_OPCTM 49 0 hidden dark
layer 106 LO_OPCTM 49 0 hidden dark
```

```

----- Arbitrary Commands -----
# 3. Change Fragmentation Defaults for Multilayer OPC
# If you have a multilayer OPC run, you can change global fragmentation
# settings for a particular layer using fragmentLayer block, as
# described in "Multilayer Fragmentation (fragmentLayer)" on page 98.
#
sse LITHO_PROMOTION_TILING 1
MRC_RULE INTERNAL {
    USE 0.13 EUCLIDEAN
}
MRC_RULE EXTERNAL {
    USE 0.16 EUCLIDEAN
}

fragmentLayer 2 {
    interfeature -ripplen 0.1450000
}

```

Fragmentation Constraint Parameters

When you fragment your features, there are several keywords that you can apply as constraint parameters to your fragmentation schemes. For instance, prior to applying fragmentation during an OPC run, you can set a minimum size constraint for the length of a fragment and the minimum size constraint for a feature to be fragmented.

Table 5-2. Fragmentation Constraint Parameters Summary

| Keyword | Description |
|--|---|
| Minimum or Maximum Size and Length Constraints | |
| maxedgelength <i>maxedge</i> [-split 0 1] | Specifies the maximum edge length. Longer edges are fragmented to less than this value. |
| minedgelength <i>minedge</i> | Limits the minimum length of an edge that is created when fragmenting. |
| minfeature <i>minsize</i> | Specifies the minimum feature size of the input GDS. This value is used to calculate default values for unspecified keywords. No edge can move more than minfeature/2 during OPC. This is a required keyword. |
| minjog <i>minjogsz</i> | Specifies the minimum length for the fragment to be considered a jog. |
| Line-End and Line-End Adjacent Fragment Constraints | |
| lineEndAdjDist <i>distanceValue</i> | Specifies the distance away from a line end which is considered to be adjacent to the line end. |
| lineEndLength <i>Length</i> | The lineEndLength parameter defines the distance criteria used to determine whether or not a fragment is a line end. |

Table 5-2. Fragmentation Constraint Parameters Summary (cont.)

| Keyword | Description |
|--|--|
| Corner Adjacent Fragment Constraints | |
| <code>convexAdjDist <i>distanceValue</i></code> | Specifies the distance away from a concave corner considered to be adjacent to the corner. |
| <code>concaveAdjDist <i>distanceValue</i></code> | Specifies the distance away from a convex corner adjacent to the corner. |

Example

```
minfeature 0.180000
minedgelength 0.130000
maxedgelength 1000.000000
cornedge 0.130000
concavecorn 0.130000
interfeature -interdistance 0.290000 -ripplelen 0.130000 -num 0
seriftype 0
minjog 0.120000
lineEndLength 0.320000
```

This example is from a setup file. It sets values for the minimum feature size (minfeature), minimum and maximum fragment edge length (minedgelength and maxedgelength), minimum length for a jog (minjog), and minimum line end length (lineEndLength). When OPC runs, the values determine the fragmentation scheme.

Note

 The minfeature keyword is *required* and must always be defined in your fragmentation section.

Fragmentation Strategies

Because the Calibre OPCpro batch tool operates on individual fragments, different fragmentation schemes result in different mask designs. And because aerial image simulation data for all the batch tools is only collected at control sites on fragments, different fragmentation schemes can result in more or less accurate rule checking and results precision. Controlling fragmentation is therefore critical for producing high yield, printable mask designs.

Fragmentation affects both simulation and OPC. Because of this, you must determine the method of fragmenting your design to make the appropriate setup and refinements required to run RET tools.

There are three basic strategies you can use to create fragmentation:

- **Global 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, interfeature, maxedgelength, island-layer, visible-layer, and implicit fragmentation.
- **Special Case Fragmentation** — This strategy uses the fragmentTag keyword to adjust specific edges in cases where you have special requirements for those particular fragments.
- **Multilayer Fragmentation** — Used for multilayer masks, this strategy uses the fragmentLayer command block to allow you to make custom settings for global fragmentation types on specified layers.

| | |
|--|------------|
| Global Fragmentation | 81 |
| Special Case Fragmentation (fragmentTag) | 94 |
| Multilayer Fragmentation (fragmentLayer) | 98 |
| Multilayer Fragmentation Issues - Managing Default Settings | 102 |

Global Fragmentation

The Calibre OPCpro tools support several different types of fragmentation:

- **Intrafeature Fragmentation** — Evaluates each polygon on the layout, independent of other polygons, and adds vertices at specified distances from the feature's corners.
- **Interfeature Fragmentation** — Evaluates each polygon in terms of proximity to nearby features and adds vertices to edges that are within a specified distance from these features.
- **Maxedgelength Fragmentation** — Breaks long fragments into shorter fragments.
- **Island-Layer Fragmentation** — Inserts vertices where the edges on an opc layer or correction layer intersect edges or vertices on an island layer.
- **Visible-Layer Fragmentation** — Inserts vertices based on two types of interactions.
- **Implicit Fragmentation** — Breaks fragments to help manage promotion of data between levels of the hierarchy.

Use keywords such as minedgelength and maxedgelength to specify minimum and maximum fragment lengths and the layer keyword to specify which types of fragmentation are permitted for the various types of edges on a specific layer. Use the layer keyword in the setup file to control the amount of fragmentation and the types of edges affected. This keyword controls many other layer-related functions as well. For more information on the layer keyword, refer to “[layer](#)” on page 207.

Additional keywords and variables allow you to refine the fragmentation scheme even further.

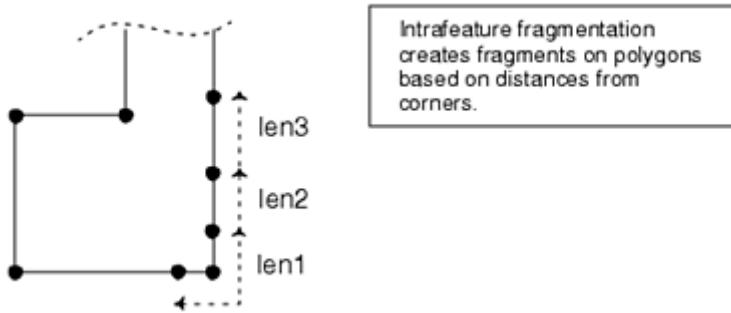
| | |
|--|-----------|
| Intrafeature Fragmentation | 81 |
| Interfeature Fragmentation | 86 |
| Maxedgelength Fragmentation. | 89 |
| Island-Layer Fragmentation. | 90 |
| Visible-Layer Fragmentation | 92 |
| Implicit Fragmentation | 92 |
| Summary of Global Fragmentation Methods | 93 |
| Global Fragmentation Issues - Hierarchical Interactions | 94 |

Intrafeature Fragmentation

Intrafeature fragmentation evaluates each polygon on the layout, independent of other polygons, and adds vertices at set distances from the polygon corners.

[Figure 5-5](#) shows how the intrafeature fragmentation parameters are applied to produce fragmentation edges.

Figure 5-5. Intrafeature Fragmentation



Intrafeature Fragmentation Controls

You control how vertices are added during intrafeature fragmentation with the keywords `cornedge` (for convex corners) and `concavcorn` (for concave corners), or `intrafeature` (for both corner types). These keywords let you define the number and lengths of fragments created at edges adjacent to corners, specify the minimum size of remaining fragments, and assign priority for fragmentation of edges based on the types of corners. [Table 5-3](#) lists the supported keywords and syntax.

Table 5-3. Intrafeature Fragmentation Summary

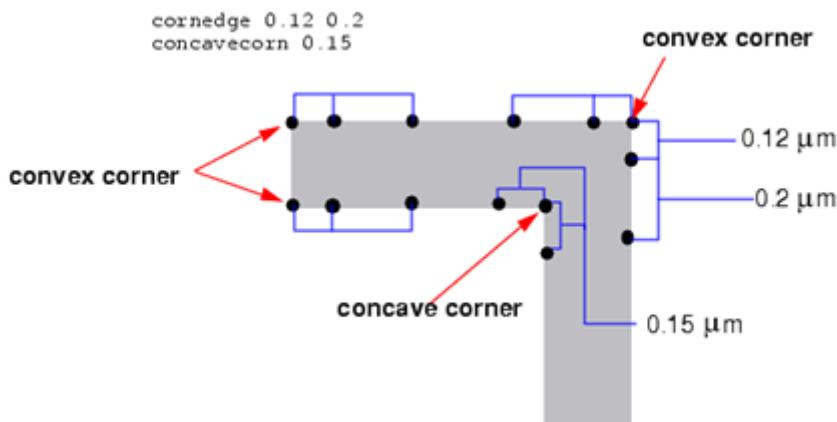
| Keyword | Description |
|--|---|
| <code>concavcorn length1 ... lengthN [rem r] [non90 distance1 ... distanceN [rem r]] [sea sea_distance1 ... sea_distanceN [rem r]]</code> | Gives precise control over the number and length of intrafeature fragmented edges at concave corners (inside corners). |
| <code>cornedge length1 ... lengthN [rem r] [priority] [non90 distance1 ... distanceN [rem r] [priority]] [lea lea_distance1 ... lea_distanceN [rem r] [priority]]</code> | Gives precise control over the number and length of intrafeature fragments created from edges at convex corners (outside corners). |
| <code>intrafeature edge_spec [fragment] rule [corner2 [fragment] rule]</code> | An alternative syntax that combines both <code>concavcorn</code> and <code>cornedge</code> functionality to control fragments at corners. This keyword syntax more finely controls fragmentation by allowing combinations of features. It also allows splitting of fragments into arbitrarily sized equal pieces. |
| <code>layer layer_num layer_name frag bits 1, 2, 4, 8, 128 -or- flaglayer layer_name intrafeature_* enable</code> | When defining the output layer, set the <code>frag</code> bit option in the <code>layer</code> keyword to 1, 2, 4, 8, or 128 to enable intrafeature fragmentation. The <code>frag</code> option defines the type of fragmentation to be performed on the layer. |

You supply the number and lengths of fragments as an ordered list of distance arguments, which you can interpret as follows:

- The total number of arguments indicates the maximum number of new fragments created on either side of the corner.
- The argument values specify the lengths of the fragments.
- The order in which you supply the arguments defines the position of the fragments. The first argument specifies the length of the fragment directly adjacent to the corner. The second argument specifies the length of the next adjacent fragment, and so on.

[Figure 5-6](#) illustrates the results of intrafeature fragmentation using the cornedge and concavecorn keywords in the setup file (this is only a partial shape and does not include all fragmentation points).

Figure 5-6. cornedge and concavecorn Example



The arguments following the cornedge and concavecorn keywords define the default fragmentation distances, determining the length of the fragments.

The [cornedge](#), [concavecorn](#), and [intrafeature](#) keywords allow you to fragment based on offsets from concave or convex corners. You can place further conditions on fragmentation (fragmenting adjacent to specified corners) based on three different categories:

- **non90**

Specifies the fragmentation values at non-90 degree concave corners when such fragments require different treatment.

- **lea (Line End Adjacent)**

Used only in conjunction with cornedge, lea specifies the fragmentation values at line-end adjacent convex corners when such fragments required different treatment.

In intrafeature, the equivalent type is “convex end_adjacent.”

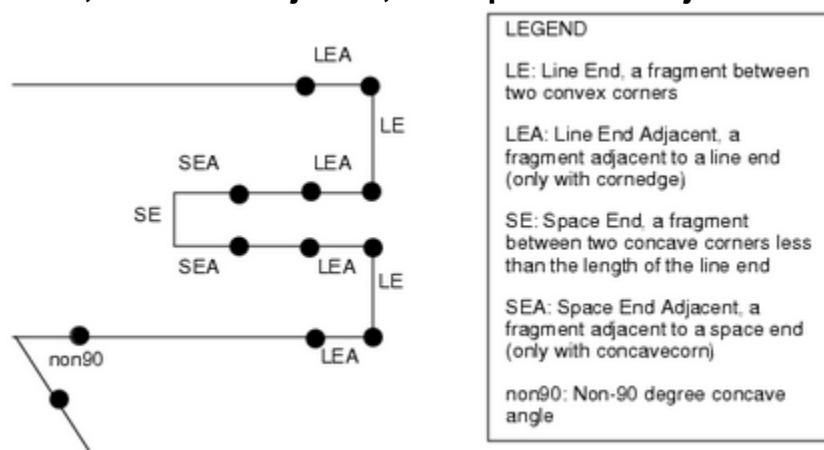
- **sea (Space End Adjacent)**

Used only in conjunction with concavecorn, sea specifies the fragmentation values at space-end adjacent concave corners when such fragments require different treatment. If two concave corners are separated by a distance of less than the line-end length, then it is called a “space end.” A space-end adjacent fragment is a fragment that touches the space end.

In intrafeature, the equivalent type is “concave end_adjacent.”

Figure 5-7 illustrates differences between non90, sea, and lea fragments (this example is only a partial shape and does not include all fragmentation points).

Figure 5-7. non90, Line-End Adjacent, and Space-End Adjacent Fragmentation



The remaining arguments define the minimum length and priority for your fragmentation:

- **split *n* [force]**

Used in conjunction with intrafeature only, the split option indicates the edge is to be broken into *n* equal-length fragments. If the fragments would be shorter than minedgelength, the transcript shows the error “Split by *n* violates minedgelength” and the edge is not split. You can override this error using the optional force keyword but this is not recommended; instead, use interval constraints with a lower bound of minedgelength * *n*.

- **rem *r***

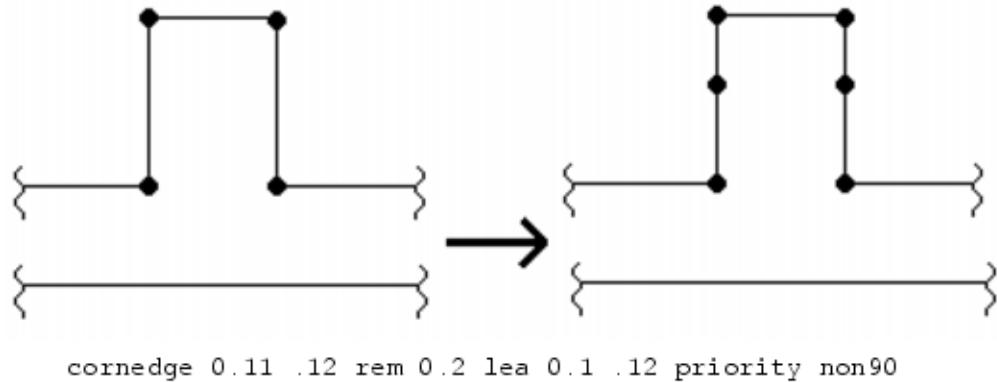
Use the rem argument to control the minimum lengths for fragments remaining after fragmentation. If rem directly follows the keyword and lists default distances, it defines a default remainder length. If it follows one of lea (cornedge only), sea (concavecorn only), or non90, it defines the remainder length for the associated type of edge.

- **priority**

Use the priority option with the cornedge keyword to give priority to convex corners (or either of the special types of convex corners, lea, and non90) when an edge has both a

convex and a concave corner, and the fragment is not long enough for fragmentation from both corners. Using priority creates fragments inward from the convex corner first. In this case, the rem value from the convex corner is used. Otherwise, the larger of the two rem values is used. [Figure 5-8](#) shows an example of using cornedge with the priority option enabled.

Figure 5-8. Priority Example



In this example, cornedge creates line-edge adjacent fragments (lea) and limits fragmentation elsewhere, using the priority option give priority to non-90 degree convex corners (in this case, a 270-degree angle).

For more information, see “[concavcorn](#)” on page 161 and “[cornedge](#)” on page 164.

Intrafeature Fragmentation Rules

When using intrafeature fragmentation, remember the following interactions:

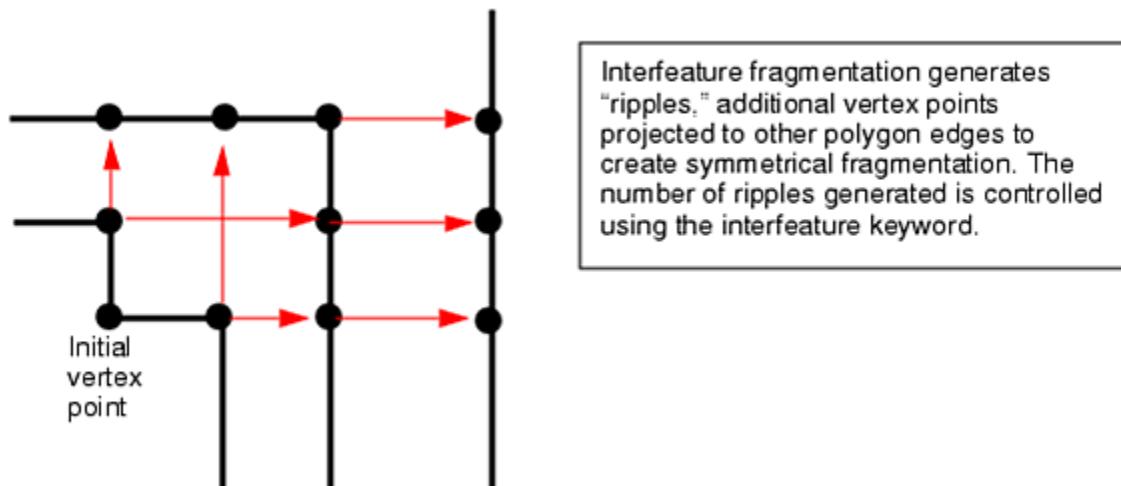
- The arguments to the cornedge, concavcorn, and intrafeature keywords define the lengths of fragments to be generated, as well as the total number.
- The [layer](#) keyword in the setup file defines the fragmentation level for the specified layer. With proper setting of the fragmentation level, you can suppress intrafeature fragmentation by corner type and fragment type.
- By default, intrafeature fragmentation *is not performed* on fragments that match the line-end definition. Notice that in [Figure 5-6](#), the small edge on the left of the figure was not fragmented because its length is less than the value of lineEndLength. However, you can override this default behavior by setting the keyword [seriftype](#) to 1.
- Intrafeature fragments *are not produced* if the resultant fragment’s length would be less than the value defined for [minedgelength](#) in the setup file.

Interfeature Fragmentation

Interfeature fragmentation evaluates each polygon in terms of proximity to nearby features and adds vertices to edges that are within a specified distance from these features. A feature is defined as a non-collinear corner. Features are typically corners of nearby polygons. However, they can also be corners on the same polygon as the edge being fragmented.

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. [Figure 5-9](#) illustrates the placement of vertices to create ripple fragments.

Figure 5-9. Interfeature Fragmentation Example



The remaining arguments define the minimum length and priority for your fragmentation.

Interfeature Fragmentation Controls

You control how vertices are added during interfeature fragmentation through the `interfeature` keyword, also in the setup file. This keyword takes optional arguments. (If you do not specify any arguments, the toolset calculates appropriate values.) You control the types of edges affected on a layer by layer basis using the `layer` keyword in the setup file.

[Table 5-4](#) lists the syntax for interfeature fragmentation keywords.

Table 5-4. Interfeature Fragmentation Summary

| Keyword | Description |
|--|---|
| <code>interfeature</code> [-num <i>x</i>] [-interdistance <i>y</i>] [-ripplelen <i>z</i>] [-rem <i>r</i>] [-shield <i>n</i>] [-distancePriority {0 1}] [-ripplestyle {0 1}] [-split {0 1}] | Defines the fragmentation parameters that influence interfeature fragmentation. |

Table 5-4. Interfeature Fragmentation Summary (cont.)

| Keyword | Description |
|--|--|
| <code>seriftype {0 1}</code> | Determines how line ends are fragmented during interfeature fragmentation. When the seriftype = 0, then any edge with the lineEnd property is not fragmented further, resulting in a hammerhead line end. If seriftype = 1, then edges marked with lineEnd have normal fragmentation applied. The <code>lineEndLength</code> keyword defines what edges are classified as line ends. |
| <code>layer layer_num layer_name frag bit 16</code> -or- <code>flaglayer layer_name interfeature_* enable</code> | When defining the output layer, set the frag bit option in the layer keyword to 16 to enable interfeature fragmentation. The <code>frag</code> option defines the type of fragmentation to be performed on the layer. |

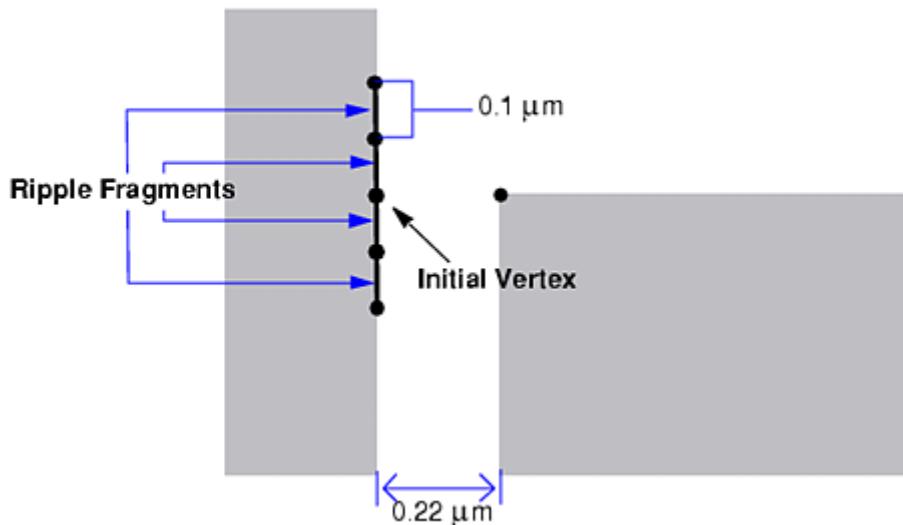
The interfeature arguments define how vertices are placed and how many ripples are generated and at what distance:

- `-interdistance y`
Defines the maximum spacing between an edge on one polygon and a corner on an adjacent polygon that causes interfeature fragmentation to occur.
- `-ripples x`
Specifies the number of ripple fragments to be created on each side of the initial vertex.
- `-ripplelen z`
Specifies the length of each ripple fragment.
- `-ripplestyle {0 | 1}`
Sets priority to resolve conflicts in fragmentation point placement during ripple generation. For best results, set `-ripplestyle` to 1.
- `-rem r`
Specifies the minimum allowed remainder resulting from interfeature fragmentation.
- `-shield n`
Limits the number of edges fragmented by 1 corner. This option prevents the fragmentation of any edge shielded by *n* edges.
- `-split {0 | 1}`
Controls the splitting of fragments. When set to 0, this option preserves a single fragment. When set to 1, the fragment is split into two equal fragments.

Figure 5-10 provides an example of using the [interfeature](#) keyword in the setup file and the resultant fragmentation.

Figure 5-10. interfeature Example

```
interfeature -interdistance 0.3 -num 2 -ripplelen 0.1
```



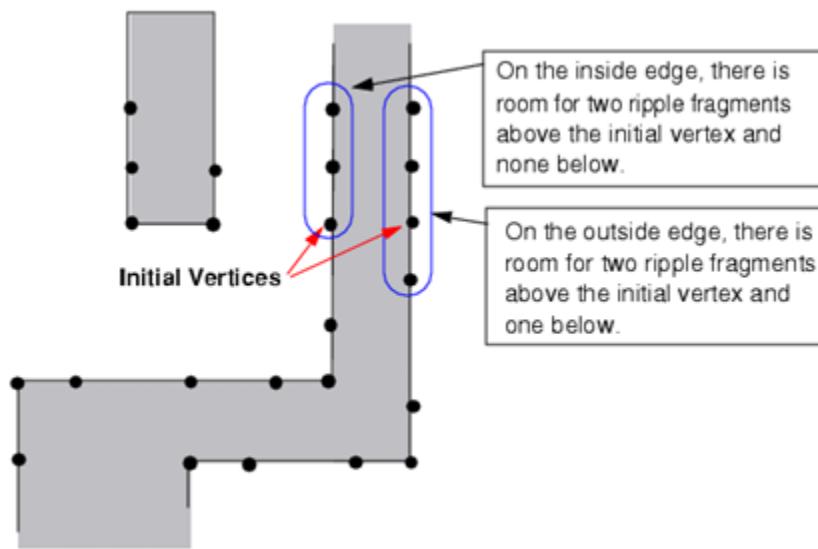
Interfeature Fragmentation Rules

When using interfeature fragmentation, keep in mind that:

1. Whenever possible, the RET batch tools create ripple fragments in pairs.
2. Ripple fragments *are not produced* if the resultant fragment's length is less than the value set for the [minedgelength](#) keyword in the setup file, or if the length of the remaining fragment is less than [-rem](#).

Figure 5-11 shows an example of interfeature fragmentation in which not all ripple fragments can be created in pairs. This occurs because doing so would result in edges smaller than [minedgelength](#). Notice the intrafeature fragmentation at the corners. The RET batch tools perform intrafeature fragmentation before interfeature fragmentation.

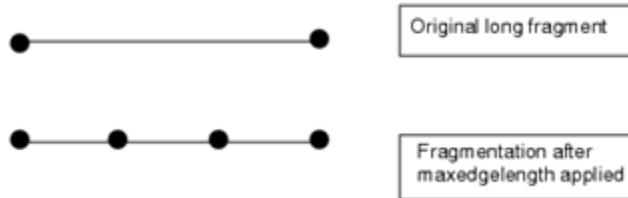
Figure 5-11. Asymmetrical Interfeature Fragmentation



Maxedgelength Fragmentation

Maxedgelength fragmentation breaks long fragments into shorter fragments.

Figure 5-12. Maxedgelength Fragmentation



Maxedgelength Fragmentation Controls

You control this type of fragmentation by setting the keyword `maxedgelength` in the setup file. [Table 5-5](#) lists the syntax for `maxedgelength` fragmentation.

Table 5-5. Maxedgelength Fragmentation Summary

| Keyword | Description |
|---|--|
| <code>maxedgelength maxedge [-split {0 1}]</code> | Splits a long fragment into shorter fragments. Longer edges are broken into three parts, two that are <code>maxedgelength</code> , and one that is \geq <code>minedgelength</code> . A long fragment is defined as $2 * \text{maxedgelength} + \text{minedgelength}$. |

The arguments for `maxedgelength` control the maximum fragment length and how the fragments are split.

- *maxedge*

A required argument specifying the maximum length of fragments generated by maxedgelength fragmentation, specified in microns.

The default value is *maxedge* = tilemicrons/2, which is so large that maxedgelength fragmentation is not performed.

- *-split {0 | 1}*

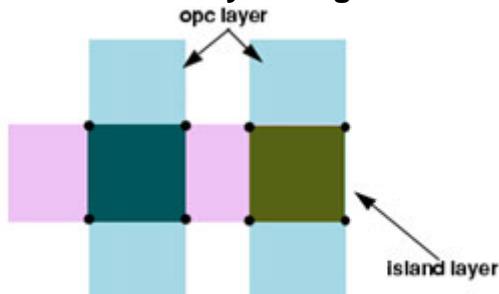
An optional argument controlling how this keyword splits fragments with a length is greater than 2*maxedgelength.

Island-Layer Fragmentation

Island-layer fragmentation inserts vertices where the edges on an opc layer or correction layer *intersect* edges or vertices on an island layer.

Depending on how you have configured island-layer fragmentation, it either adds just the vertex at each intersection or it adds ripple fragments as well, along the edges that coincide with the island layer.

Figure 5-13. Island-Layer Fragmentation



Island-Layer Fragmentation Controls

You control how and when the RET batch tools perform island-layer fragmentation using the setup controls listed in the table.

Table 5-6. Island-Layer Fragmentation Summary

| Keyword | Description |
|--|---|
| <code>islandFragment [-ripples x] [-ripplelen z] [-rem r] [-enforce {0 1}]</code> | Gives precise control over island layer fragmentation. |
| <code>layer layer_num layer_name frag bit 32 -or- flaglayer layer_name intersection_island enable</code> | When defining the output layer, set the frag bit option in the layer keyword to 32 to enable island-layer fragmentation. The <i>frag</i> option defines the type of fragmentation to be performed on the layer. |

To perform island-layer fragmentation, you must include one or more island-layers in the layer list for the LITHO operation and set the frag option on the layer keyword to permit island-layer fragmentation. You can use the islandFragment keyword to create new fragmentation points on fragmented layer edges at intersection points between fragmented layer edges and island layer edges. The arguments for the island-layer fragmentation controls are as follows:

- **-enforce {0 | 1}**

When set to 1, this option breaks a fragment at intersection points only if no fragments of lengths less than minedgelength are created.

- **-ripples *x***

Sets the number of ripples you wish to generate. The default value is 0 (no ripples are created). The maximum number of ripples you can create is 10.

- **-ripplelen *z***

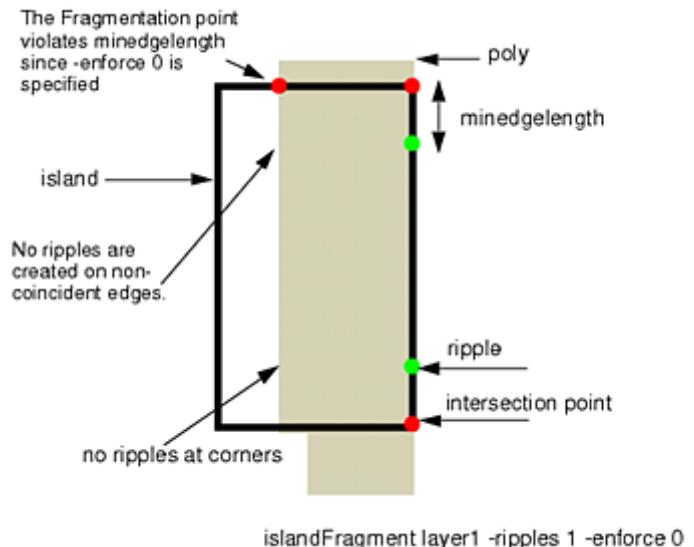
Specifies the length of a single ripple edge in microns. The default value is minedgelength.

- **-rem *r***

Specifies the minimum allowed remainder for the purposes of ripple generation. 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.

[Figure 5-14](#) shows an example use of the islandFragment keyword.

Figure 5-14. islandFragment Example



Island Layer Fragmentation Rules

When using island-layer fragmentation, you can optimize performance if you

- Limit the number of island layers passed into the LITHO operation, and pass only those layers necessary to accomplish the required task.
- Minimize the area of the island layers.

Visible-Layer Fragmentation

Visible-layer fragmentation inserts vertices where the edges on an opc layer or correction layer *intersect* edges or vertices on a visible layer.

Visible-Layer Fragmentation Controls

Visible-layer fragmentation is controlled using the [layer](#) keyword, as described in [Table 5-7](#).

Table 5-7. Visible-Layer Fragmentation Summary

| Keyword | Description |
|--|---|
| <code>layer layer_num layer_name frag bit 128</code> -or- <code>flaglayer layer_name *_visible enable</code> | When defining the layer, set the frag bit option in the layer keyword to 128 to enable visible-layer fragmentation. The frag option defines the type of fragmentation to be performed on the layer. By default, visible layers do not induce fragmentation on the opc layer. |

Visible layers contain geometries that do not receive correction yet appear on the mask and interact with other geometries to affect the final printed design. The data on visible layers can be non-printing data, such as scattering bars, or can be printing data that does not need correction. By default, visible layers are not fragmented and do not receive any sites (unless you set the frag bit to 128 to enable fragmentation).

A visible layer that induces fragmentation also induces fragmentation on jogs, regardless of how the frag bit is set on the opc layer.

Implicit Fragmentation

Implicit fragmentation is fragmentation performed by the hierarchical engine to help manage promotion of data between levels of the hierarchy. This is necessary because the simulator must be able “see” all the geometries in an area in order to generate an accurate aerial image. Implicit fragmentation occurs automatically (no controls are necessary).

Summary of Global Fragmentation Methods

This section summarizes the different global fragmentation methods and the default order in which they are processed outside of fragmentLayer blocks. Use the litho setup variable OPC_NEW_FRAG_ORDER to choose between the default order and an alternate order.

Table 5-8. Fragmentation Types Quick Summary

| Process Order | Fragmentation Type | Quick Description | Control Keywords |
|---------------|-----------------------------|---|--|
| 1 | Island-Layer Fragmentation | Island-layer fragmentation inserts vertices where the edges on an opc layer or correction layer intersect edges or vertices on an island layer. | islandFragment interfeature flaglayer layer, frag bit 32 |
| 2 | Visible-Layer Fragmentation | Visible-layer fragmentation inserts vertices based on two types of interactions. | flaglayer layer, frag bit 128 |
| 3 | Intrafeature Fragmentation | Intrafeature fragmentation evaluates each polygon on the layout, independent of other polygons, and adds vertices at specified distances from the figure's corners | cornedge concavecorn intrafeature flaglayer layer, frag bits 1, 2, 4, 8, 128 |
| 4 | Interfeature Fragmentation | Interfeature fragmentation evaluates each polygon in terms of proximity to nearby features and adds vertices to edges that are within a specified distance from these features. | interfeature flaglayer layer, frag bit 16 |
| 5 | Maxedgelength Fragmentation | Maxedgelength fragmentation breaks long fragments into shorter fragments. | maxedgelength |
| 6 | Implicit Fragmentation | Implicit fragmentation is fragmentation performed by the hierarchical engine to help manage promotion of data between levels of the hierarchy. | – |

Fragments are not generated smaller than minedgelength, except where this rule is explicitly overridden.

Related Topics

[OPC_NEW_FRAG_ORDER](#)

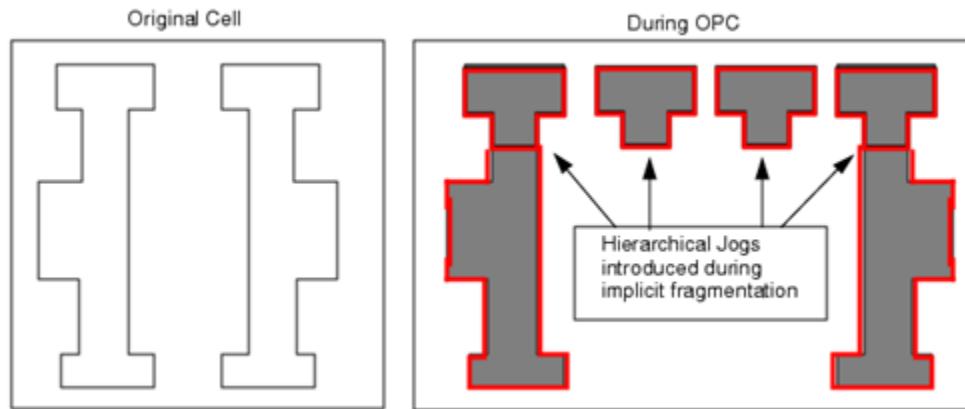
[Global Fragmentation Issues - Hierarchical Interactions](#)

Global Fragmentation Issues - Hierarchical Interactions

Fragmentation can result from hierarchical interactions. During OPC, the edges may break in unexpected fashions, introducing unwanted edge movement and jogs. The figure illustrates an example of hierarchical interactions caused during implicit (automatic) fragmentation.

Calibre attempts to preserve edges at a single level to give more consistent corrections at a small cost in run time and file size. However, some implicit fragmentation can still occur, even when you preserve edges.

Figure 5-15. Hierarchical Promotion Issues



In addition to jogs, occasionally design hierarchy can cause symmetrical layouts to not receive the same correction. This can happen when one part of the layout is promoted because of interactions that can only be resolved at the parent level. See the example and discussion in “[LITHO_SIMULATE_CONTEXT](#)” on page 311.

Special Case Fragmentation (fragmentTag)

There may be occasions when you have edges that require fragmentation according to specific requirements. In these cases, the global fragmentation models would not apply and you require a custom edge fragmentation. This “special case” fragmentation can be controlled using the fragmentTag keyword.

The fragmentTag keyword allows you to modify the fragmentation for individual sets of fragments. The fragmentTag command provides several options described in the following sections.

Note

 The fragmentTag keyword is not supported by PRINTimage. PRINTimage simulations do not reflect any fragmentTag operations.

Adding Additional Fragmentation

Use fragmentTag to add additional fragmentation vertices at user-specified positions, breaking fragments into more fragments. The syntax for this is as follows:

```
fragmentTag tagname -first {df1 df2 ...} -last {dl1 dl2 ...} [-rem r]
```

The arguments required for adding additional fragmentation are

- **-first**

A switch specifying the first point to tag. The *dfn* parameters are the distances in microns of each successive fragment vertex from the previous vertex. *df1* is the first distance specified in microns.

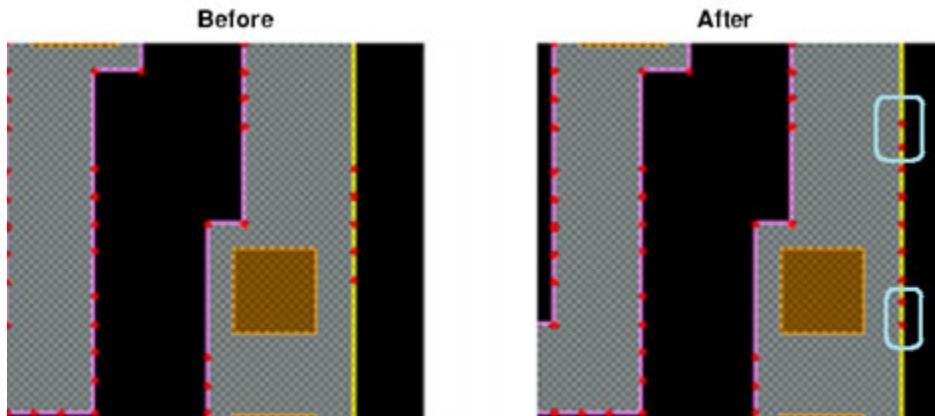
- **-last**

A switch specifying the last point to tag. The *dfn* parameters are the distances in microns of each successive fragment vertex from the previous vertex. *dl* is the last distance specified in microns.

- **-rem r**

An optional switch specifying the minimum remainder size after fragmentation. The default is minedgelength.

Figure 5-16. Adding Additional Fragmentation



```
newTag dense -how distance all all 0.001 0.5 -edgeBased -from outside -to outside
newTag iso -how subtractTags all dense
fragmentTag iso -first 0.08 0.08 -last 0.08 0.08 -unfragment
```

Removing Fragmentation and Refragmenting

You can use fragmentTag to merge adjacent fragments belonging to one or multiple tags. After this “unfragmenting” it also adds new fragmentation vertices, if configured to do so. The basic syntax is:

```
fragmentTag tagname -unfragment
```

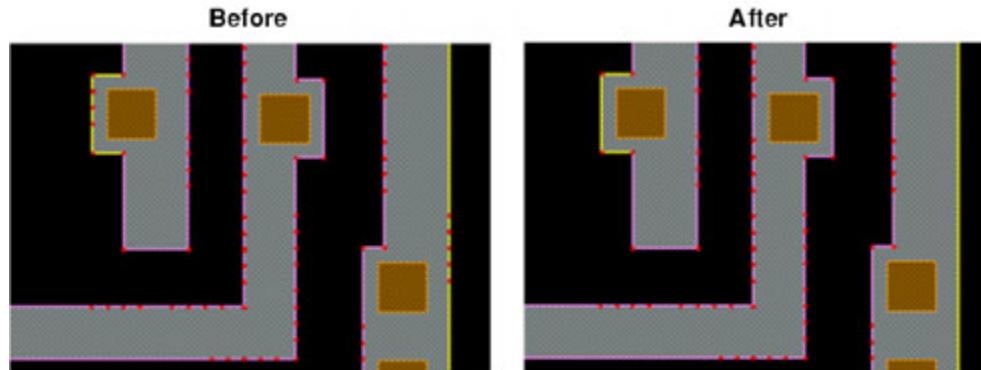
The required parameter is:

- **-unfragment**

Instructs the operation to merge consecutive fragments of the given tag name into a single fragment before performing the specified fragmentation. You can specify multiple tag sets when using the -unfragment option, as in the following example:

```
fragmentTag tagName1 [tagName2 ...] -unfragment
```

Figure 5-17. Removing Fragmentation

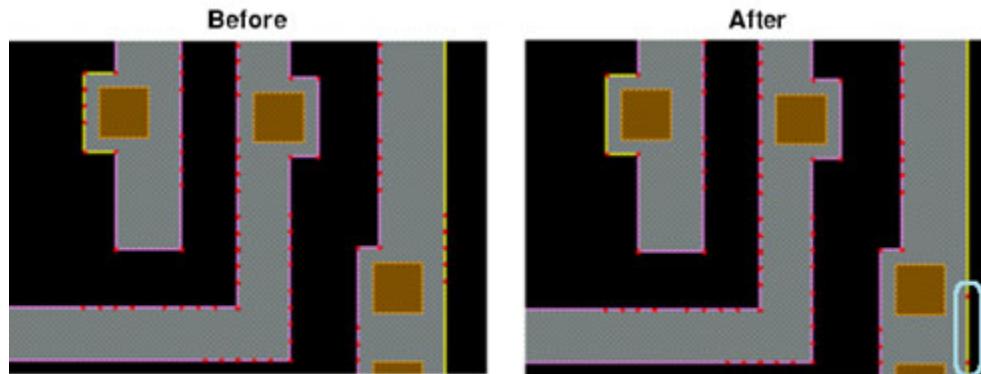


```
newTag dense -how distance all all 0.001 0.5 -edgeBased -from outside -to outside
newTag iso -how subtractTags all dense
fragmentTag iso
```

You can also write the command so that it adds new fragmentation after this “unfragmenting”. The syntax for this is:

```
fragmentTag tagname -first {df1 df2 ...} -last {dl1 dl2 ...} -unfragment
```

Figure 5-18. Unfragmenting With Re-fragmentation



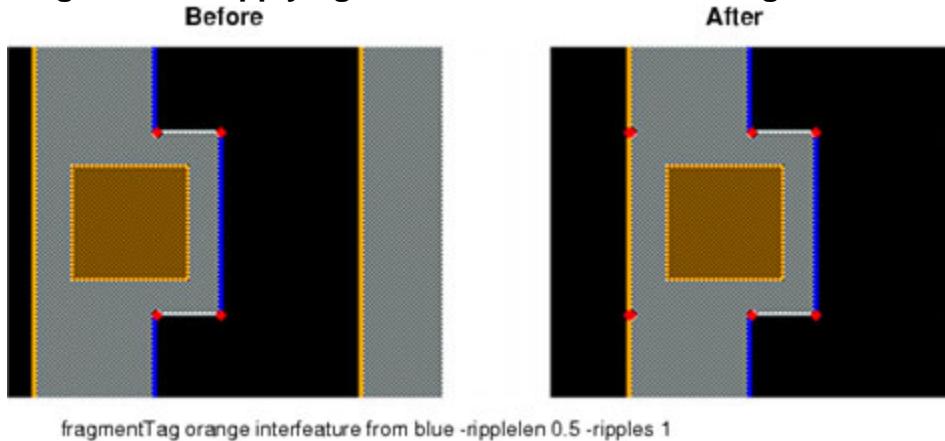
```
newTag dense -how distance all all 0.001 0.5 -edgeBased -from outside -to outside
newTag iso -how subtractTags all dense
fragmentTag iso -first 0.4 0.4 -last 0.4 0.4 -unfragment
```

Applying Additional Interfeature, Island-Layer, and maxedgelength Fragmentation

You can use fragmentTag to add additional fragmentation vertices by performing interfeature-type, island-layer-type, or maxedgelength-type fragmentation.

- Interfeature fragmentation is initiated by projecting corners from the source tag(s) onto fragments belonging to the tag being fragmented, as shown in [Figure 5-19](#).

Figure 5-19. Applying Additional Interfeature Fragmentation



```
fragmentTag orange interfeature from blue -ripplelen 0.5 -ripples 1
```

- Island layer-type fragmentation inserts vertices where the edges on an opc layer or correction layer *intersect* edges or vertices on an island layer.
- Maxedgelength-type fragmentation breaks long fragments into shorter fragments.

Note

 When performing additional interfeature-type, island layer-type, or maxedgelenngth-type fragmentation, you can specify different fragmentation control values than were used during the original fragmentation.

Special Case Fragmentation Issues

When writing tagging scripts containing fragmentTag, consider the following issues:

Fragmentation Order

When using fragmentTag, fragmentation is performed in the following order:

1. General fragmentation, in the following order:
 - a. Preprocess the layers (internally controlled)
 - b. Fragment opc and correction layers where they intersect island layers
 - c. Fragment opc and correction layers where they intersect visible layers
 - d. Apply intrafeature fragmentation
 - e. Apply interfeature fragmentation
 - f. Apply maxedgelenngth fragmentation
 - g. Perform implicit fragmentation due to hierarchy
2. Custom fragmentation (fragmentTag operations), in the order present in the setup file.

fragmentTag Impacts Upon Tagging

When the LITHO operation encounters a fragmentTag command, it performs three steps:

1. Processes the fragmentTag command
2. Removes this fragmentTag operation from the script that is resident in memory
3. Re-evaluates the tagging script from the beginning

This process, which is triggered by the fragmentTag keyword, means some tagging operations are evaluated multiple times. For more information, refer to the section “[Tagging](#)” in this chapter for more information.

Multilayer Fragmentation (fragmentLayer)

In a multilayer mask, global fragmentation settings apply to all layers by default. Typically, your fragmentation schemes are set as global defaults in the setup file or SVRF rule file and apply to all layers.

However, if you want to create customized fragmentation for each layer or a set of specified layers, you can use a special command block, `fragmentLayer`, to change a default fragmentation setting specific for a particular layer.

The `fragmentLayer` block allows greater control over fragmentation, enabling you to make changes to parameters for specific layers rather than resetting global defaults. Further benefits include:

- Enhanced control over interfeature fragmentation
- Separate controls for interfeature fragmentation by concave corners
- Ripple vertices with island layer fragmentation
- Different fragmentation schemes for different layers
- Can be used in conjunction with global fragmentation and special case fragmentation (`fragmentTag`) on each layer

A `fragmentLayer` block is defined in the “Arbitrary Section” of the setup file using the following syntax:

```
fragmentLayer layer { settings }
```

where

layer is the layer name whose fragmentation is defined by this block

settings is a sequence of lines containing fragmentation keywords with parameters enclosed by curly braces ({}). A single keyword is allowed per line. [Table 5-9](#) lists the fragmentation keywords allowed in a `fragmentLayer` block.

Table 5-9. Keywords Allowed in a fragmentLayer Block

| Keyword | Description |
|---|---|
| <code>coincidentLayer</code> <i>layer1</i> -overlay {0 1} | Modifies global interfeature and intrafeature fragmentation. This keyword can only be used within a <code>fragmentLayer</code> block. |
| <code>concavcorn</code> <i>length</i> ... [rem <i>r</i>] [non90 <i>distance</i> ... [rem <i>r</i>]] [sea <i>sea_distance</i> ... [rem <i>r</i>]] | Gives precise control over the number and length of intrafeature fragmented edges at concave corners (inside corners). |
| <code>cornedge</code> <i>length</i> ... [rem <i>r</i>] [<i>priority</i>] [non90 <i>distance</i> ... [rem <i>r</i>] [<i>priority</i>]] [lea <i>lea_distance</i> ... [rem <i>r</i>] [<i>priority</i>]] | Gives precise control over the number and length of intrafeature fragments created from edges at convex corners (outside corners) |
| <code>interfeatLayers</code> <i>layer1</i> [<i>layer2</i> ...] | Modifies global interfeature fragmentation settings. This keyword can only be used within a <code>fragmentLayer</code> block. |

Table 5-9. Keywords Allowed in a fragmentLayer Block (cont.)

| Keyword | Description |
|---|---|
| <code>interfeature [-num x] [-interdistance y] [-ripplelen z] [-rem r] [-shield n] [-distancePriority 0 1] [-ripplestyle 0 1] [-split 0 1]</code> | Defines the fragmentation parameters that influence interfeature fragmentation. |
| <code>intrafeature edge_spec [fragment] rule [corner2 [fragment] rule]</code> | An alternative syntax that combines both concavecorn and cornedge functionality to control fragments at corners. |
| <code>islandFragment [-ripples x] [-ripplelen z] [-rem r] [-enforce 0 1]</code> | Gives precise control over island layer fragmentation. |
| <code>maxedgelength maxedge [-split 0 1]</code> | Specify the maximum edge length. Longer edges are fragmented to less than this value. |
| <code>minedgelength minedge [enforce]</code> | Limits the minimum length of an edge that is created when fragmenting. |
| <code>minjog minjogsز</code> | Specifies minimum length for a jog. |
| <code>seriftype {0 1}</code> | Determines how line ends are fragmented during interfeature fragmentation. |
| <code>visibleLayers [layer1 layer2...]</code> | Modifies global visible and asraf layer fragmentation settings. This keyword can only be used within a fragmentLayer block. |

The following rules apply for fragmentLayer blocks:

- Layers must be type opc or correction.
- All fragmentLayer blocks must appear in the text portion of the setup file.
- Global fragmentation settings are defined outside of the fragmentLayer block.
- If an opc or correction layer does not have a fragmentLayer block defined for it, the global fragmentation settings apply to it.
- You only need to set those parameters needed to override the global settings. The layer inherits the global fragmentation settings for any parameter not defined in the block.

Example 1

```
#-----Fragmentation-----
minfeature 0.06
modelpath /test
opticalmodel defaultopt
resistpolyfile resist.mod
interfeature -ripples 2 -ripllen 0.1 -interdistance 0.45
#-----Layers-----
background clear
layer 1 L1 17 0 opc dark
layer 2 L2 17 0 correction dark
#-----Arbitrary Commands-----
fragmentLayer 2 {
    interfeature -ripples 1
}
```

In this example, the following global fragmentation settings are defined in the “Fragmentation” section:

```
interfeature -ripples 2 -ripllen 0.1 -interdistance 0.45
```

This example sets up **interfeature** fragmentation with a 2-ripple maximum (-ripples option), a maximum ripple length of 0.1 microns (-ripllen option), and a maximum ripple distance of 0.45 microns to the next feature (-interdistance option).

The “Layers” section defines a correction layer 2 (using the **layer** keyword).

In the “Arbitrary Commands” section, a fragmentLayer block is created to modify the -ripples default specifically for layer 2. In this case, the -ripples option is set to 1 rather than the global setting 2, as shown in the following example (the underlined text indicates where a change occurred):

```
interfeature -ripples 1 -ripllen 0.1 -interdistance 0.45
```

Example 2

The following example shows fragmentation specified for edges coincident with layer L1. Both examples use the same setup file.

```
fragmentLayer L2 {
    coincidentLayer L1
}
fragmentLayer L3 {
    coincidentLayer L1
}
```

The following figure shows the results. The highlighted dots represent interfeature vertices created on the coincident edges.

Figure 5-20. Changing the Coincident Layer to Affect Fragmentation

Multilayer Fragmentation Issues - Managing Default Settings

If a setting is not specified within a particular fragmentLayer block, a default value is applied from global settings. The table specifies the default values applied for different keywords used in a fragmentLayer block.

Table 5-10. Default Values for a fragmentLayer Block

| Keyword | Default Values |
|-----------------|--|
| coincidentLayer | No default |
| concavcorn | Global concavcorn values |
| cornedge | Global cornedge values |
| intrafeature | Global corner values (same as concavcorn and cornedge) |
| interfeature | Global interfeature values |
| interfeatLayers | All opc and correction layers plus any visible and asraf layers with frag bit set to 128, prioritized in ascending order according to layer numbers ¹ |
| islandFragment | All island layers |
| maxedgelength | Global maxedgelength value |
| minedgelength | Global minedgelength value |
| minjog | Global minjog values |
| seriftype | Global seriftype values |
| visibleLayers | All visible and asraf layers |

1. Layer numbers are only used for prioritization by default, when interfeatLayers is not specified. When you specify interfeatLayers, prioritization is based on the order in which the layers are supplied.

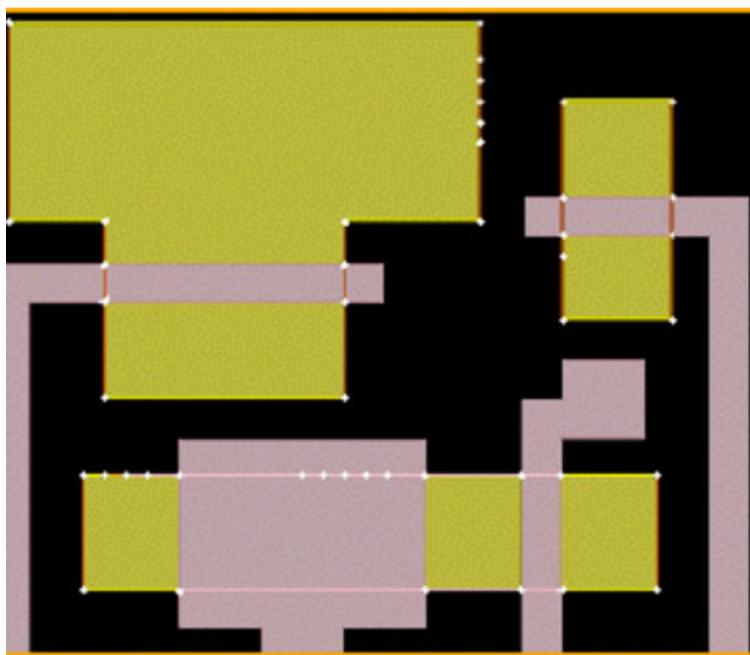
Viewing Fragmentation

Before investing time and resources in a full RET batch tool run, you should test your fragmentation settings to verify that they behave as expected. Use the Calibre WORKbench or Calibre LITHOview application to perform and plot fragmentation on a small, flat area of the layout. The **Frag** button in the RET Flow Tool lets you display fragmentation vertices as small circles without running OPC.

The following video shows how to load the fragmentation recipe and view the results.



Figure 5-21. Viewing Fragmentation in the Calibre WORKbench Application



Related Topics

[Inside Calibre WORKbench or Calibre LITHOview](#)

[Calibre WORKbench RET Flow Tool User's Manual](#)

Tagging

Once you fragment a polygon, you may only want 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. Tags are used in both the Calibre OPCpro and Calibre ORC processes.

| | |
|--|------------|
| Fragmentation Built-In Tags | 104 |
| Using Fragtype to Refine Fragmentation Settings | 105 |
| New Tags (newTag) | 107 |
| Methods to Control Edges With Tags | 108 |
| Debugging Tagging Scripts With Calibre WORKbench..... | 109 |

Fragmentation Built-In Tags

The RET batch tools provide several built-in tags. These are global tags, available for building custom tags.

The first group of tags contain edges that meet generic criteria.

Table 5-11. Global Built-In Tags

| | |
|-------------------------|-------------------|
| all | horizontal_line |
| all_angle_line | L (deprecated) |
| angle_45_line | line_end |
| concave_corner | line_end_adjacent |
| concave_corner_adjacent | opcable |
| convex_corner | vertical_line |
| convex_corner_adjacent | |

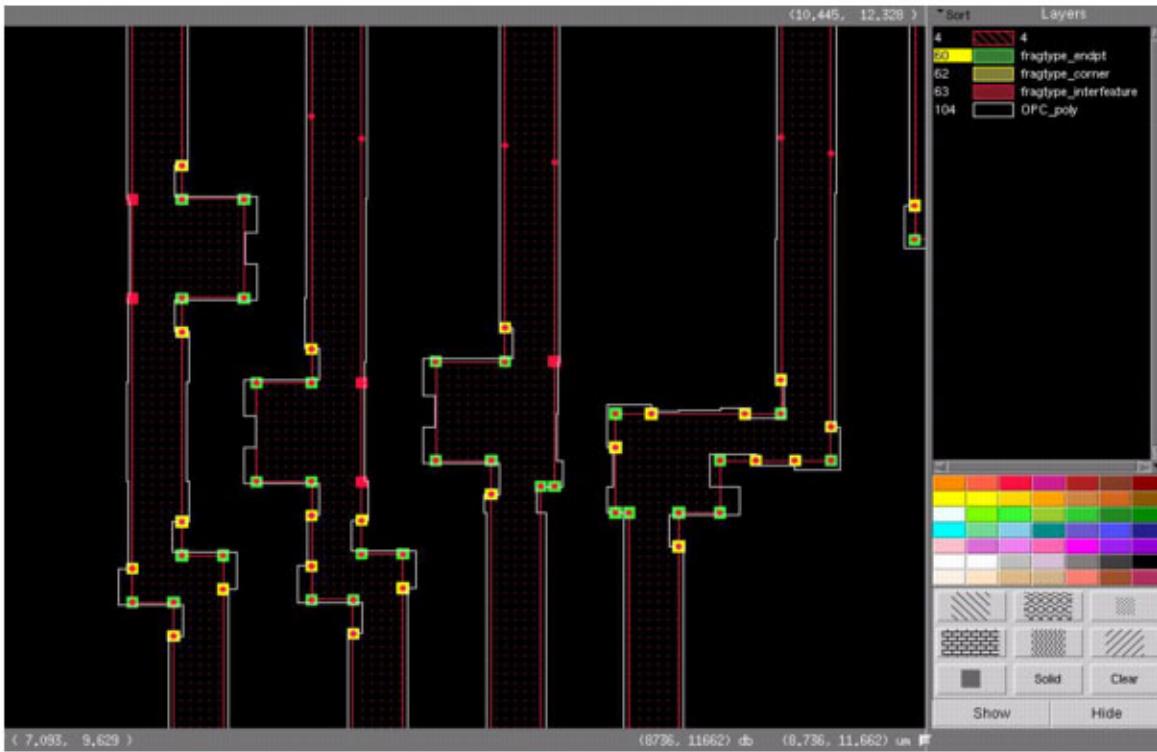
Tagging commands that start with “fragtype” help you to identify the causes of fragmentation.

Table 5-12. Fragtype Tagging Commands

| | |
|-----------------|-----------------------|
| fragtype_endpt | fragtype_builtin |
| fragtype_corner | fragtype_interfeature |
| fragtype_layer | fragtype_max |
| fragtype_tag | fragtype_other |

Using these built-in tagging commands in the setup file can help you identify where fragmentation breaks were created. Each fragmentation point is highlighted with a square box in the output of the layout (see [Figure 5-22](#)). They can be easily visible along with your existing fragmentation points for you to tune existing algorithms.

Figure 5-22. Fragtype Built-In Tagging Commands



Fragmentation ID tagging commands are different from using the **Frag** button in Calibre WORKbench, in that they aid in singling out selected operations and assist you in root cause analysis of multiple fragmentation sets that lie on the same edge. This family of tagging commands may not be helpful in less intensive fragmentation algorithms.

Using Fragtype to Refine Fragmentation Settings

One use case is to use the built-in tags when you have too much fragmentation on an edge. When viewing multiple breakpoints, it can be difficult to determine one set from another. These commands help you find out what caused them to be created. From there you can refine your fragmentation settings.

Prerequisites

- Existing layout file
- Existing DRC rule file
- Existing setup file

Procedure

1. Include the following code into your existing setup file and SVRF rule file.

In your setup file:

```
tags2boxes -tags fragtype_endpt -layers 60 -halfWidths 0.020 \
    -trimEnds 0.020 -markend
tags2boxes -tags fragtype_builtin -layers 61 -halfWidths 0.020 \
    -trimEnds 0.020 -markend
tags2boxes -tags fragtype_corner -layers 62 -halfWidths 0.020 \
    -trimEnds 0.020 -markend
tags2boxes -tags fragtype_interfeature -layers 63 -halfWidths 0.020 \
    -trimEnds 0.020 -markend
tags2boxes -tags fragtype_layer -layers 64 -halfWidths 0.020 \
    -trimEnds 0.020 -markend
tags2boxes -tags fragtype_max -layers 65 -halfWidths 0.020 \
    -trimEnds 0.020 -markend
tags2boxes -tags fragtype_tag -layers 66 -halfWidths 0.020 \
    -trimEnds 0.020 -markend
tags2boxes -tags fragtype_other -layers 67 -halfWidths 0.020 \
    -trimEnds 0.020 -markend
```

In your SVRF rule file:

```
fragtype_endpt = LITHO OPC FILE "setup/setup.in" POLY MAPNUMBER 60
fragtype_builtin = LITHO OPC FILE "setup/setup.in" POLY MAPNUMBER 61
fragtype_corner = LITHO OPC FILE "setup/setup.in" POLY MAPNUMBER 62
fragtype_interfeature = LITHO OPC FILE "setup/setup.in" POLY MAPNUMBER 63
fragtype_layer = LITHO OPC FILE "setup/setup.in" POLY MAPNUMBER 64
fragtype_max = LITHO OPC FILE "setup/setup.in" POLY MAPNUMBER 65
fragtype_tag = LITHO OPC FILE "setup/setup.in" POLY MAPNUMBER 66
fragtype_other = LITHO OPC FILE "setup/setup.in" POLY MAPNUMBER 67
```

2. Run the rule file that has this code in the setup file and open the output layout in Calibre WORKbench.

New layers with markers are created with the tags2boxes command.

3. Load the setup file in Calibre WORKbench and press the **Frag** button to see where existing fragmentation points are.
4. Single out layers in Calibre WORKbench to find out what aspect of your fragmentation needs refining. Consider changing the colors of each layer to separate one type of fragmentation from another.
5. Superimpose the original GDS or OASIS layer with the fragtype_endpt layer to see where the fragmentation breaks are in your original designs.

Results

Fragtype_corner identifies what caused the break in fragmentation to occur in convex and concave corners. These fragment breaks are created from the values specified in concavecorn and concave in the setup file.

Fragtype_max highlights the initial points of fragments created by the maxedgelength command. The maxedgelength command only fragments edges longer than twice the maxedgelength setting plus a remainder. (The remainder is controlled by -split.)

Related Topics

[Tagging Commands](#)

New Tags (newTag)

In most situations, built-in tags do not give sufficient control over the edges, which means you must create tags that are specific to your design. Use the newTag command to create custom tagging schemes to identify particular sets of edges to control.

The newTag syntax is as follows:

```
newTag tagName -how method arguments
```

The required arguments for newTag are

- **tagName**

A required argument specifying the name for the new tag.

- **-how method arguments**

A required secondary keyword followed by one of the tagging keyword described in “[Tagging Basics](#)” on page 597 plus additional arguments required by the method. The **-how method** describes the conditions a fragment must meet to be included in the output tag set.

The newTag command lets you create new tags by supplying a unique tag name or a description of the edges that belong to that tag. Descriptions of fragments that belong to a tag refer to existing tags, simulation data, or fragment properties such as length, angle, and corner type. You can create new tags using any of the following criteria:

- Distance from existing tags
- Edge length of existing tags
- Adjacency to existing tags
- Logical operations
- Aerial image constraints
- Edge placement error (EPE)

You may need to create several levels of tags (that is, tags based on tags based on other tags) to achieve the granularity necessary. The following setup file excerpt identifies all the edges with edge placement errors outside the acceptable range that are also adjacent to an endcap.

Figure 5-23. Tag Lines From Setup File

```
# Find EPEs outside the acceptable range of 0.002 microns
newTag badEPE -how EPE all -0.002 0.002 -not

# Find end caps (fragments with two convex corners) less than
# 0.3 microns long
newTag endcap -how edge -TYPE1 CONVEX -TYPE2 CONVEX -LENGTH < 0.3

# Find all fragments in "badEPE" within 0.3 - 0.9 microns of "endcap"
newTag EPE_near_endcap -how EXTERNAL >= 0.3 < 0.9 \
    INPUT1 m3 endcap INPUT2 m3 badEPE
```

You must write tag descriptions using the keyword syntax described in “[Tagging Commands](#)” on page 368.

As with fragmentation, it is important to test your tagging commands to ensure appropriate tagging of the edges. Calibre WORKbench allows you to interactively develop and test tag scripts on a small scale. For more information, refer to “[Debugging Tagging Scripts With Calibre WORKbench](#)” on page 109.

Methods to Control Edges With Tags

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 (rule-based or model-based).

The following command stops OPC processing for fragments tagged as “endcap”.

```
opcTag endcap -freeze
```

During ORC, you can count the number of edges with a specific tag, or you can create boxes around edges to make it easy to view them in Calibre WORKbench.

For a complete listing of fragmentation tags, as well as their usage and syntax, refer to the “[Tagging Commands](#)” section.

With the Calibre ORC batch tool, you use tagging commands to define conditions that are of special interest or concern. Like the Calibre OPCpro batch tool, this batch tool operates on edge fragments that result from fragmentation of polygon edges. The benefit of this is that you can identify exactly which portions of a polygon edge cause problems. A good fragmentation scheme breaks an edge into just enough fragments to ensure that portions of an edge with small errors or no errors are tagged differently than portions with larger errors. For more information on fragmentation, refer to “[Fragmentation Strategies](#)” on page 80.

The functional groupings and descriptions of the tagging commands can be found in the section “[Tagging Basics](#)” on page 597.

Debugging Tagging Scripts With Calibre WORKbench

In the Calibre RET toolset, a tagging script serves several purposes. The primary purposes are to help analyze EPEs when running the Calibre ORC batch tool and to perform OPC on selected edges only.

In order to debug tagging scripts, you need to see exactly which fragments meet each tag description. You do this by viewing the fragmentation with different tags displayed as a different color.

Prerequisites

- A layout file
- A Calibre OPCpro, ORC, or PRINTimage setup file with appropriate settings

Procedure

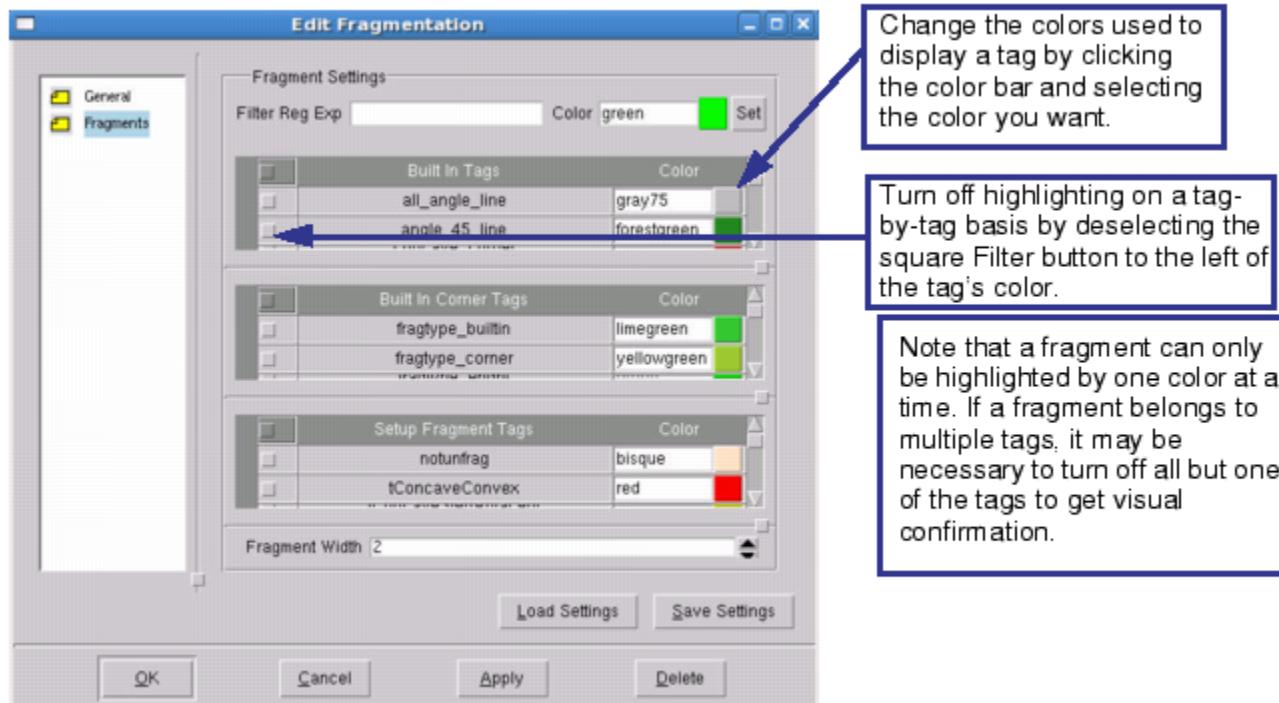
1. In Calibre WORKbench, select a small section of the layout by holding down the right mouse button and dragging diagonally from left to right.

Smaller areas are simulated more quickly. Select a large enough area that all significant features (long edges, line ends, space ends, and so on) are represented.
2. Load the setup file in the RET Flow Tool.
 - a. Select **Litho > RET Flow Tool**.
 - b. In the RET Flow Tool, select **File > New RET Flow**.
 - c. Right-click the flow icon and select **Add Existing Setup**.
 - d. Browse to your setup file. The navigator automatically filters for **.in** files; you may need to set it to show all files to see your setup file.
 - e. In the Setup Layers Mapping area at the bottom left of the RET Flow Tool window, set the layout layer number using the **WB# - Name** dropdown list for each setup file layer.
3. To fragment the layout, click the **Frag** button.

Thin lines mostly perpendicular to the feature edges appear in the layout. If you hide all layers but the selected one, you can see the thin lines intersect colored sections of the edges. The colored sections are the fragments. The colors represent one of the tags assigned to the fragment.
4. In the Calibre WORKbench main window, select **Edit > Fragmentation** to open the Edit Fragmentation dialog box.

The box shows the colors used to display each of the built-in tags, plus any user-defined tags. The initially displayed pane controls the general fragment properties, such as the dots indicating fragment endpoints.

To access the tag-specific settings, click **Fragments** on the left of the Edit Fragmentation dialog box.



Note

- Fragments often have many different tags assigned to them. Because a fragment can be displayed in only one color at a time, it may belong to a tag that is “shown,” yet not displayed, with that color. To see whether or not the fragment belongs to a particular tag, you must hide all colors that also highlight that fragment.

5. If you are dissatisfied with the tags applied to fragments, adjust the setup file in the RET Flow Tool and then click **Update** (above the Setup Layers Mapping) to make the changes to the setup in memory. Click **Frag** again to run the updated script.

The Calibre RET toolset supports special tagging options that make it easier for you to organize ORC data and visually inspect EPEs in the design.

- **newTag -how EPE** — Tags fragments with EPEs in the specified range.
- **newTag -how epeSensitivity** — Tags fragments with EPEs having a sensitivity to process variation that is within the specified range.
- **tags2boxes** — Generates a new layer containing rectangles drawn around each EPE region.

You can use the Calibre WORKbench or Calibre LITHOview application to test and debug any of these tagging options except tags2boxes, which requires the Calibre RET batch tools.

6. When you are satisfied with your tagging script, in the RET Flow Tool select **File > Save Setup File**.

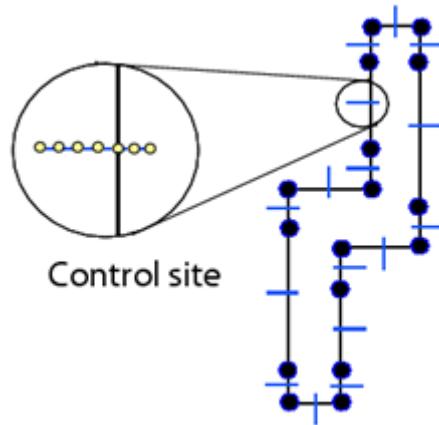
Related Topics

[Tagging Basics](#)

Control Sites

Control sites are locations on the opc layer where simulation data is evaluated for edge placement errors (EPE). Calibre RET batch tools use the information from the control sites to assess the need for OPC, to evaluate the success of an OPC run, and to generate the simulated wafer image.

Figure 5-24. Control Sites on Fragmented Polygon



Each control site consists of a line containing a set of control points. During simulation, the image intensity is calculated at each of the control points.

Several site-specific factors affect the information extracted from a control site:

- The location.
- The type of site.
- The number and spacing of control points on the site.
- How the site is aligned on the fragment.

| | |
|-------------------------------------|------------|
| Control Sites and EPE | 112 |
| Control Site Keywords | 113 |
| Adjusting Sites | 114 |
| Site Styles for Corners..... | 115 |

Control Sites and EPE

Sites are used to determine the simulated edge placement error (EPE), or the distance between the drawn edge location on the original design and the simulated edge location.

EPE is the fundamental quantity used by all RET batch tools:

- The objective of an OPCpro run is to minimize the EPE so that the simulated design closely matches the original design. It takes into account the behavior of the resist surface when reacting to light exposure.
- ORC checks the EPE-adjusted simulation to make sure that it does not violate the design rules set for the original design.
- PRINTimage displays a simulation of the EPE-adjusted edges.

The Calibre RET batch tools calculate EPEs by simulating the image intensity at predefined control sites, calculating a printing threshold, then measuring the distance between the contour and the threshold. As control sites are the locations on the simulated wafer where simulation data is evaluated, both to generate the aerial image contours and to check for edge placement errors, accuracy at your sample point is critical to create an accurate simulation model.

Control Site Keywords

You control the locations of the sites through fragmentation, the layer keyword in the setup file, and proper layer usage. Sites are automatically generated when fragmentation occurs. By default, the Calibre model-based RET batch tools create one control site on every edge fragment on the opc layer except jog fragments.

Table 5-13. Keywords Used to Control Simulation Sites

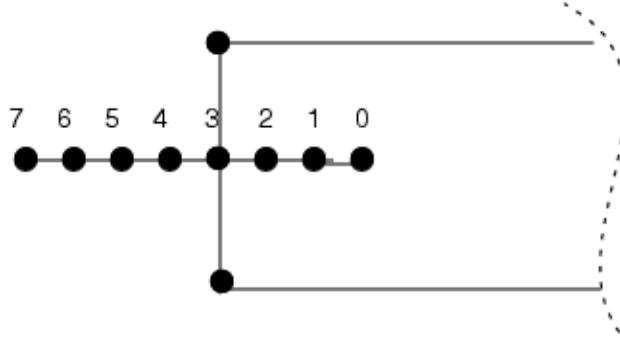
| Site Property | Keywords |
|-------------------------------------|--|
| Locations of the sites | layer plus fragmentation parameters |
| Type of sites | siteinfo |
| Number of the control points | siteinfo |
| Spacing of the control points | siteinfo |
| Site Center relative to figure edge | siteinfo |
| Alignment of sites | cornerSiteStyle |
| Rotational alignment | cornerRadius |
| Remove sites | siteControl DELETE |
| Move site by an offset | siteControl OFFSET_FROM |
| Smooth edges | siteControl SMOOTH |
| Move to a specified corner | siteControl SHIFT_TOWARD |

Adjusting Sites

Sites are divided into discrete points, starting at point 0 (sites number from the interior of the polygon outward). You can change the number of points, the angle at which the control site intersects the fragment, and how far the control site extends to each side.

Figure 5-25 shows a site with eight discrete points, where the center of the site (point 3), is on the edge of the polygon.

Figure 5-25. Example Control Site



If the default setting on a particular site does not yield accurate EPE and simulation results, you may need to adjust the site precision. You can do so by changing the number of points, angle, and center point (where the polygon edge falls) using the siteinfo keyword. The basic syntax is as follows:

```
siteinfo type [-num x] [-spacing y] [-center z]
```

The type you use depends on the type of process model specified in the setup parameters. Each type defines default values for number, spacing, and alignment of control points. Optional arguments let you override these defaults. The three default types are:

- **AERIAL** — Used when a constant threshold model is specified. A constant threshold places the edge on a single fixed contour level. It sets the options -num 3 -spacing 0.04 -center 1.
- **RESIST** — Used with a variable threshold model. A variable threshold model uses intensity and slope to calculate edge placement. It sets -num 12 -spacing 0.04 -center 3.
- **SAMPLE** — Used only when creating a variable threshold model. It sets -num 20 -spacing 0.04 -center 10.

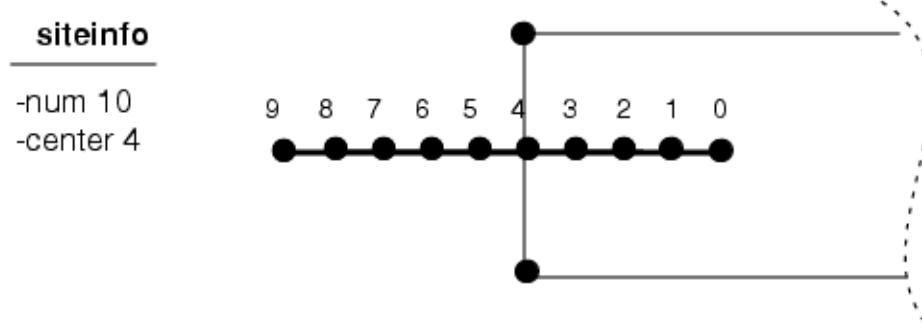
If you choose to manually reset them, then the following values need to be specified:

- **-num** is the number of points on the site.
- **-center** sets which points (numbered from 0, starting from the inside of the polygon) to place on the polygon edge.

- **-spacing** sets spacing in between EPE points (the length of the site). The spacing must be an integer multiple of grid size, and cannot be different than the spacing used in the model creation. For example, the spacing of 0.04 um is too large for 0.13 um gridsize and below.

[Figure 5-26](#) shows how the siteinfo parameters affect control point position (in this case, the site was lengthened to 10 points and centered at point 4). The length of a control site should be sufficient to find the local I_{max} and I_{min} (maximum and minimum intensity).

Figure 5-26. Site Placement With the siteinfo Keyword

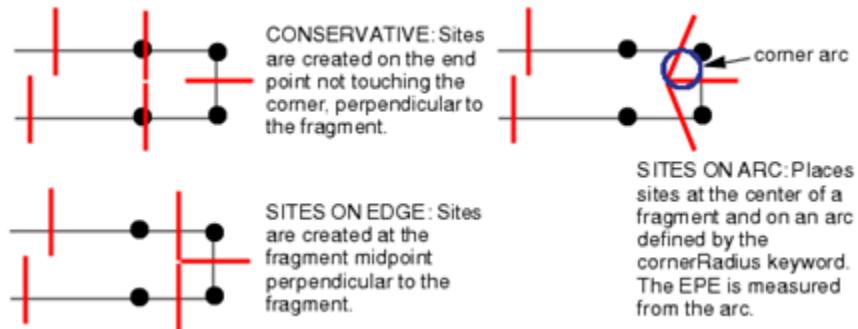


Site Styles for Corners

By default, a site is placed perpendicular at the midpoint of a fragment. However, at corners this technique potentially gives far less precise results. You can get better simulation results by adjusting the position or angle of the site relative to its fragment.

There are three basic corner site placement “styles”: conservative, sites on arc, and sites on edge, as illustrated in [Figure 5-27](#).

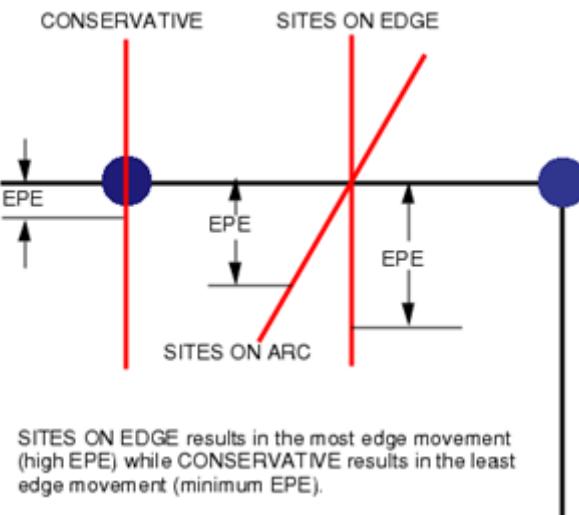
Figure 5-27. Corner Site Styles



Your choice of style will result in different EPE results, as EPE is measured at a different point for each style.

You specify the style you want using the [cornerSiteStyle](#) keyword. This keyword is required in the setup file. These styles only apply to the placement of sites on fragments adjacent to a corner. The [cornerRadius](#) keyword allows you to change the rotation of sites for the [SITES_ON_ARC](#) style.

Figure 5-28. Site Styles and EPE

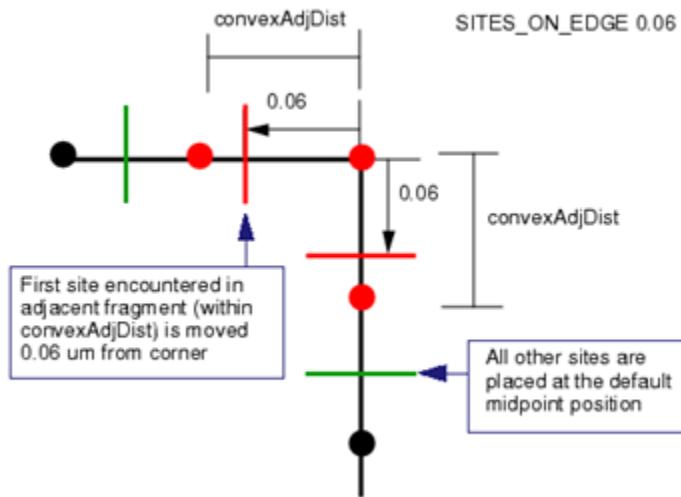


Corner Distances

In addition to the three default corner site styles, you can specify a particular distance from the corner. Use the distance option to specify the exact location of the site center (the distance from the corner), depending on the site style you set as default (see [Figure 5-29](#) and [Figure 5-30](#)).

For [SITES_ON_EDGE](#), the site passes through the fragment at exactly this distance from the corner. When you specify a distance for [SITES_ON_EDGE](#), only the first site encountered within the corner distance parameter ([convexAdjDist](#), [concaveAdjDist](#), and [lineEndAdjDist](#)) is affected. All other sites are placed in their default positions.

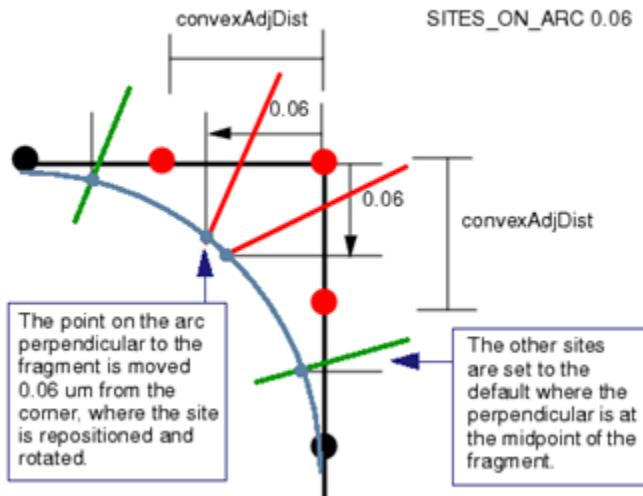
Figure 5-29. SITES_ON_EDGE Specified With Distance



For SITES_ON_EDGE, a point is set at a distance d from the corner that intersects perpendicular a point on an arc. The site is then placed orthogonal to the point on the arc (see Figure 5-30). You can determine the rotation of the site by using the `cornerRadius` keyword to change the radius of the arc.

As with SITES_ON_EDGE, if you specify a distance for SITES_ON_ARC, only the first site encountered within the corner distance parameter (`convexAdjDist`, `concaveAdjDist`, and `lineEndAdjDist`) is affected. All other sites are rotated as normal.

Figure 5-30. SITES_ON_ARC Specified With Distance



You cannot specify a distance for the CONSERVATIVE site style.

Custom Corner Settings

You can also override the default style for special-case corners (convex, concave, and line ends) using the following options:

- **concave** — Sets a site style mode specifically for concave corners.
- **lea** — Specifies a site style mode specifically for line-end adjacent fragments. You can specify either of the following:

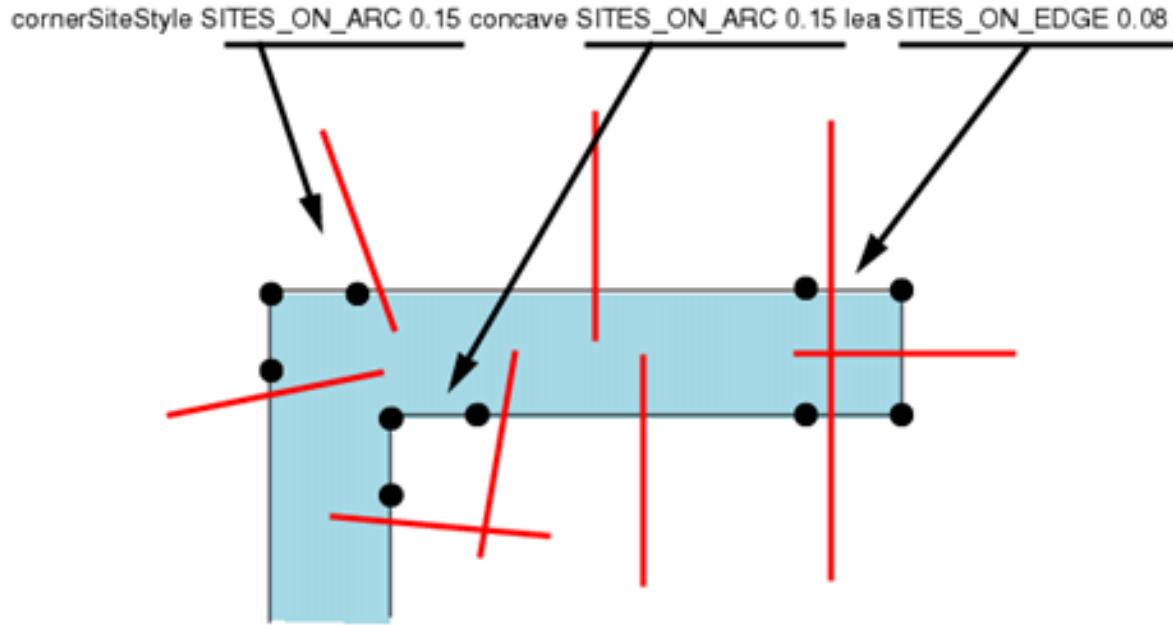
These options allow you to mix default site styles as in the following example:

```
cornerSiteStyle CONSERVATIVE concave SITES_ON_ARC lea SITES_on_EDGE
```

In this example, CONSERVATIVE is the default style, SITES_ON_ARC is set for concave corners, and SITES_ON_EDGE is set for line-end corners.

You can also set specific distances for different corners styles. [Figure 5-31](#) shows an example of setting separate modes and distances for concave corners and line-edge adjacent fragments.

Figure 5-31. Customizing Sites for Special Corners Example



Site Orthogonalization and Image Gradient

Site orthogonalization ensures that a site is created at an angle normal to the aerial image itself, or in other words, along the image gradient.

An analogy for an image gradient is a contour-line topographical map, where contour lines that are closer together indicate a steeper incline or “gradient” than contour lines that are spread

further apart. In this case, the contour lines indicate intensity at different points (as in an aerial image map).

Using the orthog option you can specify that “site orthogonalization” should be attempted for all sites. Each site is rotated until it matches the gradient, unless the gradient is too small or the slope is the wrong direction.

Corner Site Limitations

Using the corner_limit option, you can set further limits to your site placement, setting specific actions to occur based on a particular type of offset.

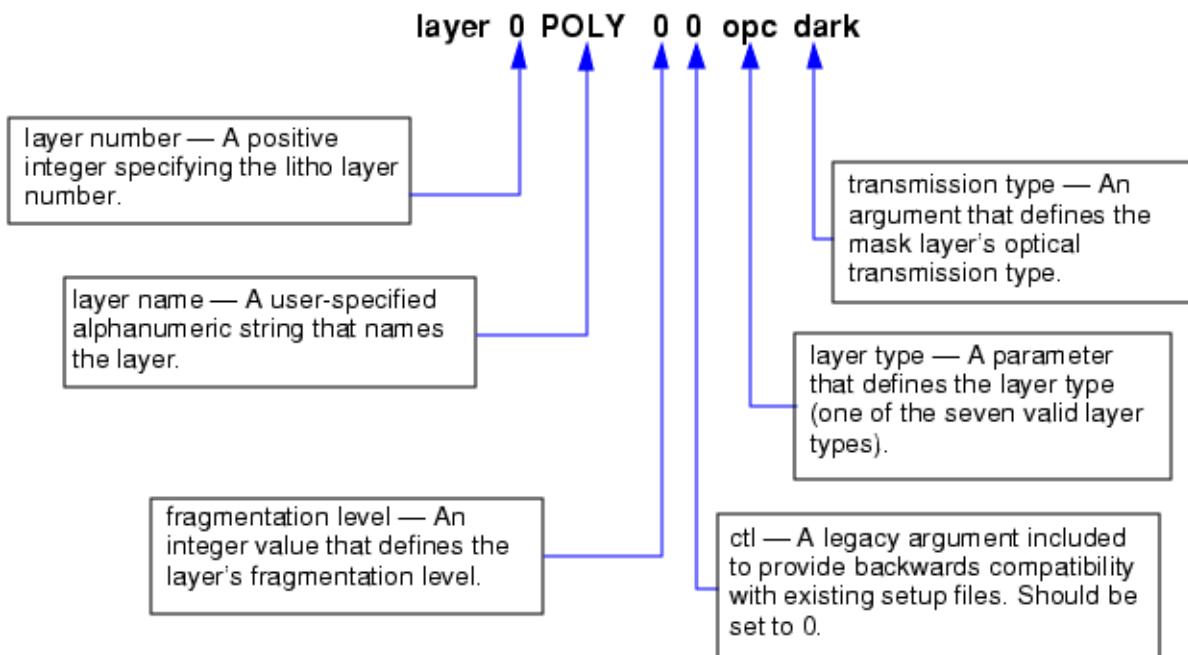
Layer Usage

The Calibre RET batch tools operate on Calibre polygon layers. The opc layer containing the image you want to print on your wafer is the minimum layout data required to run the Calibre OPCpro batch tool. This minimum data is sometimes not the only data needed. You can improve overall results by providing additional layer data to include non-printing simulation data, generate fragments, and control correction.

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

[Figure 5-32](#) shows how to read the arguments for a layer keyword. It uses the keyword from the previous example, which defined the type and layer number for the POLY layer.

Figure 5-32. Layer Keyword Arguments



You create the additional layers you pass to the OPCpro batch tool by manipulating the data in the original layout file prior to invoking the batch tool. You perform this manipulation using SVRF operations in the SVRF rule file, which is beyond the scope of this manual. However, refer to “[Introduction to SVRF Rule Files](#)” on page 47 for a basic introduction to using SVRF.

| | |
|--------------------------------------|------------|
| Layer Types | 121 |
| Additional Fragmentation..... | 124 |

| | |
|-------------------------------|-----|
| Controlling Correction | 125 |
| Layer Names and Numbers | 126 |

Layer Types

There are seven layer types, each of which is described in the following sections. These layer types have meaning only with respect to the RET toolset.

Table 5-14 provides a brief summary of how the Calibre OPCpro batch tool interprets each of the RET layer types.

Table 5-14. How RET Layer Types Affect OPC

| Layer Type | Usage | Corrected? | Optically Visible? |
|------------|--|--|--|
| opc | Serves as the target—the design that you want to print on the wafer. If no correction layers are present, this layer has its edges moved to correct for process effects. | Yes, if no correction layers are present | Yes, if no correction layers are present |
| correction | Contains geometries to be corrected when using multiple masks. | Yes | Yes |
| visible | Contains mask geometries that are not corrected but serve as input to the optical model computations. Also used to insert fragmentation vertices and identify false edges. | No | Yes |
| asraf | Contains mask geometries (generally negative SRAFs) that are not corrected, but serve as a negative input to the optical model computations. | No | Yes |
| island | Contains geometries used to insert fragmentation vertices and tag edges. | No | No |
| hidden | Not used for OPC. | No | No |
| external | Not used for OPC. | No | Yes |

Opc Layer

The opc layer serves as the target layer for LITHO operations. That is, the shapes on this type of layer represent the actual geometry that you intend to transfer onto wafer. Opc layers are the only layers on which you can create control sites, and all EPEs are calculated with respect to these layers.

During processing, an opc layer is modified as follows:

- Its edges are fragmented.
- Control sites are installed on its edge fragments and EPEs are evaluated at those sites.
- Its edges are moved during OPC, unless correction layers are present.
- It is interpreted as representing the final mask, unless external layers are present.

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.

Correction Layer

Correction layers are special layers whose edges are moved during OPC to compensate for optical and processing errors. 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 OPCpro batch tool or Calibre WORKbench application.

When you pass a correction layer to the LITHO OPC operation, the batch tool maps each edge fragment on the correction layer to a corresponding edge fragment on the opc layer and moves the layer to generate the corrected mask design.

During processing, a correction layer has:

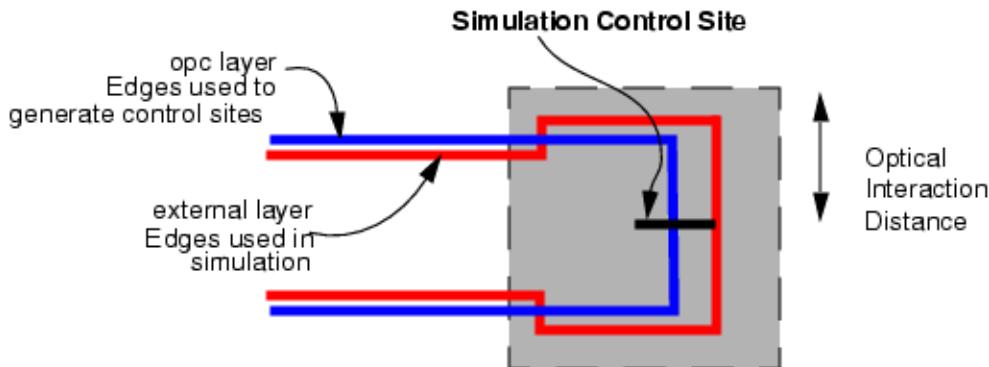
- Its edges fragmented.
- Its edge fragments moved during OPC.

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.

External Layer

External layers can only be used with the Calibre ORC and Calibre PRINTimage batch tools. External layers contain corrected mask data, and their presence indicates to the batch tool that the opc layer's geometries represent the target design rather than the mask design itself. When simulating areas in which visible layers and external layers overlap, the tools add the mask transmission values for the overlapping areas.

As mentioned previously, opc layers are the only layers on which the batch tools create control sites. Consequently, when you use an external layer for simulation, the EPEs it generates are computed relative to the opc layer for the run. [Figure 5-33](#) shows the relationship between the opc layer, the external layer, and the control sites used for simulation during ORC.

Figure 5-33. Using External Layers for ORC

During processing, an external layer:

- Has its edges evaluated by Calibre ORC with respect to the EPEs on the opc layer.
- Has its mask image generated by Calibre PRINTimage when passed to that application.
- Causes opc layers to be ignored by the simulator.

External layers are not required. If you do not pass an external layer to the batch tool, the opc layer is used for simulation.

Visible Layer

Visible layers contain geometries that do not receive correction yet are optically visible—that is, they appear on the mask and interact with other geometries to affect the final printed design. Visible layers are not fragmented and do not receive any sites. Thus, data on visible layers is used by the following:

- **Optical model simulations** — Optical models simulate aerial images at every point on the image grid, factoring in all data on visible and opc or correction layers.
- **Process model simulations** — Process models simulate printed images based on measurements taken at simulation sites. Since visible layers do not contain simulation sites, process simulations reflect visible layer data only if it is within the optical radius of opc geometry. This fact is most relevant to the use of VT5 models with densities because such models are sensitive to the density of data containing sites.

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. However, as with island layers, you can configure them using the frag argument on the layer keyword so that they cause fragmentation to occur and generate vertices wherever edges or vertices on the visible layer intersect geometries on the opc and correction layers.

Visible layers are not required.

Asraf Layer

Asraf layers are very similar to visible layers. The primary difference is that geometries on asraf layers are subject to MRC checks.

The geometries (typically anti-scattering bars) on asraf layers are not fragmented and do not receive sites, but do appear on the mask. By default, asraf layers do not induce fragmentation, but can be configured to. The geometries on asraf layers “knock out” or cancel sections of the opc layer.

Island Layer

Island layers are used for two important roles:

- **Fragmentation** — Island-layer fragmentation generates fragmentation vertices wherever edges or vertices on this type of layer intersect geometries on the opc or correction layers. This is the default behavior. If you set the frag bit for an island layer to 32 (layer keyword in setup file) the layer does not cause fragmentation to occur. Island layers are not required. If you do not pass any island layers to the batch, island layer fragmentation does not occur.
- **Tagging** — Island layers also work with certain tagging commands to identify fragments. You can use island layers to generate tags, regardless of whether they are used for fragmentation purposes.

Hidden Layer

By default, any layer not specified in the setup file is marked as a hidden layer. The data on hidden layers do not affect the outcome of a batch tool run. Hidden layers serve an important function in the Calibre WORKbench and Calibre LITHOview applications by providing a way to return data that should not be factored into a simulation.

Note

 Never pass hidden layers into any LITHO operation in the SVRF. That is, do not include the names of any hidden layers in the list of layers included in the LITHO operation.

If you include a layer defined to be of type hidden in a setup file, the layer must have a legal transmission value or it causes a setup file error.

Additional Fragmentation

The Calibre RET batch tools begin by fragmenting the edges on the opc layer and correction layers. Most fragmentation occurs based on the fragmentation parameters defined in the setup file. You can force the system to create additional fragmentation vertices using island, asraf, and visible layer geometries. This type of fragmentation generates vertices wherever edges or vertices on these types of layers intersect geometries on the opc and correction layers.

Use the *frag* option on the layer keyword to control whether or not the batch tool performs this type of fragmentation.

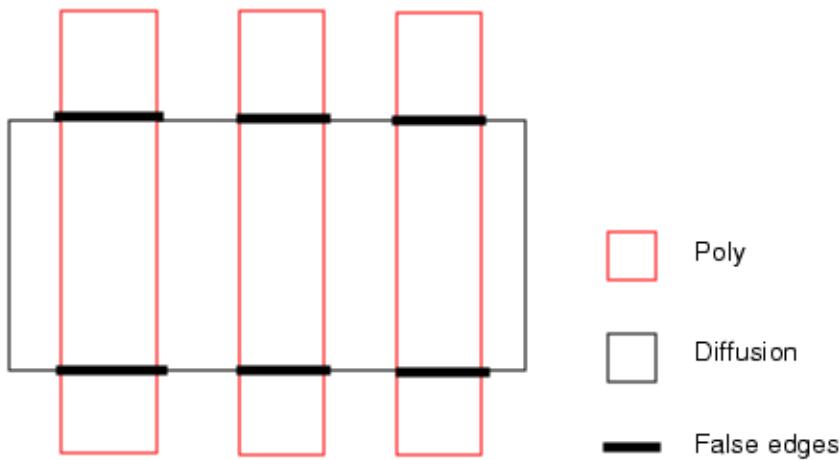
Controlling Correction

You can use the layers you pass to the tool to classify edges into related sets. Once classified, you control which edges receive correction, and how much correction they receive.

- **Topological Tagging** — Most tagging commands let you build tags by combining existing tags or evaluating fragments with respect to other fragments on the same layer. Topological tagging commands, such as insideEdge, TouchEdge, coincidentInsideEdge, and notOutsideEdge, let you build tags based on geometries on another layer. These commands take two arguments: a tag layer (either the opc layer or a correction layer) and a marker layer. They tag all fragments on the tag layer that meet the condition (inside, outside, not inside, or not outside) with respect to the marker layer.
- **False Edges** — Many people use SVRF operations to create derived layers by ANDing two existing layers. The purpose of this is usually to break polygons into two sets of polygons, those that require correction and those that do not. In order to accomplish this, the Calibre nmDRC hierarchical engine must create edges at the breaks. These system-created edges are called false edges because they did not exist in the original drawn layer. Typically these false edges should not have OPC performed on them. [Figure 5-34](#) shows the false edges formed when you use SVRF commands to create the GATE layer by ANDing the POLY and DIFF layers.

False edges on the opc layer that abut a visible layer do not receive control sites and do not move during OPC. You can use this feature to force the batch tools to ignore these false edges. In the case of [Figure 5-34](#), you can force the Calibre OPCpro batch tool to ignore the false edges by creating a visible layer with the POLY not GATE SVRF layer operation.

Figure 5-34. False Edges



Layer Names and Numbers

The setup file layer names must match the layers passed into the LITHO SVRF operation. The next few lines show an example for a layer called “POLY”.

From SVRF:

```
OPC_DATA = LITHO OPC POLY FILE setup.in
```

From the setup file:

```
layer 0 POLY 0 0 opc dark
```

The setup file layer numbers can be chosen arbitrarily. These layer numbers play a role when using concurrent operations to return multiple outputs from a single batch tool run. With concurrent operations, you specify the layer to return by referencing it with either the MAP or the MAPNUMBER keyword, as in the example below.

From SVRF:

```
Layer1 = LITHO ORC POLY FILE setup.in MAP t1
Layer2 = LITHO ORC POLY FILE setup.in MAP t2
```

From the setup file:

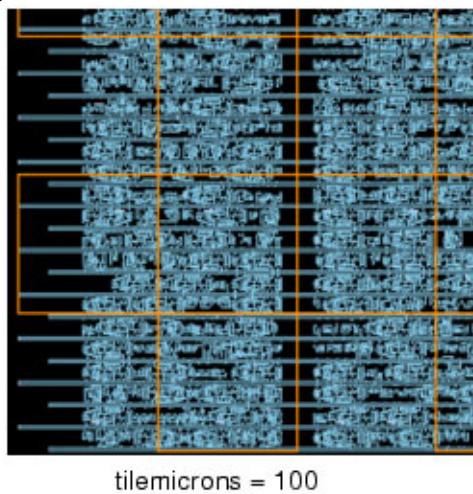
```
layer 0 POLY 0 0 opc dark
tags2boxes -tags t1 t2 -layers 1 2
```

Tiles and Tile Boundaries

A tile is a rectangular area of a design that is processed as a unit. Large designs or cells are usually divided into tiles to improve processing speed. When performing OPC, Calibre OPCpro and Calibre WORKbench process the data in chunks, called tiles. If a cell is smaller than the specified tile size, it is processed as one tile. If a cell is larger than the tile size, it is divided into square tiles extending out from the lower left corner of the cell or layout.

A tile boundary is the edge of a tile, where one tile abuts another tile. If a feature is near the edge of a tile, the software must factor in interactions with neighboring tiles. In certain cases, this can affect OPC results.

[Figure 5-35](#) shows example tile boundaries for a layout.

Figure 5-35. Tile Boundaries Example

Tile boundaries are typically used when troubleshooting your layout. If your OPC results are not what you expect, one of the first things to check is the relationship between the unexpected results and the tile boundaries.

You can change the tile size using the tilemicrons keyword in the setup file.

To Generate the New Output Layer

1. Include the keyword outLayer and its arguments in the setup file.
2. Use concurrent operations to return the results of processing plus the new tile boundary output layer. This requires including the LITHO operation in your SVRF rule deck at least two times. Both operations must include the MAPNUMBER or MAP keywords.
 - o Returning the tile boundary layer plus the OPC results:
 - The operation that returns the result should contain MAP or MAPNUMBER with the name or number of the opc or correction layer. Refer to “[Returning the Boxes Layers as Output](#)” on page 608 for further information on the use of MAP and MAPNUMBER.
 - The operation that returns the tile boundary output layer must contain the MAP or MAPNUMBER with the name or number defined using the outLayer keyword.
 - o Returning the tile boundary layer plus the ORC results:
 - The operations that return the tags2boxes layers should contain MAP or MAPNUMBER with the names or numbers of the tags2boxes layers.
 - The operation that returns the tile boundary output layer must contain the MAP or MAPNUMBER with the name or number defined using the outLayer keyword.

- Returning the tile boundary layer plus the PRINTImage results:
 - For returning a single image, the operation that returns the result should contain MAP or MAPNUMBER with the name or number of the opc or correction layer. For returning multiple contour levels on separate layers, MAPNUMBERS reflect the order in which the contours are defined. Refer to “[Calibre PRINTImage Procedures and Examples](#)” in this manual.

Impact of Promotion Tiling on the OPC Process

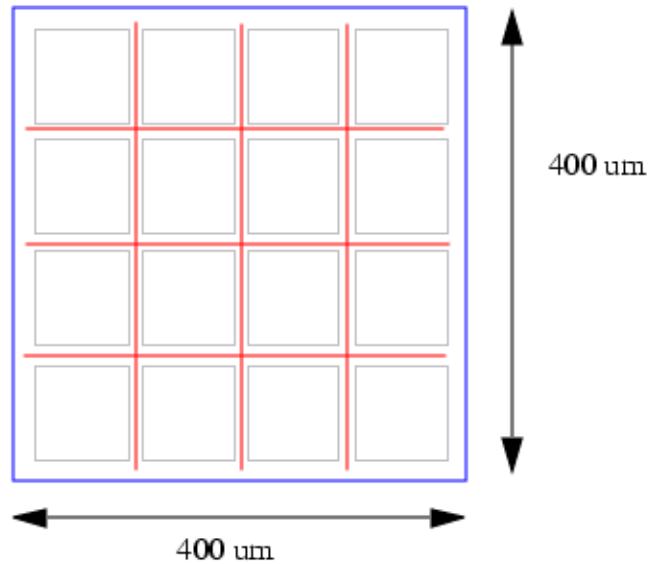
[LITHO_PROMOTION_TILING](#) creates a hierarchy of tiles. For example, if you had a 1.6 mm x 1.6 mm cell, and tilemicrons is set to 100 um in your setup file, you would get tiling that consists of:

- 16 x 16 tiles (lowest level)
- 4 x 4 tiles (mid level)
- 2 x 2 tiles (almost top level)
- 1 tile (top level)

The lowest level tiles (16 x 16 array) contain most of the data. The exception is the data that is at the border of each tile. (These border areas are not operated on.) These border regions are promoted to the next level up in the hierarchy (the 4 x 4 level). This mid level then keeps most of that data, but promotes the data that is at the boundaries of the 4 x 4 tiles. This occurs again at the 2 x 2 level, and so on.

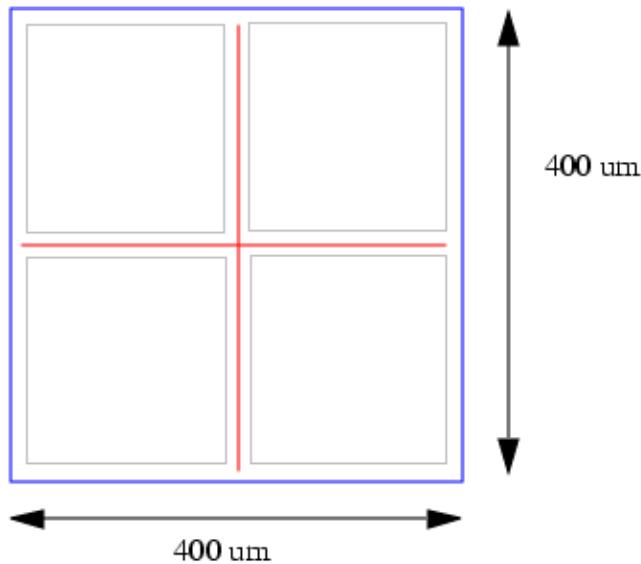
Assume we have a 400 um x 400 um area to be corrected by OPC. This is divided into 16 equal tiles of 100 um x 100 um (see [Figure 5-36](#)).

Figure 5-36. 4 x 4 Tiled Areas



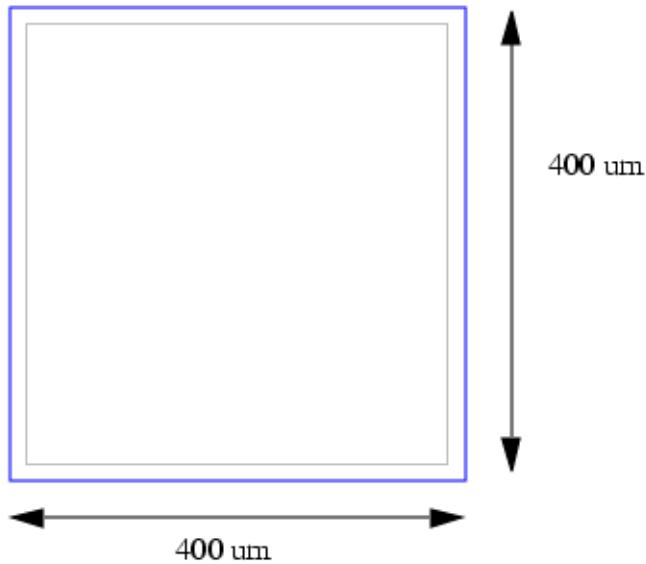
In [Figure 5-36](#), the design (the box in blue) is divided into 4×4 tiles. OPC is performed inside each of these tiles (gray boxed area) but the edges marked with red boundary lines are not processed.

Figure 5-37. Next Level of Tiling



In [Figure 5-37](#), the next level of tiling, the area is divided into 2×2 tiles. OPC is now done on the areas missed during the previous step, but the edges of the 2×2 squares in this level marked by red lines in [Figure 5-37](#) are not performed.

Figure 5-38. Top Level of Tiling



In [Figure 5-38](#), the topmost level of tiling, OPC is done on the whole square. The areas in which OPC was not performed in the second level of tiling are done in this level. OPC is never done twice on the same area; the final OPC is the result of tiling merged together.

A smaller tile size provides more scalability; that is, distribution to many processors is possible and the overlap regions get better correction because it is processed several times. See “[Recommended Settings for Best Results](#)” on page 631 for tilemicrons recommendations by technology node.

Chapter 6

Sparse OPC Command Reference

The sparse OPC commands include both SVRF command and commands that appear in setup files. These commands control Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

This chapter contains the following sections:

| | |
|--|------------|
| SVRF Commands | 136 |
| LITHO FILE | 137 |
| Litho Setup File Format | 138 |
| LITHO OPC | 146 |
| LITHO ORC | 148 |
| LITHO PRINTIMAGE | 150 |
| Litho Setup File Keywords | 152 |
| background | 155 |
| coincidentLayer | 158 |
| concaveAdjDist | 160 |
| concavecorn | 161 |
| convexAdjDist | 163 |
| cornedge | 164 |
| cornerRadius | 167 |
| cornerSiteStyle | 169 |
| flaglayer | 174 |
| fragmentLayer | 176 |
| gridsize | 179 |
| gse | 180 |
| inline_optical1 | 181 |
| inline_optical2 | 182 |
| inline_optical3 | 183 |
| inline_resist | 184 |
| interfeatLayers | 186 |
| interfeature | 187 |
| intrafeature | 197 |
| islandFragment | 201 |
| iterations | 206 |
| layer | 207 |
| lineEndAdjDist | 214 |
| lineEndLength | 215 |
| maxedgelength | 216 |
| minedgelength | 221 |
| minfeature | 222 |

| | |
|------------------------------------|------------|
| minjog | 223 |
| modelpath | 224 |
| movelimit | 225 |
| MRC_RULE | 226 |
| opcIter | 232 |
| optical_model | 241 |
| opticalmodel | 243 |
| outlayer | 244 |
| processModel | 246 |
| progversion | 248 |
| resistpolyfile | 250 |
| secondOpticalmodel | 251 |
| seriftype | 252 |
| set_kernel_count | 253 |
| set_resist_model | 255 |
| siteinfo | 257 |
| sse | 259 |
| stepsize | 260 |
| svrf_var | 261 |
| thirdOpticalmodel | 263 |
| tilemicrons | 265 |
| visibleLayers | 266 |
| Litho Variables..... | 267 |
| ALLOW_SMALL_MOVE | 273 |
| DISALLOW_SHALLOW_ANGLE | 274 |
| DONT_MOVE_ALL_ANGLE_EDGES | 276 |
| FRAG_BREAK_IN_HALF | 278 |
| GRIDS_FOR_ANGLE45 | 280 |
| INTENSITY_INTERP_CENTER | 281 |
| INTERACTION_DISTANCE | 282 |
| LITHO_ALLOW_MAP_BY_ORDER | 284 |
| LITHO_ASYM_SOURCE_CELL_CLONE | 286 |
| LITHO_AUTO_PUSH | 288 |
| LITHO_CLEAN_OPIC | 289 |
| LITHO_CONTEXT_RELAX_MRC | 290 |
| LITHO_DEANGLE | 292 |
| LITHO_HISTOGRAM_OUTPUT_FILE | 294 |
| LITHO_IMPL_FRAG_LEN | 295 |
| LITHO_MASK_PRIORITY_MOVEMENT | 296 |
| LITHO_MOVE_CONTEXT | 297 |
| LITHO_MOVEMENT_PRIORITY_TAG | 298 |
| LITHO_MRC_CLEANUP_LIMIT | 299 |
| LITHO_MRC_MODE | 300 |
| LITHO_PRECISE_PROMOTION | 301 |
| LITHO_PRINT_LEVEL | 302 |

| | |
|--|-----|
| LITHO_PRIORITY_BASED_MOVEMENT | 304 |
| LITHO_PROMOTION_TILING | 306 |
| LITHO_REPORT_REMOTE_CPU_TIME | 308 |
| LITHO_SIDE_SEGMENT_FILTER | 309 |
| LITHO_SIMULATE_CONTEXT | 311 |
| LITHO_SIMULATE_CONTEXT_DISTANCE | 314 |
| LITHO_SIMULATE_CONTEXT_SITE_SHIFT | 315 |
| LITHO_SUPPRESS_INTERACTION_ERROR | 317 |
| LITHO_SUPPRESS_MAPNUMBER_ERROR | 318 |
| LITHO_TAG_TIMER | 319 |
| LITHO_TILE_SLIVER_SIZE | 320 |
| LITHO_TOLERATE_MINOR_SKEW | 322 |
| LITHO_WRITE_DEBUG_SVRF (shell variable only) | 323 |
| MASK_CHIP_CORNER | 327 |
| OPC_AUTO_STEPBY_THRESHOLD | 329 |
| OPC_CORR_SEARCH_DISTANCE | 330 |
| OPC_CROSS_MEEF_DEADBAND | 331 |
| OPC_DO_SKew_CHECK | 332 |
| OPC_ENCLOSURE_MIN_LENGTH | 333 |
| OPC_ENFORCE_FRAG_LAYER_ORDER | 334 |
| OPC_EPE_TOLERANCE_FRAC | 337 |
| OPC_FAST_MOVE | 338 |
| OPC_FEEDBACK | 339 |
| OPC_IGNORE_ENCLOSURE_AT_CORNER | 340 |
| OPC_LOW_MEEF_DEFAULT | 341 |
| OPC_LOW_MEEF_FREEZE_TAG | 342 |
| OPC_LOW_MEEF_LIMIT | 343 |
| OPC_LOW_MEEF_NEG_SCALE | 344 |
| OPC_LOW_MEEF_NEG_TAG | 345 |
| OPC_LOW_MEEF_POS_SCALE | 347 |
| OPC_LOW_MEEF_POS_TAG | 348 |
| OPC_MAX_ITER_MOVEMENT | 349 |
| OPC_MIN_ENCLOSURE_01 | 350 |
| OPC_MIN_ENCLOSURE_10 | 352 |
| OPC_MIN_SITE_ANGLE | 354 |
| OPC_NEAREST_ADJ | 355 |
| OPC_NEW_FRAG_ORDER | 356 |
| OPC_USE_LE_DEF | 357 |
| SEM_CONSIDER_SITE_OFFSET | 359 |
| SEM_DO_MORPH | 360 |
| SEM_MAX_NONPRINT_MOVE | 361 |
| SEM_MORPH_SIZE | 362 |
| SEM_PWL_JOT_SIZE | 363 |
| SEM_USE_PWL | 364 |
| SEM_USE_SUF_FRAG | 365 |

| | |
|---|------------|
| SEM_USE_SUF_SITES | 366 |
| VERBOSITY | 367 |
| Tagging Commands..... | 368 |
| bindModelToTag | 373 |
| clearTagScript | 374 |
| dump | 375 |
| epeToleranceTag | 379 |
| fastIter | 382 |
| fragalign | 384 |
| fragcoinc | 388 |
| fragmentTag | 389 |
| fragmentTag ... interfeature | 395 |
| fragmentTag ... intersection | 397 |
| fragmentTag ... maxedgelength | 403 |
| fragmentTag ... projection | 404 |
| fragmentTag ... split | 406 |
| histogram | 407 |
| jogsmooth | 409 |
| MATRIX_OPC | 412 |
| newTag | 426 |
| newTag ... -how ANDTAGS | 429 |
| newTag ... -how COINCIDENTEDGE | 430 |
| newTag ... -how COINCIDENTINSIDEEDGE | 432 |
| newTag ... -how COINCIDENTOUTSIDEEDGE | 434 |
| newTag ... -how DISPLACEMENT | 436 |
| newTag ... -how EDGE | 437 |
| newTag ... -how EPE | 446 |
| newTag ... -how EPESENSITIVITY | 449 |
| newTag ... -how EXTERNAL | 451 |
| newTag ... -how fragAfterEdge | 454 |
| newTag ... -how fragAfterFrag | 455 |
| newTag ... -how fragBeforeEdge | 456 |
| newTag ... -how fragBeforeFrag | 457 |
| newTag ... -how FRAGMENT | 458 |
| newTag ... -how fragNextToEdge | 462 |
| newTag ... -how fragNextToFrag | 463 |
| newTag ... -how HAS_SITE | 464 |
| newTag ... -how IMAGE | 465 |
| newTag ... -how INSIDEEDGE | 472 |
| newTag ... -how INTERNAL | 474 |
| newTag ... -how LAYERAND | 477 |
| newTag ... -how LAYERANDNOT | 479 |
| newTag ... -how loopWithFrag | 480 |
| newTag ... -how MRC | 482 |
| newTag ... -how NOTCOINCIDENTEDGE | 484 |

| | |
|--|-----|
| newTag ... -how NOTCOINCIDENTINSIDEEDGE | 486 |
| newTag ... -how NOTCOINCIDENTOUTSIDEEDGE | 488 |
| newTag ... -how NOTINSIDEEDGE | 490 |
| newTag ... -how NOTOUTSIDEEDGE | 492 |
| newTag ... -how NOTTOUCHEDGE | 494 |
| newTag ... -how NOTTOUCHINSIDEEDGE | 496 |
| newTag ... -how NOTTOUCHOUTSIDEEDGE | 498 |
| newTag ... -how ORTAGS | 500 |
| newTag ... -how OUTSIDEEDGE | 501 |
| newTag ... -how SEQUENCE | 503 |
| newTag ... -how SUBTRACTTAGS | 506 |
| newTag ... -how TOUCHEDGE | 507 |
| newTag ... -how TOUCHINSIDEEDGE | 509 |
| newTag ... -how TOUCHOUTSIDEEDGE | 511 |
| opcTag | 513 |
| setMoveLimitForTag | 517 |
| siteControl DELETE | 518 |
| siteControl OFFSET_FROM | 519 |
| siteControl OFFSET_PROP | 523 |
| siteControl OFFSET_ROTATED | 525 |
| siteControl SHIFT_PROP | 527 |
| siteControl SHIFT_TOWARD | 529 |
| siteControl SMOOTH | 531 |
| sitemodify | 536 |
| tags2boxes | 539 |
| Pre-Defined Tags | 541 |

SVRF Commands

There are four SVRF commands used by the sparse OPC tools.

Table 6-1. SVRF Command Summary

| Command | Description |
|------------------|--|
| LITHO FILE | Contains the litho setup file within an SVRF rule file. |
| LITHO OPC | Layer operation used to invoke Calibre OPCpro for optical process correction. |
| LITHO ORC | Layer operation used to invoke Calibre ORC for tagging edges or verifying OPC results. |
| LITHO PRINTIMAGE | Layer operation used to invoke Calibre PRINTImage to simulate a SEM image. |

The model-based Calibre RET tools all use litho setup files to provide tool-specific settings and operations within a Calibre DRC run. The file format as discussed in this manual also generally applies to non-sparse OPC tools, although there may be differences.

LITHO FILE

Type: [SVRF Commands](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Contains the litho setup file within an SVRF rule file.

Usage

```
LITHO FILE name '['/*  
    inline_file  
*/ ']'
```

Arguments

- ***name***

A required argument providing a way for the [LITHO OPC](#), [LITHO ORC](#), or [LITHO PRINTIMAGE](#) statement to reference the setup file.

- '['
 inline_file
 ']'

A required setup file placed between brackets ([]). Characters that occur within the brackets on the same line as the brackets are ignored. The comment characters /* and */ are not part of the syntax, but are generally used to prevent the SVRF compiler from warning about the non-SVRF contents of the *inline_file*.

Each statement in the setup file must be on a line of its own. Statements can span multiple lines by ending the line with a backslash (\). The backslash character must not be followed by any other characters, even whitespace.

For the sparse OPC tools, the setup file has the general format of

```
LITHO FILE name [ /*  
    Litho Setup File Keywords  
    ----- Arbitrary Commands -----  
    Litho Variables  
    Tagging Commands  
*/ ]
```

This order is required. Not all keywords, variables, and commands are supported by all sparse OPC tools. When a control has limited support, the supported tools are listed in the Type section.

The *inline_file* may not contain brackets.

Comments must begin at the beginning of the input line and are indicated with a double forward slash (//) or pound sign (#). Comments prefaced with // appear in the transcript; comments prefaced with # do not.

Litho Setup File Format

Input for: Calibre OPCpro, Calibre ORC, and Calibre PRINTimage. It may also be used for mini-OPC and the PI button in Calibre WORKbench, but is not required.

The model-based Calibre RET tools all use litho setup files to provide tool-specific settings and operations within a Calibre DRC run. The file format as discussed in this manual also generally applies to non-sparse OPC tools, although there may be differences.

A litho setup file can be a standalone setup file, referenced using the FILE keyword in the LITHO operation, or it can be provided inside the SVRF rule file using the [LITHO FILE](#) statement. (This second form is sometimes referred to as a “setup parameter block”.) Both methods support Tcl scripting to parameterize repetitive setup code (see Examples 2 through 5 in “[Examples](#)”), and both are checked for errors before running. The only difference is that when provided as a standalone file, all lines must begin in the first column.

All parameters for a specific batch tool run must be defined in the same file, and must not change during the run. If you want to modify a setup file for a series of runs, set Calibre to save a copy to a setup parameter block when the run starts using code like the following:

```
LITHO FILE setup_block [ /*  
    INCLUDE litho.in  
 */ ]
```

Setup File Error Reporting

All setup files are checked for syntax errors before executing any SVRF operations. If the application encounters an error in the setup file, it aborts the run. The error message in the transcript reports the line in the setup file where the error was encountered. Note that if the setup file contains multiple errors, processing halts as soon as the first error is encountered.

The following example shows an error message generated during parsing.

```
-----  
AT LINE # 58  
SETUP FILE ERROR: sse: not a valid env variable OPC_USE_ADAPTIVE_STEPS  
-----
```

```
ERROR: USER ERROR in LITHO  
xxxxxxxxxxSetup file problem.
```

```
//////////  
//////////  
CALIBRE LITHO ABORTED DUE TO FATAL USER ERROR  
//////////  
//////////  
//////////
```

Format

A litho setup file must conform to the following restrictions:

Syntax Rules

- Only one command or variable per line.
- All arguments for a command must be on the same line as the keyword. A line can be extended by having a \ as the last character on the line. (Additional whitespace or comments cause a syntax error.)
- All litho keywords, arguments, option names, command names, and variable names are reserved words. Do not use them as names of layers or tags.
- Comments within the setup parameters begin with a pound sign (#) or a double forward slash (//).
- Comments must begin at the beginning of the input line, and everything on the line after the comment character is part of the comment.
- Use the double forward slash (//) character to have your comments remain in the transcript after the setup file is pre-processed. The pound sign (#) is interpreted as a Tcl comment character and any subsequent text is removed from the transcript.

Case Sensitivity

- Keywords and names written in all uppercase letters in this document are case-insensitive.
- All other keywords and names are case-sensitive.
- Because of their relationship to the directory system of the host platform, structure names and pathnames are also case sensitive. To avoid confusion, you should enclose them in quotation marks.

Order Rules

- Keywords must appear at the beginning of the setup parameters, separate from variables and tagging commands by the “arbitrary commands” comment.
- Keyword arguments must appear in the order specified in the syntax descriptions in this manual.
- Keyword options, which take the form of pairs of secondary keyword and value, can appear in any order after the keyword itself.
- Variables, which you set using the sse command, must appear in the arbitrary command section of the setup files.
- Tagging commands must appear after the variables and must appear in the order in which they are to be evaluated.

- For the sparse OPC tools, the setup file is generally formatted as follows:

```
LITHO FILE name [ /*  
    Litho Setup File Keywords  
    #----- Arbitrary Commands -----  
    Litho Variables  
    Tagging Commands  
 */ ]
```

Parameters

There are few required parameters. The set that cause a syntax error if not present includes

- [background](#)
Describes the imaging background, such as dark field, light field, or attenuated.
- [cornerSiteStyle](#)
Specifies how to handle control sites on fragments adjacent to corners.
- [gridsize](#)
Sets an internal grid size to use for calculations, typically the same as the layout's units.
- [layer](#)
Defines the properties of layers passed in by the LITHO operation.
- [minfeature](#)
Specifies the minimum design feature size in microns.
- [opticalmodel](#) or [optical_model](#)
Specifies the optical model used in simulation. One of these commands is required even if the setup file primarily uses models specified with inline_optical.
- [resistpolyfile](#)
Specifies the resist model used in simulation.

There are many possible parameters, so the remainder of this chapter covers them in three sections. See the Related Topics.

Examples

Example 1 — Simple Setup File Without Tagging

```
# ----- Simulation models -----  
modelpath ./models  
opticalmodel mymodel.opt  
resistpolyfile mymodel.mod  
# Simulation Models - Use this section to define simulation model  
# information such as:  
# - The directories to search for the models  
# - Optical model names  
# - Process model name
```

```
# ----- OPC algorithm -----
iterations 4
tilemicrons 160.000000
stepsize 0.001
gridsize 0.001
siteinfo RESIST
cornerSiteStyle SITES_ON_ARC
lineEndAdjDist 0.190000
convexAdjDist 0.140000
concaveAdjDist 0.140000

# OPC Algorithm - Use this section to define OPC parameters such as:
# - Number of iterations for OPC
# - The input GDS grid size
# - The edge movement size, or stepsize
# - Site information (location points for intensity calculations)
# - Corner site styles (placement of corner sites)

# ----- Fragmentation -----
minfeature 0.180000
minedgelength 0.130000
maxedgelength 1000.000000
cornedge 0.130000
concavicorn 0.130000
interfeature -interdistance 0.290000 -ripplelen 0.130000 -num 0
seriftype 0
minjog 0.120000
lineEndLength 0.320000

# Fragmentation - Use this section to define fragmentation parameters such
# as:
# - Fragmentation types such as interfeature, island-layer, visible-layer
# - Features to fragment
# - Fragmentation limits such as max or min edge length, corners, jog size

# ----- Layer info -----
background clear
# Layers
layer 2 DIFF 17 0 hidden dark
layer 4 POLY 17 0 opc dark
layer 5 L237 17 0 hidden dark
layer 104 HI_OPCT 49 0 hidden dark
layer 105 DEF_OPCT 49 0 hidden dark
layer 106 LO_OPCT 49 0 hidden dark

# Layer Info - Use this section to define layer information such as:
# - Layer definitions
# - Background definitions
```

```
#----- Arbitrary Commands -----
sse LITHO_PROMOTION_TILING 1

MRC_RULE EXTERNAL {USE 0.16}
MRC_RULE INTERNAL {USE 0.13}

# Arbitrary Commands Area - Use this area to add additional setup features
# such as:
# - Setup environment variables
# - Custom Tcl scripts
# - fragmentLayer blocks (command blocks used to override global or
#   default settings for specified layers)
```

Example 2 — Tcl Blocks

A Tcl script can be placed in the “Arbitrary Commands” section of the setup file, following a clearTagScript keyword. For example:

```
clearTagScript
set simtag "all"
for {set i 1} {$i <= 10} {incr i} {
    newTag negEPE$i -how EPE $simtag -0.1 0
    fastIter negEPE$i 1
    newTag posEPE$i -how EPE $simtag 0 0.1
    fastIter posEPE$i 1
}
```

Tcl commands in the setup file are pre-processed by the Tcl interpreter, then converted (or “expanded”) to Calibre LITHO commands, and finally input to the litho tool. The litho transcript echoes the Calibre translated commands and not the original Tcl commands. For instance, the following Tcl example:

```
clearTagScript
for {set i 0} {$i < 3} {incr i} {
    fastIter all 1
}
```

expands to the following executable litho commands:

```
clearTagScript
fastIter all 1
fastIter all 1
fastIter all 1
```

In addition, the Calibre WORKbench GUI tool expands any Tcl commands into translated setup file commands. You do not have to maintain the expanded form of the script.

Note

 Be sure to save the expanded setup code in a different file. Once Tcl code is translated, it cannot be restored to its original language syntax.

Tcl scripting can be used as a method of implementing tagging operations. For example, you can create a Tcl script that executes the equivalent of a series of fastIter commands, allowing you to perform custom OPC or ORC iterations on specific tags.

Example 3 — Custom OPC With Negative and Positive EPE Search

The following Tcl script was created to search for negative and positive EPEs and apply OPC on those edges:

Example 6-1. Tcl for Custom OPC

```
clearTagScript
set simtag "all"
for {set i 1} {$i <= 5} {incr i} {
    # Perform this loop five times
    newTag negEPE$i -how EPE $simtag -0.1 0
    # Find any edge with negative EPE
    fastIter negEPE$i 1
    # Apply OPC to those edges
    newTag posEPE$i -how EPE $simtag 0 0.1
    # Find any edge with positive EPE
    fastIter posEPE$i 1
    # Apply OPC to those edges
}
```

The Tcl loop expands to the following litho commands:

```
clearTagScript
newTag negEPE1 -how EPE all -0.1 0
fastIter negEPE1 1
newTag posEPE1 -how EPE all 0 1
fastIter posEPE1 1
newTag negEPE2 -how EPE all -0.1 0
fastIter negEPE2 1
newTag posEPE2 -how EPE all 0 0.1
fastIter posEPE2 1
newTag negEPE3 -how EPE all -0.1 0
fastIter negEPE3 1
newTag posEPE3 -how EPE all 0 0.1
fastIter posEPE3 1
newTag negEPE4 -how EPE all -0.1 0
fastIter negEPE4 1
newTag posEPE4 -how EPE all 0 0.1
fastIter posEPE4 1
newTag negEPE5 -how EPE all -0.1 0
fastIter negEPE5 1
newTag posEPE5 -how EPE all 0 0.1
fastIter posEPE5 1
```

Note how many newTag and fastIter commands would normally have been manually written without the use of the Tcl script.

Example 4 — Generate an Array of Tags for Use With Calibre ORC

The following Tcl loop creates an array of image tags (SLOPE, in this case). The range is from 0 to 5, in increments of 0.5.

Example 6-2. Tcl for Array of Image Tags

```
clearTagScript
newTag orcable -how EDGE LENGTH >= 0.0875
set stepSLOPE 0.5
for (set i 0) ($i <=9) (incr i) {
    set hibin [expr $i*stepSLOPE+$stepSLOPE]
    set lobin [expr $i*stepSLOPE]
    newTag "SLOPE$lobin$hibin" -how IMAGE orcable SLOPE > $lobin <= $hibin
}
```

This Tcl code block expands to the following litho commands:

```
clearTagScript
newTag orcable -how EDGE LENGTH >= 0.0875
newTag SLOPE0.00.5 -how IMAGE orcable SLOPE > 0.0 <= 0.5
newTag SLOPE0.51.0 -how IMAGE orcable SLOPE > 0.5 <= 1.0
newTag SLOPE0.01.5 -how IMAGE orcable SLOPE > 1.0 <= 1.5
newTag SLOPE0.52.0 -how IMAGE orcable SLOPE > 1.5 <= 2.0
newTag SLOPE0.02.5 -how IMAGE orcable SLOPE > 2.0 <= 2.5
newTag SLOPE0.53.0 -how IMAGE orcable SLOPE > 2.5 <= 3.0
newTag SLOPE0.03.5 -how IMAGE orcable SLOPE > 3.0 <= 3.5
newTag SLOPE0.54.5 -how IMAGE orcable SLOPE > 3.5 <= 4.0
newTag SLOPE0.04.5 -how IMAGE orcable SLOPE > 4.0 <= 4.5
newTag SLOPE0.55.5 -how IMAGE orcable SLOPE > 4.5 <= 5.0
```

Example 5 — Tags2boxes Output (ORC)

The following Tcl loop performs the same tagging operation as in [Example 4 — Generate an Array of Tags for Use With Calibre ORC](#) (although this time, the increment is 0.7). Additionally, each tagged fragment is written out with a tags2boxes operation.

Example 6-3. Tcl for Image Tagging and Tags2Boxes

```
clearTagScript
newTag orcable -how EDGE LENGTH >= 0.0875
set stepSLOPE 0.7      #increment by 0.7
for (set i 0) ($i <=9) (incr i) {
    set hibin [expr $i*stepSLOPE+$stepSLOPE]
    set lobin [expr $i*stepSLOPE]
    newTag "SLOPE$lobin$hibin" -how IMAGE orcable SLOPE > $lobin <= $hibin
}
for (set i 0) ($i <=9) (incr i) {
    set hibin [expr $i*stepSLOPE+$stepSLOPE]
    set lobin [expr $i*stepSLOPE]
    set layernum [expr $i +300]
    tags2boxes -tags "SLOPE$lobin$hibin" -layers "$layernum"
                                #output tags2boxes
}
```

This Tcl code block expands to the following litho commands:

```
clearTagScript
newTag orcable -how EDGE LENGTH >= 0.0875
newTag SLOPE0.00.7 -how IMAGE orcable SLOPE > 0.0 <= 0.7
newTag SLOPE0.71.4 -how IMAGE orcable SLOPE > 0.7 <= 1.4
newTag SLOPE0.42.1 -how IMAGE orcable SLOPE > 1.4 <= 2.1
newTag SLOPE0.12.8 -how IMAGE orcable SLOPE > 2.1 <= 2.8
newTag SLOPE0.83.5 -how IMAGE orcable SLOPE > 2.8 <= 3.5
newTag SLOPE0.54.2 -how IMAGE orcable SLOPE > 3.5 <= 4.2
newTag SLOPE0.24.9 -how IMAGE orcable SLOPE > 4.2 <= 4.9
newTag SLOPE0.95.6 -how IMAGE orcable SLOPE > 4.9 <= 5.6
newTag SLOPE0.66.3 -how IMAGE orcable SLOPE > 5.6 <= 6.3
newTag SLOPE0.37.0 -how IMAGE orcable SLOPE > 6.3 <= 7.0
tags2boxes -tags SLOPE0.00.7 -layers 300
tags2boxes -tags SLOPE0.71.4 -layers 301
tags2boxes -tags SLOPE0.42.1 -layers 302
tags2boxes -tags SLOPE0.12.8 -layers 303
tags2boxes -tags SLOPE0.83.5 -layers 304
tags2boxes -tags SLOPE0.54.2 -layers 305
tags2boxes -tags SLOPE0.24.9 -layers 306
tags2boxes -tags SLOPE0.95.6 -layers 307
tags2boxes -tags SLOPE0.66.3 -layers 308
tags2boxes -tags SLOPE0.37.0 -layers 309
```

Related Topics

[Litho Setup File Keywords](#)

[Litho Variables](#)

[Tagging Commands](#)

[Tcl Library in Calibre \[Calibre Administrator's Guide\]](#)

LITHO OPC

Type: [SVRF Commands](#). Used by Calibre OPCpro.

Layer operation used to invoke Calibre OPCpro for optical process correction.

Usage

```
LITHO OPC layer [layer ...] FILE {filename | parameter_block_name | '['
    inline_file
    '']' } [MAPNUMBER N | MAP layer_name]
```

Arguments

- **layer** [layer...]

A required original, derived polygon, or derived edge layer upon which to perform Calibre OPCpro operations. You can specify **layer** any number of times in one statement. The first **layer** is the target layer for correction. Any additional **layers** are supplementary (for example, non-printing islands).

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

A required keyword followed by one of the following:

A **filename** indicating the path to the setup file, generally enclosed in quotation marks. The **filename** parameter can contain environment variables. For information regarding the use of environment variables in the **filename** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in the [SVRF Manual](#).

The **name** parameter of a [LITHO FILE](#) specification statement. The setup file is defined in the SVRF rule file.

Inline_file, a copy of the setup file within the LITHO OPC statement, enclosed in brackets ([]). Characters within the brackets on the same line as them are ignored. In-line files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAPNUMBER** *N*

An optional keyword followed by the tag layer to output. The parameter *N* is the layer specified in the setup file’s layer statement. MAPNUMBER cannot be used with the MAP keyword.

- **MAP** *layer_name*

An optional 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. MAP cannot be used with MAPNUMBER.

Description

Specifies to run Calibre OPCpro during Calibre nmDRC or Calibre nmDRC-H. The command takes in a target layer and outputs a new layer with OPC-corrected shapes.

This command can be specified any number of times in your rule file. By default, it produces a log file named *litho.log* in the working directory and, if MAP or MAPNUMBER is specified, a layer. The layer must be defined within the setup file.

Concurrent Operations

When LITHO OPC generates multiple result layers (for example, running OPC on two or more mask layers), the Calibre nmDRC hierarchical engine performs the LITHO operation once if the LITHO OPC statement differs only by the MAP or MAPNUMBER value.

Examples

Example 1

This example shows how the *layer* in the SVRF call is matched by a layer command in the setup file. It uses the *inline_file* syntax, but only shows the relevant setup instruction.

```
OPC_OUT = LITHO OPC poly FILE [  
    ...  
    layer 0 poly 0 0 opc dark  
    ...  
]
```

Example 2

This example shows the difference between using MAP and MAPNUMBER in LITHO OPC for PSM masks. Using MAP and the layer name has the advantage of using the layer name “PHASE0”, as opposed to the setup file map number.

Using MAP:

```
PHASE0_OPCT = LITHO OPC PHASE0 PHASE180 BLOCK TARGET FILE "setup.in"  
MAP PHASE0
```

Using MAPNUMBER:

```
PHASE0_OPCT = LITHO OPC PHASE0 PHASE180 BLOCK TARGET FILE "setup.in"  
MAPNUMBER 100
```

The litho setup file (*setup.in*) for both cases includes the following line:

```
layer 100 PHASE0 0 0 correction clear 1 1.0
```

Related Topics

[Model-Based OPC Examples](#)

LITHO ORC

Type: [SVRF Commands](#). Used by Calibre ORC.

Layer operation used to invoke Calibre ORC for tagging edges or verifying OPC results.

Usage

```
LITHO ORC FILE {filename | parameter_block_name |  
  '['  
  inline_file  
  ']'} layer_in [layer_aux ...] [MAPNUMBER value | MAP name]
```

Parameters

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

A required keyword followed by one of the following:

A *filename* indicating the path to the setup file, generally enclosed in quotation marks.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in the *SVRF Manual*.

The *name* parameter of a [LITHO FILE](#) specification statement. The setup file is defined in the SVRF rule file.

Inline_file, a copy of the setup file within the LITHO ORC statement, enclosed in brackets ([]). Characters within the brackets on the same line as them are ignored. In-line files contain setup instructions for OPC and ORC and span multiple lines. The instructions may not use environment variables.

- **layer_in**

A required original or derived polygon layer that serves as input. In the setup file, it must have type opc.

- **layer_aux**

An optional layer or layers that provides additional simulation or fragmentation data. In the setup file, these layers can be of type visible, asraf, external, or island.

- **MAPNUMBER value**

An optional keyword followed by the tag layer to output. For ORC applications, the parameter value is the layer specified in the technology setup file with the tags2boxes command. MAPNUMBER cannot be used with the MAP keyword.

- **MAP name**

An optional 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. MAP cannot be used with MAPNUMBER.

Description

Runs optical rule checking (ORC) during Calibre nmDRC or Calibre nmDRC-H on a single layer, *layer_in*, and outputs a new layer containing edges that have either been tagged or are in error.

This command can be specified any number of times in your rule file. By default, it produces a log file named *litho.log* in the working directory.

Concurrent Operations

When LITHO ORC generates multiple result layers (for example, outputting different types of errors to different layers), the Calibre nmDRC hierarchical engine performs the LITHO operation once if the LITHO ORC statement differs only by the MAP or MAPNUMBER value.

Examples

In the following example, POLY is the input layer to Calibre ORC and *orcpoly.in* is the setup file. Calibre ORC derives the layer MY_ORC, which contains poly edges that were in error. This derived layer is then output using the ORC rule check and sent to a GDSII database file using DRC Check Map.

```
MY_ORC = LITHO ORC FILE orcpoly.in POLY
ORC.1 { copy MY_ORC }
DRC CHECK MAP ORC.1 orc.gds GDSII
```

Related Topics

[Calibre ORC Procedures and Examples](#)

LITHO PRINTIMAGE

Type: [SVRF Commands](#). Used by Calibre PRINTimage.

Layer operation used to invoke Calibre PRINTimage to simulate a SEM image.

Usage

```
LITHO PRINTIMAGE FILE {filename | parameter_block_name |  
  '['  
  inline_file  
  ']'} layer [layer ...] [MAPNUMBER value]
```

Parameters

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

A required keyword followed by one of the following:

A *filename* indicating the path to the setup file, generally enclosed in quotation marks.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in the *SVRF Manual*.

The *name* parameter of a [LITHO FILE](#) specification statement. The setup file is defined in the SVRF rule file.

inline_file, a copy of the setup file within the LITHO PRINTIMAGE statement, enclosed in brackets ([]). Characters within the brackets on the same line as them are ignored. In-line files contain setup instructions and span multiple lines. The instructions may not use environment variables.

- **layer**

A required original or derived polygon layer upon which to perform Calibre PRINTimage operations. The first layer is the target layer and the optional layers are supplementary.

- **MAPNUMBER value**

An optional keyword followed by the contour level to output. If this keyword is not used then all contours are returned in a single layer.

Description

Invokes Calibre PRINTimage during a Calibre nmDRC or Calibre nmDRC-H run. Calibre PRINTimage simulates a scanning electron microscope (SEM) image of the polygons on the first *layer*. The simulation data is generated as a set of contours.

Generally LITHO PRINTIMAGE commands provide at least two *layers*: the target layer and an OPC-corrected layer. In the setup file, the target layer (the first in the LITHO PRINTIMAGE statement) must be of type opc. The corrected layer must be of type external. Any other layers

must be of type visible, asraf, external, or island. The supplemental layers cannot be hidden, correction, or opc types.

The LITHO PRINTIMAGE command can be specified any number of times in your rule file. By default, it produces a log file named *litho.log* in the working directory.

Concurrent Operations

When LITHO PRINTIMAGE generates multiple result layers, the Calibre nmDRC hierarchical engine performs the LITHO operation once if the statement differs only by the MAP or MAPNUMBER value.

Examples

This example generates an SEM image for a simple binary mask. The LITHO PRINTIMAGE command is part of a DRC rule check.

```
MY_SEM { LITHO PRINTIMAGE FILE printimage.in POLY }
DRC CHECK MAP MY_SEM 3

LITHO FILE printimage.in [
    ...
    layer 2 poly 0 0 opc dark
    ...
]
```

Because the example provides only the target layer, the simulated image does not reflect OPC correction.

Related Topics

[Calibre PRINTImage Procedures and Examples](#)

Litho Setup File Keywords

Setup file keywords define OPC and simulation parameters and define the layers used in Calibre OPCpro, Calibre ORC, and Calibre PRINTimage operations. Keywords may only be specified once per setup file.

All keywords and arguments are case-sensitive unless explicitly noted otherwise on the reference page.

Table 6-2. Summary of Litho Setup File Keywords and Commands

| Keyword | Description |
|------------------------------|--|
| <code>background</code> | <i>Required.</i> Describes the imaging background. |
| <code>coincidentLayer</code> | Specifies coincident layer fragmentation for a specified layer. This can only be used within a <code>fragmentLayer</code> block. |
| <code>concaveAdjDist</code> | Fragmentation parameter specifying the distance away from a concave corner considered to be adjacent to the corner. |
| <code>concavcorn</code> | An optional fragmentation keyword controlling the number and length of intrafeature fragmented edges at concave corners. |
| <code>convexAdjDist</code> | Fragmentation parameter specifying the distance away from a convex corner adjacent to the corner. |
| <code>cornedge</code> | Fragmentation parameter providing precise control over the number and length of intrafeature fragmented edges at convex corners. |
| <code>cornerRadius</code> | Controls how sites are rotated on corner fragments. |
| <code>cornerSiteStyle</code> | <i>Required.</i> Specifies the model used to place control sites on fragments adjacent to a corner. |
| <code>flaglayer</code> | Enables fragmentation operations on a per-layer basis, overriding the frag bit settings of the layer keyword. |
| <code>fragmentLayer</code> | Overrides global parameter settings for a specified layer. |
| <code>gridsize</code> | <i>Required.</i> Specifies the internal grid size in microns. |
| <code>gse</code> | Returns the value of a litho variable. |
| <code>inline_optical1</code> | Allows you to include an optical model inside your setup file, rather than as a separate file. |
| <code>inline_optical2</code> | Allows you to include a second optical model inside your setup file rather than as a separate file. |
| <code>inline_optical3</code> | Allows you to include a third optical model inside your setup file rather than as a separate file. |
| <code>inline_resist</code> | Allows you to include a resist model inside your setup file, rather than as a separate file. |

Table 6-2. Summary of Litho Setup File Keywords and Commands (cont.)

| Keyword | Description |
|------------------------------|---|
| <code>interfeatLayers</code> | Specifies interfeature fragmentation for a specified layer. This can only be used within a fragmentLayer command block. |
| <code>interfeature</code> | Defines the fragmentation parameters that influence interfeature fragmentation. |
| <code>intrafeature</code> | Alternative to cornedge and concavecorn for implementing intrafeature fragmentation. |
| <code>islandFragment</code> | Controls island layer fragmentation. |
| <code>iterations</code> | Specifies how many iterations of OPC movement to perform on the design. |
| <code>layer</code> | <i>Required.</i> Defines how the given mask layer is treated by OPC and simulation. |
| <code>lineEndAdjDist</code> | Specifies the distance away from a line end that is considered to be adjacent to the line end. |
| <code>lineEndLength</code> | Defines the distance criteria used to determine if an edge is on a line end or not. |
| <code>maxedgelength</code> | Breaks long edges into fragments by specifying the maximum length of a fragment. |
| <code>minedgelength</code> | Sets the smallest length that can be produced by fragmentation. |
| <code>minfeature</code> | <i>Required.</i> Specifies the minimum design feature size in microns. |
| <code>minjog</code> | Controls the internal definition of a jog and ensures that jogs are not corrected. |
| <code>modelpath</code> | <i>Required.</i> Defines the search path for optical and process models. |
| <code>movelimit</code> | Specifies the maximum inward and outward movements allowed for any fragment. |
| <code>MRC_RULE</code> | Sets constraints for EXTERNAL and INTERNAL rule checks for specific tags. |
| <code>opcIter</code> | Applies a different OPC iteration algorithm than fastIter. |
| <code>opticalmodel</code> | <i>Required.</i> Specifies the optical model for aerial image simulation during OPC and analysis. |
| <code>outlayer</code> | Creates a supplementary output layer that displays the tile boundaries used when processing your layout. |
| <code>processModel</code> | Registers the name of a resist model file. |
| <code>progversion</code> | Reports the litho program version in the setup file. |
| <code>resistpolyfile</code> | <i>Required.</i> Defines the resist model to use for simulations. |

Table 6-2. Summary of Litho Setup File Keywords and Commands (cont.)

| Keyword | Description |
|------------------------------------|---|
| secondOpticalmodel | Specifies a second mask in multiple exposure simulations. |
| seriftype | Determines how line ends are fragmented for OPC. |
| set_kernel_count | Changes number of active kernels in the optical model. |
| set_resist_model | Changes the resist model during tagging operations. |
| siteinfo | Specifies where and how Calibre WORKbench and Calibre OPCpro install control sites. |
| sse | Sets or changes the value of a litho setup variable. |
| stepsize | Specifies the movement distance in microns for each edge in one OPC iteration. |
| svrf_var | Enables LITHO operation access to variables defined elsewhere in the SVRF rule file. |
| thirdOpticalmodel | Specifies a third mask in multiple exposure simulations such as implant modeling. |
| tilemicrons | Specifies the tile size for breaking up the design into smaller pieces for processing. |
| visibleLayers | Changes fragmentation caused by asraf and visible layers. Requires a fragmentLayer statement. |

background

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Required. Describes the imaging background.

Usage

background *setting* [*setting*]

Arguments

- *setting*

A required argument indicating the background type. This is related to the layer transmission value. (For example, a clear field mask does not support any layers that are defined as clear.)

You must specify one background type per mask exposure. When multiple masks are used, the first argument is applied to Mask 0, the second to Mask 1, and so on.

One of the following four arguments is required:

dark

A literal argument designating dark field imaging.

clear

A literal argument designating clear field imaging.

attenuated *factor*

The **attenuated** keyword (or **atten**) defines the layer to be an attenuated phase-shifting layer (180-degree shift) and the optical transmission to be a percent of incident light, as specified by factor. The *factor* value is a required numerical argument to the attenuated option. It is the attenuation factor for attenuated backgrounds. The unit is a percentage. For an 8% attenuated phase shift mask, the factor should be 0.08. The maximum allowed value is 0.36 (36%).

When Calibre litho tools encounter an attenuated factor, they translate the factor into a pair of real and imaginary values. The basic principle behind transmission layers can be summarized as follows:

actual transmission = background + layer transmission values

When using the **atten** keyword to specify background transmission, it uses the following real imaginary pair:

RE(bg) = -sqrt(*percent*/100)

IM(bg) = 0

For example:

background atten 0.06

is equivalent to

```
background -0.245 0
```

In this example, the factor of 6% (0.06) with a 180-degree phase-shifted transmission is applied to the background. This corresponds to an electric field (real, imaginary) pair of $(-\sqrt{0.06}, 0) = (-0.245, 0)$.

There are no limitations to the layer transmission values when the **atten** option is used for the background definition. Do not use an attenuated background with a **phase180** layer, as this causes a user error.

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:

```
background (real imag) + layer (real imag) = transmission_value
```

For example:

```
background 1 0
```

| number | name | frag | ctl | opt | type |
|--------|--------|------|-----|-----|------|
| 1 | chrome | 19 | 0 | opc | 0 0 |

The background is specified with a (1, 0) pair, while chrome layer 1 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 $(1,0) + (0,0) = \text{clear } (1,0)$.

```
background 1 0
```

| number | name | frag | ctl | opt | type |
|--------|------|------|-----|---------|------|
| 1 | p180 | 12 | 0 | visible | -1 0 |

In this case, the layer is defined as (-1 0). With clear background (1,0) and a layer specified with (0,0), the resulting transmission layer is $(1,0) + (-1,0) = \text{dark } (0,0)$.

Examples

Example 1

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 0 VIA 0 0 opc clear
```

Example 2

The following example specifies a valid double mask exposure. Mask 0 is dark, and mask 1 is clear.

```
background dark clear
```

This could also be expressed using the ***real imag*** syntax:

```
background 0 0 1 0
```

Related Topics

[layer](#)

coincidentLayer

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies coincident layer fragmentation for a specified layer. This can only be used within a fragmentLayer block.

Usage

coincidentLayer *layer1* -overlay {0 | 1}

Arguments

- ***layer1***

A required layer name with which edges must be coincident in order to be fragmented.
Layer1 must be of type opc, correction, island, asraf, or visible.

There is no default value for this keyword.

- **-overlay {0 | 1}**

A required keyword and value. When it is set to 1, the fragmentation points from the specified **layer1** are overlaid onto the current layer, keeping those points from **layer1**. The coincident edge copies the fragmentation, and the non-coincident edges are still fragmented.

When it 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 fragmentLayer block. Edges on the layer to which the fragmentLayer block applies are fragmented if and only if they are coincident with the layer specified by this keyword.

Examples

The following example shows fragmentation specified for edges coincident with layer L1. (Both examples use the same setup file.)

```
fragmentLayer L2 {
    coincidentLayer L1
}
fragmentLayer L3 {
    coincidentLayer L1
}
```

[Figure 6-1](#) shows the results. The highlighted dots represent interfeature vertices created on the coincident edges.

Figure 6-1. Changing the Coincident Layer to Affect Fragmentation



Related Topics

[fragmentLayer](#)

[fragcoinc](#)

concaveAdjDist

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Fragmentation parameter specifying the distance away from a concave corner considered to be adjacent to the corner.

Usage

concaveAdjDist *distanceValue*

Arguments

- ***distanceValue***

A required argument specifying the distance in microns. The ***distanceValue*** must be greater than or equal to 0.

Description

An optional keyword fragmentation parameter.

The concaveAdjDist is the distance away from a concave corner considered to be adjacent to the corner. This distance controls which fragments are tagged with the built-in tag concave_corner_adjacent, which is used in conjunction with the [layer](#) keyword's ***layer_type*** parameter to force or prevent OPC at concave corners. Only fragments having both endpoints within concaveAdjDist to a concave corner vertex are tagged as concave_corner_adjacent.

Examples

```
concaveAdjDist 0.0800
```

Related Topics

[concavecorn](#)

[layer](#)

concavecorn

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

An optional fragmentation keyword controlling the number and length of intrafeature fragmented edges at concave corners.

Usage

```
concavecorn len1 ... lenN [rem r]  
[non90 [d1 ... dN [rem r]]]  
[sea [seed1 ... seedN [rem r]]]
```

Arguments

- ***len1***
A required argument specifying the default distance in microns from the concave corner to the first fragmentation vertex that is produced. The value must be greater than or equal to [minedgelength](#). This value is used for all concave corners unless separate distances are specified for non-90 or space end adjacent corners.
- ***lenN***
A required argument specifying the default distance in microns from the previous fragmentation vertex to the next fragmentation vertex, measured sequentially from the corner inward. The value must be greater than or equal to [minedgelength](#). This value is used for all concave corners unless separate distances are specified for non-90 or space end adjacent corners.
- **rem *r***
An optional argument specifying the minimum allowed remainder. If intrafeature fragmentation would produce an edge with length smaller than *r*, then it does not fragment the edge. If not specified, the default value used is [minedgelength](#). The value must be greater than or equal to [minedgelength](#).
You can specify different remainders for non-90, sea, and standard concave corners.
This argument is not case-sensitive.
- **non90 [*d1*... *dN*]**
An optional argument used to specify the fragmentation values at non-90 degree concave corners when such fragments require different treatment. If not specified, non-90 degree corners receive no special treatment and are fragmented using *len1*... *lenN*.
The parameters *d1*...*dN* are the distances in microns of each successive fragment vertex from the previous vertex. Each value must be greater than 0. If non90 appears without values, fragmentation is disabled for non-90 degree corners.
This argument is not case-sensitive.

- *sea sead1 ... seadN*

An optional argument used to specify the fragmentation values at space-end adjacent concave corners when such fragments require different treatment. If not specified, space-end adjacent corners receive no special treatment and are fragmented using *len1 ... lenN*.

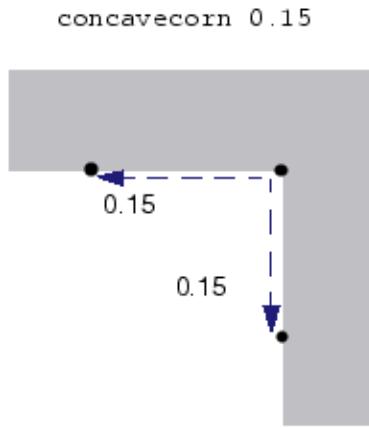
The parameters *sead1 ... seadN* are the distances in microns of each successive fragment vertex from the previous vertex. Each value must be greater than 0.

This argument is not case-sensitive.

Description

The concavecorn keyword is an optional fragmentation parameter. This keyword gives you precise control over the number and length of intrafeature fragmented edges at concave corners. You can create up to 20 fragments. The fragmentation process occurs automatically before OPC begins. In intrafeature fragmentation, the edges connected to corners are broken up according to the [cornedge](#) and concavecorn parameters. [Figure 6-2](#) shows how the intrafeature fragmentation parameters are applied to produce fragmentation vertices.

Figure 6-2. Intrafeature Fragmentation With concavecorn



Examples

Turn off fragmentation for non-90 degree corners.

```
concavecorn 0.1 non90
```

Related Topics

[concaveAdjDist](#)
[cornedge](#)
[intrafeature](#)
[layer](#)

convexAdjDist

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Fragmentation parameter specifying the distance away from a convex corner adjacent to the corner.

Usage

convexAdjDist *distanceValue*

Arguments

- *distanceValue*

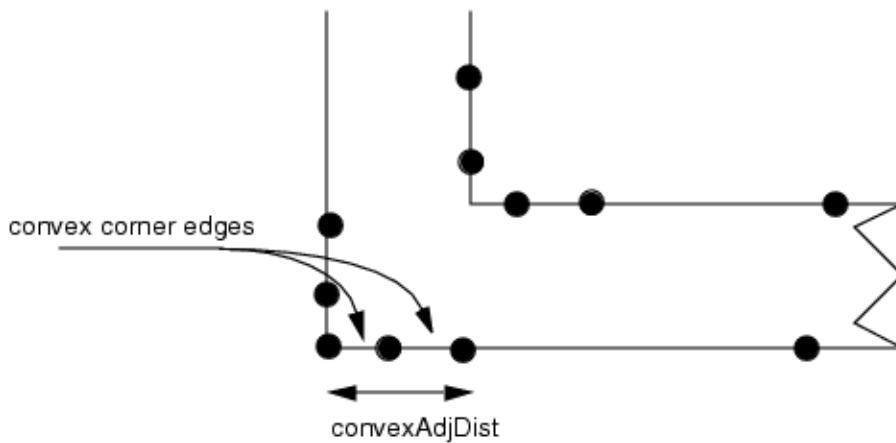
A required argument specifying distance in microns. The *distanceValue* must be greater than or equal to 0.

Description

This is an optional fragmentation parameter. The convexAdjDist is the distance away from a convex corner which is adjacent to the corner. The convexAdjDist can be used in combination with the [layer](#) keyword's *layer_type* parameter to force or prevent OPC at convex corners. All edge fragments completely within the convexAdjDist are considered adjacent to the corner.

Figure 6-3 shows edges that are convex edges by satisfying the adjacency requirements.

Figure 6-3. Adjacency Requirements



This keyword controls which fragments are tagged with the [convex_corner_adjacent](#) tag. Only fragments which have both endpoints within convexAdjDist to a convex corner vertex are tagged with [convex_corner_adjacent](#).

Related Topics

[cornedge](#)

[layer](#)

cornedge

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Fragmentation parameter providing precise control over the number and length of intrafeature fragmented edges at convex corners.

Usage

```
cornedge len1 ... lenN [rem r] [priority]  
[non90 d1 ... dN [rem r] [priority]]  
[lea lea_d1 ... lea_dN [rem r] [priority]]
```

Arguments

- ***len1***

A required argument specifying the distance in microns from the convex corner to the first fragmentation vertex that is produced. The value must be greater than or equal to [minedgelength](#). This value is used for all convex corners unless separate distances are specified for non-90 or line-end adjacent corners.

- ***lenN***

A required argument specifying the distance in microns from the previous fragmentation vertex to the next fragmentation vertex, measured sequentially from the corner inward. The value must be greater than or equal to [minedgelength](#). This value is used for all convex corners unless separate distances are specified for non-90 or line-end adjacent corners.

- ***rem r***

An optional argument specifying the minimum allowed remainder. If the specified fragmentation would produce an edge with length smaller than *r*, then that fragmentation does not occur. If not specified, the default value used is [minedgelength](#). The value must be greater than or equal to [minedgelength](#).

You can specify different remainders for non-90, lea, and standard convex corners.

This argument is not case-sensitive.

- ***priority***

An optional keyword indicating that convex corners have priority over concave corners. When an edge has both a convex and a concave corner, and the fragment is not long enough for fragmentation from both corners, giving priority to convex corners causes fragments to be created inward from the convex corner. In this case the rem value from the convex corner is used. Otherwise, the larger of the two rem values is used. This flag allows users to control fragmentation of “stubby” features.

You can control the priority separately for different types of convex corners: non-90, lea, and standard.

This argument is not case-sensitive.

- non90 [$d1 \dots dN$]

An optional argument used to specify the fragmentation values at non-90 degree convex corners when such fragments required different treatment. If not specified, nonsquare corners receive no special treatment and are fragmented using $\text{len}1 \dots \text{len}N$.

The parameters $d1 \dots dN$ are the distances in microns of each successive fragment vertex from the previous vertex. If non90 appears without values, fragmentation is disabled for non-90 degree corners. Each value must be greater than 0.

This argument is not case-sensitive.

- lea $\text{lea_d1} \dots \text{lea_dN}$

An optional argument used to specify the fragmentation values at line-end adjacent convex corners when such fragments required different treatment. If not specified, line-end adjacent corners receive no special treatment and are fragmented using $\text{len}1 \dots \text{len}N$.

The parameters $\text{lea_d1} \dots \text{lea_dN}$ are the distances in microns of each successive fragment vertex from the previous vertex. Each value must be greater than 0.

This argument is not case-sensitive.

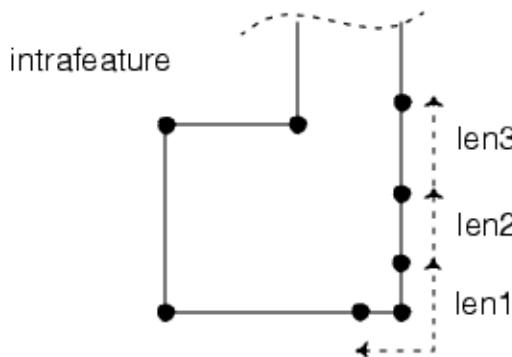
Description

This keyword is an optional fragmentation parameter. It gives you precise control over the number and length of intrafeature fragments created from edges at convex corners. In intrafeature fragmentation, the edges connected to corners are broken up according to the cornedge and concavecorn parameters. The fragmentation process occurs automatically before OPC begins. [Figure 6-4](#) shows how the intrafeature fragmentation parameters are applied to produce fragmentation vertices.

Note

 The number of cornedge fragments was limited to 10 in Calibre versions before 2008.3. This has been increased to 20 fragments for 2008.3 and beyond.

Figure 6-4. Intrafeature Fragmentation Example (cornedge)



Examples

Example 1

Turn off fragmentation for non-90 degree corners.

```
cornedge 0.1 non90
```

Example 2

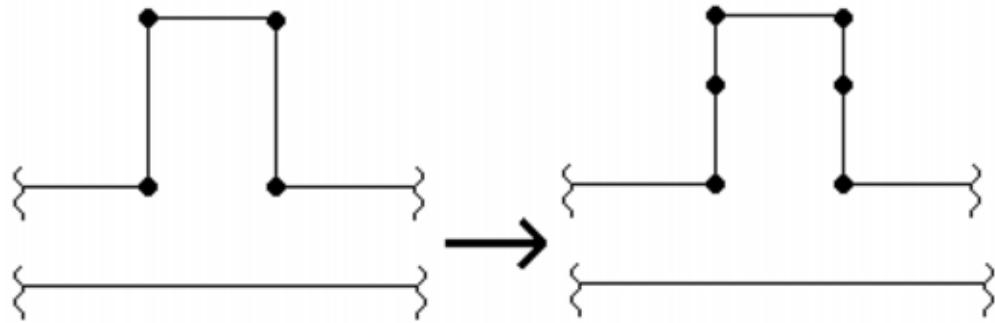
Turn on stubby feature fragmentation, where “stubby” features are defined as edges such that:

```
minedgelength + cornedge <= LENGTH < minedgelength + (cornedge len1) +  
(concavecorn len1)
```

To activate stubby feature fragmentation, use the following command:

```
cornedge 0.1 priority
```

Figure 6-5. Stubby Feature Fragmentation



To activate stubby feature fragmentation for line ends only, and limit fragmentation elsewhere

```
cornedge 0.11 0.12 rem 0.2 lea 0.1 0.12 priority non90
```

Related Topics

[concavecorn](#)

[convexAdjDist](#)

[intrafeature](#)

[layer](#)

cornerRadius

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Controls how sites are rotated on corner fragments.

Usage

cornerRadius *Radius*

Arguments

- ***Radius***

The radius in microns. The value must be between 0 and 0.3, inclusive. The default value is

$$\left(\frac{\lambda}{(1+\sigma)NA}\right)^{1/2}$$

where:

λ is optical model wavelength.

σ is optical model partial coherence.

NA is optical model numerical aperture.

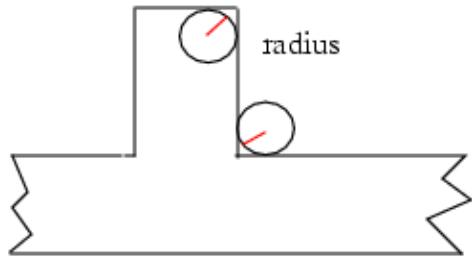
Description

This optional keyword controls how sites are rotated on corner fragments.

- A site that is placed within the cornerRadius is placed on the arc defined by cornerRadius and rotated normal to the arc.
- If the [cornerSiteStyle](#) is set to SITES_ON_ARC, cornerRadius also defines the radius of the inscribed arc. The cornerRadius setting is only in effect when SITES_ON_ARC is used. Otherwise, it is ignored.
- The rotation of sites by cornerRadius is independent of [convexAdjDist](#) or [concaveAdjDist](#). It depends only on the site being on the radius and [lineEndAdjDist](#).
- Changing the placement of corner sites, whether by the CONSERVATIVE, SITES_ON_ARC, or SITES_ON_EDGE site styles, applies only to the fragment that is touching a corner. The convexAdjDist or concaveAdjDist keywords do not apply, except in the following cases:
 - If the first fragment is not completely inside convexAdjDist or concaveAdjDist, it is not considered a corner fragment and its site is placed in the center. Rotation is not affected by whether a fragment touches a corner or not.
 - If the radius specified by cornerRadius is > linewidth/2, then cornerRadius is set to linewidth/2.

Generally, set cornerRadius equal to the corner rounding that is considered acceptable after OPC. For example, setting the radius to 0.10 microns instructs Calibre OPCpro to perform corrections such that corners are rounded on an arc of 0.10 microns. An example of this is shown in [Figure 6-6](#).

Figure 6-6. Corner Radius



cornerSiteStyle

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Required. Specifies the model used to place control sites on fragments adjacent to a corner.

Usage

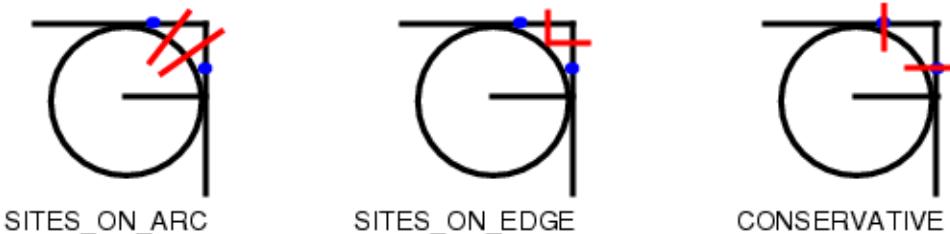
```
cornerSiteStyle mode [d] [concave mode [d]] [lea mode [d]] [orthog type]  
[corner_limit offset_mode action]
```

Arguments

- **mode**

A required argument specifying the default site placement mode for edge fragments within the corner radius. The modes have these basic forms:

Figure 6-7. Corner Site Styles



SITES_ON_ARC

Sites are created on an arc inscribed inside the corner, with the site direction normal to the arc. The radius of the inscribed arc is as large as possible, up to [cornerRadius](#) or linewidth/2. The site's center point is on the arc rather than the edge. This changes the location of zero EPE to the arc.

SITES_ON_EDGE [ignore_radius]

Sites are created perpendicular to the fragment. They are placed at the midpoint unless *d* is specified.

- **ignore_radius** — An optional keyword that prevents the corner radius from being used in determining a site's position. The only value used in determining the site's location is the value *d*, or the midpoint if *d* is not specified. This option may be useful when *d* is greater than the corner radius.

CONSERVATIVE [ignore_jogs [length]]

Sites are created on the fragment end point not touching the corner, perpendicular to the fragment. If the fragment at a convex corner is longer than [convexAdjDist](#), the fragment is not considered to be a corner fragment and thus the site is placed at the center of the fragment. Similarly, if it is a concave corner and the fragment is longer than [concaveAdjDist](#), then the site is centered rather than be pushed away from the corner.

- ignore_jogs [*length*] — Jogs are ignored if they are below a certain *length*. This causes sites to be placed at the center of a fragment rather than opposite a corner with a short jog. This only occurs when the jog is below the minimum length; if the jog is at or beyond the specified length, the corner is handled like any other (which means that the site is placed opposite the corner.)

If ignore_jogs is specified without a *length*, the default length (the [minjog](#) value) is used. If needed, a jog length value can be specified following ignore_jogs that indicates the maximum length of a jog to be ignored. For example, if “ignore_jogs 0.0035” is specified, all corners on jogs greater than 0.0035 microns in length are considered to be real corners, while all jogs less than or equal to that length are ignored (the site is centered as if there were no corner at all).

Note

 Any usage of ignore_jogs requires that a fragment has a jog at one corner and no corner of any kind on the opposite side (the fragment is connected to another fragment going in the same direction). This option has no effect on fragments with a corner at both ends, or no corners at either end.

- *d*

An optional argument used to specify the exact location of the site center, specified as a distance from the corner:

- For SITES_ON_EDGE, the site passes through the fragment at exactly this distance from the corner. See [Figure 6-8](#).
- For SITES_ON_ARC, a point is set at a distance *d* from the corner that intersects perpendicular a point on an arc. The site is then placed orthogonal to the point on the arc. You can change the rotation of the site by changing the arc radius using the [cornerRadius](#) keyword (to a maximum of linewidth/2). [Figure 6-9](#) shows an example of SITES_ON_ARC specifying a distance of 0.06 um.
- This argument has no meaning when mode is CONSERVATIVE.

When you specify a distance for SITES_ON_EDGE or SITES_ON_ARC, only the first site encountered within the corner distance parameter ([convexAdjDist](#), [concaveAdjDist](#), and [lineEndAdjDist](#)) is affected. All other sites are placed in their default positions.

Note

 If *d* is greater than the corner radius, the site may be placed short of where you indicated. For SITES_ON_EDGE, you can override this with the [ignore_radius](#) option.

Figure 6-8. SITES_ON_EDGE Specified With Distance

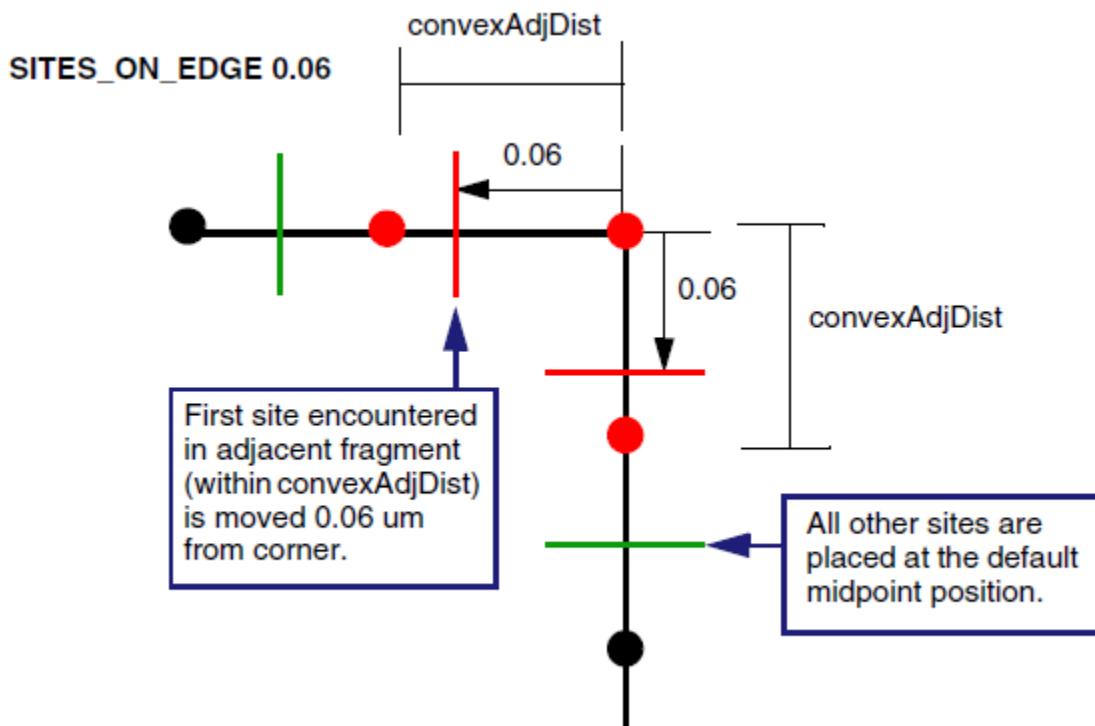
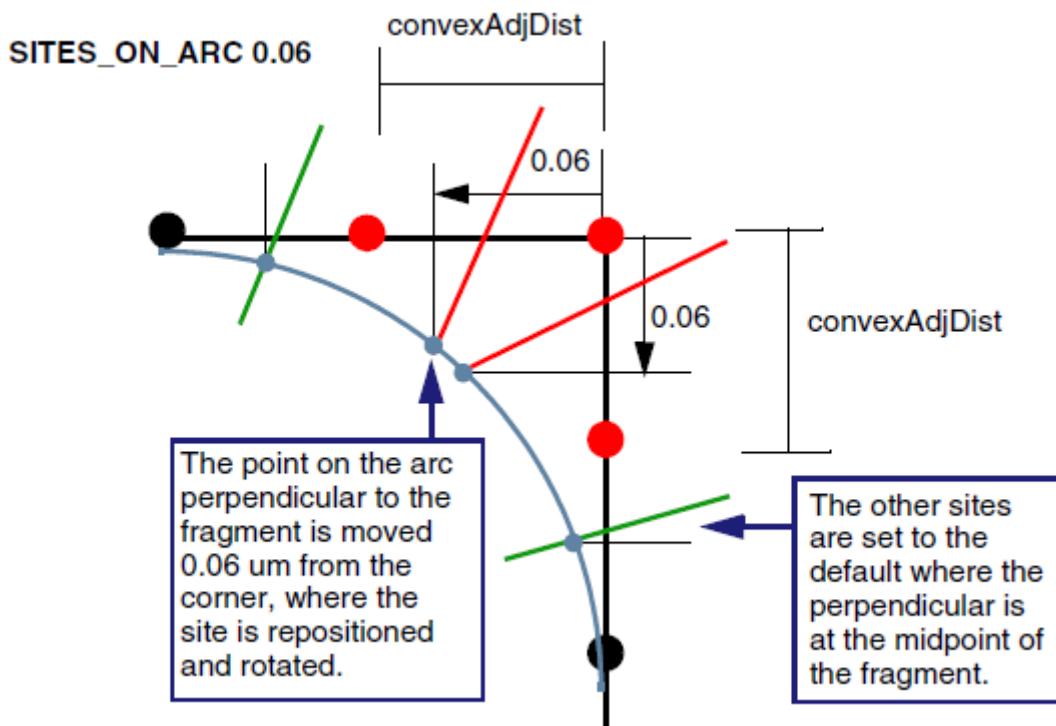


Figure 6-9. SITES_ON_ARC Specified With Distance



- **concave mode [d]**
An optional argument used to override the default site placement mode for fragments touching concave corners. The *d* option cannot be used with CONSERVATIVE.
- **lea mode [d]**
An optional argument used to override the default site placement mode for line-end adjacent fragments. The *d* option cannot be used with CONSERVATIVE.
- **orthog type**
An optional argument used to indicate that “site orthogonalization” should be attempted for the type of fragments specified by *type*. Site orthogonalization ensures that a site is created at angle normal to the aerial image itself, or in other words, along the image gradient. When orthog is specified, each site is rotated to match the gradient unless:
 - gradient is too small
 - slope is wrong direction

Note

 This operation adds approximately 15-30% overhead to the run time per iteration.

The *type* parameter specifies the type of fragments that are affected. Valid values are:

NONE — No sites are orthogonalized.

ALL — All sites are orthogonalized.

CORNER — All sites within corner radius are orthogonalized.

CONVEX — All sites within corner radius of a convex corner are orthogonalized.

CONCAVE — All sites within corner radius of a concave corner are orthogonalized.

Note that ALL causes all fragments to have their sites orthogonalized, not just fragments at corners. The default is NONE.

- **corner_limit offset_mode action**

An optional argument used to indicate limitations to account for corner offsets. Specify *offset_mode* to allow one of the following models of offsets:

EDGE — Describes a corner offset that goes beyond the end of the edge

EDGE_CENTER — Describes a corner offset that goes beyond the center of the edge

FRAGMENT — Describes a corner offset that goes beyond the fragment end.

NO_LIMIT — Any corner offset is allowed. If NO_LIMIT is selected, then you do not need to specify *action*.

Specify *action* for one of the following:

IGNORE — Corner offset is not used. The center point on the fragment is used instead.

SNAP — Places sites according to one of the following conditions set by *offset_mode*:

- If EDGE is used, place the site at the end of the edge.
- If EDGE_CENTER is used, place the site at the center of the edge.
- If FRAGMENT is used, place the site at the end of the fragment.

Description

The cornerSiteStyle command specifies the mode used to place control sites on fragments adjacent to a corner. It must be specified in the setup file.

Examples

Example 1

In the following example, the cornerSiteStyle keyword specifies the SITES_ON_ARC method, which allows corner rounding for all corners except concave corners, which have their sites placed on edge.

```
cornerSiteStyle SITES_ON_ARC concave SITES_ON_EDGE
```

Example 2

The following example causes all fragments with a corner on a jog of less than or equal to 0.0035 microns in length to center the site on the fragment. (The corner is ignored.) If a fragment has a corner on a jog of greater than 0.0035 microns in length, that fragment has its site placed opposite the corner with the jog.

```
cornerSiteStyle CONSERVATIVE ignore_jogs 0.0035
```

flaglayer

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Enables fragmentation operations on a per-layer basis, overriding the frag bit settings of the layer keyword.

Usage

flaglayer *layer* *flagName* {enable | disable}

Arguments

- ***layer***

A required argument specifying the name of the layer to be affected.

- ***flagName***

A required argument that indicates the type of fragmentation operation. It must be one of the following keywords:

| | |
|------------------------|------------------------|
| intrafeature_smalljogs | interfeature_smalljogs |
| intrafeature_convex | intersection_island |
| intrafeature_concave | interfeature_visible |
| intrafeature_lineend | intersection_visible |
| intrafeature_false | |

The intersection operations cause fragmentation to occur on other layers when *layer* is the asraf or visible layer (intersection_visible) or island layer (intersection_island).

- **enable | disable**

A setting indicating whether or not the fragmentation operation should be performed for this layer. Either enable or disable must be specified.

Description

This optional keyword controls the fragmentation operation for *layer*. These operations are normally controlled by the *frag* bit settings specified in the [layer](#) keyword. This keyword provides a more human-readable form for the same control. It does not change the bit settings but overrides them; the default operations must still be set using the layer command.

Examples

The following two lines both control intrafeature fragmentation at small jogs:

```
layer 3 TARGET 16 0 correction clear
flaglayer TARGET intrafeature_smalljogs enable
```

In layer, the value “16” enables intrafeature fragmentation for jogs, but turns off jogs for interfeature. The flaglayer command sets jog fragmentation only for intrafeature; interfeature small jogs are not changed.

fragmentLayer

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Overrides global parameter settings for a specified layer.

Usage

```
fragmentLayer layer [horizontal | vertical | non90] '{'  
    settings  
}'
```

Arguments

- *layer*

A required argument specifying the layer name to which the fragmentation parameters within this keyword apply.

- horizontal | vertical | non90

An optional argument used to indicate that *settings* within this block only apply to the specified edge types. Use only one per command.

- '{

settings

'}

A required sequence of lines containing fragmentation keywords with parameters. A single keyword is allowed per line. The lines must be enclosed in braces ({}).

The following fragmentation keywords are allowed in a fragmentLayer block:

Table 6-3. Supported Fragmentation Keywords in fragmentLayer Block

| | | |
|-----------------|----------------|---------------|
| coincidentLayer | interfeature | minedgeLength |
| concavecorn | intrafeature | minjog |
| cornedge | islandFragment | seriftype |
| interfeatLayers | maxedgeLength | visibleLayers |

Description

This keyword defines a fragmentLayer 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. When used, the fragmentLayer keyword must be placed at the end of the setup file, along with the tagging commands and the commands used to set environment variables.

The fragmentLayer syntax includes a direction argument to handle double dipole lithography simulations. This argument causes fragmentation to be applied only to edges in the specified direction. Edges in any other direction are not fragmented.

The direction specified can be “horizontal”, “vertical,” or “non90”. If it is omitted, it means “all directions” (the default). A layer may be listed multiple times to carry out fragmentation in different directions.

Directional fragmentation does have some limitations. It only applies to interfeature fragmentation, corner fragmentation, and maxedgelen length fragmentation. Any of the layer-based commands, like coincident layer, apply to all edges regardless of the direction specified.

Examples

Example 1 - Layer-Specific Ripples

```
#-----Fragmentation-----
minfeature 0.06
modelpath /test
opticalmodel defaultopt
resistpolyfile resist.mod
interfeature -ripples 2 -ripplelen 0.1 -interdistance 0.45

#-----Layers-----
background clear
layer 1 TARGET 17 0 opc dark
layer 2 CORRECTION 17 0 correction dark

#-----Arbitrary Commands-----
fragmentLayer CORRECTION {
    interfeature -ripples 1 -ripplelen 0.1 -interdistance 0.45
}
```

This example defines the following settings:

```
interfeature -ripples 2 -ripplelen 0.1 -interdistance 0.45
```

The fragmentLayer block modifies the interfeature -ripples default specifically for layer 2, setting the ripples to 1 rather than the default 2, as shown in the following example (the underline section indicates where a change occurred):

```
interfeature -ripples 1 -ripplelen 0.1 -interdistance 0.45
```

Example 2 - Directional Fragmentation

This example shows how directional fragmentation can be carried out on a poly layer with different fragmentation settings for edges in various orientations.

```
fragmentLayer POLY horizontal {
    cornedge 0.100 0.100
    concavecorn 0.100 0.100
    interfeature -interdistance 0.740 -ripplelen 0.200 -num 1
}
fragmentLayer POLY vertical {
    cornedge 0.120 0.120
    concavecorn 0.120 0.120
    interfeature -interdistance 0.740 -ripplelen 0.200 -num 1
}
fragmentLayer POLY non90 {
    cornedge 0.130 0.130
    concavecorn 0.130 0.130
    interfeature -interdistance 0.740 -ripplelen 0.200 -num 1
}
```

Related Topics

[coincidentLayer](#)
[interfeatLayers](#)
[islandFragment](#)
[visibleLayers](#)

gridsize

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Required. Specifies the internal grid size in microns.

Usage

gridsize *gridsz*

Arguments

- ***gridsz***

A required argument specifying gridsize in microns. The value must be greater than 0. The default is 0.001.

Description

This required keyword specifies the internal grid size, in microns. The grid size should normally be set to the database unit size in the layout file. Typically, this value is the reciprocal of the SVRF Precision setting.

Note

 You must include the gridsize keyword in your setup file, otherwise the RET tools exit with an error.

Related Topics

[Precision \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

gse

Type: [Litho Setup File Keywords](#). Used by Calibre WORKbench.

Returns the value of a litho variable.

Usage

gse [*variable*]

Arguments

- *variable*

An optional string that specifies the name of the litho variable.

Description

This command returns the value of a currently-set setup variable.

If no arguments are supplied, then the command returns a list of argument value pairs for all setup variables set within the current application run.

Unlike other litho setup file commands, this command appears in the Arbitrary Commands section of a setup file as it is used after variables are set.

Examples

```
gse ALLOW_SMALL_MOVE
```

Related Topics

[Litho Variables](#)

[sse](#)

inline_optical1

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Allows you to include an optical model inside your setup file, rather than as a separate file.

Usage

```
inline_optical1 '{  
    optical model parameters  
}'
```

Arguments

- '{
 optical model parameters
}'

The model description, which must be supplied using the [optical parameters file format](#), as described in the *Calibre WORKbench User's and Reference Manual*. The model description must be enclosed in braces ({}). The open brace must appear on the same line as the keyword. The close brace must be on a line by itself, directly following the last optical model parameter.

Description

Use this keyword to include an optical model inside your setup file rather than as a separate external file. The optical model is dynamically generated in memory, used, and then deleted.

To set the use of in-line optical models, set the [opticalmodel](#) keyword to “inline,” then place the contents of the optical text file between the braces ({}). The easiest way to obtain this data is to cut and paste it out of an optical model file or out of the Preview panel on the Optical Model Tool. You can also create it in a text editor following the optical parameters file syntax.

Examples

```
opticalmodel inline  
...  
inline_optical1 {  
    version 4  
    opticsystem 0.248 0.500 0.600  
    defocuslevels 9 0.000 0.100  
    approxorder 8  
    hoodpix 1.280  
    kerngrid 0.010  
    illumtype STANDARD  
    engine TCCcalc  
}
```

inline_optical2

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Allows you to include a second optical model inside your setup file rather than as a separate file.

Usage

```
inline_optical2 '{'  
    optical model parameters  
'}'
```

Arguments

- '{
 optical model parameters
'}'

The model description, which must be supplied using the [optical parameters file format](#), as described in the *Calibre WORKbench User's and Reference Manual*. The model description must be enclosed in braces ({}). The open brace must appear on the same line as the keyword. The close brace must be on a line by itself, directly following the last optical model parameter.

Description

Use this keyword to include an second optical model inside your setup file rather than as a separate file. You use a second optical model when using multiple masks for the same layer. This keyword is used in conjunction with the [secondOpticalmodel](#) keyword.

To set the use of in-line optical models, set the secondOpticalmodel keyword to “inline,” then place the contents of the optical text file between the braces ({}). The easiest way to obtain this data is to cut and paste it out of an optical model file or out of the Preview panel on the Optical Model Tool. You can also create it in a text editor following the optical parameters file syntax.

Examples

```
secondOpticalmodel inline  
...  
inline_optical2 {  
version 6  
opticalsystem 0.248 0.500  
defocuslevels 1 0.000 0.100  
approxorder 12  
hoodpix 1.280  
kerngrid 0.010  
illumtype COMPOSITE...  
}
```

inline_optical3

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Allows you to include a third optical model inside your setup file rather than as a separate file.

Usage

```
inline_optical3 '{  
    optical model parameters  
'}'
```

Arguments

- '{
 optical model parameters
'}'

The model description, which must be supplied using the [optical parameters file format](#), as described in the *Calibre WORKbench User's and Reference Manual*. The model description must be enclosed in braces ({}). The open brace must appear on the same line as the keyword. The close brace must be on a line by itself, directly following the last optical model parameter.

Description

Use this keyword to include a third optical model inside your setup file rather than as a separate file. You use a third optical model when using multiple masks for the same layer, such as when including substrate effects while modeling the implant layer. This keyword is used in conjunction with the [thirdOpticalmodel](#) keyword.

To set the use of in-line optical models, set the [thirdOpticalmodel](#) keyword to “inline,” then place the contents of the optical text file between the braces ({}).

Examples

```
thirdOpticalmodel inline  
...  
inline_optical3 {  
version 8  
approxorder 5  
hoodpix 1.280  
kerngrid 0.005  
kernel 0 TYPE gauss 0.0754 SCALE -0.563138 SQUARE no USESODCS no  
...  
kernel 4 TYPE gauss 0.0847 SCALE 0.780006 SQUARE no USESODCS no  
}
```

inline_resist

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Allows you to include a resist model inside your setup file, rather than as a separate file.

Usage

```
inline_resist [name] '{  
    model_parameters  
}'
```

Arguments

- '{
 model_parameters
}'

A required resist model description, which must be supplied using the [VT5 file format](#) as described in the *Calibre WORKbench User's and Reference Manual*. The braces are required for this keyword. The open brace ({) must appear on the same line as **inline_resist**. The close brace (}) must be on a line by itself, directly following the last parameter.

- *name*

An optional argument that specifies the name of the model for later reference in [bindModelToTag](#).

Description

Use this optional keyword to include a resist model inside your setup file, rather than as a separate file. This keyword is used in conjunction with the [resistpolyfile](#) and [processModel](#) keywords to define the contents of in-line resist models.

The braces ({ }), which are required for this keyword, must surround the text that would have been used as the contents of the .mod file for the resist model.

To use the model specified by this keyword, set resistpolyfile to “inline”. When using multiple process models, specify *name*. The processModel *modelname modelfile* commands are replaced by **inline_resist** *name* commands.

Note

 You should set up the in-line resist models prior to any additional operations in the setup file.

Examples

```
resistpolyfile inline
inline_resist vt5.mod {
    type VT-5
    ttermCount 1
    TTERM 0.25
}
```

interfeatLayers

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies interfeature fragmentation for a specified layer. This can only be used within a fragmentLayer command block.

Usage

interfeatLayers *layer* ...

Arguments

- *layer*

A required argument, specified one or more times, that lists the layer names whose corners define interfeature fragmentation, listed from highest priority to lowest priority. If you specify no layers, no interfeature fragmentation occurs.

Description

An optional user control for interfeature fragmentation. Use to add layers that cause interfeature fragmentation to occur within a [fragmentLayer](#) block.

Note

 This keyword can only be used within a fragmentLayer block.

This keyword defines a prioritized list of layers whose corners are used to trigger generation of interfeature fragmentation vertices on the layer to which the fragmentLayer block applies. The order of the layers in the keyword defines the priority. The first *layer* has the highest priority, the second has the second highest priority, and so on.

In this case, priority defines the order in which the fragmentation vertices are generated. Interfeature fragmentation caused by features on the highest priority layer is performed first. Fragmentation vertices caused by the interfeatLayers with the second highest priority are only generated if they do not violate [minedgelength](#) with respect to existing vertices, and so on.

interfeature

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Defines the fragmentation parameters that influence interfeature fragmentation.

Usage

Syntax 1:

interfeature off

Syntax 2:

interfeature [-num *x*] [-interdistance *y*] [-ripplelen *z*] [-rem *r*] [-shield *n*]
[-distancePriority {0 | 1}] [-ripplestyle {0 | 1}] [-split {0 | 1}] [-skipCorners] [-sym]
[-internalOnly | -externalOnly] [-shift *z*] [-cornerDist *w*]

Arguments

- **off**

In Syntax 1, a required keyword that specifies to turn off all interfeature fragmentation. This keyword cannot be combined with any other arguments.

- **-num *x***

An optional argument specifying the number of ripples. The default value for -num is $0.5 * (\lambda/NA) / \text{ripplelen}$, rounded to the nearest whole number. The maximum is ten (10).

- **-interdistance *y***

An optional argument specifying the maximum distance at which interfeature fragmentation occurs. This parameter is specified in microns. The value must be greater than 0, and less than the promotion radius. The default value for is $1.5 * (\lambda/NA)$.

- **-ripplelen *z***

An optional argument specifying the length of a single ripple edge in microns. The default for ripplelen is [minedgelength](#).

- **-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 does not occur. The value must be greater than or equal to 0. If not specified, the default value used is [minedgelength](#).

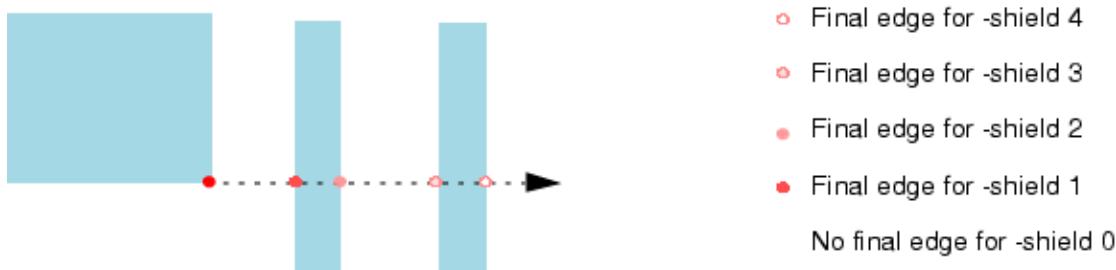
- **-shield *n***

An optional argument used to ensure that interfeature fragmentation can only occur if the edge undergoing fragmentation is shielded by less 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 set to 0, then shielding is turned off. The default value for -shield is 0. [Figure 6-10](#) shows an example of shielding.

If -internalOnly is specified, -shield is automatically set to 1 and cannot be changed.

Figure 6-10. Interfeature Shielding Example



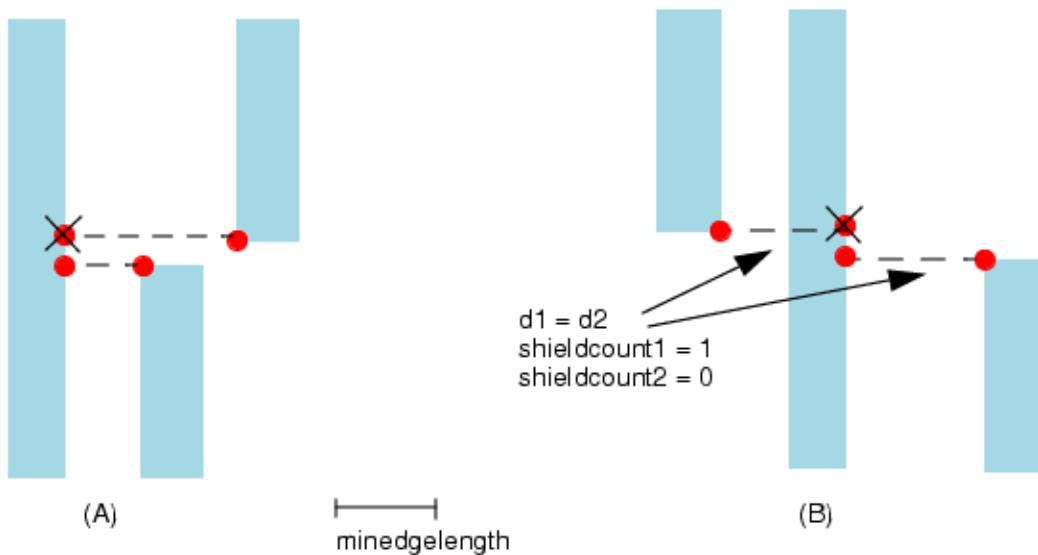
- -distancePriority {0 | 1}

An optional argument controlling how the application responds to situations in which multiple close features could cause fragmentation vertices. With this keyword you can choose either of:

- 0 — Based on order of processing, which cannot be determined ahead of time. This is the default.
- 1 — Based on the distance from the feature causing fragmentation; the feature closest to the fragment generates the new vertex.

When two features are equally distant from the fragment, the one with a smaller shielding count “wins.” [Figure 6-11](#) shows the effects of using the -distancePriority argument.

Figure 6-11. Using -distancePriority With Interfeature Fragmentation



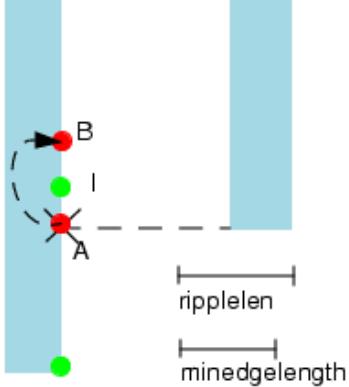
- `-ripplestyle {0 | 1}`

An optional argument used to control ripple generation.

Choose one of the following:

- 0 — All conflicts between fragmentation points are resolved according to the `-distancePriority` argument. Interfeature fragmentation points that were blocked by the `-rem` constraint do not generate ripples. This setting cannot be used with the `-sym` setting.
- 1 — Ripples are generated by *all* interfeature fragmentation points, whether successfully inserted or blocked by `-rem` constraints, as shown in [Figure 6-12](#). This is the default.

Figure 6-12. Example -ripplestyle



In this example, fragmentation point A is not inserted due to a `minedgelength` violation with intrafeature fragmentation point I.

Fragmentation point B, however, ripples out from point A and is inserted when `-ripplestyle 1` is specified.

If `-ripplestyle 0` is specified, then fragmentation point B is not be inserted.

`interfeature -ripplestyle 1 -ripples 1`

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:
 - i. The distance between the fragmentation point and the fragmentation-causing corner.
 - ii. The number of shielding edges between the fragmentation point and the fragmentation causing corner.
 - iii. The distance from the fragmentation point to the middle of the fragmented edge.
 - iv. 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 the corners are equal, the fragmentation point with a smaller number of shielding edges is inserted.

- The priority of ripples from interfeature fragmentation points is determined (in order of importance) by:
 - i. The distance from a ripple to the interfeature fragmentation point that the ripple originated from.
 - ii. 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:

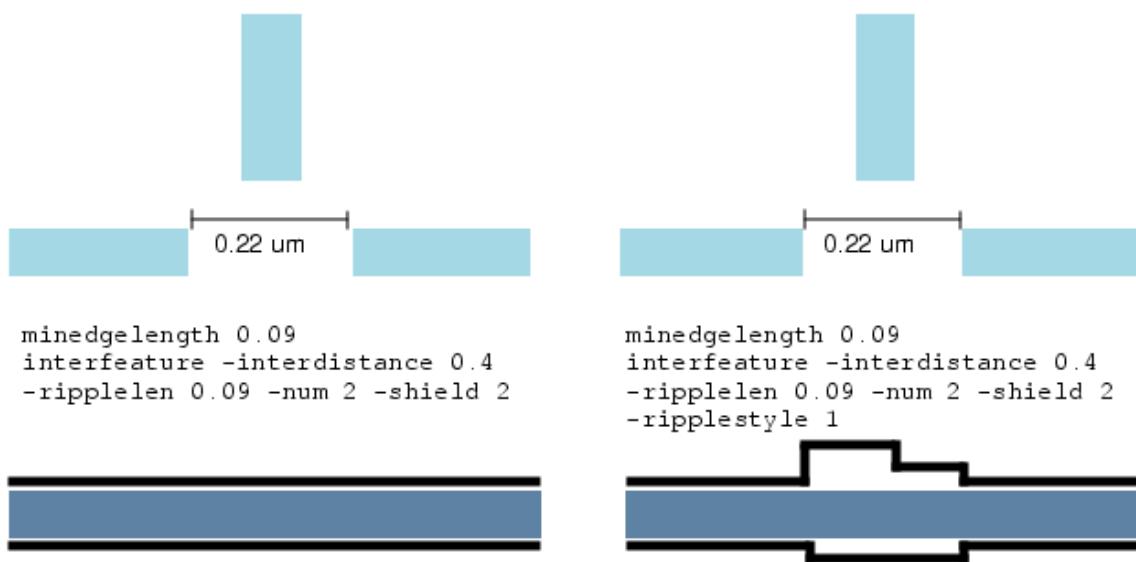
- Ripple points of the same priority are divided into sets such that there are no -rem conflicts between ripple points in different sets and there are -rem conflicts between neighboring ripple points inside each set.
- Each set is processed according to the size of the set:

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 their distance from the first point in the edge.

Without the -ripplestyle 1 setting, the interfeature fragmentation results can be far less precise, as illustrated in [Figure 6-13](#).

Figure 6-13. Differences Between -ripplestyle 0 and -ripplestyle 1 Settings

- `-split {0 | 1}`

An optional argument controlling the splitting of fragments with the following possible values:

- 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. See (A) in [Figure 6-14](#).
- 1 — Specifies splitting into two equal fragments any fragment between interfeature projection points with:
 - $(2 * \text{remainder}) \leq \text{length} < (2 * \text{ripplelen} + \text{remainder})$ if -ripplestyle 0 is selected.
 - $(2 * \text{ripplelen}) \leq \text{length} < (2 * \text{ripplelen} + \text{remainder})$ if -ripplestyle 1 is selected.

Note

 If the -ripplestyle is set to 1, -split is set to 1, overriding the specified -split user setting in the setup file.

Figure 6-14. -split Example



- **-skipCorners**

Interfeature fragmentation can interfere with intrafeature fragmentation. Fragmentation from the cornedge or concavecorn 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 6-17](#) in “[Example 1 - skipCorners and cornerDist](#)” on page 194.) Use this option to preserve corner fragmentation.

- **-sym**

An optional argument indicating fragmentation should be symmetric around a fragment-generating line end. Fragments are [minedgelength](#) in size and only occur on horizontal or vertical edges. See “[Example 2 - Scattering Bars](#)” on page 195 for an example of how this option changes the output.

The -sym argument cannot be used with -ripplestyle 0.

- **-internalOnly**

An optional argument that limits interfeature fragmentation to the initiating polygon and sets -shield to 1. Corners from one polygon do not cause interfeature fragmentation on another polygon. The shielding prevents the corner from generating fragments on edges of the polygon that are separated from the corner by a space.

You cannot specify -internalOnly with -externalOnly.

- **-externalOnly**

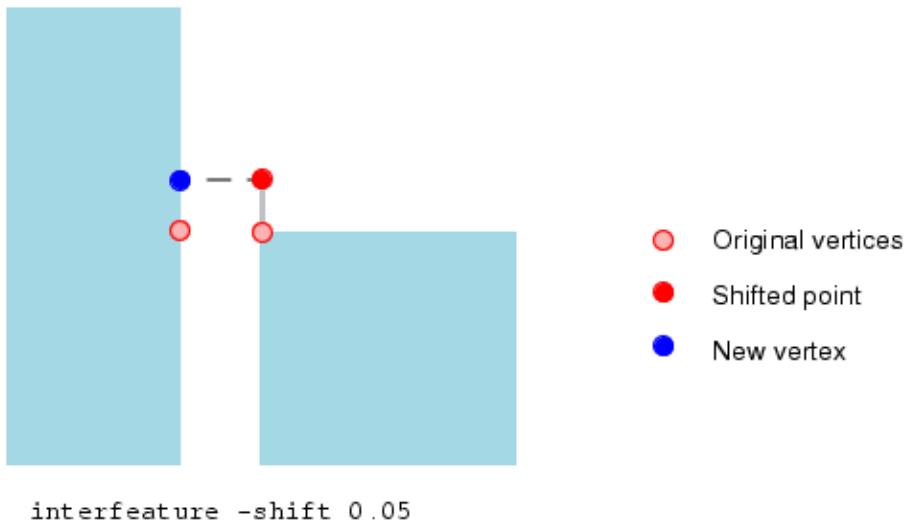
An optional argument that limits interfeature fragmentation to polygons other than the one the initiating corner is on. This option cannot be specified with -internalOnly.

- **-shift z**

An optional argument that shifts the coordinates of a vertex-generating corner a distance *z* in microns. The point is shifted in both the X and Y directions, creating two points. Only the point that has moved parallel to the receiving edge, though, creates interfeature fragmentation. (See [Figure 6-15](#).)

Positive numbers project the generating point outward. Negative numbers move the generating point along the edges of the projecting shape. The receiving edge is then fragmented across from the shifted point. The default is to not shift coordinates.

Figure 6-15. -shift Example



It is recommended that the `-shift` option only be used in a `fragmentLayer` block to minimize unexpected results.

- `-cornerDist w`

An optional argument that indicates the minimum distance from a corner at which a vertex may be inserted. If the vertex is less than distance w in microns from a corner, it is not inserted. If the distance is equal to or greater than w , the vertex is inserted.

In contrast, `-skipCorners` blocks all interfeature fragmentation on a fragment that has one endpoint on a corner without regard to distance. See “[Example 1 - skipCorners and cornerDist](#)” on page 194.

Description

This keyword defines the fragmentation parameters that influence interfeature fragmentation.

Fragmentation is the process of breaking a layout polygon’s edges (or parts of edges) into smaller segments by adding vertices. The purpose of fragmentation is to prepare the layout for correction by creating more edge fragments wherever the layout needs more correction.

Interfeature fragmentation adds vertices based on proximity to other features. The interfeature fragmentation algorithm adds new vertices to edges that meet the following criteria:

- The distance from the edge to the corner causing the fragmentation is less than or equal to the distance specified by `-interdistance`.

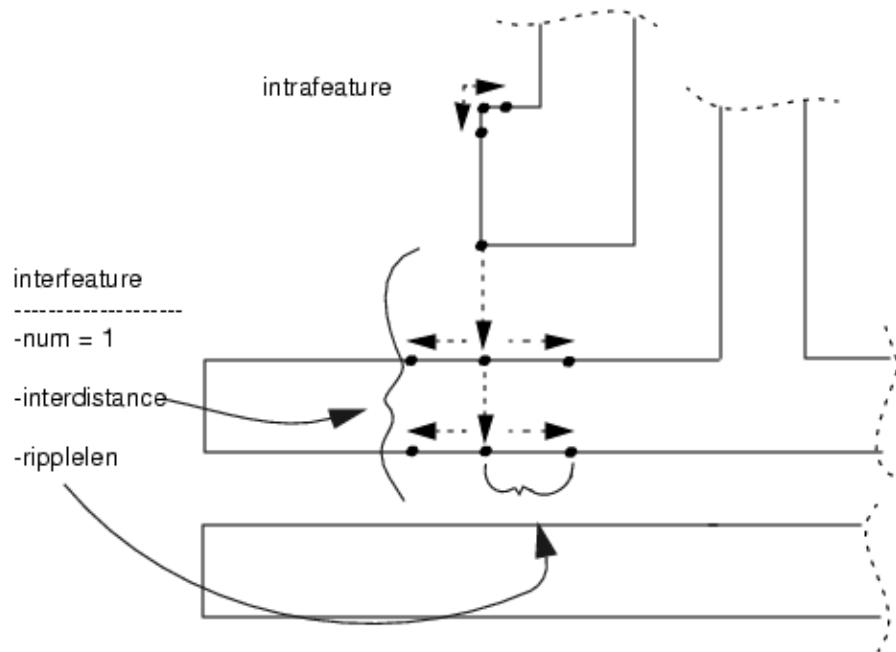
- None of the fragments that would result from fragmentation is shorter than the distance specified by -rem.
- The edge is separated from the corner by fewer than n edges, where n is specified by -shield.

After the initial breakpoint is inserted, additional vertices are inserted on both sides of the original point, with the length of the fragments formed defined by the -ripplelen parameter.

If none of the optional arguments are specified, the interfeature fragmentation algorithm calculates appropriate values.

The diagram in [Figure 6-16](#) shows how the interfeature fragmentation occurs.

Figure 6-16. Using Shield With Interfeature Fragmentation



Examples

Example 1 - skipCorners and cornerDist

In this example, the cornedge command forms two fragments on the convex corner. The following interfeature command places two additional fragment points in between the original fragment formed with cornedge.

```
cornedge 0.1 0.05
interfeature -interdistance 0.13 -skipCorners
```

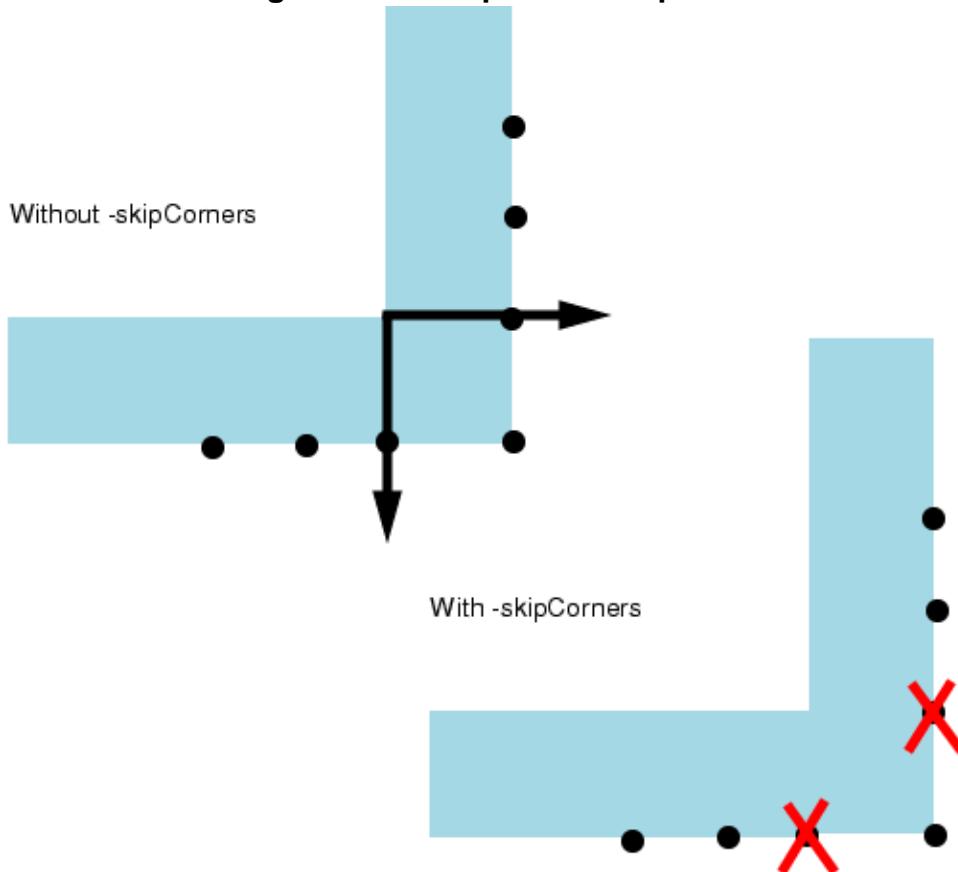
However, the -skipCorners option is enabled in this case, to prevent from breaking the initial set of intrafeature frags.

Another way to achieve the same effect in this specific instance is by setting -cornerDist to the *len1* of cornedge:

```
cornedge 0.1 0.05
interfeature -interdistance 0.13 -cornerDist 0.1
```

In general, use -skipCorners if intrafeature fragmentation runs first, as 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 you do not want further fragmented.

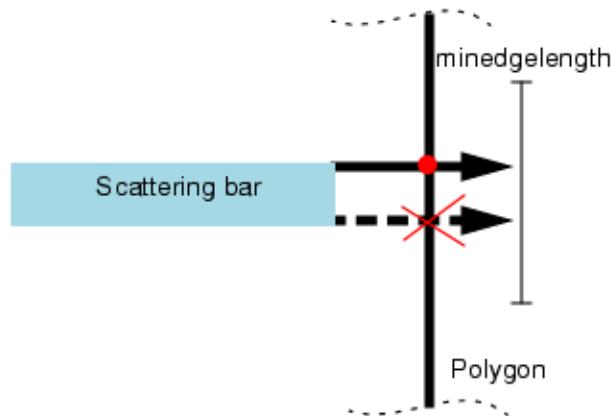
Figure 6-17. -skipCorners Option



Example 2 - Scattering Bars

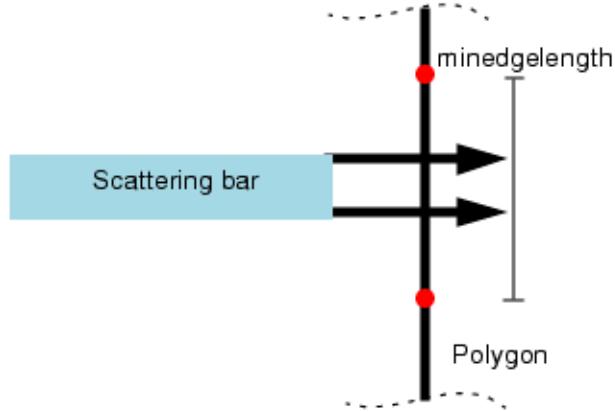
Scattering bars are generally quite thin, usually less than the minimum edge length. This means that when the corners of a scattering bar fragment the edge of a polygon, one corner generates a break point but the other does not because the fragment would violate the minimum length as shown in Figure 6-18.

Figure 6-18. 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 6-19](#). (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 6-19. Symmetric Fragmentation Caused by a Scattering Bar and -sym



Related Topics

[layer](#)

intrafeature

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Alternative to cornedge and concavecorn for implementing intrafeature fragmentation.

Usage

intrafeature *edge_spec* [fragment] *rule* [corner2 [fragment] *rule*] [breakinhalf]

Arguments

- ***edge_spec***

A required set of keywords that defines the corner type and fragment lengths. The *edge_spec* argument can be broken down into the following syntax:

corner_type [*corner_subtype*] [*corner_type* [*corner_subtype*]] [*length constraint*]
[*length1 constraint*] [*length2 constraint*]

corner_type

A required argument that specifies a constraint that determines whether the rule applies to one or both corners of a particular edge. The allowable corner types are **concave** and **convex**.

corner_subtype

An optional argument that specifies a constraint that determines whether the rule applies to one or both corners of a particular edge. The choices are **end_adjacent**, **not_end_adjacent**, **angle_90**, and **angle_non90**. The **end_adjacent** subtype specifies that the corner satisfies the line-end-adjacent or space-end-adjacent criteria. The **angle_90** subtype specifies that the angle is a 90 degree or 270 degree angle.

length constraint

length1 constraint

length2 constraint

Length of the edge, the edge at corner 1, and the edge at corner 2, respectively. All three may be specified jointly or independently of each other. The constraint takes the form of a DRC constraint and the value is specified in microns. For example:

```
length > 0.1 <= 0.2  length1 <= 0.15  length2 < 0.2
length > 0.2
length == 0.3  length2 > 0.18
```

- **fragment**

An optional keyword that can be used to separate the fragmentation rule from the edge specification. It can be useful if length is specified so that the length value is separated from the fragmentation values.

- ***rule***

Defines the rules for intrafeature fragmentation. The *rule* parameter has two forms.

- **len1 ... lenN** [priority] [rem *r*]
- **split *n*** [force]

len1

The default distance in microns from the concave corner to the first fragmentation vertex that is produced. The value must not be less than [minedgelength](#).

len*N*

The default distance in microns from the previous fragmentation vertex to the next fragmentation vertex, measured sequentially from the corner inward. The value must not be less than [minedgelength](#).

priority

An optional flag 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.

rem *r*

An optional argument specifying the minimum allowed remainder. If intrafeature fragmentation would produce an edge with length smaller than *r*, then it does not fragment the edge. If not specified, the default value used is [minedgelength](#).

split *n*

An optional keyword and value specifying to break the fragment into *n* equal parts. For example, if the following command is applied to a 0.4 micron edge:

intrafeature convex length <= 0.685 split 4

the edge is split into four 0.1 micron fragments.

The *n* value must be an integer between 2 and 40, inclusive. If the resulting fragments would violate [minedgelength](#), the command generates an error. The above example would likely generate an error since lengths down to 0 would be split into four fragments. You can prevent this by providing a lower bound, as in this example:

```
minedgelength 0.050
intrafeature convex length >= 0.200 <= 0.685 split 4
```

force

An optional keyword for use with split. It overrides the [minedgelength](#) error. Use only when you are certain that violating [minedgelength](#) does not cause an error.

- **corner2**

If corner2 is specified, it must be followed by a fragmentation rule for the second endpoint. It is only legal to specify corner2 if there are two corner-types listed and if they are not ambiguous. For example, if both corner types are convex, then it is not possible to distinguish between the two and therefore the fragmentation must be the same for both.

- breakinhalf

If breakinhalf is specified, it divides a fragment into two equal parts when the resulting intrafeature fragmentation violates the remainder constraint.

For example, if the following command is applied to a 0.12 micron edge:

```
intrafeature 0.05 0.05 rem 0.04 breakinhalf
```

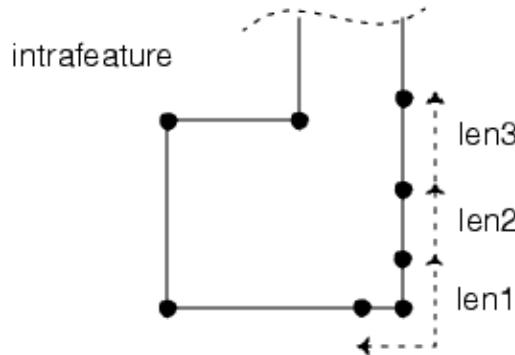
A 0.12 micron edge cannot have ripples inserted without violating the remainder constraint of 0.04 microns. Normally, no changes would be made to the edge. However, since breakinhalf was specified, the edge is split in the middle to create two fragments 0.06 microns in length.

Description

This keyword is an optional fragmentation parameter. It combines the functionality of both [cornedge](#) and [concavcorn](#) parameters and can be used to implement intrafeature fragmentation with a description of the edge and a rule for generating fragments. The edge description takes into account the edge length and the type of corners that it contains.

In intrafeature fragmentation, the edges connected to corners are broken up according to the cornedge and concavcorn parameters. The fragmentation process occurs automatically before OPC begins. [Figure 6-20](#) shows how the intrafeature fragmentation parameters are applied to produce fragmentation vertices.

Figure 6-20. Intrafeature Fragmentation Example



Interaction With Hierarchy

Be careful in hierarchical designs to take promotion radius into account. When two corners are specified in conjunction with a length greater than the automatically calculated promotion radius or the manually set [INTERACTION_DISTANCE](#) value, the software cannot always determine what type the other corner is. When this happens, an error occurs. The solution is to have two forms of the rule: one for lengths up to the promotion radius and the other for lengths greater than the promotion radius.

For example, if the promotion radius is 0.91 this rule occasionally generates errors:

```
intrafeature concave concave length > 0.8 fragment 0.15 0.15
```

The problem occurs when the two corners are separated by an edge greater than 0.91 microns. The context of the first corner does not include the second corner, so Calibre cannot determine its type. A better setup file would use this pair of rules:

```
intrafeature concave concave length > 0.8 <= 0.91 fragment 0.15 0.15
intrafeature concave length > 0.91 fragment 0.15 0.15
```

This permits correct fragmentation whether or not the edge is interrupted by a boundary and generates the same results in both flat and hierarchical mode.

The promotion radius is listed in the log file as “promotion radius” whether calculated automatically or set with the INTERACTION_DISTANCE variable.

Related Topics

[concavecorn](#)

[cornedge](#)

[layer](#)

[OPC_USE_LE_DEF](#)

islandFragment

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Controls island layer fragmentation.

Usage

```
islandFragment layer_name [-ripples x] [-ripplelen z] [-rem r] [-enforce {0 | 1}]  
[-rippleinside | -rippleboth] [-dontcross] [-offset offset_value]
```

Arguments

- *layer_name*

A required argument that specifies the layer name. The layer must be defined as island or opc in the [layer](#) statement.

- -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 the shortest legal value, [minedgelength](#).

- -rem *r*

An optional argument specifying the minimum allowed remainder for ripple generation. If the ripple would produce an edge with length less than *r*, the vertex is not inserted. The default value is [minedgelength](#). The remainder must be greater than 0.

- -enforce {0 | 1}

When set to 1, a fragment is broken at intersection points only if no fragments of lengths less than [minedgelength](#) are created. If not present, fragmentation can result in [minedgelength](#) violations. The default value is 0.

- -rippleinside

Generates ripples toward the inside of the islandFragment layer polygons at any intersection. The default is to only generate ripples where coincident edges are present.

For example:

```
fragmentLayer POLY {  
    islandFragment 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 are generated, and no fragments less than [minedgelength](#) in length are created.

Either -rippleinside or -rippleboth may be specified, but not both.

- **-rippleboth**

Generates ripples toward both the inside and the outside of the islandFragment layer polygons at any intersection. The default is to only generate ripples where coincident edges are present.

For example:

```
fragmentLayer POLY {  
    islandFragment 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 are generated, two ripples of length 0.04 microns towards the outside of the DIFF layer polygon are generated, and no fragments less than minedgelength in length are created in either direction.

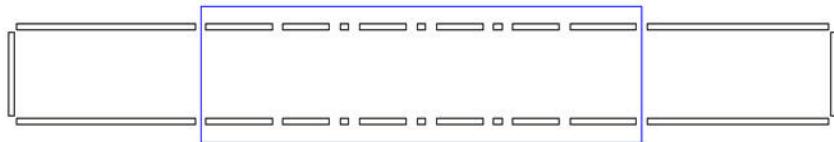
Either -rippleinside or -rippleboth may be specified, but not both.

- **-dontcross**

This option prevents ripples from crossing an island layer, or crossing one another.

[Figure 6-21](#) shows an example of what happens when this option is not specified.

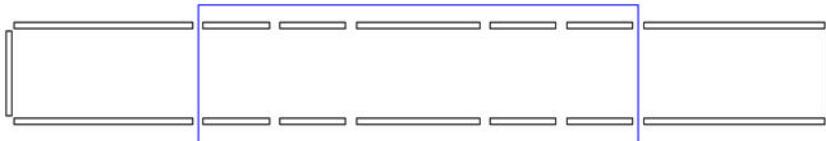
Figure 6-21. Without -dontcross Option



In [Figure 6-21](#), the island layer shown in blue has fragmented the layer shown in black with these options:

```
fragmentTag all intersection from island_layer -ripples 4 \  
    -ripplelen 0.07 -rem 0.0005 -enforce 0 -rippleinside
```

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 6-22](#) shows an example of what happens when this option is specified.

Figure 6-22. 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.

- **-offset *offset_value***

An optional argument specifying an offset for all ripples. Normally, an island break occurs at exactly 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 exactly 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 an offset is specified versus when an offset is not specified.

Description

This keyword controls island layer fragmentation. Island layer fragmentation inserts fragmentation points where edges on the specified island or opc layer intersect edges on an opc or correction layer.

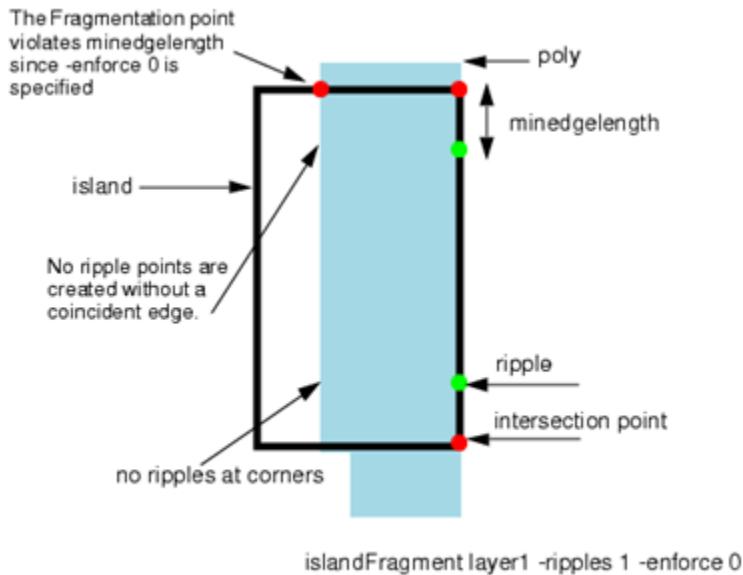
If **-enforce 0** is specified, fragmentation points are created unconditionally at intersections as described previously.

If **-enforce 1** is specified, fragmentation points are inserted only if no **minedgelength** 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.

Figure 6-23. Island Layer Fragmentation Example



Examples

This example covers the effects of using island fragmentation with and without an offset. [Figure 6-24](#) shows the effect of the setup file excerpt below. The island layer is marked by the blue hatched region while the resulting fragment breaks on the OPC layer are shown in black. Note that two ripples were created in the OPC layer, one on either side of the island layer crossing, and there is a break centered on the crossing.

```
islandFragment island_layer -ripples 1 -ripplelen 0.07 -rippleboth
```

The next example, [Figure 6-25](#), shows how a specified offset effects fragmentation position.

```
islandFragment island_layer -ripples 1 -ripplelen 0.07 -rippleboth \
-offset 0.035
```

Figure 6-24. Island Fragmentation Without an Offset

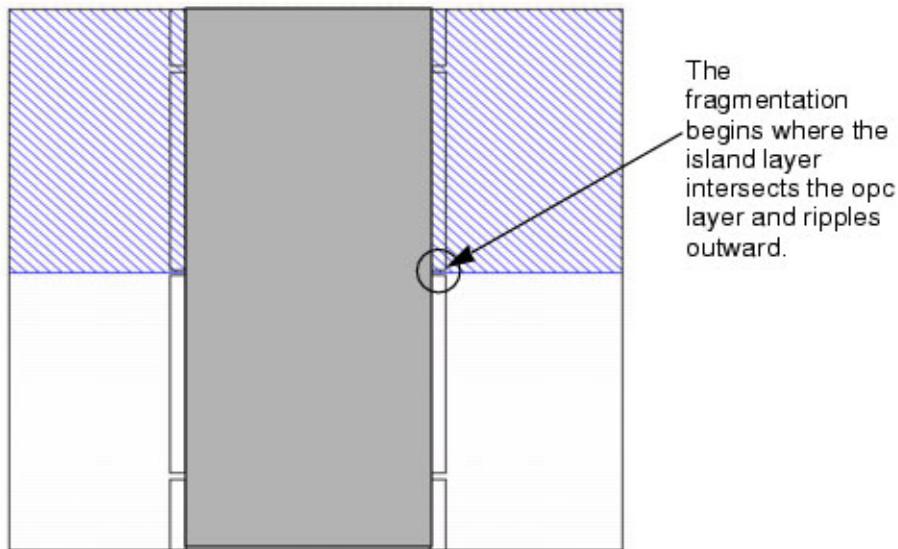
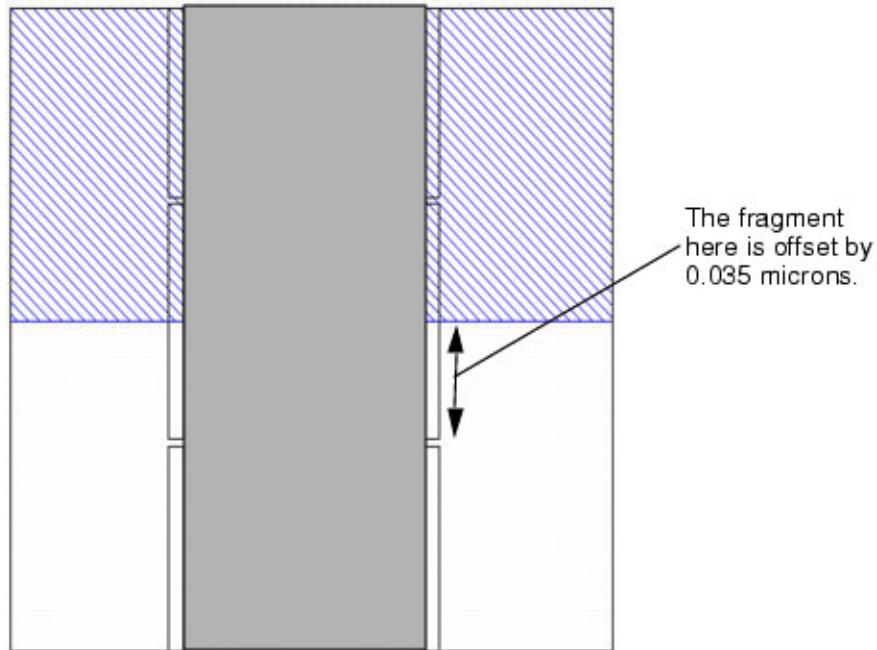


Figure 6-25. Island Fragmentation With an Offset



Note that only one ripple was created, and there is no break where the island intersects the opc layer. Instead, there is just one break on either side of the island crossing offset 0.035 microns. A break occurs on both sides of the island crossing as specified by the -rippleboth option.

iterations

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies how many iterations of OPC movement to perform on the design.

Usage

iterations *number*

Arguments

- ***number***

Number of iterations. Iterations only apply to edge movement operations. For Calibre ORC and Calibre PRINTimage this value is ignored. The default value for Calibre WORKbench is 4. The value must be greater than or equal to 0.

Description

This keyword 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 [stepsize](#) keyword's *stepsz* argument in microns, according to the simulations models. The iterations are run at the end of the tagging script, assuming that no [fastIter](#) commands are present in the tagging script. If any fastIter commands are present, this keyword is ignored, as though *number* were set to 0.

layer

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Required. Defines how the given mask layer is treated by OPC and simulation.

Usage

```
layer layer_num[.datatype] layer_name frag ctl layer_type transmission [mask_num dose]  
[bias x_value y_value]
```

Arguments

- ***layer_num[.datatype]***

A required number associated with the layer. The number can be either a positive integer or include a decimal portion to indicate a particular datatype layer.

When using Calibre WORKbench Litho File Tool or RET Flow Tool, the layer number should match the desired design layer number. For the Calibre LITHO operations in this manual, the layer number is arbitrary, and the layer name is used to match the desired Calibre layer.

- ***layer_name***

A required argument specifying the name of the layer.

Layer names must be alphanumeric strings less than 20 characters in length and must match the name that was assigned to the layer in the SVRF file from which the litho toolset is invoked.

- ***frag***

A required decimal value that defines the types of fragmentation to be performed.

To manually calculate the ***frag*** value, add the values shown in [Table 6-4](#) for all fragmentation options that apply. For example, to select the fragmentation options “turn off interfeature fragmentation for jogs” (16) and “turn off intrafeature fragmentation for jogs” (1) add the values together: 16 + 1 = 17. The ***frag*** value would be 17.

While this argument is required for all layers, non-zero values are meaningful only for layers that can either be fragmented or cause other layers to be fragmented. Some ***frag*** values control the type of fragmentation performed on the layer. Other values control the type of fragmentation caused by the layer. The last column in [Table 6-4](#) indicates the types of layers for which each bit is applicable.

Table 6-4. layer Keyword frag bit Functionality

| Bit | Turns... | Type of Fragmentation | On | Applies to |
|-----|----------|----------------------------|---------------|--------------------|
| 0 | ON | Full fragmentation | all fragments | all layers |
| 1 | OFF | Intrafeature fragmentation | jogs | opc, correction |

Table 6-4. layer Keyword frag bit Functionality (cont.)

| Bit | Turns... | Type of Fragmentation | On | Applies to |
|-----|----------|---|--|--------------------|
| 2 | OFF | Intrafeature fragmentation | convex corners | opc, correction |
| 4 | OFF | Intrafeature fragmentation | concave corners | opc, correction |
| 8 | OFF | Intrafeature fragmentation | fragments adjacent line ends | opc, correction |
| 16 | OFF | Interfeature fragmentation | jogs | opc, correction |
| 32 | OFF | Island layer fragmentation (at intersections) | fragments overlapping this layer | island |
| 64 | ON | Intrafeature fragmentation | false edges | opc, correction |
| 128 | ON | Interfeature fragmentation | caused by this layer | visible, asraf |
| 256 | OFF | Visible layer fragmentation (at intersections) | fragments overlapping this layer | visible, asraf |

- *ctl*

A legacy argument included to provide backwards compatibility with existing setup files. This argument should be set to 0.

- *layer_type*

A required parameter that specifies how the layer is used by OPC and the aerial image simulator. [Table 6-5](#) lists the valid *layer_types* for quick reference with detailed explanations following.

Table 6-5. Valid layer_types

| | | |
|---------|------------|----------|
| asraf | correction | external |
| hidden | island | opc |
| visible | | |

asraf

Asraf layers contain geometries representing holes cut into the pre-opc layer. Usually the holes serve as anti-SRAFs, also known as negative scattering bars. Asraf layer geometries do not receive correction but do appear on the mask and affect the final printed design. Edges are not fragmented and do not receive sites.

The asraf layer must have the same **transmission** value as the opc layer.

Unlike visible layers, geometries on asraf layers are subject to MRC rules.

correction

Correction layers contain mask geometries to be corrected by OPC. If present, correction layers have their edges fragmented, and the fragments on the correction layer are moved instead of moving edges on the opc layer.

To accomplish this, the correction layer edge fragments are mapped to the nearest opc layer fragment and moved to correct EPEs calculated with respect to the opc layer fragment. The mapping occurs only when the midpoints of the two fragments are separated by a distance less than a user-defined search range. You specify the search range using the [OPC_CORR_SEARCH_DISTANCE](#) setup variable.

Correction layers are used primarily for performing incremental OPC or performing OPC on multiple masks.

external

External layers contain corrected mask data. You use external layers for simulation. If present, external layers are evaluated for EPEs during ORC or be simulated by PRINTimage. In ORC, EPEs are measured between the simulation of the external layer and the OPC target layer. External layers are not used by OPCpro.

When simulating areas in which visible or asraf layers and external layers overlap, the tools add the mask transmission values for the overlapping areas.

hidden

Hidden layers do not have any impact on any simulations and should not be passed to any of the litho tools. These layers serve an important function in Calibre WORKbench and Calibre LITHOview by providing a means of “turning off” data that should not be factored into a simulation.

Never include a hidden layer in the SVRF LITHO statement layer list.

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.

opc

Opc layers contain the actual mask geometry that you intend to transfer onto silicon. An opc layer

- has its edges fragmented
- has control sites installed and EPEs evaluated at those sites
- has its edges moved during OPC if no correction layers are present
- is used to generate the print image if no external layers are present

At least one opc layer must be specified in the setup file. Using additional opc layers for multiple layer corrections in a single pass can also be performed.

visible

Visible layers contain geometries that do not receive correction yet appear on the mask and interact with other geometries to affect the final printed design. The data on visible layers can be non-printing data, such as scattering bars, or can be printing data that does not need correction. Visible layers are not fragmented and do not receive any sites.

By default, visible layers do not induce fragmentation on the opc layer. However, you can override this behavior by setting the *frag* argument on a visible layer to 128. A visible layer that induces fragmentation also induces fragmentation on jogs, regardless of how frag bit #4 is set on the opc layer.

Each *layer_type* has unique well-defined properties. [Table 6-6](#) lists the OPC and simulation properties associated with each layer.

Table 6-6. Layer Properties Matrix

| Type | Optional or Required | Fragmented & Tagged | Edges moved during OPC | Sites on Edges | Used by the simulator to compute Aerial Image |
|------------|----------------------|---------------------|---|----------------|--|
| asraf | Optional | no | no | no | yes |
| correction | Optional | yes | yes | no | yes |
| external | Optional | no | no | no | yes |
| hidden | Optional | no | no | no | no |
| island | Optional | no | no | no | no |
| opc | Required | yes | yes if no correction layers are present. | yes | yes, if no correction or external layers are present. |
| | | | no if correction layers are present. | | no if correction or external layers are present. |
| visible | Optional | no | no | no | yes, if no external layers are present. |

- ***transmission***

A required argument that defines the mask layer's optical transmission type. Up to 50 values can be defined for visible layers.

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 [background](#) keyword in this chapter for more information about physical layer transmission in relation to background. [Table 6-7](#) lists the valid ***transmission*** choices.

Table 6-7. layer Keyword transmission Argument Settings

| 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 attenuated <i>factor</i> , they translate the factor into a real imaginary pair such that the attenuated background has a transmission value: $RE(layer) = -\sqrt{percent/100}$ $IM(layer) = 0$ The maximum allowed attenuation factor is 36. |
| clear | Defines the layer's optical transmission to be 100%. (clear features) |
| dark | Defines the layer's optical transmission to be 0%. (dark features) |
| phase60 phase90 phase120 phase180 phase270 | Defines the layer to be a phase shifting layer transmitting light with the given phase relative to the incident light. |
| <i>real imag</i> | Defines the layer's optical transmission to be as specified by the <i>real</i> and <i>imaginary</i> value pair. Up to 50 different transmission values can be defined for visible layers. Note that the attenuated syntax for the transmission is recommended over the <i>real imag</i> pair syntax. |

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,IMAGINARY pair is resolved relative to the background pair. The layer transmission value is resolved according to the following equation:

$$\text{background } (\text{real imag}) + \text{layer } (\text{real imag}) = \text{transmission value}$$

For example:

```
background 1 0
number name frag ctl opt type
1       chrome 19   0     opc   0 0
```

The background is specified with a (1, 0) pair, while chrome layer 1 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).

For example:

```
background 1 0
number name frag ctl opt type
1       p180  12   0     visible -1 0
```

In this case, the layer 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).

- *mask_num dose*

An optional parameter pair used with phase shifting masks to allow the simulators to support multiple masks.

mask_num — The number of the mask to which the layer belongs. The default is mask 0. When multiple masks are used, they must be numbered sequentially, beginning with 0. The opc layer must appear on mask 0.

dose — The layer's light energy dose, expressed as a percentage of the full energy dose for the mask. The *dose* is specified as a positive real number. For example; 0.5 is 50% of the full energy dose. Values greater than 1 represent over exposures. Values less than 1 represent under exposures. In most cases, the dose setting used to build a calibrated VT5 model should be the same dose setting used when running Calibre OPCpro, Calibre ORC, or Calibre PRINTimage with the model.

- *bias x_value y_value*

Optional parameters that specify a bias to be applied before optical simulations are performed. Biases must be specified in microns. A negative bias moves the edge inward. A positive bias moves the edge outwards. If bias is used, both x and y bias values must be supplied. Different biases per layer is allowed, even for layers of the same transmission value.

Such biases do not effect the position of your mask edges, rather they perturb locally the mask transmission by changing the edge positions of the layer before it is input to the optical simulation. An example usage for this is a specification of a positive bias to a layer containing assist features that causes them to behave optically as if they were larger. These biases, applied before optical simulations are sometimes adequate approximations to modeling 3D electromagnetic field (EMF) mask effects. The bias values must be determined after a careful study of the actual 3D EMF effects.

The *x_value* applies to vertical edges and represents a bias in the X-direction. The *y_value* applies to horizontal edges with movement in the Y-direction. Both values are in microns. Edges that are non-Manhattan receive a bias equal to the average of x and y. Movement occurs orthogonally along the edge.

Description

The layer keyword assigns a layer number to the specified layer and defines how OPC and simulation treat the layer. Every layer specified in the LITHO command must be declared in the setup parameters, using this keyword. Also, any layer declared in the setup parameters must be specified in the LITHO command.

Examples

Example 1 - Pre-simulation Bias

This example shows how a pre-simulation bias is applied to the opc layer.

```
# ----- Layer info -----
background clear

layer 16 OPC_PROC      0 0 opc      atten 0.06 bias -0.0065 0.0000
layer 17 OPC_VISIBLE   128 0 opc      atten 0.06
layer 18 SRAF          0 0 visible  atten 0.06
layer 19 SRAF_ISLAND   0 0 island   atten 0.06
```

Example 2 - Anti-Scattering Bars

Previous versions required several lines of code to handle negative scattering bars. The asraf layer introduced in version 2009.4 makes this simpler. Assume the target layer is 185 and the negative scattering bars are on layer 186.

```
background -0.251 0
layer 185 MxTARGET 0 0 opc 1.251 0
layer 186 ASRAF 128 0 asraf 1.251 0
```

Related Topics

[background](#)

[flaglayer](#)

[OPC_CORR_SEARCH_DISTANCE](#)

lineEndAdjDist

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies the distance away from a line end that is considered to be adjacent to the line end.

Usage

`lineEndAdjDist distanceValue`

Arguments

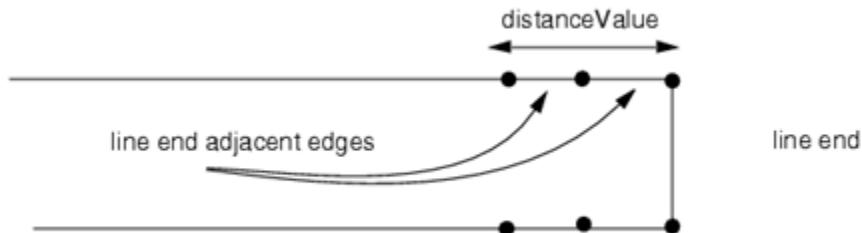
- *distanceValue*

A number representing a distance in microns.

Description

This optional keyword is a fragmentation parameter. It has no default value. The *distanceValue* is the distance away from a line end which is considered to be adjacent to the line end. The lineEndAdjDist keyword can be used in combination with the [layer](#) keyword's *layer_type* parameter to force OPC at line ends only. All edges completely within the *distanceValue* are considered adjacent to the line end, as shown by [Figure 6-26](#).

Figure 6-26. Line End Adjacent Distance



Related Topics

[lineEndLength](#)

lineEndLength

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Defines the distance criteria used to determine if an edge is on a line end or not.

Usage

lineEndLength *length*

Arguments

- ***length***

The maximum length in microns for an edge that should be classified as a line end. The ***length*** must be greater than or equal to 0. The default is [minfeature](#).

Description

This optional keyword defines the distance criteria used to determine whether an edge is on a line end. A line end is defined as an edge that is shorter than or equal to ***length*** and between two convex corners, each of which is at least ***length*** long. Line ends are automatically tagged by the built-in tag [line_end](#).

Any edge that is a line end is treated differently than other edges. Line ends are fragmented by the [seriftype](#) keyword.

maxedgelength

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Breaks long edges into fragments by specifying the maximum length of a fragment.

Usage

maxedgelength maxedge [-split {0 | 1}]

Arguments

- **maxedge**

A required argument specifying the maximum length of fragments generated by maxedgelength fragmentation, specified in microns.

The default value of **maxedge** is ([tilemicrons](#)/2). The default value is large enough that not much fragmentation occurs when it is left at its default value. The value must be greater than [minedgelength](#).

- **-split {0 | 1}**

An optional argument controlling how this keyword splits fragments with a length greater than 2*maxedgelength. [Table 6-8](#) and [Table 6-9](#) show the effects of this argument, and [Figure 6-27](#) depicts these effects graphically.

Table 6-8. How MAXEDGELENGTH Splits Long Fragments (-split=0)

| Relationship | -split = 0 |
|--|--|
| < 2MAX ¹ + MIN ² | do not break |
| $\geq (2\text{MAX} + \text{MIN}) \leq (3\text{MAX})$ | fragments of length maxedge , with a centered remainder fragment with length greater than minedgelength . |

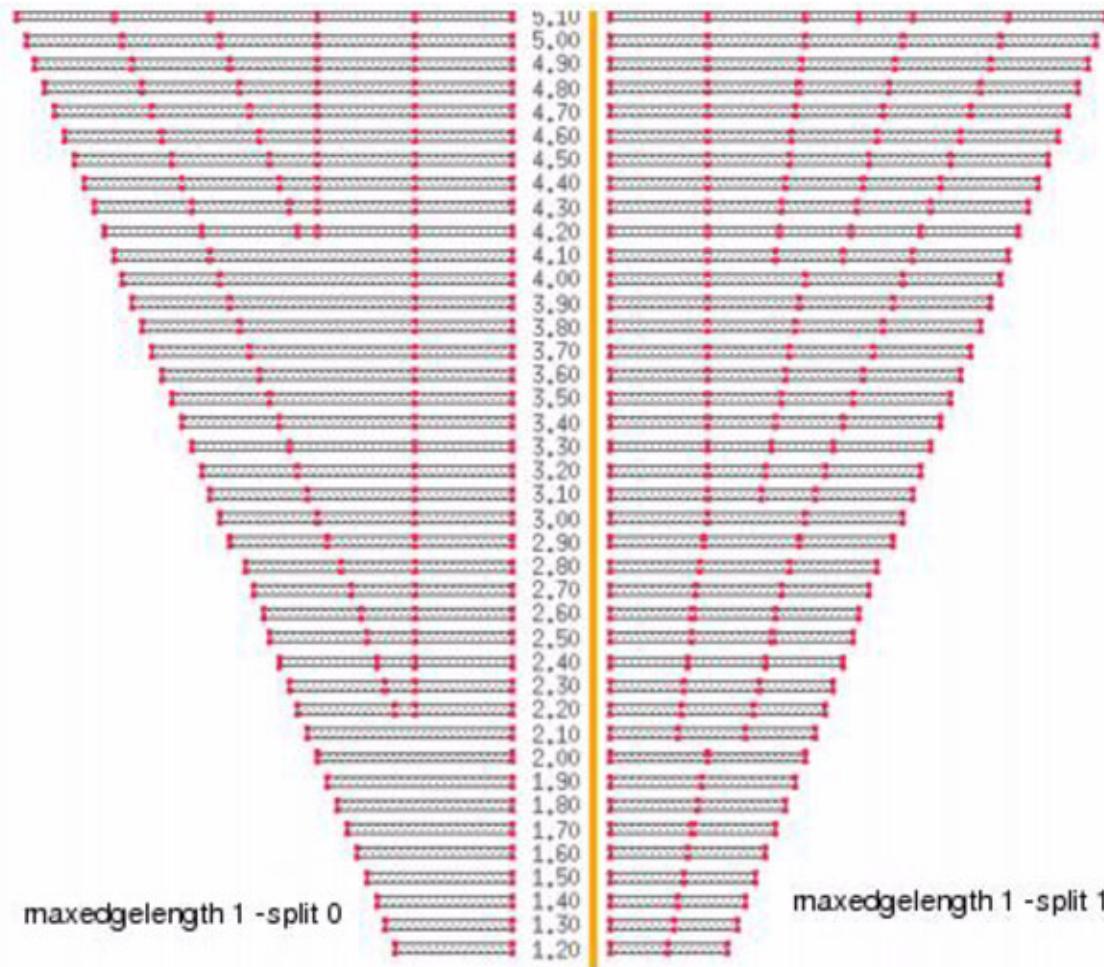
1. MIN = [minedgelength](#)
2. MAX = [maxedgelength](#)

Table 6-9. How MAXEDGELENGTH Splits Long Fragments (-split=1)

| Relationship | -split = 1 |
|--|---|
| $< 2\text{MIN}$ 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 6-9. How MAXEDGELENGTH Splits Long Fragments (-split=1) (cont.)

| Relationship | -split = 1 |
|---------------------|--|
| > 3MAX | fragments of length maxedge , with a centered remainder split into two or three equal fragments that are as long as possible with none longer than maxedge , and none shorter than minedgelength . Note: if the original long fragment is < 3MAX, there are no fragments of maxedge . |

Figure 6-27. Comparing MAXEDGELENGTH -split 0 and 1

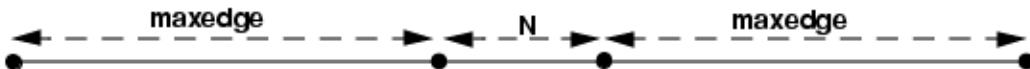
Description

This keyword is a fragmentation parameter. It is used during **maxedgelength** fragmentation, which breaks up very long edges into shorter fragments.

The **maxedgelength** fragmentation algorithm attempts to break all edges equal to or longer than $2 * \text{maxedge} + \text{minedgelength}$. Subsequently, these edges are split into two **maxedge** parts and one part greater than or equal to **minedgelength**. Fragmentation is symmetric with respect to the

center of the edge. [Figure 6-28](#) shows the results of fragmenting an edge of length $2 * \text{maxedge} + N$, where $N > \text{minedgelength}$.

Figure 6-28. Maxedgelength Fragmentation



No fragment smaller than `minedgelength` is generated by this operation. In the figure above, if N had been shorter than `minedgelength`, the edge would not have been fragmented.

Note that the `maxedgelength` is not the maximum fragment length. The true maximum fragment length is $2 * \text{maxedge} + \text{minedgelength} - \text{epsilon}$.

Examples

In this example, `minedgelength` and `maxedgelength` are set to the following:

- `minedgelength` (or MIN) = 0.0750 microns
- `maxedgelength` (or MAX) = 0.1000 microns

[Table 6-10](#) shows how `minedgelength` and `maxedgelength` are interpreted based on scale and measurement unit (such as nm, microns, or dbus).

Table 6-10. `minedgelength` and `maxedgelength` Scale and Measurement Unit

| Scale | MIN (nm) | MAX (nm) | MIN (microns) | MAX (microns) | MIN (dbu) | MAX (dbu) |
|-------|----------|----------|---------------|---------------|-----------|-----------|
| 1x | 75 | 100 | 0.0750 | 0.1000 | 150 | 200 |
| 2x | 150 | 200 | 0.1500 | 0.2000 | 300 | 400 |
| 3x | 225 | 300 | 0.2250 | 0.3000 | 450 | 600 |

For this example, the precision is set to 2000, and `minedgelength` and `maxedgelength` are measured in dbus.

Using these parameters, the following tables show the results of using the `-split` option, showing how fragments are split, depending if the fragment length falls into certain ranges of `minedgelength` (MIN) and `maxedgelength` (MAX), and whether the `-split` option is set to either 0 (see [Table 6-11](#)) or 1 (see [Table 6-12](#)).

Table 6-11. Results With `-split = 0`

| Length (dbu) | Relationship | <code>-split = 0</code> | Comments |
|--------------|--------------|-------------------------|-----------------------|
| 149 | MIN-1 | no splits | < 2MIN and \leq MAX |
| 150 | MIN | no splits | < 2MIN and \leq MAX |

Table 6-11. Results With -split = 0 (cont.)

| Length (dbu) | Relationship | -split = 0 | Comments |
|---------------------|---------------------|--------------------------|---------------------------|
| 151 | MIN+1 | no splits | < 2MIN and <= MAX |
| 199 | MAX-1 | no splits | < 2MIN and <= MAX |
| 200 | MAX | no splits | < 2MIN and <= MAX |
| 201 | MAX+1 | no splits | < 2MIN and <= MAX |
| 299 | 2MIN-1 | no splits | < 2MIN and <= MAX |
| 300 | 2MIN | no splits | >=2MIN, > MAX, and <=2MAX |
| 301 | 2MIN+1 | no splits | >=2MIN, > MAX, and <=2MAX |
| 399 | 2MAX-1 | no splits | >=2MIN, > MAX, and <=2MAX |
| 400 | 2MAX | no splits | >=2MIN, > MAX, and <=2MAX |
| 401 | 2MAX+1 | no splits | >2MAX and <=3MIN |
| 449 | 3MIN-1 | no splits | >2MAX and <=3MIN |
| 450 | 3MIN | no splits | >2MAX and <=3MIN |
| 451 | 3MIN+1 | no splits | >2MAX and <=3MIN |
| 549 | (2MAX+MIN)-1 | no splits | < 2MAX + MIN |
| 550 | 2MAX+MIN | two splits (into thirds) | >=2MAX + MIN and <=3MAX |
| 599 | 3MAX-1 | two splits (into thirds) | >=2MAX + MIN and <=3MAX |
| 600 | 3MAX | two splits (into thirds) | >=2MAX + MIN and <=3MAX |
| 601 | 3MAX+1 | two splits (into thirds) | >3MAX |

Table 6-12. Results With -split = 1

| Length (dbu) | Relationship | -split = 1 | Comments |
|---------------------|---------------------|-------------------|-------------------|
| 149 | MIN-1 | no splits | < 2MIN and <= MAX |
| 150 | MIN | no splits | < 2MIN and <= MAX |
| 151 | MIN+1 | no splits | < 2MIN and <= MAX |
| 199 | MAX-1 | no splits | < 2MIN and <= MAX |

Table 6-12. Results With -split = 1 (cont.)

| Length (dbu) | Relationship | -split = 1 | Comments |
|--------------|--------------|--------------------------|---------------------------|
| 200 | MAX | no splits | < 2MIN and <= MAX |
| 201 | MAX+1 | no splits | < 2MIN and <= MAX |
| 299 | 2MIN-1 | no splits | < 2MIN and <= MAX |
| 300 | 2MIN | one split (into halves) | >=2MIN, > MAX, and <=2MAX |
| 301 | 2MIN+1 | one split (into halves) | >=2MIN, > MAX, and <=2MAX |
| 399 | 2MAX-1 | one split (into halves) | >=2MIN, > MAX, and <=2MAX |
| 400 | 2MAX | one split (into halves) | >=2MIN, > MAX, and <=2MAX |
| 401 | 2MAX+1 | one split (into halves) | >2MAX and <=3MIN |
| 449 | 3MIN-1 | one split (into halves) | >2MAX and <=3MIN |
| 450 | 3MIN | one split (into halves) | >2MAX and <=3MIN |
| 451 | 3MIN+1 | two splits (into thirds) | >2MAX and <=3MIN |
| 549 | (2MAX+MIN)-1 | two splits (into thirds) | < 2MAX + MIN |
| 550 | 2MAX+MIN | two splits (into thirds) | >=2MAX + MIN and <=3MAX |
| 599 | 3MAX-1 | two splits (into thirds) | >=2MAX + MIN and <=3MAX |
| 600 | 3MAX | two splits (into thirds) | >=2MAX + MIN and <=3MAX |
| 601 | 3MAX+1 | two splits (into thirds) | >3MAX |

Related Topics

[minedgelength](#)

[tilemicrons](#)

minedgelength

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Sets the smallest length that can be produced by fragmentation.

Usage

minedgelength *minedge*

Arguments

- ***minedge***

A required argument specifying the minimum allowed length of a fragmented edge in microns. The value must be greater than 0. The default value is 0.1 microns.

Note

 Setting ***minedge*** to a value smaller than OPC_MIN_INTERNAL or OPC_MIN_EXTERNAL causes fragmentation to produce edge fragments that violate the width and spacing rules, even before movement.

Description

This keyword is a fragmentation parameter. It defines the minimum edge that can be produced by fragmentation. An edge is not fragmented if it results in a new edge shorter than ***minedge***.

The minedgelength value only applies to fragments created by the application. It does not apply to edges drawn in the design. Although this keyword defines the minimum length of a fragmented edge, there may still be smaller original edges.

Related Topics

[maxedgelength](#)

minfeature

Type: [Litho Setup File Keywords](#). Calibre OPCpro only.

Required. Specifies the minimum design feature size in microns.

Usage

minfeature *minsize*

Arguments

- ***minsize***

A required argument specifying the minimum design feature size in microns. The value must be greater than 0.

Description

This is a required keyword that sets the minimum design feature size in microns. The minimum design feature size is the traditional smallest feature in the design. This parameter is constrained by the limitations of lithographic resolution for printing small features. The default value is 0.28 microns.

Related Topics

[iterations](#)

[stepsize](#)

minjog

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Controls the internal definition of a jog and ensures that jogs are not corrected.

Usage

minjog minjogsز

Arguments

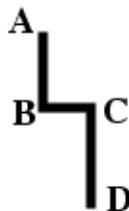
- **minjogsز**

A required argument that defines the minimum length for a movable jog edge in microns. The default value is (minfeature/4). The value must be greater than or equal to 0. Any edge less than or equal to **minjogsز** is considered a jog.

Description

The optional minjog keyword controls the definition of a jog, which ensures that short edges are not corrected. Any edge less than or equal to **minjogsز** is a jog, and is not moved during OPC.

Figure 6-29. Minimum Jog Size



If BC < **minjogsز** it is a jog and does not move during OPC.

Jogs do not receive control sites.

By default, sites are prevented from being placed on line ends that are less than **minjogsز** in length. However, if [ALLOW_SMALL_MOVE](#) is set to 1, sites are placed on the line ends irrespective of the value for minjog.

Note

 Small jogs should be turned off, because if there is an attempt to correct them, it may result in patterns with excess weight put on correcting small jogs while more important features do not get adequate correction.

Related Topics

[minfeature](#)

[opcIter](#)

modelpath

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Required. Defines the search path for optical and process models.

Usage

modelpath *directory_path*

Arguments

- ***directory_path***

A required argument specifying the directory path of the models to be used. The directories should be separated by colons (:).

Description

This required keyword lets you specify which models Calibre RET tools use for performing simulations.

When searching for optical and process models, the simulation tools search the directory specified by modelpath and use the first models they find matching the model names. If the application is unable to find any in the path supplied with the modelpath keyword in the setup file, the application automatically checks the current working directory (./) for optical models. The scope of this keyword is the duration of the batch tool run. When new setup parameters are loaded, the previous value for modelpath is replaced.

The modelpath keyword is different from ::MODEL_PATH, which is a Tcl variable you can set in the *wbinit.tcl* file for use with GUI applications.

Examples

The following modelpath keyword specifies to look for the *vt5out.mod* resist model and the *opticalD0* optical model first in the *./litho/models* directory, then the current working directory, and then the *models* subdirectory.

```
modelpath      litho/models:.:models
opticalmodel   opticalD0
resistpolyfile vt5out.mod
```

If the models are not found, Calibre checks again in the current working directory before generating an error.

Related Topics

[opticalmodel](#)
[resistpolyfile](#)

movelimit

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies the maximum inward and outward movements allowed for any fragment.

Usage

movelimit *d1* [*d2*]

Arguments

- *d1*
A required argument specifying the maximum outwards movement allowed, in microns. The default value is the largest of 0.08 micron or [minfeature](#)/2.
- *d2*
An optional argument specifying the maximum inwards movement allowed, in microns. If not specified, this value defaults to *d1*.

Description

The movelimit keyword gives global control over the maximum movement allowed for any fragment. To override this limit for one or more fragments, use the [setMoveLimitForTag](#) command. To limit the movement in a single iteration, use [OPC_MAX_ITER_MOVEMENT](#) setup variable.

MRC_RULE

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Sets constraints for EXTERNAL and INTERNAL rule checks for specific tags.

Usage

Syntax 1:

MRC_RULE {EXTERNAL | INTERNAL} *name1 name2 OFF*

Syntax 2:

MRC_RULE {EXTERNAL | INTERNAL} [*name1 name2*] {'

rule

[LENGTH *max_length rule*]...

[PROJECTING [*dist*]

[LENGTH *max_length rule*]

[LENGTH *max_length rule*]...]

[NOT_PROJECTING [*dist*]

[LENGTH *max_length rule*]

[LENGTH *max_length rule*]...]

'}

Arguments

- **EXTERNAL | INTERNAL**

A required argument that specifies the check type. You must specify either **EXTERNAL** or **INTERNAL**, but not both.

- ***name1 name2***

A pair of arguments limiting the fragment sets to which the rule applies. They are required for Syntax 1, and optional for Syntax 2.

The ***name1*** and ***name2*** values can be any valid layer or tag name. These allow specific MRC rules to be applied between tagged fragments or fragments between two layers. Up to 64 tag names can be used in MRC_RULE statements, and any of the opc, correction, asraf, and visible layers defined in the setup file can be used.

By default, layers defined on the same mask are always compared using the MRC rule that applies. This can be overridden by using the **OFF** option.

Note that ***name1*** and ***name2*** must both either be layer names or tag names. You cannot combine a layer and a tag.

- When ***name1*** and ***name2*** are tag names, the rule is applied to fragments.
- When ***name1*** and ***name2*** 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.

- **OFF**

For Syntax 1, a required keyword that disables constraint checking between the indicated tag groups of fragments.

- **{ and }**

For Syntax 2, required enclosing braces. The opening brace must appear on the same line as MRC_RULE, and be the last character on the line. The closing brace must be on a line of its own.

- ***rule***

For Syntax 2, a required argument that must be specified at least once. (The default rule.) Up to eight additional MRC rules can be specified with LENGTH, PROJECTING, and NOT_PROJECTING sections for a total of 24 cases.

The rule is specified using the following syntax:

USE *d1* {*metric* | *d2*}

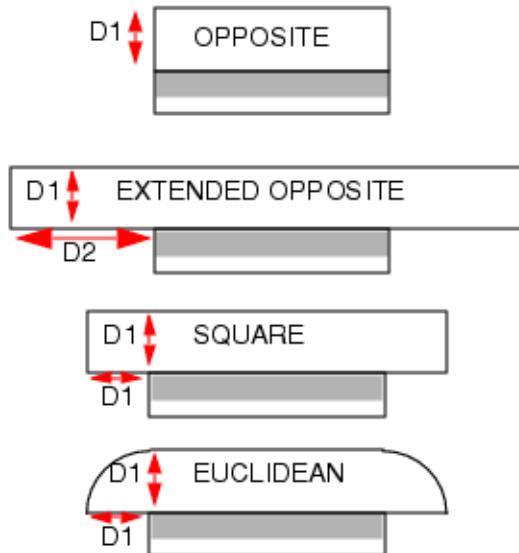
Distances that satisfy the MRC check are greater than or equal to ***d1***. The value for ***d1*** is required and is the default value used if none of the LENGTH constraint rules apply. You must specify either ***metric*** or ***d2***.

The possible values for ***metric*** are OPPOSITE, SQUARE, and EUCLIDEAN.

When ***d2*** is specified, it implies EXTENDED OPPOSITE with ***d2*** as the extension value.

[Figure 6-30](#) shows the measurement regions created for each of the different metrics, and the dimensions used with each.

Figure 6-30. The Four Metrics



- PROJECTING [*dist*]

An optional keyword that instructs Calibre to measure the separation between two fragments only when one fragment projects onto the other fragment. If *dist* is specified, the fragments are classified as non-projecting unless their projection length is greater than or equal to this value.

The PROJECTING keyword can only appear once per MRC_RULE. Any **rule** that follows PROJECTING must also meet the PROJECTING constraint.

- NOT_PROJECTING [*dist*]

An optional keyword that instructs Calibre to measure the separation between two fragments only when neither fragment projects onto the other fragment. If *dist* is specified, fragments are classified as non-projecting unless the projection length is greater than this value.

The NOT_PROJECTING keyword can only appear once per MRC_RULE. Any **rule** that follows NOT_PROJECTING must also meet the NOT_PROJECTING constraint.

- LENGTH *max_length*

An optional argument specifying the maximum length of fragment to which the **rule** applies. If the length of the minimum of the two fragments being compared is less than or equal to *max_length* and greater than the previous *max_length*, then this subcheck is used rather than the main check. The USE value then specifies the valid separation to use for this case.

There are two constraints on subcheck violation regions:

- A violation region for a subcheck must be completely covered by or be equal to all violation regions corresponding to subchecks with larger *max_length* that are within the same rule and same PROJECTING, NOT_PROJECTING, or default clause.
- When an MRC_RULE contains a PROJECTING or NOT_PROJECTING clause, the violation region of an edge or a fragment of any length under the PROJECTING clause must completely cover or be equal to the violation region under the NOT_PROJECTING clause.

Description

MRC_RULE sets constraints for **EXTERNAL** or **INTERNAL** rule checks for specific tags. With MRC_RULE, 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_RULE.

Similarly, tags on the same mask can have the check between them disabled by setting the **MRC_RULE OFF** option. When multiple rule statements are defined, the first MRC_RULE with **name1 name2** specified in the setup file that matches the two fragments under consideration is the one selected. The default check is selected only if no **name1 name2** checks match the two fragments under consideration.

You typically define an MRC_RULE syntax block in the “Arbitrary Commands” section of the setup file.

The [LITHO_PROMOTION_TILING](#) environment variable must be explicitly set to 1 in order to use the MRC_RULE syntax, as shown in the following example:

```
----- Arbitrary Commands Can Follow This Line -----
sse LITHO_PROMOTION_TILING 1

MRC_RULE EXTERNAL poly poly {
    USE 0.160 EUCLIDEAN
    LENGTH 0.01 USE 0.030 OPPOSITE
    LENGTH 0.1 USE 0.050 EUCLIDEAN
    PROJECTING
    USE 0.160 EUCLIDEAN
    LENGTH 0.02 USE 0.050 EUCLIDEAN
}
```

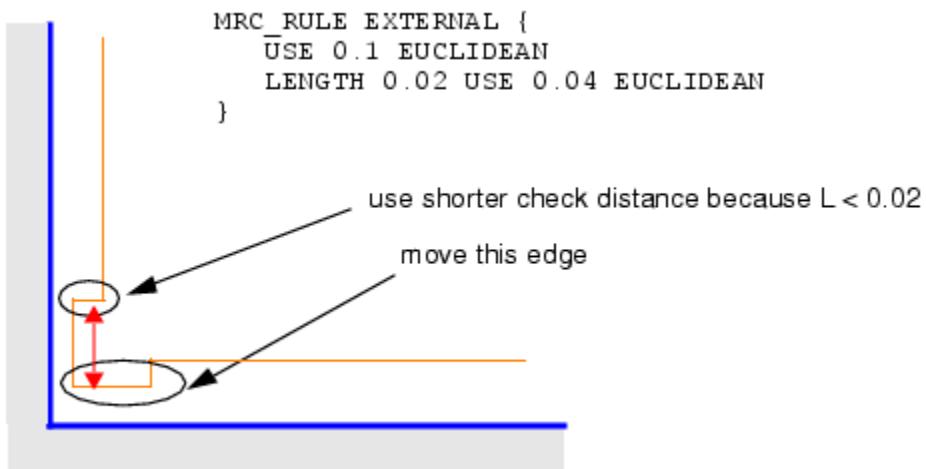
MRC_RULE cannot be used with the deprecated alternative, the combination of OPC_MIN_EXTERNAL, OPC_MIN_INTERNAL, and opcMinCheckTag.

Examples

MRC_RULE allows the flexibility to perform INTERNAL or EXTERNAL operations for a variety of unique situations.

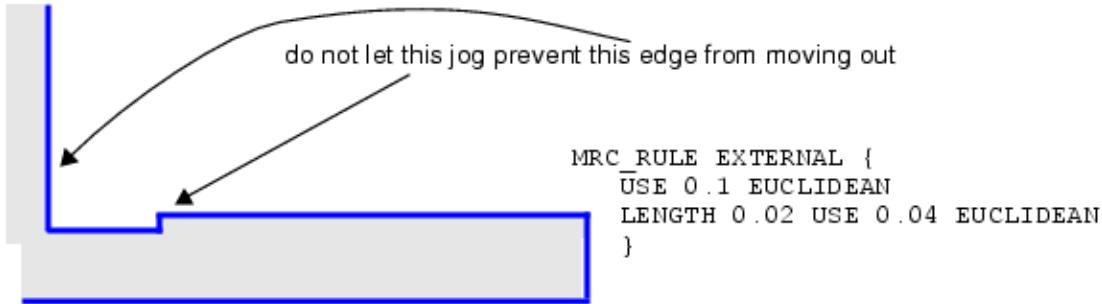
Example 1 - Moving a Fragment From a Concave Corner

The following example allows a fragment at a concave corner to move outwards.



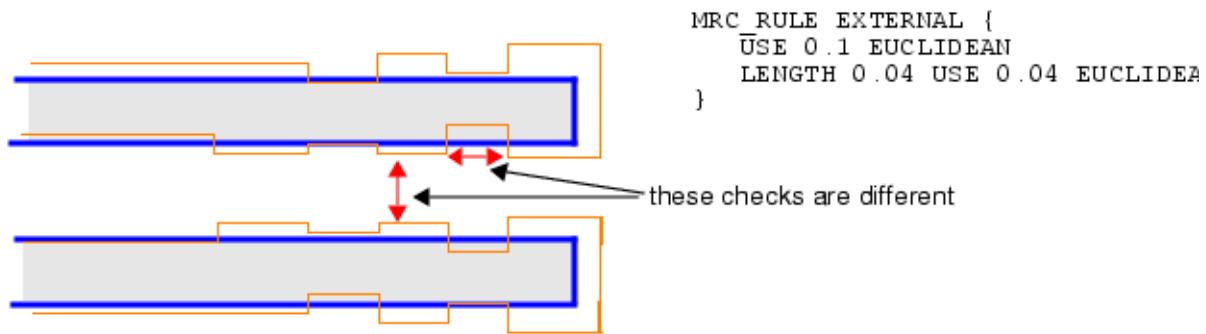
Example 2 - Jogs

In this example, a rule is set to ignore a small jog on the input layout when preventing a fragment from moving during OPC. The LENGTH should be the same as the value used in minjog.



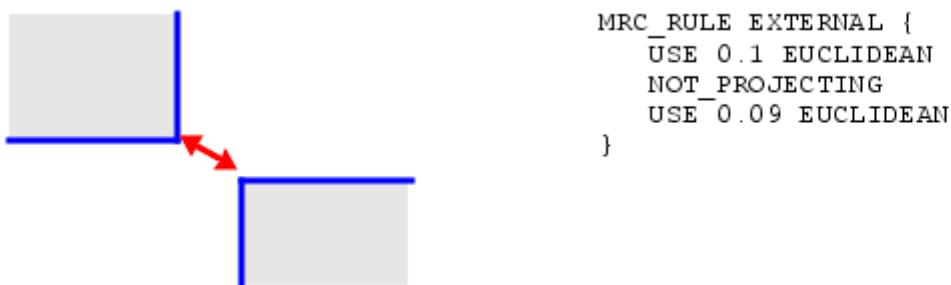
Example 3 - Notches and Feature-To-Feature Checks

In the following example, a distinction is made between notches and feature-to-feature checks.



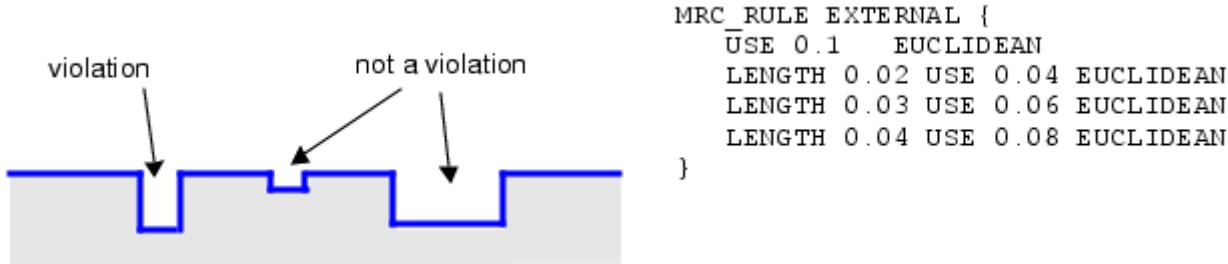
Example 4 - Corner-to-Corner Checks

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.



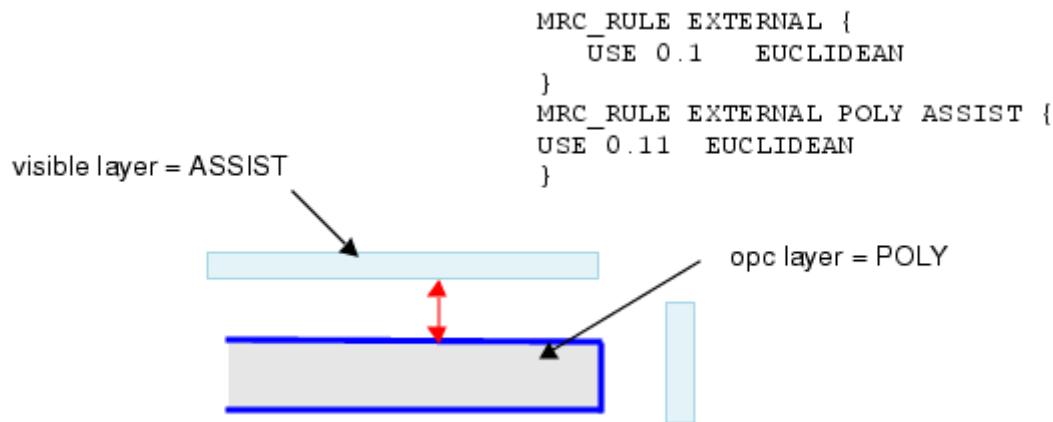
Example 5 - Stair-Stepping

The following example implements a “stair-stepped aspect ratio” of 2:1 when features are below a certain size.



Example 6 - Specifying Unique Spacing Rules

The following example implements different spacing rules for POLY to ASSIST versus POLY to POLY spacing.



Example 7 - Line-End Checks

The following example specifies different spacing rules for line-end to line-end check versus other checks.

```

MRC_RULE EXTERNAL {
    USE 0.1 EUCLIDEAN
}
MRC_RULE EXTERNAL line_end line_end {
    USE 0.95 EUCLIDEAN
}

```



opcIter

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Applies a different OPC iteration algorithm than fastIter.

Usage

```
[exec] opcIter [adaptiveSites {true | false}]
[cornerRadius value]
[cornerRadiusConcave value]
[cornerRadiusConvex value]
[criticalDistance value]
[enclose layer by value]
[fastIterMode {true | false}]
[freezeStepFragment {true | false | length}]
[globalEpeTolerance value]
[jogSmooth value [jogSmoothTagSet tagName]]
[limitTag tagName to value]
[lineEndRatio value]
[lineEndWidth value]
[maxJog size]
[modifiedTarget {true | false}]
[nIterations value]
[parameterOutputFile file]
[preferMid {true | false}]
[midSpanExt value]
[maxMidSpan value]
[printApd {true | false}]
[printHeader {true | false}]
[printTime {true | false}]
[roundCorner {true | false}]
[roundedCornerRadius value]
[siteControl {true | false}]
[spaceEndRatio value]
[spaceEndWidth value]
[statisticsFileName filename]
[targetLayer layer]
[turnOffSlivers width]
[-outLayer {child_owned_fragments | corners | epe_tolerance | false_fragments |
           jogs | limit_tags | modified_target | mrcBlocked_iter_layer |
           parent_owned_fragments | site_extents | sites |
           target_band | tolerance_band | valid_fragments | violations}]
```

Note

 The “exec” portion of the command is optional and does not explicitly need to be specified in the setup file for Calibre versions 2008.3 and later.

Arguments

- adaptiveSites {true | false}

If adaptiveSites is set to true, control points outside the previous simulation’s minimum and maximum intensity locations are ignored in the current simulation. This reduces simulation time in most cases with very little change in final output.

- cornerRadius *value*

An optional argument that is used at corners to estimate the achievable target. Users 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 an integer 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 an integer greater than 0. The default is to use the cornerRadius *value* specified for opcIter.

- 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 an integer greater than 0. The default is opcIter’s cornerRadius.

Note

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

- criticalDistance *value*

The smallest feature that must be resolved. Normally this should not be left at its default value, $(R_o/3)$, where R_o is the resolution value obtained from the optical model parameters (lambda/NA). Optimize this argument around the default value by plus or minus 50% to obtain the best results for their applications.

- enclose *layer* by *value*

An optional argument that ensures all features on the named layer are enclosed by at least *value* distance in the final results. By default, no such check is made.

- fastIterMode {true | false}

If fastIterMode is set to true, then fastIter is executed instead of opcIter. If you want to use fastIter with opcIter siteControl, set siteControl to true and fastIterMode to true. If

siteControl is false and fastIterMode is true then the results should be identical to using fastIter directly.

- **freezeStepFragment {true | false | *length*}**

Controls whether step fragments (fragments less than $2 * \text{cornerRadius}$ connecting two fragments longer than itself) are simulated or not.

In version 2009.2 and earlier, step fragments were not treated differently from other fragments; this is the same as the “false” setting. In version 2009.3 and later, step fragments are not simulated by default.

The *length* parameter allows you to control which step fragments are simulated based on length in microns. Fragments shorter than *length* are not simulated.

- **globalEpeTolerance *value***

The EPE tolerance on all fragments that are not otherwise specified by using the limitTag. The default value is set to $0.06 * \text{criticalDistance}$. Users should optimize this parameter from **stepsize** up to a value of $5 * \text{stepsize}$ or more.

- **jogSmooth *value***

When specified, jogSmooth smooths all jogs created by OPC that are smaller than or equal to *value*. There is no default.

- **jogSmoothTagSet *tagName***

This argument restricts smoothing to the named tag set, otherwise everything gets smoothed.

- **limitTag *tagName* to *value***

An optional argument that limits *tagName* fragments to being no more than *value* distance from their EPE tolerance.

tagName specifies the name of a tag set you want to impose restrictions on.

value is a limit value in user units.

For example, “limitTag line_end to 0.001” puts a tolerance of one nanometer on all fragments in the line_end tag set. The EPE tolerance values set by the limitTag arguments override the value set globally by globalEpeTolerance. There can be any number of limitTag arguments for opcIter. There are no default values for these arguments.

There are two built-in tags for limitTag: “corners” and “straights”. The argument corners specifies fragments that touch corners or fragments that have simulation sites within the cornerRadius value of a corner. Jogs are ignored when these corners are defined. Use the -outLayer corners option to see how corners are classified. The argument straights is defined as anything that is not a corner.

- **lineEndRatio *value***

An optional argument that moves the simulation sites for line-end adjacent fragments to control hammerheads. Setting this to a small value creates large hammerheads. Setting it to

a value larger than one disables the feature and simulation sites do not move. The default site placement is equivalent to a value of 0.5.

The image at line-end adjacent fragments is expected to cross the target at (*value* * D) from the line end, where D is the width of the line. The simulation sites for line-end adjacent fragments are moved to this location if reasonable. Adjusting this value makes the algorithm match line-end targets much better, but often causes image ringing far back from the line end.

- **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*cornerRadiusConvex.

- **maxJog *size***

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 *size* are user units. The default value is approximately 0.292893*radius, where radius is the minimum of cornerRadiusConcave, cornerRadiusConvex, and cornerRadius.

- **maxMidSpan *value***

An optional argument used only when preferMid is true. This argument determines the maximum length of a span of fragments that can be a straight line instead of a curve. Single fragments or edges longer than *value* are represented by curves. The valid range of *value* is 0 to 1 microns, in user units. The default is 2*criticalDistance/3.

- **midSpanExt *value***

An optional argument used only when preferMid is true. This argument indirectly controls the slope of the line drawn for the span. The valid range of *value* is 0 to 10. The default is 1.

A value of 1 draws a straight line at a 45-degree angle through the midpoint of a jog within the span. Setting midSpanExt to 0.5 draws a line half as long, resulting in a steeper angle. Setting it to 2 draws a line twice as long as the default setting, which results in a much shallower angle.

- **modifiedTarget {true | false}**

Note



As of version 2011.3, modifiedTarget is deprecated. Use the targetLayer argument.

If modifiedTarget is set to true, the OPC target is redefined by the positions of the fragments. If it is set to false (the default) then the original fragment biases are ignored.

If “modifiedTarget true” is specified, the simulation sites are moved to the biased fragment locations even if “siteControl false” is specified.

- **nIterations *value***

The maximum number of iterations. Set it to zero if you only want to perform site control or create the auxiliary output layers. The default value is 8.

- preferMid {true | false}

An optional argument that affects how curved targets are created. When true, straight lines are used instead of curves to represent a span of the polygon edge. The lines can be skew. This sometimes provides a much better fit than a true curve. The default setting is false.

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

An optional parameter that causes opcIter to round concave corners formed by edges both shorter than cornerRadiusConcave. When roundCorner is set to true, the target is moved out towards the printed image by roundedCornerRadius. The default setting is false (no adjustments).

Use roundedCornerRadius to increase the distance from the original corner from which the target is moved. The default value is derived from cornerRadiusConcave and is typically smaller than optimum. The roundedCornerRadius setting is ignored when roundCorner is false.

- siteControl {true | false}

By default, opcIter does automatic site control, moving sites to improve convergence. If you want to perform site control using other techniques this feature can be turned off with “siteControl false”.

- spaceEndRatio *value*

An optional argument that overrides the lineEndRatio setting, which controls hammerheads, for space ends. Smaller values are more aggressive than larger ones. The default value is the setting of lineEndRatio.

- 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*cornerRadiusConcave.

- targetLayer *layer*

Use this argument to prebias a layer. The initial sites are placed relative to *layer* rather than the opc layer. The opc and correction layers are still used for fragmentation. By default, *layer* is the opc layer specified by the [layer](#) command.

- turnOffSlivers *width*

The width at which a polygon is determined to be a sliver. Slivers are polygons that should not receive corrections because they are thin enough that correcting them is likely to degrade their neighbors' corrections without improving overall results. Generally slivers occur in the contextual halo around cells. The default value is criticalDistance/2.

The arguments turnOffSlivers and modifiedTarget are mutually exclusive. (There is no conflict with targetLayer, the preferred alternative to modifiedTarget.) If “modifiedTarget true” is specified, then you must also specify “turnOffSlivers 0”.

Fragments on a sliver still receive control sites but are not corrected.

Debugging Arguments

Note

 These parameters do not affect the operation of opcIter.

They generate reports that are helpful in understanding the results. Be aware that writing out a statistics file causes one extra optical simulation of every simulation site in the analyzed area. This is so the data reported on the final result uses accurate results, not results from earlier simulations that may have been incomplete.

- **statisticsFileName *filename***

The name of the file where statistics should be reported. If this argument is not specified, a report is not written.

- **parameterOutputFile *file***

An optional argument specifying a file for parameter values. All parameters used by opcIter are written to this file for reference; it cannot be used as input to OPCpro. By default, this file is not created.

- **printApd {true | false}**

If set to true then APD data is also printed to the statistics file. The default value is false.

- **printHeader {true | false}**

By default header information is printed to the statistics file before writing the data. You should set printHeader to false if you use the statistics file to collect performance data on fastIter. In that case, you want the header information only on the first opcIter call and not on subsequent opcIter calls. The default value is true.

- **printTime {true | false}**

If this argument is set to true, then timing information is printed to the log file, not the statistics file. The default value is false.

- **-outLayer {child_owned_fragments | corners | epe_tolerance | false_fragments | jogs | limit_tags | modified_target | parent_owned_fragments | site_extents | sites | target_band | tolerance_band | valid_fragments | violations}**

Output layers are additional layers that can be written by opcIter. They are helpful for understanding problem areas. The -outLayer parameters must be the last arguments on the opcIter call. A separate -outLayer declaration needs to be specified for each generated layer. An -outLayer parameter with a different keyword generates an error message.

child_owned_fragments

Draws a box around each fragment that is child owned in the current hierarchy.

corners

Draws a box around fragments classified as corners.

epe_tolerance

Puts a box around every fragment showing the tolerance bounds for each fragment.

false_fragments

Draws a box around each fragment that is false in the current hierarchy. False fragments are introduced to break large shapes.

jogs

Draws a box around fragments that are classified as jogs.

limit_tags

Draws a box around any fragment specified in a `limit_tag` argument.

modified_target

If the command-line option “`modifiedTarget true`” is used, then `opcIter` uses the initial fragment biases to define a modified target shape. This output layer is that modified target. If `modifiedTarget` is set to false or left at its default value, then this output layer is the same as the original OPC target layer.

mrcBlocked_iter_layer

Outputs a layer with boxes indicating fragment movements that were blocked by MRC or other constraints.

To see fragments specified on a particular iteration, supply a number for `iter`. For example, `mrcBlocked_1` outputs boxes for fragments blocked on the first iteration.

To see fragments from a particular layer, specify the layer name for `layer`. For example, `mrcBlocked_m1` outputs boxes for `m1` fragments whose movements were blocked on the last iteration.

The basic form is simply `mrcBlocked`, which outputs polygons for blocked fragments movements for all layers on the last iteration. Forms such as `mrcBlocked_1_m1` are permissible.

parent_owned_fragments

Draws a box around each parent owned fragment in the current hierarchy.

site_extents

Draws a box around every control point of every simulation site. This is particularly useful for checking `adaptiveSites` output.

sites

Draws a box around every simulation site.

target_band

This is the difference between the original drawn target and the achievable target computed by `opcIter`. The drawn target is the original polygon that `opcIter` receives. The `target_band` argument rounds off the corners according to other parameters given to `opcIter`.

tolerance_band

This is the area between the outer and inner acceptable print images.

valid_fragments

Draws a box around each fragment that is valid in the current hierarchy. Along with the next three output layers this can help you understand what is happening in the hierarchy.

violations

Outputs a layer containing the violations remaining after the algorithm completes. The box shows the size and direction of the error.

Description

OpcIter is a setup file keyword similar to [fastIter](#). OpcIter has its own set of parameters used for tuning purposes. These parameters are only meant to be used with opcIter. When opcIter is used, the [iterations](#) keyword must be set to zero, otherwise additional iterations may occur.

Outlined below are three methods of carrying out OPC.

- Iterations Keyword

Using the iterations keyword in the setup file carries out full chip simulations in batch mode. When the iterations keyword is used in Calibre WORKbench, OPCpro iterates over the area currently displayed in the window.

- fastIter Setup Keyword

The fastIter setup file keyword is used to target a specific set of tags in a design. This command is used in cases where a full iteration over the chip is not necessary. When particular sites are tagged for an iteration, fastIter provides a targeted way to iterate only those selected areas. In most cases, fastIter is tuned with the [opcTag -hintOffset](#) option. The iterations keyword is ignored when fastIter is used.

- OpcIter Setup Keyword

OpcIter belongs in the same family of iteration commands as the iterations keyword and fastIter, but can also provide better results for denser layers in the way that it handles each iteration. OpcIter requires less tuning than fastIter and converges on an answer while fastIter continually iterates over simulation sites for the specified number of times. OpcIter has its own options for performance tuning that should not be used with other setup commands. OpcIter must be the final entry in your setup file. Output layers generated with the [-outLayer](#) option must be the last arguments specified in the opcIter call. A separate [-outLayer](#) declaration needs to be specified for each generated layer.

Optimizations

Two keywords can decrease your run time. They are available also to fastIter, when fastIter is specified using the [fastIterMode](#) keyword with the [opcIter](#) command.

- The argument [jogSmooth](#) smooths the design after every opcIter iteration. It reduces run-time by 10 to 15 percent at a small cost of image quality. A good starting value for jogSmooth is one or two database units. Initial trials with the [jogSmooth](#) keyword

should not include any limiting tag sets to first assess the impact and results of jogSmooth and opcIter together.

- The argument adaptiveSites begins with all control points on the control sites. After the first iteration, it turns off control points outside the minimum and maximum intensity. In a typical layout, each iteration turns off more control points and calculates fewer intensities and EPEs. This technique is not recommended for layouts where minimum and maximum intensity occur near the edges of the control site because the few control points trimmed from runs are offset by the time required to check.

Neither of these optimizations is available with siteControl false or nIterations 0.

Examples

```
exec opcIter criticalDistance 0.065 \
limitTag line_end to 0.001 \
-outLayer sites -outLayer tolerance_band
```

In this example, opcIter is called in the last line in the setup file. This example creates two output layers, the first showing the location of sites and a second layer that shows the tolerance band. In normal production runs, neither of these would be present.

Related Topics

[iterations](#)

[fastIter](#)

optical_model

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies an optical model. Can appear up to 12 times in a setup file, as opposed to opticalmodel, secondOpticalmodel, and thirdOpticalmodel.

Usage

optical_model number {modelname | inline_model}

Arguments

- **number**

A required argument that specifies the index of the model. The first model must be 0 (zero) and is used for aerial image simulation.

Instances of the optical_model command must be sequentially numbered starting at 0. (That is, the next optical_model instance must be 1, then 2, then 3, and so on up to 11.) Each **number** must be unique.

- **modelname | inline_model**

A required argument that specifies the optical model to use.

modelname — The filename of the optical model. The first matching file in [modelpath](#) is used.

inline_model — An optical model enclosed in braces ({}). The optical model must adhere to the format described in “[Optical Parameters File Format](#)” in the *Calibre WORKbench User’s and Reference Manual*. The open brace must appear on the same line as the keyword. The close brace must be on a line by itself, directly following the last optical model parameter.

Description

A keyword that defines an optical model to use for simulations. Either opticalmodel or optical_model must appear in the litho setup file. To use an optical model defined in a separate file, provide the name of the optical model.

Note

 Resist models created using Model Center contain the path of the optical model file used in the creation of the resist model. When such a resist model is specified in the setup parameters, **modelname** must match the value in the resist model file.

Use optical_model when you need more than three optical models for your process. A rule file must use either numbered optical_model keywords or opticalmodel, secondOpticalmodel, and thirdOpticalmodel; the optical model keywords cannot be mixed.

Examples

Example 1 — Four Optical Models

Multi-patterning processes sometimes require more than three optical models that the original opticalmodel, secondOpticalmodel, and thirdOpticalmodel keywords allowed for.

```
modelpath ./models
optical_model 0 small_248_nom
optical_model 1 small_248_dpos
optical_model 2 small_248_dneg
optical_model 3 small_248_f+10
```

Example 2 — In-line Optical Model

This example shows the proper placement of the braces for an in-line optical model.

```
optical_model 0 {
    version 4
    engine TCCcalc
    opticsystem 0.248 0.600 0.750
    defocuslevels 8 0.001 0.100
    approxorder 16
    hoodpix 1.280
    kerngrid 0.010
    illumtype STANDARD
}
```

Related Topics

[opticalmodel](#)

opticalmodel

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Required. Specifies the optical model for aerial image simulation during OPC and analysis.

Usage

opticalmodel {*modelname* | **inline**}

Arguments

- ***modelname***

A required argument specifying the name of the optical model to use for simulation. Cannot be specified with **inline**.

- **inline**

A required literal string that directs the tools to use the optical model that is provided within the same setup file, rather than in a separate file. Cannot be used with ***modelname***.

Description

A keyword that defines an optical model to use for simulations. Either **opticalmodel** or **optical_model** must appear in the litho setup file. To use an optical model defined in a separate file, provide the name of the optical model. If you want to use an in-line optical model, supply the string “**inline**” instead of a model name.

An optical model is always required for aerial image simulation during OPC and analysis. An invalid ***modelname*** makes the setup parameters invalid. The optical model file must be located in the directories defined by the [modelpath](#) keyword. When resolving model paths, Calibre uses the first optical model file matching ***modelname***.

Note

 Resist models created using Model Center contain the path of the optical model file used in the creation of the resist model. When such a resist model is specified in the setup parameters, ***modelname*** must match the value in the resist model file.

Related Topics

[inline_optical1](#)

[secondOpticalmodel](#)

[resistpolyfile](#)

outlayer

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Creates a supplementary output layer that displays the tile boundaries used when processing your layout.

Usage

outlayer *layer_number* *layer_name* -tiles

Arguments

- ***layer_number***

A required argument assigning a unique layer number for the layer to be created. You can use ***layer_number*** with the MAPNUMBER argument in the LITHO operation.

- ***layer_name***

A required argument assigning a unique name to the layer to be created. You can use ***layer_name*** with the MAP argument to the LITHO operation.

- **-tiles**

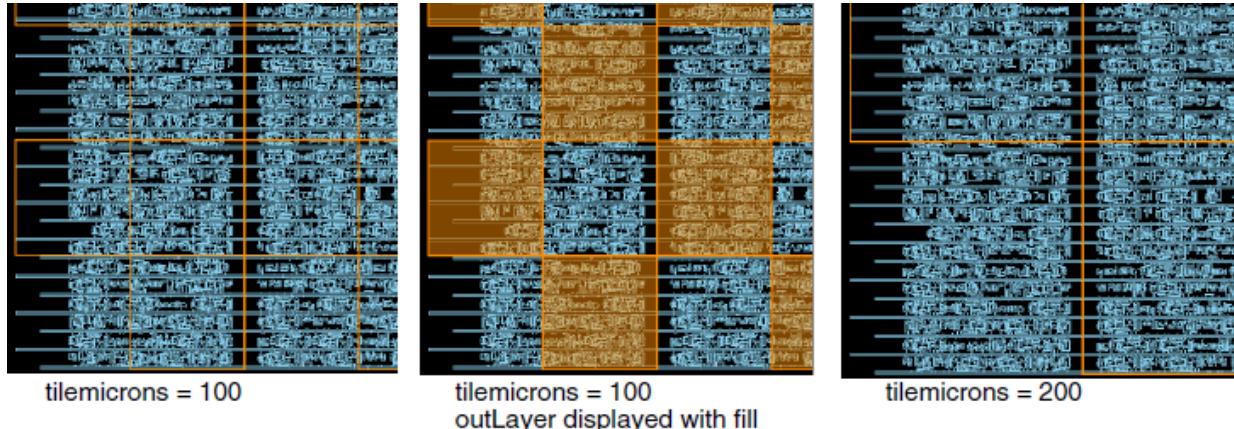
A required switch indicating that the output is tile boundaries.

Description

The outlayer command creates a supplementary output layer that displays the tile boundaries used when processing your layout. When you use this command, you must place it at the end of the setup file, along with the tagging script and sse command for setting environment variables.

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. In [Figure 6-31](#), you can see the original geometries displayed in blue and the tile boundary output geometries in orange.

Figure 6-31. Tile Boundaries



Examples

Example Setup File

```
#=====
modelpath .:models:~/calibrewb_workspace/models
opticalmodel defaultopt
resistpolyfile test.mod
iterations 1
gridsize 0.001
stepsize 0.001
#-----
tilemicrons 150
#-----
cornerSiteStyle CONSERVATIVE
siteinfo RESIST -spacing 0.0300 -num 17 -center 3
minfeature 0.120000
minedgelength 0.040000
maxedgelength 1.000000
minjog 0.035000
seriftype 1
concavecorn 0.080000
interfeature off
background clear
layer 1 L1 17 0 opc dark
#-- Arbitrary Commands Can Follow This Line. Don't delete this line! --
outLayer 101 tileLayer -tiles
#=====
```

Example SVRF

```
LAYOUT PATH "sample.gds"
LAYOUT PRIMARY "TOP"
LAYOUT SYSTEM GDSII
PRECISION 1000
RESOLUTION 1

DRC MAXIMUM RESULTS ALL
DRC MAXIMUM VERTEX 199
DRC RESULTS DATABASE "tiles_output.gds" GDSII
DRC SUMMARY REPORT "tiles_output.log"

LAYER L1 0
OPC1 = LITHO OPC L1 FILE tiles_1.in MAP L1
tileLayer1 = LITHO OPC L1 FILE tiles_1.in MAP tileLayer

L1 {COPY L1} DRC CHECK MAP L1 1
OPC1 {COPY OPC1 } DRC CHECK MAP OPC1 10
tileLayer1 {COPY tileLayer1 } DRC CHECK MAP tileLayer1 101
```

processModel

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Registers the name of a resist model file.

Usage

Syntax 1:

processModel *modelname* *modelfile*

Syntax 2:

processModel *modelname* [C *constr*] [I *constr*]

Arguments

- ***modelname***

A required argument specifying the name of the resist model.

- ***modelfile***

A required argument specifying the name of the resist model file.

- **C *constr***

An optional argument indicating that the aerial image factor is used to identify the areas for which this model is used. *constr* defines the range of aerial image factor values applicable to this model.

- **I *constr***

An optional argument that defines the peak intensity ranges for which this model is used.

Description

For Syntax 1, this keyword registers the name of a resist model file for subsequent use by [resistpolyfile](#) and [bindModelToTag](#).

For Syntax 2, this keyword specifies the named model to be used whenever the indicated conditions are met. The keyword [resistpolyfile](#) references the primary resist model, and the keyword [processModel](#) references each additional resist model. Optional arguments to the keyword identify the areas to which each process model applies.

Each setup file can contain up to four occurrences of the [processModel](#) keyword.

Examples

Example 1

```
resistpolyfile ctr.30.mod      # primary model
processModel mymod ctr.20.mod # name the line end model (usage 1)

processModel mymod C < -0.2   # use the named model for certain image
                            # constr (usage 2)
```

Example 2

```
resistpolyfile mymodel.mod          # primary model
processModel    thresh25 ctr.25.mod # 0.25 ctr model
processModel    thresh30 ctr.30.mod # 0.30 ctr model
processModel    thresh25 C > 0.3   # 0.25 model for line end
processModel    thresh30 C < -0.2  # 0.30 model for inv line end
```

Related Topics

[resistpolyfile](#)

[bindModelToTag](#)

progversion

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Reports the litho program version in the setup file.

Usage

progversion [arch | date | libs | name | number | version | check *constraint version_string*]

Arguments

- **arch**
Returns the architecture. For example: Sun SPARC.
- **date**
Returns the compilation date. For example:

```
Wed Oct 15 19:29:35 PST 2008
```
- **libs**
Returns the litho libraries compiled into the program. For example:

```
// Litho Libraries v2008.4_2.20 Wed Oct 15 19:29:35 PST 2008
```
- **name**
Returns the program name. For example: Calibre WORKbench or Calibre DESIGNrev.
- **number**
Returns the program internal ID number: 1 for Calibre WORKbench and 2 for LITHO.
- **version**
Returns the program version. For example: version 2008.4_2.20.
- **check *constraint version_string***
Checks if the current version meets the indicated constraint for version number string.
Returns 0 or 1.

Where:

- *constraint* is one of the following: “==”, “>=”, “<=”, “>”, “<”.
- *version_string* is a valid program version.

Description

This command is used to report the litho program version information in the setup file.

Use progversion to conditionally change setup files for specified versions of the litho program. This helps to maintain operability in an environment where multiple versions are present. The progversion command can also be used to:

- Assist in migration to newer software versions which have minor incompatibilities with older versions.
- Enforce usage of a frozen version of litho or Calibre WORKbench with a certain setup file.

Examples

Ensure that a certain version is being used with this setup file.

```
set v [progversion version]
if { [lindex $v 1] != "2004.2.1.1" } {
    error "This setup file is locked to 2004.2 build 1 patch 1"
}
```

An alternative method of doing same check is as follows:

```
if ![progversion check == 2004.2.1.1] { error "Wrong version" }
```

resistpolyfile

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Required. Defines the resist model to use for simulations.

Usage

resistpolyfile {*resist_model* | **inline**}

Arguments

- ***resist_model***

A required argument that specifies the name of the resist model to use for simulation.

- ***inline***

A required argument that specifies to use the resist model that is provided within the same setup file, rather than in a separate file.

Description

A required keyword that defines resist model to use for simulations. You indicate that you want to use a resist model defined in a separate file by providing the name of the resist model. You indicate that you want to use an in-line resist model by supplying the string “**inline**” instead of a model name.

The resist model is required for simulating resist and process effects during OPC and analysis. An invalid resist model makes the setup parameters invalid. The resist model must be located somewhere in the path defined by the [modelpath](#) keyword.

Related Topics

[inline_resist](#)

[opticalmodel](#)

secondOpticalmodel

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies a second mask in multiple exposure simulations.

Usage

secondOpticalmodel {*modelname* | **inline**}

Arguments

- ***modelname***

A required argument specifying the name of the optical model to use for simulation. Cannot be used with **inline**.

- **inline**

A required literal string that directs the tools to use the optical model that is provided within the same setup file, rather than in a separate file. Cannot be used with ***modelname***.

Examples

This example shows how to do an image comparison using negative dose. By taking advantage of the Litho tools' double exposure capability, you can calculate the difference between the aerial image of two layers, a "difference image." You can compute the difference in the image when using an optical model with aberrations versus one without.

Note

 The resultant difference image is not the same as the difference between EPEs of two models.

You create this difference image by using the double exposure capability. The following example setup file excerpt demonstrates one method of doing this:

```
opticalmodel      small_248_opt
secondOpticalmodel small_248_opt_coma
background clear clear

# Layers
layer 2 POLY 0 0 opc dark 0 1
layer 3 COPY 0 0 visible dark 1 -1
# where POLY is a layer and COPY is an exact copy of POLY
# the -1 dose subtracts the image of the COPY from the image of POLY
```

Related Topics

[inline_optical2](#)

[opticalmodel](#)

seriftype

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Determines how line ends are fragmented for OPC.

Usage

seriftype {0|1}

Arguments

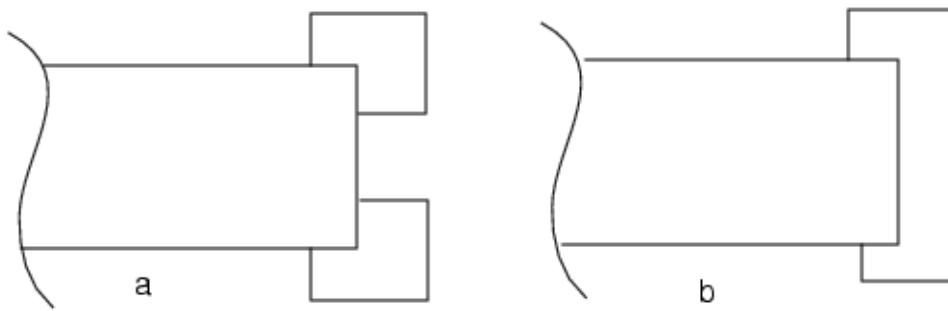
- **0**
A required argument indicating that line ends should be fragmented with hammerheads. This is the default behavior when seriftype is not specified in the setup file.
- **1**
A required argument indicating that line ends should be fragmented with serifs.

Description

The seriftype determines how line ends are fragmented during interfeature fragmentation. When the seriftype = 0, then any edge with the lineEnd property is not fragmented further, resulting in a hammerhead line-end. If seriftype = 1, then edges marked with lineEnd receive normal fragmentation. The [lineEndLength](#) keyword defines what edges are classified as line ends.

This keyword is optional.

Figure 6-32. (a) Full Serifs (b) Hammerheads



Related Topics

[lineEndAdjDist](#)

[lineEndLength](#)

[OPC_USE_LE_DEF](#)

set_kernel_count

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Changes number of active kernels in the optical model.

Usage

set_kernel_count *exposure* [*count*]

Arguments

- ***exposure***

A required argument that selects the first (0) or second exposure (1) for changing the kernel count.

- ***count***

An optional argument that sets the number of kernels to use within the currently active optical model.

If one of the following conditions occur, the keyword resets to using the total number of kernels in the active model:

- *count* is omitted.
- *count* is set less than or equal to 0.
- *count* is set greater than the number of kernels present in the model.

Use the Calibre WORKbench batch command kernsym to choose a kernel count to result in symmetric simulations.

Description

This optional keyword is used to change the number of kernels activated in the active optical model. This keyword enables you to set optical models with different kernel counts to optimize OPC speed and accuracy. For instance, OPC iterations could be run on an optical model with fewer kernels (to increase speed), then be switched back to a full optical model in the final iterations to preserve accuracy.

Examples

In this example, several iterations are performed with an optical model with fewer kernels to increase OPC speed. The full optical model is then switched back to complete the iterations. This improves performance without a degradation in accuracy. This is due the final iterations using the full optical model, resulting in an accurate final convergence.

```
modelpath ./models
opticalmodel full_optical
...
set_kernel_count 0 3
fastIter 5
# restore the full kernel count
set_kernel_count 0
fastIter 2
```

Related Topics

[opticalmodel](#)

[secondOpticalmodel](#)

[kernsym \[Calibre WORKbench User's and Reference Manual\]](#)

set_resist_model

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Changes the resist model during tagging operations.

Usage

```
set_resist_model [{threshold sites  
| use_ref_thresh [sites]}]
```

Arguments

- *threshold*

An optional value between 0 and 1. This changes the resist model to a CTR model with the given threshold. When using *threshold*, you must also specify “sites.”

- sites

An optional keyword that changes the [siteinfo](#) arguments to the following:

- -num x = 3
- -spacing y = 0
- -center z = 0

- use_ref_thresh

An optional argument that sets a CTR model as the resist model. The threshold is chosen to be the reference threshold if the original model is VT5, and if the original model is equivalent to CTR itself, this just copies the CTR value from that model to the new model.

Description

This optional keyword allows the resist model to be changed during OPC iterations, allowing further optimization of OPC speed and accuracy. When no optional arguments are specified, `set_resist_model` restores the default resist model and sites.

For example, to optimize the speed for OPC, several iterations could be performed with a CTR model. To maintain accuracy, the model can be switched back to its default, restoring the sites, in the final iterations, resulting in an accurate final convergence. This approach may gain stability during the first few iterations if the pattern contains bad image properties, because the use of a CTR maintains stability.

Examples

The following example changes the resist model to CTR for the first five iterations, then switches back to the default resist model and sites for the final two iterations.

```
...
set_resist_model 0.1 sites
fastIter 5
# restore the default resist model and sites
set_resist_model
fastIter 2
```

Related Topics

[resistpolyfile](#)

[siteinfo](#)

siteinfo

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies where and how Calibre WORKbench and Calibre OPCpro install control sites.

Usage

```
siteinfo type [-num x | -numx x] [-spacing y] [-center z]
```

Arguments

- *type*

A required argument specifying the type of control sites to be created. Valid types are:

AERIAL — Used when a constant threshold model is specified. It sets the options
-num 3 -spacing 0.04 -center 1.

RESIST — Used with a variable threshold model. It sets -num 12 -spacing 0.04
-center 3.

SAMPLE — Used only when creating a variable threshold model. It sets -num 20
-spacing 0.04 -center 10.

- -num *x* | -numx *x*

An optional argument used to override the default number of control points in the site.
Either form can be used. The value, *x*, must be a positive integer. The default is set by the
type argument.

- -spacing *y*

An optional argument used to define the spacing between control points. The value, *y*, must
be a positive number specified in microns, and must be an integer multiple of the internal
grid size.

This argument is only needed when creating a process model with spacing other than the
default value of 0.04.

When running any of the batch tools, the value of -spacing is inherited from the model. If
you do specify -spacing in the setup file, it must match the value stored in the model file, as
any other values result in a user error.

- -center *z*

An optional argument used to override the default setting for the alignment of the control
points relative to the edge fragment. The value, *z*, must be a positive integer less than -num
and corresponds to the control point (numbered from 0 beginning inside the polygon) that
lies on the edge fragment itself. The default is set by the *type* argument.

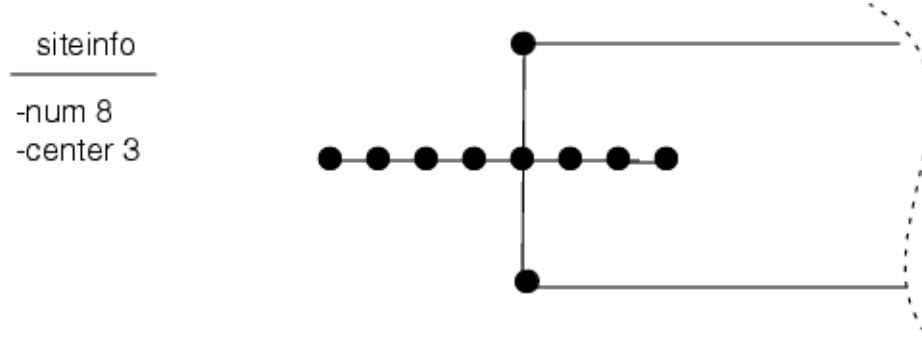
Description

The optional siteinfo keyword tells Calibre how to install control sites. Control sites are the
locations on the simulated wafer where simulation data is evaluated, both to generate the aerial

image contours and to check for edge placement errors. Each control site is comprised of a line containing a set of control points. During simulation, the image intensity is calculated at each of the control points.

At run time, Calibre sparse OPC tools create control sites on edge fragments on the mask target (opc) layer. The siteinfo parameters define the control site length, the number of control points, and their alignment relative to the fragment edge. [Figure 6-33](#) shows how the siteinfo parameters affect control point position. The control points are numbered from the interior of the polygon out, beginning at 0.

Figure 6-33. Site Placement With the siteinfo Keyword



The tools support three types of control sites. The type you use depends on the type of process model specified in the setup parameters. Each type defines default values for number, spacing, and alignment of control points. Optional arguments let you override these defaults.

sse

Type: [Litho Setup File Keywords](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Sets or changes the value of a litho setup variable.

Usage

sse *variable* *value*

Arguments

- ***variable***
Required argument defining the setup variable name.
- ***value***
The new value, or values, for the setup variable.

Description

This command sets or changes the value of most litho setup variables.

Unlike other litho setup file commands, this command appears in the Arbitrary Commands section of a setup file as it is used with variables. Commands in this section are processed sequentially and may appear multiple times.

If a litho setup variable is set both with sse and by setting it in the shell environment before starting a Calibre run, the sse setting has priority.

Examples

```
sse OPC_MAX_ITER_MOVEMENT 0.12
```

Related Topics

[Litho Variables](#)

[gse](#)

stepsize

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies the movement distance in microns for each edge in one OPC iteration.

Usage

stepsize *stepsz*

Arguments

- ***stepsz***

A required argument specifying the OPC iteration step size in microns. The default is [0.010](#) microns. The value must be greater than or equal to [gridsize](#).

Description

This optional keyword specifies the minimum distance in microns that any edge can move in one OPC iteration. During each iteration of the OPC algorithm, fragments move some multiple of ***stepsz***. That multiple is calculated based on the size of the edge placement error for the fragment.

Related Topics

[iterations](#)

svrf_var

Type: [Litho Setup File Keywords](#). Cannot be used with Calibre WORKbench Litho File Tool.
Enables LITHO operation access to variables defined elsewhere in the SVRF rule file.

Usage

svrf_var var_name

Arguments

- **var_name**

A required argument set to the name of the SVRF variable.

Description

The svrf_var command gives the Litho operation access to variables defined elsewhere in the SVRF rule file. Use the svrf_var command in place of a numeric value in any Litho keyword or environment variable statement or any tagging command. To do so, enclose the command in brackets ([]). When the Litho operation encounters the command, it replaces everything from the open square bracket to the close square bracket with the value of the SVRF variable. It writes the resulting Litho statement to the transcript.

To use this command in a setup file, define the variable in the SVRF rule file using the Variable statement. If you fail to define the variable in the SVRF rule file, the Litho operation returns an error and aborts. The SVRF syntax is as follows:

```
VARIABLE name {value[...value] | ENVIRONMENT}
```

Note that the Variable statement in SVRF lets you set the value of a variable to either a specific numeric value or to the value of an environment variable with the same name. For more information about the Variable statement in the SVRF, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).

Note

 You cannot load a setup file into Calibre WORKbench if the file contains the svrf_var command. Any attempt to do so results in an error.

Examples

In the SVRF rule file:

```
VARIABLE mystepsize 0.002
```

In the litho setup file:

```
stepsize [svrf_var mystepsize]
```

In the transcript:

```
stepsize 0.002
```

thirdOpticalmodel

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies a third mask in multiple exposure simulations such as implant modeling.

Usage

thirdOpticalmodel {*modelname* | **inline**}

Arguments

- ***modelname***

A required argument specifying the name of the optical model to use for simulation. Cannot be used with **inline**.

- **inline**

A required literal string that directs the tools to use the optical model that is provided within the same setup file with [inline_optical3](#). Cannot be used with ***modelname***.

Description

An optional keyword that defines the optical model to use for the third mask in multiple exposure simulations. The usage is the same as [opticalmodel](#).

Note that when a third optical model is specified, the [background](#) keyword must provide values for three masks, and there must be a [layer](#) for each.

Examples

Model Loading With Three Optical Models

You might use three optical models to account for the effect of substrate on implant layers. At 22 nm and below, the substrate has a significant effect on implant and cannot be ignored. A technique to handle this is to create a nominal optical model for the implant layer and auxiliary optical models for the “inside” and “outside” of the underlying layer. The masks for the auxiliary models should be specified as “visible”.

```
modelpath ./models
opticalmodel      small_248_opt
secondOpticalmodel aux_inner
thirdOpticalmodel aux_outer
resistpolyfile    ctr.mod

background clear dark dark

# Layers
layer 2 target_implant 0 0 opc dark 0 1
layer 3 aux_inside 128 1 visible clear 1 1
layer 4 aux_outside 128 2 visible clear 2 1
```

Related Topics

[optical_model](#)
[inline_optical3](#)
[opticalmodel](#)
[secondOpticalmodel](#)

tilemicrons

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Specifies the tile size for breaking up the design into smaller pieces for processing.

Usage

tilemicrons *tilesz*

Arguments

- ***tilesz***

A required argument specifying the tiling size in microns. The default is 100 microns. The value must be greater than 0.

Description

When performing optical process correction, Calibre OPCpro and Calibre WORKbench process the data in chunks, called tiles. The data is processed internally in areas no larger than ***tilesz***. If a cell is smaller than ***tilesz***, it is processed as one tile. If a cell is larger than ***tilesz***, it is divided up into areas of ***tilesz*** by ***tilesz*** square microns and each area is processed. Interactions with neighboring areas are always included in the processing.

This keyword is optional.

visibleLayers

Type: [Litho Setup File Keywords](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Changes fragmentation caused by asraf and visible layers. Requires a fragmentLayer statement.

Usage

visibleLayers [*layer* ...]

Arguments

- *layer*

An optional list of layers whose corners are to be used to define visible layer fragmentation, listed from highest priority to lowest priority. All layers must be of type visible or asraf.

If no arguments are given, then no visible layers affect fragmentation or define false edges on the target layer.

Description

An optional user control for visible and asraf layer fragmentation. This keyword can only be used within a [fragmentLayer](#) block.

Note

 If you do not want visible layers to introduce interfeature fragmentation you must use this statement.

This keyword defines a prioritized list of layers whose geometries are used to control generation of fragmentation vertices on the layer to which the fragmentLayer 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 named layers, as long as the fragments created are not smaller than [minedgelength](#). Any fragments which are coincident with or overlapped by active visible or asraf layers are marked as “false edges” that do not move during OPC.

Related Topics

[fragmentLayer](#)

Litho Variables

This section describes setup variables and litho-specific environment variables that control the functioning of the sparse OPC simulation tools. There are two ways to set these variables:

- Use the [sse](#) command. This command is placed at the start of the Arbitrary Commands section of a litho setup file. For example:

```
...
----- Arbitrary Commands Can Follow This Line -----
sse LITHO_ASYM_SOURCE_CELL_CLONE ENABLE
...
clearTagScript
newTag ...
```

- Use the appropriate command in your Linux shell for setting environment variables before invoking Calibre. This method works well with scripts. For example, in a C shell:

```
setenv LITHO_ASYM_SOURCE_CELL_CLONE ENABLE
```

All variables are case-sensitive and must be spelled as shown.

If a variable is set using both methods, the value in the setup file is used. A variable can only be defined once in the litho setup file.

Variables are like [Litho Setup File Keywords](#) in that they set global, generic values. These values can be overridden for specific groups of fragments in some cases using [Tagging Commands](#), which appear in the litho setup file after the variables.

Table 6-13. Litho Setup Variable Summary

| Variable | Description |
|---------------------------|--|
| ALLOW_SMALL_MOVE | Controls whether sites are placed on small isolated fragments. |
| DISALLOW_SHALLOW_ANGLE | Ignores shallow angles during OPC. |
| DONT_MOVE_ALL_ANGLE_EDGES | Inhibits fragment movements to correct for Optical and Process errors. |
| FRAG_BREAK_IN_HALF | Breaks edges in half if fragmentation results in fragments that are smaller than the specified remainder. |
| GRIDS_FOR_ANGLE45 | Controls the distance that 45-degree angle lines move while keeping their endpoints within a specified grid. |
| INTENSITY_INTERP_CENTER | Disables symmetry enhancements for computing intensity. |

Table 6-13. Litho Setup Variable Summary (cont.)

| Variable | Description |
|------------------------------|---|
| INTERACTION_DISTANCE | Overrides the default interaction distance used for hierarchical promotion when simulating an aerial image. |
| LITHO_ALLOW_MAP_BY_ORDER | Allows layers in the setup file to be mapped strictly by order to the SVRF LITHO call. |
| LITHO_ASYM_SOURCE_CELL_CLONE | Controls how the simulations handle cell orientation when calculating the aerial image for a hierarchical design. |
| LITHO_AUTO_PUSH | For hierarchical runs, improves Calibre MTflex performance by pushing some geometries down the hierarchy. |
| LITHO_CLEAN_OPCTM | Indicates the use of postprocessing hierarchical gap and sliver removal. |
| LITHO_CONTEXT_RELAX_MRC | Controls the number of iterations to be carried out with relaxed MRC rules. |
| LITHO_DEANGLE | Improves performance by adjusting input geometries that would introduce snapping problems before simulation. |
| LITHO_HISTOGRAM_OUTPUT_FILE | Specifies the name of the file to which histograms are written. |
| LITHO_IMPL_FRAG_LEN | Specifies minimum fragment length for implicit fragmentation. |
| LITHO_MASK_PRIORITY_MOVEMENT | Defines one mask to have higher priority than other masks when performing OPC in the presence of enclosure rule checks. |
| LITHO_MOVE_CONTEXT | Controls the context fragment optimization in the hierarchical processing scheme. |
| LITHO_MOVEMENT_PRIORITY_TAG | Overrides the LITHO_MASK_PRIORITY_MOVEMENT prioritization for a set of fragments identified by the tag name. |
| LITHO_MRC_CLEANUP_LIMIT | Enables cleanup of MRC violations after OPC has been performed. |
| LITHO_MRC_MODE | Changes default CORNERDIST violation exceptions. |
| LITHO_PRECISE_PROMOTION | Controls the default interaction distance used for hierarchical promotion when simulating an aerial image. |

Table 6-13. Litho Setup Variable Summary (cont.)

| Variable | Description |
|--|--|
| LITHO_PRINT_LEVEL | Controls how much output to send to the Calibre transcript during the litho run. |
| LITHO_PRIORITY_BASED_MOVEMENT | Enables algorithm for adjusting fragments based on EPE and MRC cleanup. |
| LITHO_PROMOTION_TILING | Enables MRC_RULE syntax. |
| LITHO_REPORT_REMOTE_CPU_TIME | Outputs the time on remote runs during OPC processing. |
| LITHO_SIDE_SEGMENT_FILTER | Relaxes LITHO_MRC_MODE 1 violations. |
| LITHO_SIMULATE_CONTEXT | Controls the context fragment optimization in the hierarchical processing scheme. |
| LITHO_SIMULATE_CONTEXT_DISTANCE | Controls how much context is simulated. |
| LITHO_SIMULATE_CONTEXT_SITE_SHIFT | Shifts control sites on long fragments closer to short fragments to increase the context for simulation. |
| LITHO_SUPPRESS_INTERACTION_ERROR | Suppresses interaction distance errors. |
| LITHO_SUPPRESS_MAPNUMBER_ERROR | Suppresses errors produced when MAPNUMBER is not present and there are tags2boxes in the setup file. |
| LITHO_TAG_TIMER | Prints information in the transcript about time spent in each tagging command for single CPU runs. |
| LITHO_TILE_SLIVER_SIZE | Defines a context region near tile boundaries in which to check for non-printing slivers. |
| LITHO_TOLERATE_MINOR_SKEW | Used in conjunction with DONT_MOVE_ALL_ANGLE_EDGES to set a tolerance for angle skews. |
| LITHO_WRITE_DEBUG_SVRF (shell variable only) | Generates additional information useful for creating test cases or debugging. <i>This is not a setup file variable and must be set in a command shell.</i> |
| MASK_CHIP_CORNER | Emulates the behavior of mask corner rounding when performing optics calculations. |

Table 6-13. Litho Setup Variable Summary (cont.)

| Variable | Description |
|--------------------------------|--|
| OPC_AUTO_STEPBY_THRESHOLD | Speeds up simulation time by prevents short jogs from being created, which causes some decrease in precision. |
| OPC_CORR_SEARCH_DISTANCE | Controls the search range for mapping correction layer edge fragments to OPC layer edge fragments. |
| OPC_CROSS_MEEF_DEADBAND | Creates a “dead band,” a range of MEEF values to be treated as if they were 0 MEEF. |
| OPC_DO_SKEW_CHECK | Checks OPC input for skewed edges. |
| OPC_ENCLOSURE_MIN_LENGTH | Defines a minimum fragment length for the enclosure check to be enforced. |
| OPC_ENFORCE_FRAG_LAYER_ORDER | Controls the user-defined fragmentation order. |
| OPC_EPE_TOLERANCE_FRAC | Defines a tolerance factor used when calculating whether or not an edge is within tolerance. |
| OPC_FAST_MOVE | Attempts to determine if a fragment will encounter an MRC restriction prior to movement. |
| OPC_FEEDBACK | Defines a fixed OPC factor by which EPEs are multiplied to calculate the edge movement during each iteration. |
| OPC_IGNORE_ENCLOSURE_AT_CORNER | Turns off enclosure checking in corner fragments in situations where a corner on the block mask overlaps a corner on the phase-shifted mask. |
| OPC_LOW_MEEF_DEFAULT | Specifies the default movement to apply to a low-MEEF fragment. |
| OPC_LOW_MEEF_FREEZE_TAG | Tags low-MEEF fragments that should not be moved. |
| OPC_LOW_MEEF_LIMIT | Sets a limit for low-MEEF handling to occur. |
| OPC_LOW_MEEF_NEG_SCALE | Scales the movement to apply to a low-MEEF fragment marked with the tag OPC_LOW_MEEF_NEG_TAG. |
| OPC_LOW_MEEF_NEG_TAG | Tags low-MEEF fragments that should receive negative edge movements. |

Table 6-13. Litho Setup Variable Summary (cont.)

| Variable | Description |
|---------------------------------|---|
| OPC_LOW_MEEF_POS_SCALE | Scales the movement to apply to a low-MEEF fragment marked with the tag OPC_LOW_MEEF_POS_TAG. |
| OPC_LOW_MEEF_POS_TAG | Tags low-MEEF fragments that should receive positive edge movements. |
| OPC_MAX_ITER_MOVEMENT | Defines the maximum fragment movement per iteration. |
| OPC_MIN_ENCLOSURE_01 | Turns on enclosure checking between Mask 0 and Mask 1 (that is, for fragments such that the Mask 0 fragments enclose the Mask 1 fragments) |
| OPC_MIN_ENCLOSURE_10 | Turns on enclosure checking between Mask 1 and Mask 0 (that is, for fragments such that the Mask 1 fragments enclose the Mask 0 fragments). |
| OPC_MIN_SITE_ANGLE | Ignores control sites between 0 and specified angle. |
| OPC_NEAREST_ADJ | Controls which type of feature is given priority when placing sites on a fragment. |
| OPC_NEW_FRAG_ORDER | Changes the default order of fragmentation. |
| OPC_USE_LE_DEF | Allows line-end fragmentation using the intrafeature command. |
| SEM_CONSIDER_SITE_OFFSET | Calculates a site offset along the direction perpendicular to the edge that gives a correct EPE result. |
| SEM_DO_MORPH | Enables morphological over/under sizing of the Calibre PRINTimage result. |
| SEM_MAX_NONPRINT_MOVE | Controls the handling of non-printable edges. |
| SEM_MORPH_SIZE | Sets the amount of the morphological overunder sizing operation. |
| SEM_PWL_JOT_SIZE | Controls filtering on edges which are excluded from piecewise linear conversion. |
| SEM_USE_PWL | Controls whether a rectilinear or piecewise linear image is drawn. |

Table 6-13. Litho Setup Variable Summary (cont.)

| Variable | Description |
|-------------------|---|
| SEM_USE_SUF_FRAG | Controls the use of fragmentation from the associated setup controls for simulated SEMs. |
| SEM_USE_SUF_SITES | Controls the position and orientation of sites when generating a SEM image with Calibre PRINTimage. |
| VERBOSITY | Controls how errors, warnings, and status messages are reported. |

ALLOW_SMALL_MOVE

Type: [Litho Variables](#). Used by Calibre OPCpro.

Controls whether sites are placed on small isolated fragments.

Usage

`sse ALLOW_SMALL_MOVE {0 | 1}`

Arguments

- **0**

A required argument that disables ALLOW_SMALL_MOVE. This is the default.

- **1**

A required argument that enables ALLOW_SMALL_MOVE.

Description

This setup variable controls whether sites are placed on small, isolated fragments. In this case, a small isolated fragment has a corner on either end, and is less than [minjog](#) in length. By default, ALLOW_SMALL_MOVE is “0” or “false”, which prevents sites from being placed on these fragments. In general, this doesn’t present a problem, since such locations are small and don’t have very much impact on the results. However, there are occasions when this is not the case.

For example, a line end can be omitted from OPC under certain conditions. If the line end is less than minjog in length, it does not receive a site. This may result in inadequate OPC. One simple solution to this problem is set ALLOW_SMALL_MOVE to “1”, so a site is placed on the line end. This ensures that the line end receives corrections, despite the fact that it is less than minjog in length.

Note

 Turning on ALLOW_SMALL_MOVE slightly increases the number of sites, which reduces performance.

Related Topics

[sse](#)

DISALLOW_SHALLOW_ANGLE

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Ignores shallow angles during OPC.

Usage

sse **DISALLOW_SHALLOW_ANGLE {0 | 1}**

Arguments

- **0**
A required argument that applies OPC to all edges at any angle.
- **1**
A required argument that omits edges at an angle of 11.4 degrees or less from OPC processing. This is the default, and allows logos to be processed without creating unwritable masks.

Description

The variable DISALLOW_SHALLOW_ANGLE protects against attempting to apply OPC to edges with sharp spikes (angles <= 11.4 degrees) or very shallow angles (between 168.5 and 180 degrees). It is ignored when [DONT_MOVE_ALL_ANGLE_EDGES](#) is set to 1.

When using DISALLOW_SHALLOW_ANGLE in conjunction with [LITHO_TOLERATE_MINOR_SKEW](#) and [DONT_MOVE_ALL_ANGLE_EDGES](#), there are four possible states:

- **Normal** (default behavior)

```
DONT_MOVE_ALL_ANGLE_EDGES      0
LITHO_TOLERATE_MINOR_SKEW     anything
DISALLOW_SHALLOW_ANGLE        1
```

With DONT_MOVE_ALL_ANGLE_EDGES off, most edges receive OPC corrections. However, since DISALLOW_SHALLOW_ANGLE is set, edges with sharp spikes or very shallow angles do not receive corrections.

- **Off** (Do not perform angle checking)

```
DONT_MOVE_ALL_ANGLE_EDGES      0
LITHO_TOLERATE_MINOR_SKEW     anything
DISALLOW_SHALLOW_ANGLE        0
```

No angle checking is performed when both DONT_MOVE_ALL_ANGLE_EDGES and DISALLOW_SHALLOW_ANGLE are set to 0. This means that all edges receive OPC correction, including logos or any sharp-angled features.

- **Allow ±1 database unit skewed edges**

```
DONT_MOVE_ALL_ANGLE_EDGES      1
LITHO_TOLERATE_MINOR_SKEW      1
DISALLOW_SHALLOW_ANGLE      anything
```

In this case, OPC is performed on edges within ± 1 database unit (dbu) of a 0-45-90 degree boundary. If an edge is not with ± 1 dbu of 0-45-90 degrees, an error message is generated and OPC is not performed on that edge.

- **Strict enforcement of 0-45-90 degree angles**

```
DONT_MOVE_ALL_ANGLE_EDGES      1
LITHO_TOLERATE_MINOR_SKEW      0
DISALLOW_SHALLOW_ANGLE      anything
```

In this case, OPC is performed on edges that are exactly on a 0-45-90 degree boundary. If an edge is not exactly at 0-45-90 degrees, an error message is issued and OPC is not performed.

Related Topics

[DONT_MOVE_ALL_ANGLE_EDGES](#)

[LITHO_TOLERATE_MINOR_SKEW](#)

DONT_MOVE_ALL_ANGLE_EDGES

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Inhibits fragment movements to correct for Optical and Process errors.

Usage

sse **DONT_MOVE_ALL_ANGLE_EDGES{0 | 1}**

Arguments

- **0**

A required argument that disables all angle checking so that all fragments are eligible for OPC. This is the default.

- **1**

A required argument that enables OPC for fragments with corners at 0, 45, and 90 degrees. Skew angles do not receive OPC.

Description

This setup variable applies only to the Calibre products that move fragments to correct for Optical and Process errors.

If set to 1, this variable turns off OPC for all angle fragments (edges with a corner) other than multiples of 45 degrees because in some instances performing OPC on all-angle edges may not be desired. Fragments with no corners are unaffected and do not receive OPC.

When DONT_MOVE_ALL_ANGLE_EDGES is set to 1, it can be used in conjunction with [LITHO_TOLERATE_MINOR_SKEW](#) to permit edges that are 1 dbu off from being at a multiple of 45 degrees to be processed by OPC.

When DONT_MOVE_ALL_ANGLE_EDGES is set to 0 or is left at the default behavior, LITHO_TOLERATE_MINOR_SKEW has no effect but another variable, [DISALLOW_SHALLOW_ANGLE](#), can be used to ignore shallow angles (angles <= 11.4 degrees) during OPC. This allows logos to be processed without creating unwritable masks.

Using LITHO_TOLERATE_MINOR_SKEW and DISALLOW_SHALLOW_ANGLE in conjunction with DONT_MOVE_ALL_ANGLE_EDGES, there are four possible states:

- **Normal** (default behavior)

| | |
|---------------------------|----------|
| DONT_MOVE_ALL_ANGLE_EDGES | 0 |
| LITHO_TOLERATE_MINOR_SKEW | anything |
| DISALLOW_SHALLOW_ANGLE | 1 |

With DONT_MOVE_ALL_ANGLE_EDGES off, most edges receive OPC corrections. However, since DISALLOW_SHALLOW_ANGLE is set, edges with sharp spikes or very shallow angles do not receive corrections.

- **Off** (Do not perform angle checking)

```
DONT_MOVE_ALL_ANGLE_EDGES      0
LITHO_TOLERATE_MINOR_SKEW     anything
DISALLOW_SHALLOW_ANGLE        0
```

No angle checking is performed when both DONT_MOVE_ALL_ANGLE_EDGES and DISALLOW_SHALLOW_ANGLE are set to 0. This means that all edges receive OPC correction, including logos or any sharp-angled features.

- **Allow ±1 database unit skewed edges**

```
DONT_MOVE_ALL_ANGLE_EDGES      1
LITHO_TOLERATE_MINOR_SKEW     1
DISALLOW_SHALLOW_ANGLE        anything
```

In this case, OPC is performed on edges within ±1 database unit (dbu) of a 0-45-90 degree boundary. If an edge is not with ±1 dbu of 0-45-90 degrees, an error message is generated and OPC is not performed on that edge.

- **Strict enforcement of 0-45-90 degree angles**

```
DONT_MOVE_ALL_ANGLE_EDGES      1
LITHO_TOLERATE_MINOR_SKEW     0
DISALLOW_SHALLOW_ANGLE        anything
```

In this case, OPC is performed on edges that are exactly on a 0-45-90 degree boundary. If an edge is not exactly at 0-45-90 degrees, an error message is issued and OPC is not performed.

Related Topics

[DISALLOW_SHALLOW_ANGLE](#)
[LITHO_TOLERATE_MINOR_SKEW](#)

FRAG_BREAK_IN_HALF

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Breaks edges in half if fragmentation results in fragments that are smaller than the specified remainder.

Usage

sse **FRAG_BREAK_IN_HALF {0 | 1}**

Arguments

- **0**
A required argument that specifies that when the intrafeature remainder value prevents further [intrafeature](#) fragmentation, the central fragment is not split. This is the default.
- **1**
A required argument that specifies that when the intrafeature fragmentation remainder is greater than the fragment that would result from further fragmentation, the remaining portion is split in half.

Description

Setting this value changes the behavior of intrafeature fragmentation on an edge. If **FRAG_BREAK_IN_HALF** is set to 1, intrafeature fragmentation divides a fragment that is less than remainder + 2 ripple lengths into two equal parts instead. Fragments that are not at least remainder + 2 ripple lengths long would violate the remainder constraint if the last ripples were applied.

Examples

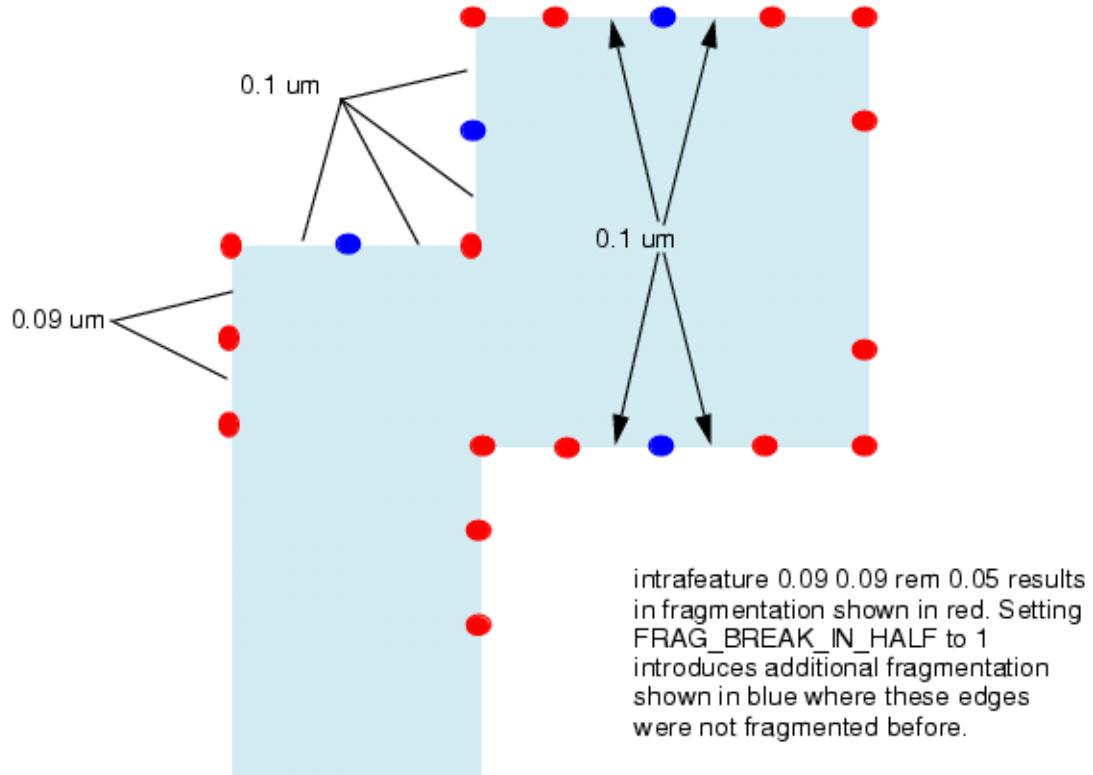
Assume the setup file contains the following:

```
intrafeature 0.09 0.09 rem 0.05
```

Applied to a 0.2 micron edge, the edge would not be fragmented by default because $0.9 + 0.9 + 0.5 > 0.2$. However, with **FRAG_BREAK_IN_HALF** set to 1, the edge is fragmented into two 0.1 micron fragments.

Applied to a 0.38 micron edge, by default the fragments would be 0.09, 0.2, and 0.09 microns long, because there is room for only one ripple on each corner. As shown in [Figure 6-34](#), by setting **FRAG_BREAK_IN_HALF** to 1, the edge is broken into 4 fragments. The red endpoints are default intrafeature fragmentation, and the blue endpoints are added because of **FRAG_BREAK_IN_HALF**.

Figure 6-34. Effect of FRAG_BREAK_IN_HALF



Related Topics

[intrafeature](#)

GRIDS_FOR_ANGLE45

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Controls the distance that 45-degree angle lines move while keeping their endpoints within a specified grid.

Usage

`sse GRIDS_FOR_ANGLE45 floating_point_number`

Arguments

- *floating_point_number*

A required argument specifying the grid size to use for snapping 45-degree lines. The default value is set by [stepsize](#).

Description

This variable 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. This assumes the following conditions hold true:

- The original data is on a grid at least 5 nm.
- The value of the [stepsize](#) keyword is at least 5 nm.

```
sse GRIDS_FOR_ANGLE45 0.005
```

INTENSITY_INTERP_CENTER

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Disables symmetry enhancements for computing intensity.

Usage

sse INTENSITY_INTERP_CENTER {0 | 1}

Arguments

- **0**

A required argument that turns off symmetry enhancements, restoring behavior from before 2003.3 release. This is not recommended.

- **1**

A required argument that turns on symmetry enhancements in intensity computation. This is the default.

INTERACTION_DISTANCE

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Overrides the default interaction distance used for hierarchical promotion when simulating an aerial image.

Usage

sse INTERACTION_DISTANCE *d*

Arguments

- *d*

A required argument that specifies the promotion radius around control sites in microns.
The default is the optical radius defined for the optical model.

Description

This variable affects the aerial image simulations for hierarchical layouts. It allows you to control how much geometry in an adjacent cell reference gets promoted, making it visible to the optical model. Refer to the following figures.

Figure 6-35. INTERACTION_DISTANCE

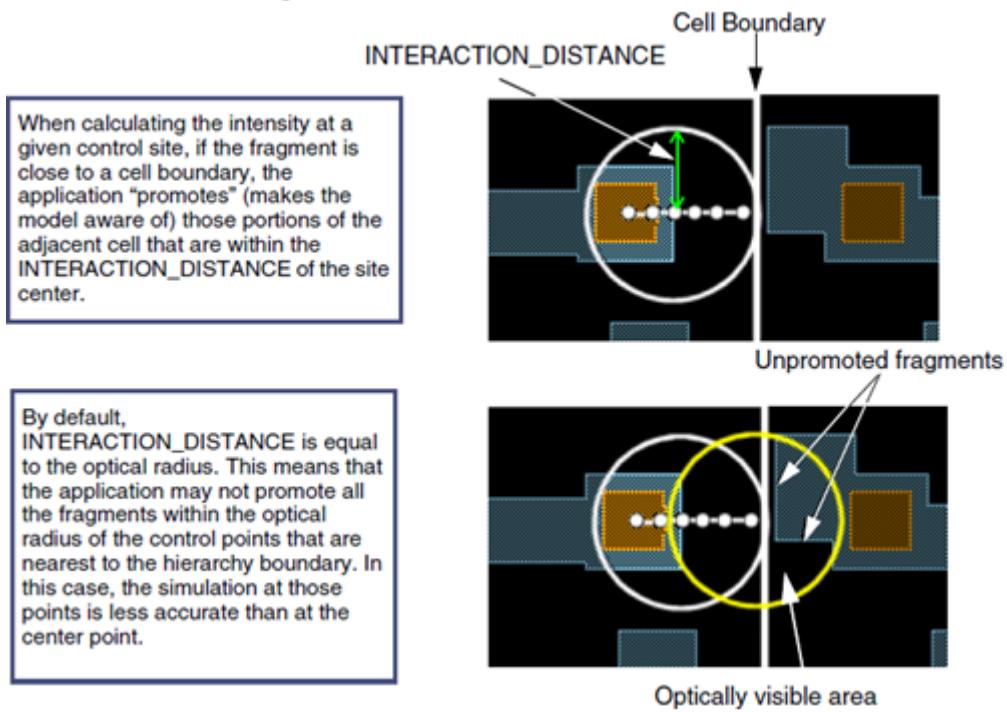
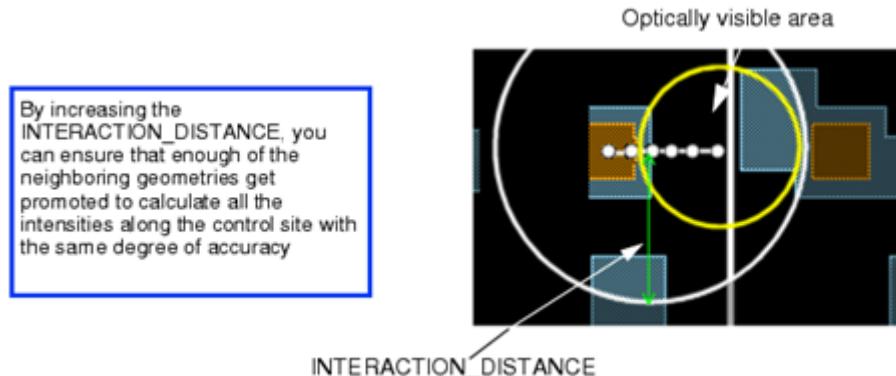


Figure 6-36. Improving Accuracy by Increasing INTERACTION_DISTANCE



Related Topics

[opticalmodel](#)

LITHO_ALLOW_MAP_BY_ORDER

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Allows layers in the setup file to be mapped strictly by order to the SVRF LITHO call.

Usage

sse **LITHO_ALLOW_MAP_BY_ORDER {0 | 1 | 2}**

Arguments

- **0**

A required argument that sets mapping to use layer names only. If the names do not match, Calibre returns an error. This is the default value.

- **1**

A required argument that sets mapping to use names first and then order. The input layers listed in the [LITHO OPC](#), [LITHO ORC](#), or [LITHO PRINTIMAGE](#) call are initially mapped by name to layers in the setup file. The remaining layers are mapped by order to the layer numbers in the setup file. It is best to number the setup file layers in increasing order for cases where 1 is used.

- **2**

A required argument that sets mapping to use layer order only. Names are ignored.

Description

Normally mapping of inputs is done by name and the layers in the setup file must match the names of the layers on the SVRF LITHO call. With this option, the mapping can be done strictly by order.

Examples

This example shows how LITHO_ALLOW_MAP_BY_ORDER maps input layers into the setup file. When LITHO_ALLOW_MAP_BY_ORDER is set to 1 with the following LITHO calls and setup file settings:

In the SVRF rule file:

```
PHASE2_OPCT
    LITHO OPC FILE "setup.in" TAG TARGET MASK2 PHASE1 PHASE2 MAPNUMBER 78 }
MASK2_OPCT
    LITHO OPC FILE "setup.in" TAG TARGET MASK2 PHASE1 PHASE2 MAPNUMBER 10 }
TARGET_OPCT
    LITHO OPC FILE "setup.in" TAG TARGET MASK2 PHASE1 PHASE2 MAPNUMBER 99 }
PHASE1_OPCT
    LITHO OPC FILE "setup.in" TAG TARGET MASK2 PHASE1 PHASE2 MAPNUMBER 2 }
```

In the setup file:

```
layer 2 FOR1 31 0 correction phase180 1 1.10
layer 5 TAG 31 0 island dark 0 1.000
layer 10 MASK2 0 0 correction dark 0 1.000
layer 78 FOR2 31 0 correction clear 1 1.10
layer 99 TARGET 0 0 opc 0.00 0.00 0 1.000
```

the layer TAG is mapped to layer 5, MASK2 is mapped to layer 10, and TARGET is mapped to layer 99. PHASE1 is mapped to layer 2 and PHASE2 is mapped to 78.

When the LITHO_ALLOW_MAP_BY_ORDER is set to 2 with the following setup file layers list:

```
layer 24 FOR_PHASE1 31 0 correction phase180 1 1.10
layer 5 FOR_TAG 31 0 island dark 0 1.000
layer 19 FOR_MASK2 0 0 correction dark 0 1.000
layer 65 FOR_PHASE2 31 0 correction clear 1 1.10
layer 13 FOR_TARGET 0 0 opc 0.00 0.00 0 1.000
```

the input layer TAG is mapped to layer 5, TARGET is mapped to layer 13, MASK2 is mapped to 19, PHASE1 is mapped to layer 24, and PHASE2 is mapped to layer 65.

LITHO_ASYM_SOURCE_CELL_CLONE

Type: [Litho Variables](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Controls how the simulations handle cell orientation when calculating the aerial image for a hierarchical design.

Usage

```
sse LITHO_ASYM_SOURCE_CELL_CLONE {ENABLE | DISABLE | FORCE |  
          PARTIAL}
```

Arguments

- **ENABLE**

A required argument specifying that Calibre checks whether any asymmetric sources are present in the optical model. If so, the Calibre database constructor clones cells which have rotated or mirrored placements (that is, transformed cells), so that hierarchical simulation of the rotated structures is correct. If the optical model references a missing source map, the source is assumed to be asymmetric.

ENABLE is the default behavior if the variable is not specified.

- **DISABLE**

A required argument that specifies Calibre should behave as though the optical source is symmetric. This prevents automatic cloning of transformed cells.

- **FORCE**

A required argument that specifies Calibre should behave as though the optical source is asymmetric. This forces transformed cells to be cloned.

- **PARTIAL**

A required argument that specifies cloning is to be applied only to cells that are rotated, but not those that are mirrored, regardless of the source shape.

Description

Note

 Since 2012, all the Calibre RET tools automatically determine whether cell placements require cloning based not just on the optical source but also based on how computations are shared across processors. Generally you do not need to set this variable.

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.

These calculations require checking the optical model. When the source in an optical model is defined as a source map and the source map is missing, then Calibre cannot check whether the source map is symmetric. To be safe, an unknown source is treated as asymmetric, which leads to cloning unless you explicitly set it to DISABLE or PARTIAL.

Examples

This example enables optical model checking for cell cloning:

```
sse LITHO_ASYM_SOURCE_CELL_CLONE ENABLE
```

Related Topics

[opticalmodel](#)

[siteControl OFFSET_FROM](#)

[siteControl OFFSET_ROTATED](#)

LITHO_AUTO_PUSH

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

For hierarchical runs, improves MTflex performance by pushing some geometries down the hierarchy.

Usage

`sse LITHO_AUTO_PUSH {0 | 1}`

Arguments

- **0**

A required argument that disables LITHO_AUTO_PUSH. Polygons are not pushed down the hierarchy and Calibre processes them at the current level.

- **1**

A required argument that enables a light automatic push of layer data. Polygons are pushed partially down the hierarchy to speed up run time. This is done if the layer is used as input to another SVRF operation where polygons are processed at its current hierarchy level. If the layer is copied to the output then a Push operation is not performed.

This is the default behavior if the variable is not set.

Description

This environment variable performs a push on output polygons to improve scaling in Calibre MTflex runs. This operation is a faster and less complete version of the SVRF command Push. When Push is used in SVRF, an entire layer is pushed down the hierarchy to the lowest possible level. Using the SVRF Push operation results in longer runs than setting LITHO_AUTO_PUSH to 1. Auto push performs a quick push on output results to keep them contained in bin cells. This feature can be disabled by setting LITHO_AUTO_PUSH to 0.

Auto push may be skipped in situations where the result is being written to disk and is not going to be used in further hierarchical operations.

Related Topics

[Push \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

LITHO_CLEAN_OP

Type: [Litho Variables](#). Used by Calibre OPCpro.

Indicates the use of post-OPC hierarchical gap and sliver removal.

Usage

```
sse LITHO_CLEAN_OP {nsteps | HEIGHT h [MINLEN m] }
```

Arguments

- ***nsteps***

A required argument that indicates the maximum size of notches to be cleaned up during postprocessing. The value should be a positive integer. It directs Calibre OPCpro to fill in notches up to *nsteps**[stepsize](#) high that are shorter than [minedgelength](#) in length. Normally, 1 or 2 should be used for *nsteps*.

- **HEIGHT *h***

A required argument that defines the upper bounds in microns on the height constraint that a gap or sliver must meet in order to be cleaned up. The application fills in notches up to *h* microns high that are shorter than MINLEN microns in length.

- **MINLEN *m***

An optional argument used in conjunction with **HEIGHT** to further constrain the notches to be cleaned up. MINLEN *m* defines the maximum length in microns for a gap or sliver to be considered a “notch”. Any notch shorter than *m* is filled or removed. The default is [minedgelength](#).

Description

The LITHO_CLEAN_OP setup variable instructs the application to remove hierarchical gap and sliver features (outward and inward notches) during post-OPC processing.

You specify the filter used to identify gaps and slivers using one of the following methods:

- Specify the height in terms of ***nsteps***. The application fills in notches with height \leq *nsteps**[stepsize](#) and length of strictly less than the minimum edge length.
- Specify the height and length constraints in microns. The application fills in notches up to ***h*** in microns which are shorter than *m* microns in length.

Related Topics

[jogsmooth](#)

[LITHO_TILE_SLIVER_SIZE](#)

LITHO_CONTEXT_RELAX_MRC

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Controls the number of iterations to be carried out with relaxed MRC rules.

Usage

```
sse LITHO_CONTEXT_RELAX_MRC {count}
```

Arguments

- **count**

A required argument that indicates the number of iterations where MRC should be relaxed. For example, if this is set to 1, the fragment ignores MRC for the first iteration. As of version 2015.3, the default is $N-2$, where N is the number of iterations.

Description

[LITHO_SIMULATE_CONTEXT](#) enforces MRC against two positions for promoted fragments: the current position and the final position, which was determined in a lower level of hierarchy. Sometimes this restriction impacts how fragments converge. When MRC constraints are relaxed with this variable, MRC is enforced against the current locations for the iterations specified in **count**. When the iterations have exceeded the value specified in **count**, [LITHO_SIMULATE_CONTEXT](#) enforces MRC against the final positions, imposing a stricter set of constraints.

When [LITHO_SIMULATE_CONTEXT](#) is enabled, OPC iterates and moves the context fragments even though the final location is known. (The location is known because it was determined at a lower level of hierarchy.) This helps nearby fragments to converge on an answer in the same way that they would in a flat simulation. This extra computation increases the run time but this can be controlled using [LITHO_SIMULATE_CONTEXT_DISTANCE](#).

In particular, there are cases where fragments receive a hintOffset to start and then reduce that initial bias as the iterations progress. If the MRC restrictions prevent that initial hintOffset then the result might be different. This variable should be used with care but it can improve matching results between hierarchical and flat runs.

Setting a value of **count** that is greater than the total number of OPC iterations can lead to improved flat versus hierarchical results consistency. However, it is possible that such a setting can lead also to MRC violations after OPC. The value of **count** should be set to at least $(N-1)$ or preferably $(N-2)$, where N represents the number of iterations, in order to ensure that MRC violations do not exist after OPC.

Related Topics

[LITHO_SIMULATE_CONTEXT](#)

[opcTag](#)

newTag ... -how MRC

LITHO_DEANGLE

Type: [Litho Variables](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Improves performance by adjusting input geometries that would introduce snapping problems before simulation.

Tip

i If your run uses both Calibre nmOPC and sparse OPC, use the SVRF command DEANGLE instead of LITHO_DEANGLE. This makes the rule file easier to maintain by using the same preparation for all OPC tools.

Usage

sse LITHO_DEANGLE *max_deviation*

Arguments

- ***max_deviation***

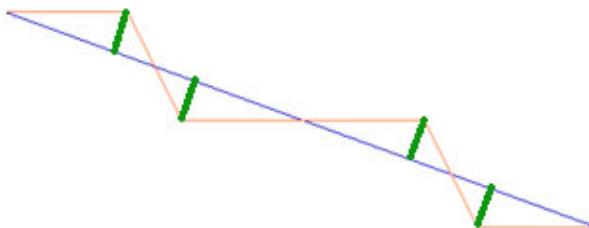
A required argument that specifies the maximum deviation from the original edge to the deangled edge, expressed in microns. Setting ***max_deviation* <=0** turns de-angling off. The default is **0** (off).

Description

This command improves performance by adjusting input geometries that would introduce snapping problems. When used, this feature preprocesses all input layers, converting those edges that are not 45 degree multiples to combinations of 45 and 90 degree edges. The preprocessed input layers are then simulated as usual. If the input layers do not contain skew edges, there is no reason to set this variable.

In [Figure 6-37](#), the green lines show the distance between the original edge (blue) and the approximation (red), which cannot exceed ***max_deviation***. If the deviation from the original fragment exceeds the specified amount, the edge is left untouched.

Figure 6-37. De-angling a Skew Fragment With Stairsteps



Using this feature gives different results from runs that do not use this feature. The differences are because de-angling breaks skew edges up into smaller edges. Any fragmentation is applied after de-angling to these smaller edges, rather than the original skew edges.

Whenever LITHO_DEANGLE encounters an off-by-one skew edge, it corrects it by moving one of the vertices as needed to make it a single 45 degree edge. A skew edge is considered off-by-one when the difference between the change in the x direction and the change in the y direction is exactly one database unit.

Generally, setting ***max_deviation*** to small values increases run times without significant benefit. For most technology processes, good results can be obtained with values between 0.001 and 0.010. Setting ***max_deviation*** to less than 0.001 is not recommended.

Related Topics

[LITHO_TOLERATE_MINOR_SKEW](#)

[DEANGLE \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

LITHO_HISTOGRAM_OUTPUT_FILE

Type: [Litho Variables](#). Used by Calibre ORC and Calibre OPCpro.

Specifies the name of the file to which histograms are written.

Usage

sse **LITHO_HISTOGRAM_OUTPUT_FILE** *filename*

Arguments

- *filename*

A required argument that provides the name of the file that you want the histogram written to.

Description

This setup variable is used in conjunction with the [histogram](#) tagging command, which generates statistics on the frequency of the occurrence of specified tags. This variable allows you to specify the name of the file to which histogram data is written.

LITHO_IMPL_FRAG_LEN

Type: [Litho Variables](#). Used by Calibre ORC and Calibre OPCpro.

Specifies minimum fragment length for implicit fragmentation.

Usage

sse **LITHO_IMPL_FRAG_LEN** *length*

Arguments

- *length*

A required argument that specifies the minimum length of a fragment that qualifies for implicit fragmentation. A value of 0 or less disables implicit fragmentation. The default value is 1.5 x optical diameter.

Description

This setup variable specifies the minimum fragment length to qualify for implicit fragmentation. Implicit fragmentation is performed near hierarchy boundaries. This can improve performance when long fragments have small sections that interact with something outside of the cell. In this case, the fragment is split near the hierarchy boundary. The exact point where the fragmentation is performed is near the interaction boundary, but it is adjusted such that no fragment shorter than minedgelength is created.

Since implicit fragmentation results from hierarchy boundaries, it creates differences in fragmentation between flat and hierarchical runs. In order to minimize these differences, it is generally performed only for long fragments. The default value of (1.5 x optical radius) is recommended.

LITHO_MASK_PRIORITY_MOVEMENT

Type: [Litho Variables](#). Used by Matrix OPC.

Defines one mask to have higher priority than other masks when performing OPC in the presence of enclosure rule checks.

Usage

sse **LITHO_MASK_PRIORITY_MOVEMENT** *mask_number*

Arguments

- ***mask_number***

A required argument that specifies a mask to have higher priority.

Related Topics

[MATRIX_OPC](#)

[LITHO_MOVEMENT_PRIORITY_TAG](#)

LITHO_MOVE_CONTEXT

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Controls the context fragment optimization in the hierarchical processing scheme.

Usage

sse LITHO_MOVE_CONTEXT {0 | 1}

Arguments

- **0**

A required argument that does not enable additional functionality. This is the default.

- **1**

A required argument that causes context fragments to be shifted according to their EPE on the first iteration only.

Description

This variable controls optimization of context fragments when processing hierarchical layouts. A context fragment is a fragment from a shape that was moved in a child cell but that must be promoted to the parent cell in order to simulate nearby promoted shapes. Generally, when a fragment has been simulated in a child cell it is unnecessary to simulate it again in the parent.

When this variable is set, context fragments receive sites and are shifted according to their EPE during the first iteration. (Compare this to [LITHO_SIMULATE_CONTEXT](#), which moves the fragments on all iterations.) Setting this variable increases simulation time but can improve results when MRC rules block promoted edges from reaching target.

This variable achieves a result similar to [LITHO_SIMULATE_CONTEXT](#) in less time but also with less precision.

Note

 Do not set both [LITHO_SIMULATE_CONTEXT](#) and [LITHO_MOVE_CONTEXT](#) in the same run.

See the [LITHO_SIMULATE_CONTEXT](#) “Examples” for more information on context fragments.

LITHO_MOVEMENT_PRIORITY_TAG

Type: [Litho Variables](#). Used by Matrix OPC.

Overrides the LITHO_MASK_PRIORITY_MOVEMENT prioritization for a set of fragments identified by the tag name.

Usage

sse **LITHO_MOVEMENT_PRIORITY_TAG** *tagname*

Arguments

- *tagname*

A required argument that identifies a set of tagged fragments to override prioritization set by LITHO_MASK_PRIORITY_MOVEMENT.

Description

When using [LITHO_MASK_PRIORITY_MOVEMENT](#) to give one mask priority over the other, this variable lets you override the prioritization for a set of fragments identified by the tag name.

Related Topics

[Tagging Commands](#)

[Pre-Defined Tags](#)

[MATRIX_OP](#)

LITHO_MRC_CLEANUP_LIMIT

Type: [Litho Variables](#). Used by Calibre ORC and Calibre OPCpro.

Enables cleanup of MRC violations after OPC has been performed.

Usage

sse **LITHO_MRC_CLEANUP_LIMIT** *value*

Arguments

- *value*

A required argument that indicates the maximum distance in microns that a fragment is allowed to move in order to resolve MRC violations. The default value is **0** (disabled).

Description

This variable enables the cleanup of MRC violations that exist after OPC has been performed on a tile of a cell (or the entire cell in the event that it is not large enough to be tiled). The cleanup routine checks each fragment to see if it is in a legal spot. If not, the bias is adjusted up to the specified number of steps in an attempt to resolve the violation. If the violation cannot be resolved by this process, then the bias is left alone. This variable performs the same function as the [fastIter -mrc_cleanup](#) tagging command.

LITHO_MRC_MODE

Type: [Litho Variables](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Changes default CORNERDIST violation exceptions.

Usage

sse LITHO_MRC_MODE {0 | 1}

Arguments

- **0**
A required argument that retains the original filtering method.
- **1**
A required argument that enables an alternative filtering method. This is the default setting.

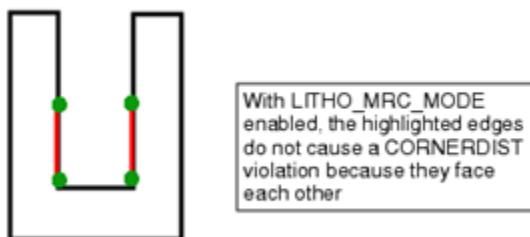
Description

This variable changes the method by which CORNERDIST filters violations. CORNERDIST is a secondary keyword in the deprecated keywords `opcMinCheckTag`, `OPC_MIN_INTERNAL`, and `OPC_MIN_EXTERNAL`. When enabled with `LITHO_MRC_MODE 1`, a CORNERDIST violation is excluded only if:

- The edges causing the error are separated by less than the CORNERDIST value
- The fragments that they are derived from must be part of the same polygon loop
- The fragments that they are derived from must not face each other

For example, two fragments on the opposite side of a “U” shape would not qualify for a CORNERDIST exclusion (see [Figure 6-38](#)).

Figure 6-38. CORNERDIST Using LITHO_MRC_MODE 1



You can also enable this mode by using CORNERDIST 0.

Note
OPC results differ with `LITHO_MRC_MODE 1` as the MRC restrictions on movement differ slightly.

LITHO_PRECISE_PROMOTION

Type: [Litho Variables](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Controls the default interaction distance used for hierarchical promotion when simulating an aerial image.

Usage

sse LITHO_PRECISE_PROMOTION {0 | 1}

Arguments

- **0**

A required argument that promotes everything within the optical radius of center of each control site.

- **1**

A required argument that promotes everything that is within (optical radius + site spacing * (site num - site center)). This is the default.

Description

This variable instructs the applications to use an interaction distance (also known as promotion distance) for hierarchical promotion calculated based on the length of the control sites.

You would set this variable to ensure that data from sampling points at the end of a control site are as accurate as data at the center. Refer to the figures in “[INTERACTION_DISTANCE](#)” for a graphical description of this problem.

When used, the application calculates the interaction distance as:

```
(optical radius + site spacing * (site num - site center))
```

This variable is overridden by [INTERACTION_DISTANCE](#) when it is set.

[INTERACTION_DISTANCE](#) directly sets the interaction distance for hierarchical promotion.

LITHO_PRINT_LEVEL

Type: [Litho Variables](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Controls how much output to send to the Calibre transcript during the litho run.

Usage

sse LITHO_PRINT_LEVEL *level*

Arguments

- ***level***

A required argument that specifies the amount of output. Valid settings are:

0 — No extra printing. This is the default.

1 — Generates a Cell Completion Report for each cell. The format of this report is fixed; consequently, you can write a transcript parser for extracting the information you need. The format is:

```
>>> CELL COMPLETION REPORT
cell           mycellname
cell_number    83/83
cell_time      0.26 min
LVHEAP = 2/17/17
<<<
```

cell — The cell being processed.

cell_number — An ordinal from 1 to maximum cell number.

cell_real_time <time> elapsed_total <total_time> — The real time which passed from the time the cell started processing to the time it completed processing. In an MTflex or MT environment, it is noted that many other cells could have also been processed during this time, so the time should be interpreted appropriately. The **elapsed_total** is the total REAL time since the first cell began processing.

When the total processing time for the cell is less than 4 seconds, the COMPLETION REPORT is abbreviated to the following:

```
Completed cell (7330/8372)      mycellname
```

2 — Generates a Tile Completion Report for each tile when using [LITHO_PROMOTION_TILING](#). The format is:

```
>>> TILE COMPLETION REPORT
cell           mycellname
tile          7/10          70.0%
x              3/3
y              1/3
depth         1/2
tile_real_time 0.15 min   elapsed_in_cell 0.15
LVHEAP = 11/17/17
<<<
```

The information listed is:

cell — The cell being processed.

tile — The tile number, ordered from 1 to the total number of tiles in this cell.

x — The current and total number of x tiles.

y — The current and total number of y tiles.

depth — The level of the current tile within the dynamic pseudo-hierarchy of the cell.

tile_real_time — The tile_time is the real time which passed from the time the tile started processing to the time it completed processing. In an MTflex or MT environment, many other cells could also have been processed during this time, so the time should be interpreted appropriately.

elapsed_in_cell — The total REAL time thus far into processing of the current cell.

LVHEAP — Shows the memory usage in Mbytes as (current/allocated/max).

The tile completion report is skipped when tile_time is less than 4 seconds.

3 — Specifies counting the number of edges, fragments, sites and promoted area in tiles. The tile completion report contains the following lines:

tot_edges — Total number of edges input to the litho tools in the tile.

pfrags — Number of previously corrected fragments.

tot_frags — Total number of fragments on all layers in the tile.

tot_sites — Total number of sites in the tile.

ia_cover — The percentage of the cell which can be process independently of all other cells; if this value is close to 100 percent, then the hierarchical processing efficiency is degraded.

4 — Specifies periodically reporting the CPU load scaling; output is in the following format:

cpu_load *number* *timestamp* *time*

Extracting this information and plotting the load versus the timestamp gives a plot of CPU scalability over time during the run.

5 — Specifies reporting the CPU load scaling before and after the completion of each tile. This gives a more granular plot of scalability.

Description

This variable controls the level of information the litho tools write to the transcript (stdout). The transcript includes an echo-back of the setup file and all models as well as all the information needed to completely reconstruct the setup file and models. In addition, the transcript contains useful information for analyzing the run time and memory usage of LITHO during its operation.

LITHO_PRIORITY_BASED_MOVEMENT

Type: [Litho Variables](#). Used by Calibre OPCpro.

Controls algorithm for adjusting fragments based on EPE and MRC cleanup.

Usage

sse LITHO_PRIORITY_BASED_MOVEMENT {0 | 1}

Arguments

- **0**
A required argument that disables priority-based movement.
- **1**
A required argument that enables priority-based movement. This is the default.

Description

This variable controls an algorithm that prioritizes fragment movement based on EPE. When the correction for two or more fragments interfere with each other and encounter MRC restrictions, there are cases where suboptimal OPC corrections occur. The algorithm prioritizes fragments for certain situations in order to get a better solution. As of version 2015.3, this algorithm is on (1) by default.

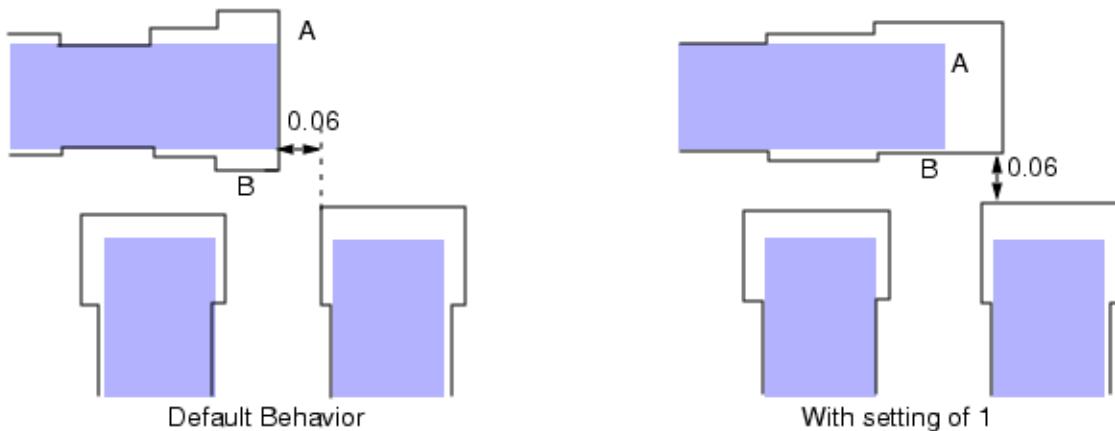
Prior to 2015.3, this algorithm (although recommended) was not run by default because it caused a significant change in output compared to earlier releases.

Examples

Example 1 - Frozen Neighbors and a Race Condition

Suppose you have a rule file that sets a minimum distance between line ends of 0.060 and a target layout as shown in [Figure 6-39](#). Without the algorithm, a race condition moves fragment B first, keeping A in a position that creates severe line-end pullback. Fragment A has worse EPE than fragment B. Setting LITHO_PRIORITY_BASED_MOVEMENT to 1 allows A to move first.

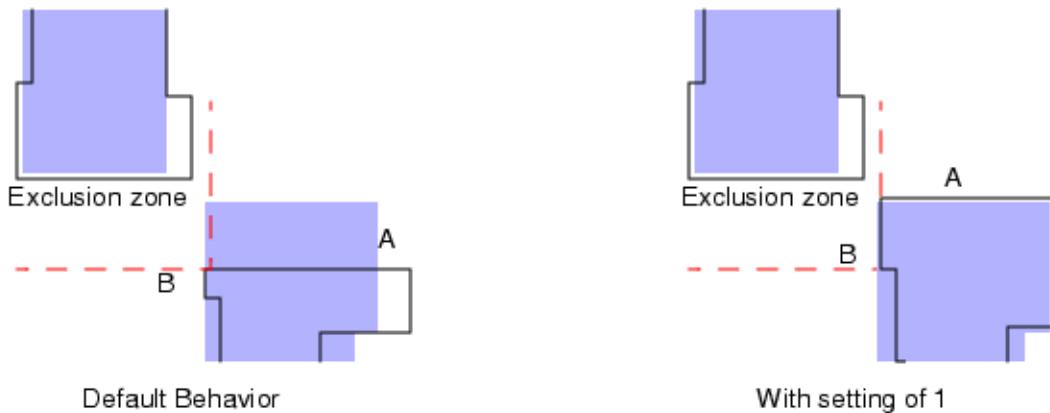
Figure 6-39. LITHO_PRIORITY_BASED_MOVEMENT Effects



Example 2 - Problems Caused by Cleaning Up an MRC Violation

After OPC, MRC flags a space violation where the corners of two hammerheads come too close to each other because of a slight overlap in the extended area. The default correction involves a line end (fragment A) being moved back significantly, leading to pullback. With the priority algorithm (default as of 2015.3), the correction code also checks the side of the hammerhead (fragment B). It calculates that the violation can be removed by moving fragment B in a single stepsize. The priority-based algorithm uses the smaller correction movement instead.

Figure 6-40. Better MRC Violation Clean-Up



LITHO_PROMOTION_TILING

Type: [Litho Variables](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Enables MRC_RULE syntax.

Usage

sse LITHO_PROMOTION_TILING {0 | 1}

Arguments

- **0**

A required argument that disables MRC_RULE syntax. This is the default.

- **1**

A required argument that enables MRC_RULE syntax.

Description

This variable enables the [MRC_RULE](#) syntax, one of the rule-checking methods for internal and external checks. The other is a combination of the variables OPC_MIN_EXTERNAL, OPC_MIN_INTERNAL, and opcMinCheckTag command.

LITHO_PROMOTION_TILING also addresses interstitial issues that regular tiling can create. Currently, when tiles are reassembled into the parent cell, there can be small differences at the boundaries of tiles, due to numeric noise, which are resolved with a post-tiling resolution step. In cells with many tiles, this operation can, on occasion, take excessively long.

LITHO_PROMOTION_TILING makes use of implicit fragmentation. If the recipe turns off implicit fragmentation with “sse LITHO_IMPL_FRAG_LEN 0”, the run exits with the following setup file error:

```
SETUP FILE ERROR: Promotion Tiling
    Implicit fragmentation value is set to 0 (off) which is not compatible
    with promotion tiling
```

LITHO_PROMOTION_TILING creates additional opportunities for multiple CPUs to do this post-tiling resolution step such that the cell completion takes less time. Enabling LITHO_PROMOTION_TILING can cause minor changes to your OPC output.

Note

 If LITHO_PROMOTION_TILING is set to 1, but MRC_RULE syntax is absent from the rule file, Calibre reverts to OPC_MIN_EXTERNAL and OPC_MIN_INTERNAL as the default rule-checking method if they are included.

Related Topics

[MRC_RULE](#)

LITHO_IMPL_FRAG_LEN

LITHO_REPORT_REMOTE_CPU_TIME

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Outputs the time on remote runs during OPC processing.

Usage

sse LITHO_REPORT_REMOTE_CPU_TIME {0 | 1}

Arguments

- **0**
A required argument that does not report CPU time for remote runs. This is the default.
- **1**
A required argument that provide CPU time for execution of remote runs in transcript.

Description

This variable enables reporting of CPU time consumed during the cell processing loop of a remote Calibre OPCpro run. When this is set to 1 before invoking Calibre, the remote processes report progress as they finish each major operation. It may impact performance because of the much larger number of writes to the output, so is recommended primarily for debugging remote runs.

When this variable is not enabled (default behavior), the final log still includes full CPU time statistics including those of remote processes and total time by command group.

LITHO_SIDE_SEGMENT_FILTER

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Relaxes LITHO_MRC_MODE 1 violations.

Usage

sse LITHO_SIDE_SEGMENT_FILTER {0 | 1}

Arguments

- **0**
A required argument that does not change [LITHO_MRC_MODE](#). This is the default.
- **1**
A required argument that relaxes LITHO_MRC_MODE 1 restrictions on side segments.

Description

This variable relaxes restrictions on side segments in the case that the moving edge and the opposing edge project onto each other and have a pre-opc violation. It only affects runs with LITHO_MRC_MODE 1.

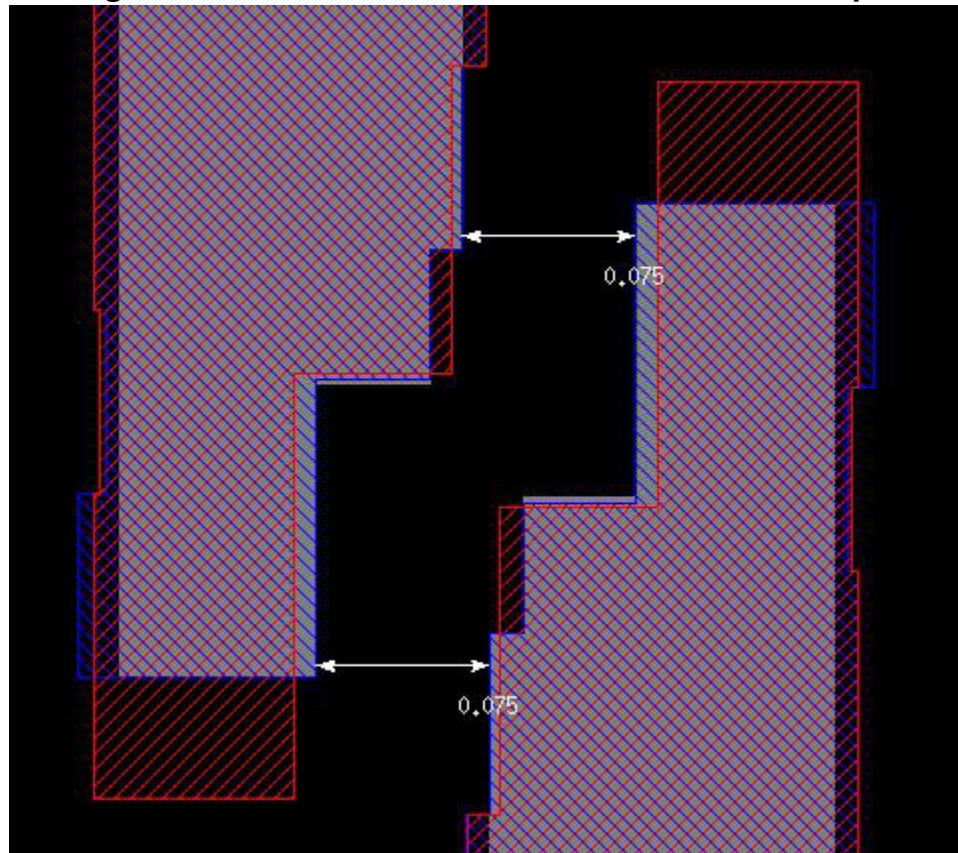
This variable is most useful when upgrading across version 2006.2, when the LITHO_MRC_MODE changed defaults.

Examples

[Figure 6-41](#) shows a case where LITHO_SIDE_SEGMENT_FILTER may be helpful. The initial input had pre-OPC violations at the indicated line-end adjacent edges. When the OPC engine tries moving the line-end fragments out, the side segments are checked for violations. The pre-OPC violation restricts the movement on the line end fragments.

In the figure, the underlying blue shapes represent the default OPC results. The red shapes are the result of setting LITHO_SIDE_SEGMENT_FILTER to 1. The red shapes provide better protection against line-end pullback and yield a better print image.

Figure 6-41. LITHO_SIDE_SEGMENT_FILTER Example



Related Topics

[LITHO_MRC_MODE](#)

LITHO_SIMULATE_CONTEXT

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Controls the context fragment optimization in the hierarchical processing scheme.

Usage

sse LITHO_SIMULATE_CONTEXT {0 | 1}

Arguments

- **0**

A required argument that disables LITHO_SIMULATE_CONTEXT.

- **1**

A required argument that enables LITHO_SIMULATE_CONTEXT. This is the default.

Description

This variable controls the context fragment optimization in the hierarchical processing scheme. Generally, when a fragment has been simulated in a child cell it is never necessary to simulate it again in the parent. Some of these fragments are promoted to the parent to provide context for neighbors and these are affected by this option.

Note

 Do not enable [LITHO_MOVE_CONTEXT](#) and [LITHO_SIMULATE_CONTEXT](#) in the same run. The variables use different algorithms to achieve the same effect and interfere with each other. Generally, [LITHO_MOVE_CONTEXT](#) is faster and [LITHO_SIMULATE_CONTEXT](#) is more precise.

Default Change

In Calibre version 2011.3, the default changed from 0 (not enabled) to 1 (enabled). The default [LITHO_SIMULATE_CONTEXT_DISTANCE](#) also changed from 0 to 0.25.

Linked Defaults

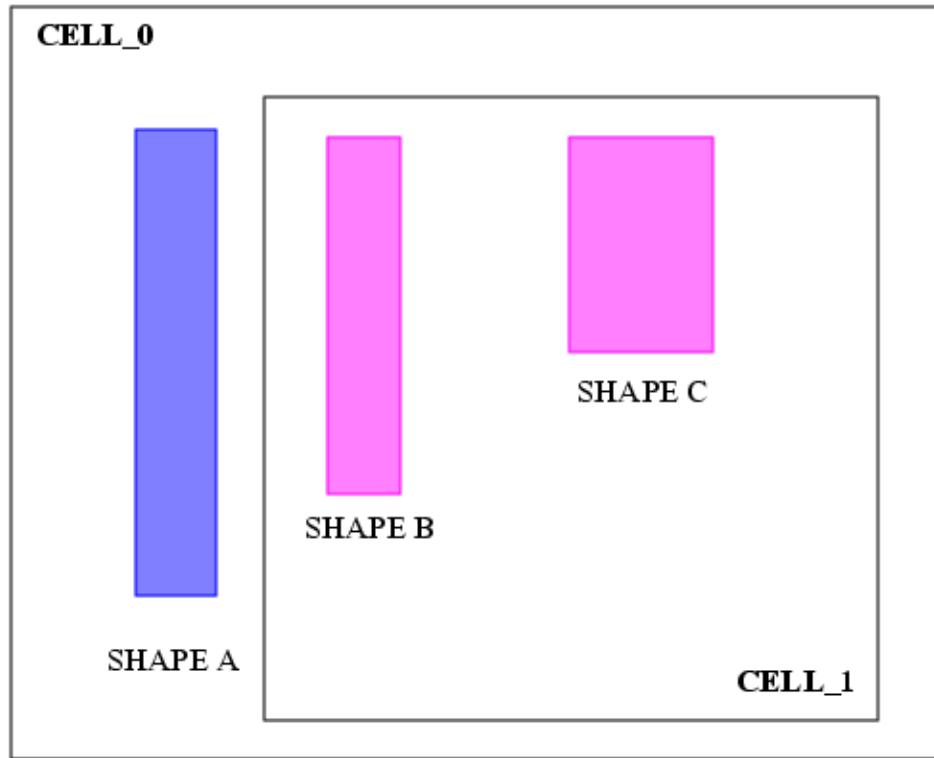
If you explicitly set LITHO_SIMULATE_CONTEXT, even to its default, 1, the default value of LITHO_SIMULATE_CONTEXT_DISTANCE becomes unlimited. It is recommended to also set this variable to avoid longer run times. Normally the default is 0.25 microns.

Examples

In [Figure 6-42](#), shape A is in a parent (Cell_0) and shapes B and C are in a child (Cell_1). Final OPC for shape B cannot be determined in the child because of its proximity to shape A. Shape C does receive OPC corrections.

During the parent cell's optimization, shapes A and B receive their correction. Shape C must also be visible in order for shape B to be corrected so it is promoted to the parent and its fragments are promoted as "context fragments."

Figure 6-42. LITHO_SIMULATE_CONTEXT Example



- **When LITHO_SIMULATE_CONTEXT is set to 0:**

The context fragments do not receive sites. During optical simulations they are biased to their correct final location.

- **When LITHO_SIMULATE_CONTEXT is set to 1:**

Context fragments receive sites and are shifted according to their EPE with each iteration. The movement during each iteration should match the movement that occurred in the child cell.

Impact on Tagging

If a tag script requires knowledge of image parameters on neighboring parameters then this option is important. For example, if a fragment in shape B is treated differently based on an image property of a fragment in shape C (i.e. EPE, slope, Imax, etc.) then it is important for the fragment on shape C to have a site.

Impact on LAPI

When LAPI is used, this option is turned on by default because LAPI code can easily be incompatible with the promoted fragment optimization.

When the context fragment optimization is on, the fragments are immediately shifted to a location which took several iterations to determine. If a tag script does a few iterations and then considers image information, then these context fragments distort the picture by being several iterations ahead of their neighbors.

Impact on MRC

When fragments are moved, the MRC constraints must consider both the temporary location of a context fragment and its final location. If LITHO_SIMULATE_CONTEXT is enabled, Calibre iterates and moves the context fragments even though the final location is known. This helps nearby fragments to converge on an answer in the same way that they would in a flat simulation. This extra computation increases the run time but this can be controlled using LITHO_SIMULATE_CONTEXT_DISTANCE.

Related Topics

[LITHO_CONTEXT_RELAX_MRC](#)

[LITHO_SIMULATE_CONTEXT_DISTANCE](#)

[newTag ... -how MRC](#)

LITHO_SIMULATE_CONTEXT_DISTANCE

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Controls how much context is simulated.

Usage

sse **LITHO_SIMULATE_CONTEXT_DISTANCE** *dist*

Arguments

- *dist*

A required argument that specifies a distance in user units. If the value is less than zero or if it is not specified, all the context fragments are simulated. The default value is [0.25 um](#) unless [LITHO_SIMULATE_CONTEXT](#) is explicitly set. In that case, the default value is unlimited.

Description

Use this variable to reduce the run time penalty of **LITHO_SIMULATE_CONTEXT**.

This variable controls how much context is simulated. Context fragments which are within this distance of a “valid” fragment are simulated and the others are not. A “valid” fragment is one for which the correction is being determined in the current cell.

If the value is small then the context fragments which are close to a valid fragment get re-simulated. Context fragments which are not nearby are visible and do not receive a site, but they are used to calculate correct intensities at the surrounding fragments.

Fragments that are not re-simulated are biased using the correction which was determined in the child cell and they stay at that position throughout the iteration process.

The transcript shows the value used for **LITHO_SIMULATE_CONTEXT_DISTANCE** in the line beginning “//INFO: simulate context distance”.

LITHO_SIMULATE_CONTEXT_SITE_SHIFT

Type: [Litho Variables](#). Used by Calibre OPCpro.

Shifts control sites on long fragments closer to short fragments to increase the context for simulation.

Usage

```
sse LITHO_SIMULATE_CONTEXT_SITE_SHIFT {0 | 1}
```

Arguments

- **0**
A required argument that does not enable site shifting. This is the default behavior.
- **1**
A required argument that enables site shifting.

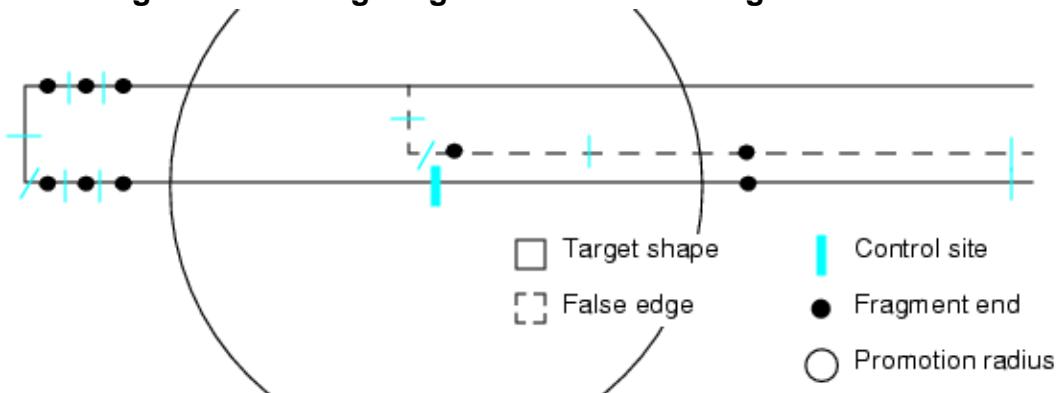
Description

Site shifting can be useful in hierarchical runs when a false edge caused by promotion results in a notch in the final output.

Examples

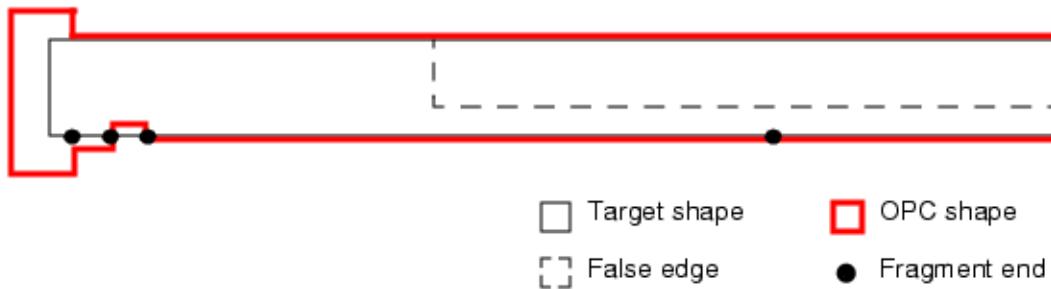
Consider a layout with long wires. Depending on the layout hierarchy and specific settings in the rule file, a fragment within a tile could see the wire as in [Figure 6-43](#). The fragment being simulated is shown with a thicker control site, at the center of the promotion radius. All fragments with control sites within the promotion radius constitute the fragment's context.

Figure 6-43. Long Fragment Near False Edge With Context



In particular, notice that when the long fragment is simulated none of the line end fragments are within its context. Due to the tile boundary, the true opposite edge of the polygon is shielded by a false edge. Similarly, when the line end fragments are simulated, the long fragment is seen in its final position, but is not moved because the control site is outside the context distance. The final OPC results include a notch near the line end. They are shown in [Figure 6-44](#).

Figure 6-44. Bad OPC Results Near Long Fragment



The reason for the notch is that the short fragments near the line end and the long fragment are adjusted in different contexts. The long fragment's control site is too far from the short fragments.

When the litho setup file sets LITHO_SIMULATE_CONTEXT_SITE_SHIFT to 1, the control site is moved closer to the short fragment by half the promotion radius as shown in [Figure 6-45](#). Notice that the two line-end adjacent fragments and the long fragment now share a context. This results in a smoother shape, as shown in [Figure 6-46](#).

Figure 6-45. Long Fragment Site and Context Shifted

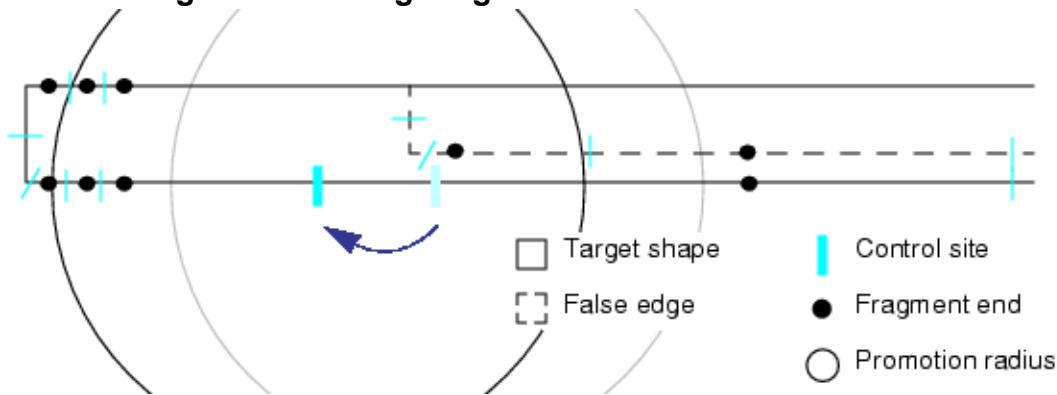
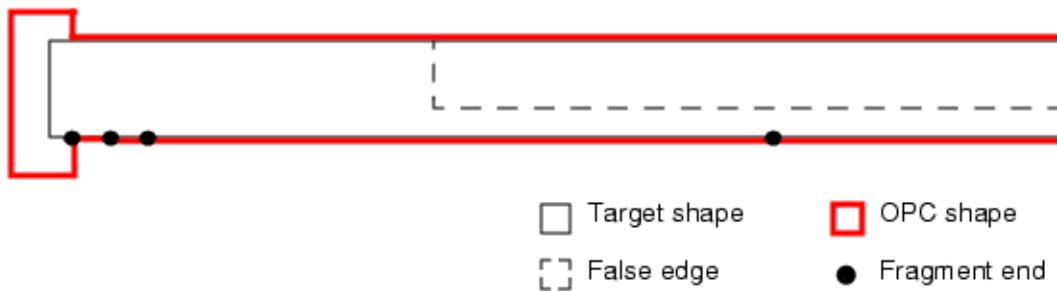


Figure 6-46. Better OPC Results With Site Shifting



Related Topics

[LITHO_SIMULATE_CONTEXT_DISTANCE](#)

LITHO_SUPPRESS_INTERACTION_ERROR

Type: [Litho Variables](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.
Suppresses interaction distance errors.

Usage

sse LITHO_SUPPRESS_INTERACTION_ERROR {0 | 1}

Arguments

- **0**
A required argument that disables error suppression. This is the default.
- **1**
A required argument that suppresses the errors produced when tagging operations exceed an interaction distance greater than or equal to what is specified by the program automatically.

Description

Suppresses interaction distance errors generated during setup file compilation stage.

Fragmentation and tagging operations such as newTag -how EDGE, SEQUENCE, FRAGMENT, EXT, and INT tagging check whether the operations require an interaction distance greater than or equal to what is set by the program automatically, based on the size of the optical radius, which is calculated as:

$0.5 * \text{hoodpix} + (\text{numsites} - \text{sitecenter} - 1) * \text{sampleSpacing}$

If the operation requires a larger interaction range, then the tools generate an error during the setup file compilation stage.

Note

 Suppressing interaction errors is *not recommended*.

LITHO_SUPPRESS_MAPNUMBER_ERROR

Type: [Litho Variables](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Suppresses errors produced when MAPNUMBER is not present and there are [tags2boxes](#) in the setup file.

Usage

sse LITHO_SUPPRESS_MAPNUMBER_ERROR {0 | 1}

Arguments

- **0**
A required argument that disables error suppression. This is the default.
- **1**
A required argument that suppresses the errors produced when MAPNUMBER is not present and there are [tags2boxes](#) in the setup file.

Description

Suppresses errors produced when MAPNUMBER is not present and there are [tags2boxes](#) in the setup file. Normally, the litho tools disallow placing [tags2boxes](#) outputs on the same output layer as the OPC geometry in litho.

Note

 Suppressing MAPNUMBER errors is *not recommended*.

Related Topics

[tags2boxes](#)

[LITHO ORC](#)

LITHO_TAG_TIMER

Type: [Litho Variables](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.
Prints information in the transcript about time spent in each tagging command for single CPU runs.

Usage

`sse LITHO_TAG_TIMER {0 | 1}`

Arguments

- **0**
A required argument that suppresses writing the information to the transcript. This is the default.
- **1**
A required argument that writes the cumulative time for each tagging operation into the transcript.

Description

This variable instructs the applications to write the cumulative REAL time for each tagging operation for single CPU runs.

Note

 This information is valid when running with a single CPU. If you use this option with multi-threaded or MTflex runs, then the transcribed times can be unreliable.

Related Topics

[LITHO_PRINT_LEVEL](#)

[Tagging Commands](#)

LITHO_TILE_SLIVER_SIZE

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Defines a context region near tile boundaries in which to check for non-printing slivers.

Usage

sse **LITHO_TILE_SLIVER_SIZE** *dist*

Arguments

- *dist*

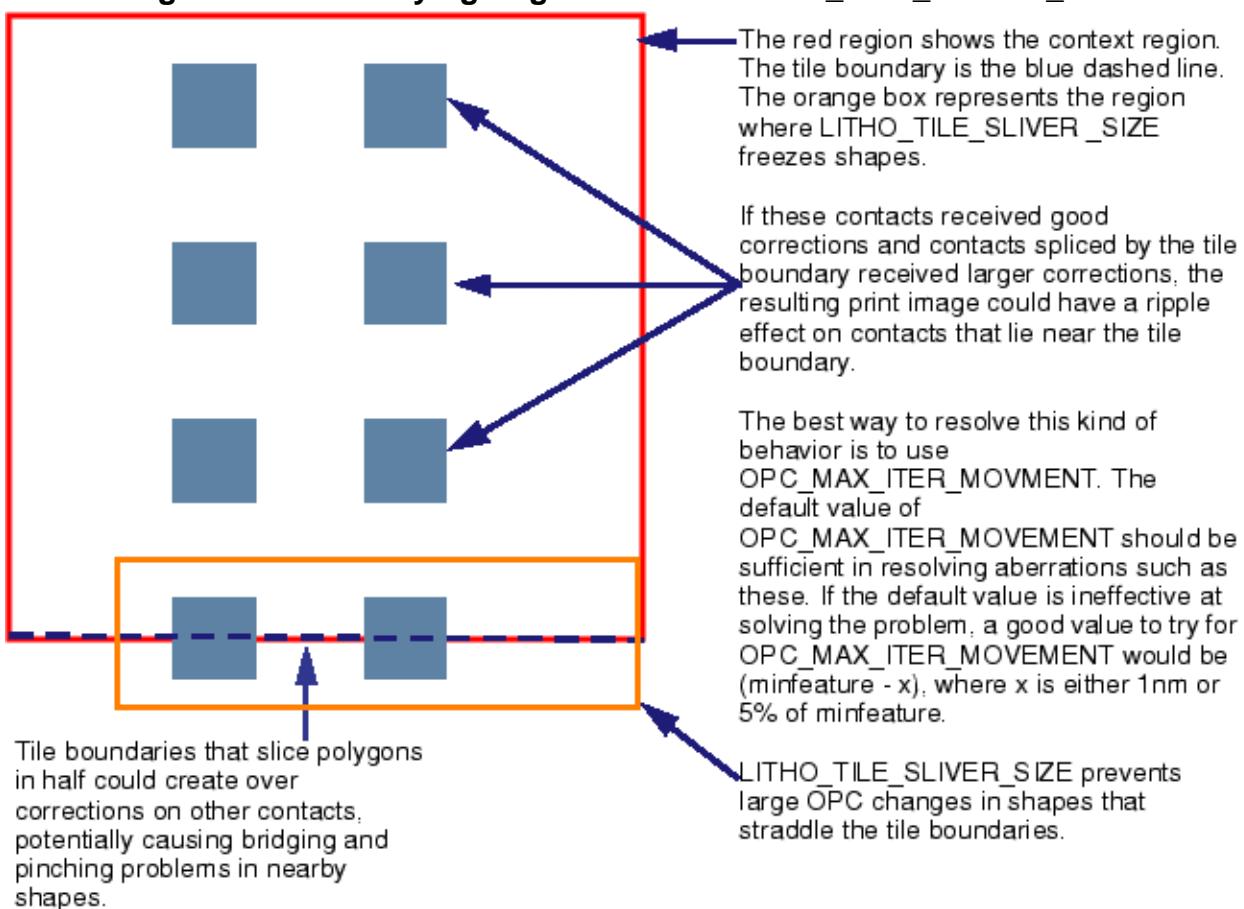
A required argument that specifies a buffer distance in microns around the context region to leave uncorrected in the layout. The default value is 0 microns.

Description

In some recipes, when a tile boundary clips a contact into two pieces, the non-printing sliver that results causes oscillations and can produce unstable OPC results. This problem can be greatly reduced or eliminated by using [OPC_MAX_ITER_MOVEMENT](#). This is mitigated with [OPC_MAX_ITER_MOVEMENT](#), but sometimes it isn't desirable to use this setting to control the problem. Another approach is to identify the fragments in these slivers and not to correct them.

The LITHO_TILE_SLIVER_SIZE setting defines a distance for doing this. A ring is created around the edge of the context region and anything within this ring is left uncorrected as shown in [Figure 6-47](#). These uncorrected fragments are only temporary fragments that are not part of the final results.

Figure 6-47. Identifying Regions to Use LITHO_TILE_SLIVER_SIZE



LITHO_TOLERATE_MINOR_SKEW

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Used in conjunction with DONT_MOVE_ALL_ANGLE_EDGES to set a tolerance for angle skews.

Usage

sse LITHO_TOLERATE_MINOR_SKEW {0 | 1}

Arguments

- **0**
A required argument that applies strict enforcement of 0-45-90 degree edges.
- **1**
A required argument that allows for 0-45-90 degree angles with ± 1 dbu skews to be processed by OPC. This is the default.

Description

This setup variable applies only to the Calibre products that move edges to correct for Optical and Process errors. It is ignored if [DONT_MOVE_ALL_ANGLE_EDGES](#) is not set, or set to 0.

When DONT_MOVE_ALL_ANGLE_EDGES is 1, LITHO_TOLERATE_MINOR_SKEW controls how much skew is tolerated.

When LITHO_TOLERATE_MINOR_SKEW is 1, edges that are 1 dbu off from a multiple of 45 degrees are processed by OPC.

When LITHO_TOLERATE_MINOR_SKEW is 0, only edges that are strictly 45 degree multiples are processed by OPC.

When edges are more skew than tolerated, Calibre outputs an error message and the edge is not processed.

Related Topics

[DONT_MOVE_ALL_ANGLE_EDGES](#)

[DISALLOW_SHALLOW_ANGLE](#)

LITHO_WRITE_DEBUG_SVRF (shell variable only)

Type: [Litho Variables](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Generates additional information useful for creating test cases or debugging. *This is not a setup file variable and must be set in a command shell.*

Usage

```
setenv LITHO_WRITE_DEBUG_SVRF id | name
```

Arguments

- ***id***

A required argument that specifies the LITHO statement to which the variable applies. An ***id*** of “1” references the first LITHO statement executed, “2” would be the second, and so on. Set ***id*** to -2 to output files for all relevant operations.

Either ***id*** or ***name*** is required, but they cannot be specified together.

- ***name***

A required argument that specifies the ***name*** of a LITHO FILE statement. When this form is used, the operations that use the specified litho setup file are written out.

Either ***id*** or ***name*** is required, but they cannot be specified together.

Description

Use this variable to create simpler test cases for specific LITHO operations.

The LITHO_WRITE_DEBUG_SVRF environment variable causes the Calibre run to echo the input layers of the referenced LITHO operation to an OASIS file named *litho_opc_debug.oas*. It also creates an SVRF file named *litho_opc_debug.svrf* that when executed performs the referenced LITHO operation upon *litho_opc_debug.oas*.

Often several LITHO operations run concurrently. Concurrent operations count as one invocation, as shown in “[Example 2 - Counting in Transcript](#)” on page 324. To determine the value to use for ***id***:

1. Save the transcript from running the real Calibre job. The job does not need to complete but does need to start the operation(s) intended for separation.
2. In the transcript, move past the rule file compilation stage. One way to do this is to search for the string

COMPILATION MODULE COMPLETED

3. Search for the string LITHO occurring during execution, and count the number of jobs being started. This gives the ***id***.

- A job consists of any cluster of “*layer = LITHO tool*” lines appearing together. The clustering indicates they are being processed concurrently.
- Each LITHO command in the SVRF file appears at least twice in the operation stage of the transcript: once when it starts being processed, and again when it ends. Only count the ones starting the job.
- Do not count LITHO FILE statements, environment variables, or other LITHO occurrences.

Note

 You can obtain complete test cases by setting *id* to -2. If test case size is not a concern, this is the most reliable method for creating a complete test case.

Examples

Example 1 - Counting in SVRF File

Assume you have an SVRF file containing the following statements:

```
LAYER metal1 11
LAYER metal3 15

opc_m1 = LITHO DENSEOPC FILE m1 metal1 MAP metal1_opc
opc_m1 {COPY opc_m1} DRC CHECK MAP opc_m1 oasis 111 m1_out.oas
opc_m3 {LITHO OPC metal3 FILE m3} DRC CHECK MAP opc_m3 m3_opc.gds

LITHO FILE m1 [...]
    layer metal1 hidden atten 0.06
    denseopc_options opts {...}
        layer metal1 opc atten 0.06
    }
    setlayer metal1_opc = denseopc metal1 MAP metal1 OPTIONS opts
]

LITHO FILE m3 [...]
    layer 15 metal3 0 0 opc dark
]
```

If you do not set LITHO_WRITE_DEBUG_SVRF, your output is the files *m1_out.oas* and *m3_opc.gds*, and the usual result databases and reports.

To produce files for a Calibre OPCpro test case, set LITHO_WRITE_DEBUG_SVRF to 2 in the shell before invoking Calibre. (This points to the second LITHO command, on the line beginning “opc_m3”.) This generates the additional litho_opc_debug files in addition to the usual output.

Example 2 - Counting in Transcript

In the following transcript excerpt, setting LITHO_WRITE_DEBUG_SVRF to 2 would return a group of 3 concurrent operations.

LITHO operation invocations are in bold. Duplicate invocations are in green.

```
Running Calibre: $CALIBRE_HOME/calibre -remotefile gridflex.cfg -64 -hyper
-turbo -turbo_litho -drc -hier rules

Environment variable LITHO_WRITE_DEBUG_SVRF ('=2') is being used
// Calibre v2021.1_33.19   Mon Mar 1 16:54:23 PST 2021
// Calibre Utility Library  v0-10_13-2017-1   Wed Jun 3 07:42:12 PDT 2020
// Litho Libraries v2021.1_33.19  Mon Mar 1 16:10:01 PST 2021
//
//           Copyright Siemens 1996-2021
//           All Rights Reserved.
// THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
// WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION
// OR ITS LICENSORS AND IS SUBJECT TO LICENSE TERMS.
//
// 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.
//
// Mentor Graphics software executing under x86-64 Linux

...
Layer FOM::<1> DELETED -- LVHEAP = 349/415/415  SHARED = 121/192
WARNING: 20 polygon result(s) segmented in DRC RuleCheck M1.

M2_fill = LITHO OPC m2_opc diff FILE ./litho/m2_opc.in
-----
Operation EXECUTING on HDB 0 (T506 S15 E0:383 H0:195 A16:644 L0:186 C5)

...
TIME FOR LITHO CORMERGE:          CPU TIME = 238 + 1687  REAL TIME = 158
TIMESTAMP 2276
<MON:E 2275 O 69>

M2_fill = LITHO OPC m2_opc diff FILE ./litho/m2_opc.in
-----
M2_fill (HIER-PMF TYP=1 CFG=0 HGC=3259010...

...
Layer xG_67m.2h::<1> DELETED -- LVHEAP = 1284/1332/1332  SHARED = 970/992

m2_90::<1> = LITHO ORC m2_orc m2_targ MAPNUMBER 300 FILE ./litho/m2_orc.in
m2_85::<1> = LITHO ORC m2_orc m2_targ MAPNUMBER 301 FILE ./litho/m2_orc.in
m2_80::<1> = LITHO ORC m2_orc m2_targ MAPNUMBER 302 FILE ./litho/m2_orc.in
-----
Operation EXECUTING on HDB 0 (T2441 S15 E0:472 H4:287 A0:947 L0:229 C5)

...
Layer v12 DELETED -- LVHEAP = 1750/3390/3393  SHARED = 1371/1696
Layer fin DELETED -- LVHEAP = 1750/3390/3393  SHARED = 1371/1696

m2_90::<1> = LITHO ORC m2_orc m2_targ MAPNUMBER 300 FILE ./litho/m2_orc.in
```

```
m2_85::<1> = LITHO ORC m2_orc m2_targ MAPNUMBER 301 FILE ./litho/m2_orc.in
m2_80::<1> = LITHO ORC m2_orc m2_targ MAPNUMBER 302 FILE ./litho/m2_orc.in
-----
m2_90::<1> (HIER TYP=1 CFG=1 HGC=0 FGC=0 HEC=0 FEC=0 VHC=F VPC=F)
...
Layer cp1m.orc.2: p1m.pinch ::<1> DELETED -- LVHEAP = 3026/3871/3872 SHARED =
2809/3584

via_Mopc = LITHO OPC via_opcIn FILE ./litho/vim/setup/via_opc.in
-----
Operation EXECUTING on HDB 0 (T3371 S15 E0:476 H2:450 A0:970 L0:229 C3)
...
via_Mopc = LITHO OPC via_opcIn FILE ./litho/vim/setup/via_opc.in
...
...
```

MASK_CHIP_CORNER

Type: [Litho Variables](#). Used by Calibre OPCpro and Calibre PRINTimage.

Emulates the behavior of mask corner rounding when performing optics calculations.

Usage

```
sse MASK_CHIP_CORNER d1 [d2]
```

Arguments

- *d1*

A required argument that specifies the amount of corner to chip for convex corners. The default is 0. *d1* is expressed in microns.

- *d2*

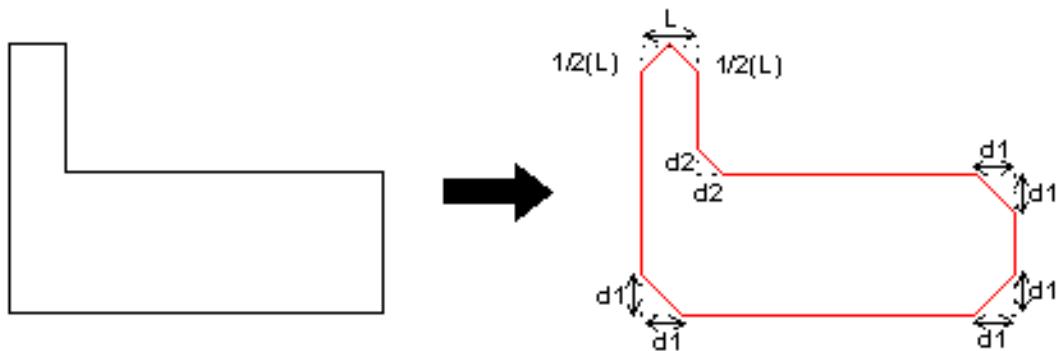
An optional argument that specifies the amount of corner to chip for concave corners. The default is 0. *d2* is expressed in microns.

Description

The MASK_CHIP_CORNER setup file variable controls whether or not the simulation tools chip the corners of input data prior to performing optics calculations. Chipping data corners emulates the behavior of mask corner rounding. Chipping corners does not impact the data on the input layers. MASK_CHIP_CORNER allows the user to add a constant corner rounding amount to account for known levels of rounding in a mask. Additional edges created by this command introduce more sites into your designs and increases simulation time.

The *d1* and *d2* arguments provide separate controls for convex corners and concave corners. When an edge is too small to be chipped the full distance, Calibre OPCpro chips the corner a distance of one half the length of the limiting edge.

Figure 6-48. Effects of MASK_CHIP_CORNER

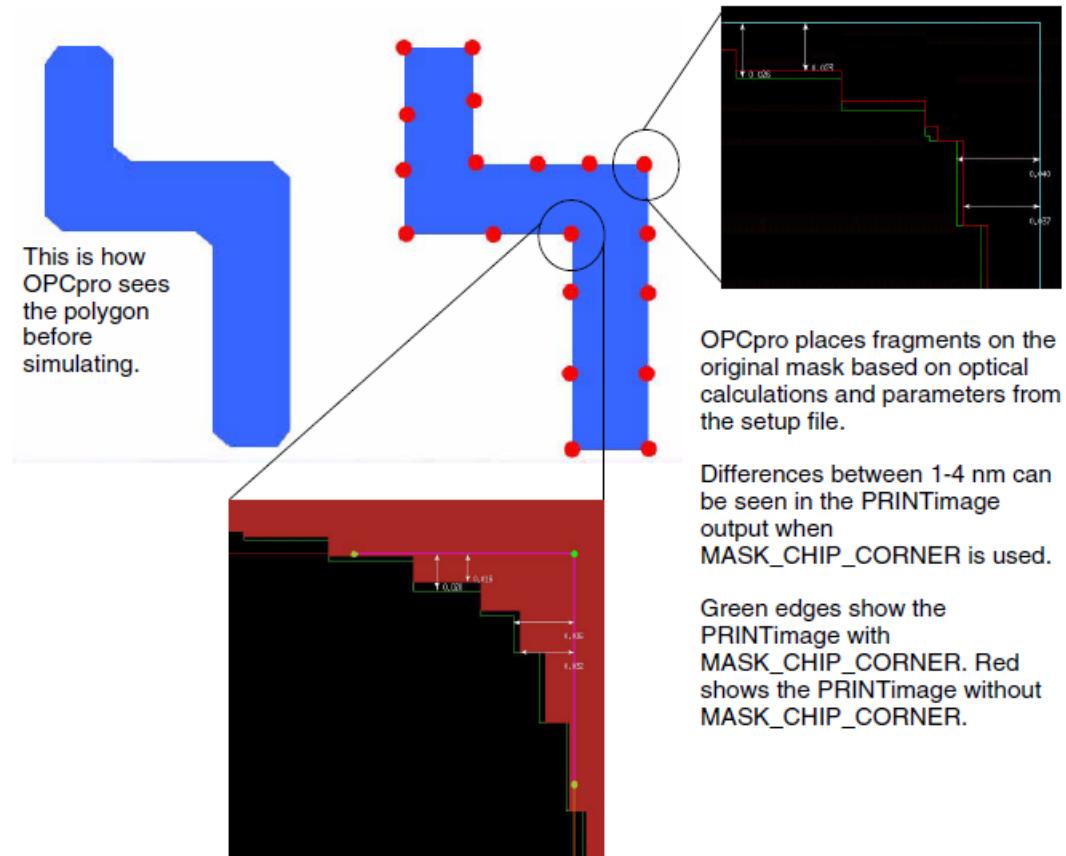


The following example shows how MASK_CHIP_CORNER can be used in a test flow.

1. Take the modeling test pattern and use SVRF to add corner rounding to the test pattern.
2. Generate your optical and resist models with the new test pattern.

3. Add “sse MASK_CHIP_CORNER *d1*” to your setup file to use in PRINTimage simulations to gauge how it affects your results. Replace *d1* with your own experimental value.
4. Run a Calibre OPCpro simulation followed by a Calibre PRINTimage simulation to assess the results.

Figure 6-49. Effects of Using MASK_CHIP_CORNER



Related Topics

[LITHO OPC](#)

OPC_AUTO_STEPBY_THRESHOLD

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Speeds up simulation time by prevents short jogs from being created, which causes some decrease in precision.

Usage

sse OPC_AUTO_STEPBY_THRESHOLD *upper_limit*

Arguments

- *upper_limit*

A required argument that sets an upper limit in microns on when to use the auto stepby algorithm. The default value is 0.0004 microns.

Description

This variable sets a threshold for a performance-enhancing algorithm that can affect OPC quality. The algorithm improves performance by preventing short jogs from being created, which results in polygons with fewer vertices. This shortens simulation time. However, for larger stepsize values, patterns may lose small details.

Note

 The *upper_limit* should be tested thoroughly on your own data to find a balance between performance and data quality.

Auto Stepby Algorithm

The auto stepby algorithm is triggered when the [stepsize](#) setting is less than the value set on this variable, or 0.0004 microns if the variable is unset. The algorithm moves fragments by decreasing multiples of stepsize over the iterations. For a setting of [iterations](#) *i*, the values are as follows:

Table 6-14. Auto Stepby Movements

| Iteration | Movement | Equivalent fastIter Call |
|-------------------------------|---------------|------------------------------------|
| <i>i</i> - 4 and earlier | 10 * stepsize | fastIter all <i>i</i> -4-stepby 10 |
| <i>i</i> - 3 and <i>i</i> - 2 | 5 * stepsize | fastIter all 2-stepby 5 |
| <i>i</i> - 1 and <i>i</i> | stepsize | fastIter all 2 |

Exceptions

When you use the [fastIter](#) command, the auto stepby algorithm is not used. Each iteration moves edges by the same number of steps.

When [stepsize](#) is greater than OPC_AUTO_STEPBY_THRESHOLD's *upper_limit*, the algorithm is not used. Each iteration applies a movement of stepsize.

OPC_CORR_SEARCH_DISTANCE

Type: [Litho Variables](#). Used by Calibre OPCpro.

Controls the search range for mapping correction layer edge fragments to OPC layer edge fragments.

Usage

sse OPC_CORR_SEARCH_DISTANCE *microns*

Arguments

- ***microns***

A required argument that sets the search range in microns for mapping correction layer fragments to opc layer fragments. The fragments are mapped only if the midpoints of the fragments are within ***microns***. The default value is **0.05** um.

Description

This variable is primarily used when performing OPC on phase shift masks (PSM) without [MATRIX_OP](#)C.

Related Topics

[layer](#)

[Layer Types](#)

OPC_CROSS_MEEF_DEADBAND

Type: [Litho Variables](#). Used by Matrix OPC.

Creates a “dead band,” a range of MEEF values to be treated as if they were 0 MEEF.

Usage

sse OPC_CROSS_MEEF_DEADBAND *value*

Arguments

- *value*

A required argument that specifies a floating-point number range that is to be treated as 0 for MEEF calculations. The default value is 0.00001.

Description

When MEEF is calculated, fragments with MEEF values close to 0 can cause slight movements in an unpredictable fashion. To prevent this, use OPC_CROSS_MEEF_DEADBAND to create a “dead band,” a value range around 0 that you can treat as though the MEEF were 0. The *value* specified gives the range to treat as MEEF 0. The range is from negative *value* to positive *value*.

OPC_DO_SKew_CHECK

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Checks OPC input for skewed edges.

Usage

sse OPC_DO_SKew_CHECK {0 | 1}

Arguments

- **0**
A required argument that does not check input for skew edges. This is the default setting.
- **1**
A required argument that instructs Calibre OPCpro to check input for skew edges.

Description

The OPC_DO_SKew_CHECK environment variable enables checking of skewed edges in OPC input. If no skew edges are found, the output is guaranteed to also not have skew edges.

OPC_ENCLOSURE_MIN_LENGTH

Type: [Litho Variables](#). Used by Matrix OPC.

Defines a minimum fragment length for the enclosure check to be enforced.

Usage

sse OPC_ENCLOSURE_MIN_LENGTH *length*

Arguments

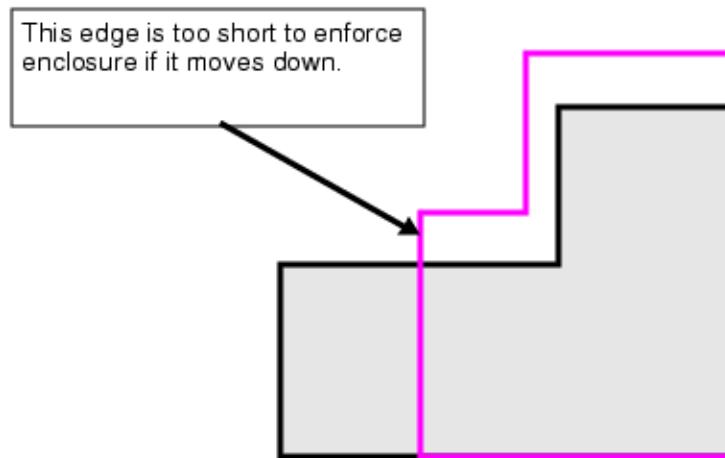
- *length*

A required argument that defines minimum fragment length to enforce enclosure rule check.

Description

Defines a minimum fragment length for the enclosure check to be enforced. If either fragment is shorter than this length, the enclosure rule does not apply. This variable allows you to turn off enclosure checking for pairs of edges in which one edge is a user-created jog (it does not matter which mask they are on).

Figure 6-50. Minimum Fragment Length for Enclosure Check



Related Topics

[OPC_IGNORE_ENCLOSURE_AT_CORNER](#)

OPC_ENFORCE_FRAG_LAYER_ORDER

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Controls the user-defined fragmentation order.

Usage

```
sse OPC_ENFORCE_FRAG_LAYER_ORDER {standard | partial | full}
```

Arguments

- **standard**

A required argument that specifies that [fragmentLayer](#) fragmentation is carried out in its default order (see [Table 6-15](#)). This is the default value.

- **partial**

A required argument that specifies that fragmentation within a fragmentLayer block is applied in the order listed in the file. If only one or two fragmentation schemes are specified in the block, the default order of execution is carried out afterwards.

- **full**

A required argument that specifies a full custom-ordered fragmentation scheme within the fragmentLayer block. If only one or two fragmentation schemes are specified in the block, it leaves the others unprocessed. In order to have it process the remaining fragmentation schemes, they must be explicitly defined in the fragmentLayer block.

Description

This variable changes the default processing order of the fragmentation schemes (intrafeature, interfeature, and so on) to instead use the order as specified in the fragmentLayer block. The default value for this variable is **standard**.

For example, if intrafeature fragmentation is specified before island fragmentation, then intrafeature fragmentation is performed first, followed by island fragmentation. Any form of fragmentation not specified in the fragmentLayer command is then handled using default values after all of the forms of fragmentation listed in the fragmentLayer command have been performed. The order of the unspecified methods of fragmentation is the same as the default order, but after the explicitly specified forms of fragmentation.

[Table 6-15](#) shows the forms of fragmentation allowed within a fragmentLayer command block along with the keywords that control them.

Table 6-15. Allowed Fragmentation in a fragmentLayer Command

| Default Order | Method | Keyword(s) |
|----------------------|--------------------------|---------------------------------|
| 1 | coincident | coincidentLayer |
| 2 | visible and asraf layers | visibleLayers |
| 3 | island layer | islandFragment |

Table 6-15. Allowed Fragmentation in a fragmentLayer Command (cont.)

| Default Order | Method | Keyword(s) |
|----------------------|---------------|------------------------------------|
| 4 | intrafeature | concavcorn, cornedge, intrafeature |
| 5 | interfeature | interfeature, interfeatLayers |
| 6 | maxedgelength | maxedgelength |

The following fragmentLayer keywords have no effect on the order of fragmentation if OPC_ENFORCE_FRAG_LAYER_ORDER is set to **partial** and they may be specified at any point in the fragmentLayer block.

- [minedgelength](#)
- [minjog](#)
- [seriftype](#)

Examples

Example 1

```
sse OPC_ENFORCE_FRAG_LAYER_ORDER partial

fragmentLayer OPCTARGET {
    intrafeature convex 0.065 0.065
    intrafeature concave 0.08
    interfeature -interdistance 0.600000 -ripplelen 0.110000 -num 0
    islandFragment RX -enforce 1
}
```

Fragmentation on the “OPCTARGET” layer occurs in the following order:

1. Intrafeature fragmentation
2. Interfeature fragmentation
3. Island fragmentation

After all of the explicitly defined forms of fragmentation have been performed, the remaining forms of fragmentation are carried out with their default parameter values, as follows:

4. Coincident fragmentation
5. Visible layer fragmentation
6. Maxedgelength fragmentation

The order of fragmentation is determined when a keyword for a type of fragmentation is first encountered. If a keyword for a particular type of fragmentation is encountered more than once, the later occurrences have no effect on the order of fragmentation. The initial occurrence determines the order of fragmentation.

Example 2

This example shows how the first-encountered fragmentation works.

```
sse OPC_ENFORCE_FRAG_LAYER_ORDER partial

fragmentLayer OPCTARGET {
    intrafeature convex 0.065 0.065
    interfeature -interdistance 0.600000 -ripplelen 0.110000 -num 0
    intrafeature concave 0.08
    islandFragment RX -enforce 1
}
```

Example 2 is identical to Example 1, with the exception that the interfeature command has been swapped with the last intrafeature command. While the order in the example is now intrafeature, interfeature followed by intrafeature, the actual order of execution is still the same as before (intrafeature followed by interfeature and island fragmentation). The location of the second intrafeature command is irrelevant.

Example 3

```
sse OPC_ENFORCE_FRAG_LAYER_ORDER partial

fragmentLayer OPCTARGET {
    intrafeature convex 0.065 0.065
    intrafeature concave 0.08
    interfeature -interdistance 0.600000 -ripplelen 0.110000 -num 0
    islandFragment RX -enforce 1
    minedgelength 0.065
}
```

Example 3 is identical to Example 1, with the addition of the **minedgelength** keyword as the last statement within the fragmentLayer block. The order of fragmentation here remains unchanged. The **minedgelength** specification applies to all operations inside the fragmentLayer block, despite the fact that it is specified last in the block.

Note

 Once the **OPC_ENFORCE_FRAG_LAYER_ORDER** flag is set to **full**, all fragmentLayer blocks are processed in the order described in the file. It is not possible to specify that one fragmentLayer block be processed in user-defined order while another is processed in default order. If you need to have a block processed in default order while the **OPC_ENFORCE_FRAG_LAYER_ORDER** flag is set to **full**, you must manually specify the operations in the same order as the default.

Related Topics

[fragmentLayer](#)

OPC_EPE_TOLERANCE_FRAC

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Defines a tolerance factor used when calculating whether or not an edge is within tolerance.

Usage

sse OPC_EPE_TOLERANCE_FRAC *value*

Arguments

- *value*

A required argument that defines a tolerance factor. Legal values are positive real numbers less than or equal to 100. The default value is 0.6.

Description

This variable defines a tolerance factor used when calculating whether or not an edge is within tolerance. Calibre moves only those edges having EPEs such that

$$(|EPE| \geq |OPC_EPE_TOLERANCE_FRAC * stepsize|)$$

The default value of this variable is 0.6. When multiplied by the default stepsize, which is 0.01 um, you can calculate that by default, the smallest EPE corrected is 0.006 um, or 6 nm. Using the default settings, those edges with EPEs smaller than 0.006 um receive no correction.

OPC_FAST_MOVE

Type: [Litho Variables](#). Used by Calibre OPCpro and Calibre ORC.

Attempts to determine if a fragment will encounter an MRC restriction prior to movement.

Usage

sse OPC_FAST_MOVE {0 | 1}

Arguments

- **0**

A required argument that moves fragments incrementally, checking at each step. This is the default.

- **1**

A required argument that moves fragments to their final positions if there are no MRC restrictions.

Description

This variable attempts to determine if a fragment will encounter an MRC restriction prior to moving it. The traditional method for applying OPC corrections is “step and check”. A fragment is moved one step toward its desired displacement and then a check is performed for MRC violations. If no violation is found, the movement is accepted and the next fragment is done.

This can be a slow process when the movements consist of tens or hundreds of steps. When OPC_FAST_MOVE is enabled (set to 1), an attempt is made to determine whether a fragment will encounter any MRC restrictions or not. If there is no violation, then the fragment movement is done in one step.

OPC_FEEDBACK

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Defines a fixed OPC factor by which EPEs are multiplied to calculate the edge movement during each iteration.

Usage

```
sse OPC_FEEDBACK factor [factor ...]
```

Arguments

- *factor*

A required argument that sets a fixed OPC factor to be used. This value should be a floating point number. The default is -0.4.

You may specify up to 15 additional factor arguments (for a total of 16 factors) to provide additional fixed OPC factors. Use these additional factor values when you want to calculate movement differently for different iterations. If you supply fewer factor values than iterations, the last value in the list is used for any additional iterations.

Description

This variable defines a fixed OPC factor by which EPEs are multiplied to calculate the edge movement for each edge during each iteration.

Advanced users only can supply a list of feedback values to define different feedback factors for the various iterations. This can improve the stability of the OPC and reduce the total number of iterations. Typically, the first iterations require a smaller feedback factor than subsequent iterations because the initial EPE values can be very large.

Examples

```
sse OPC_FEEDBACK -0.1 -0.2 -0.4
```

The preceding setting results in the following values used for each iteration:

- Iteration 1 uses -0.1 feedback.
- Iteration 2 uses -0.2 feedback.
- Iteration 3 uses -0.4 feedback.
- Iteration 4 and on use -0.4 feedback.

OPC_IGNORE_ENCLOSURE_AT_CORNER

Type: [Litho Variables](#). Used by Matrix OPC.

Turns off enclosure checking in corner fragments in situations where a corner on the block mask overlaps a corner on the phase-shifted mask.

Usage

`sse OPC_IGNORE_ENCLOSURE_AT_CORNER`

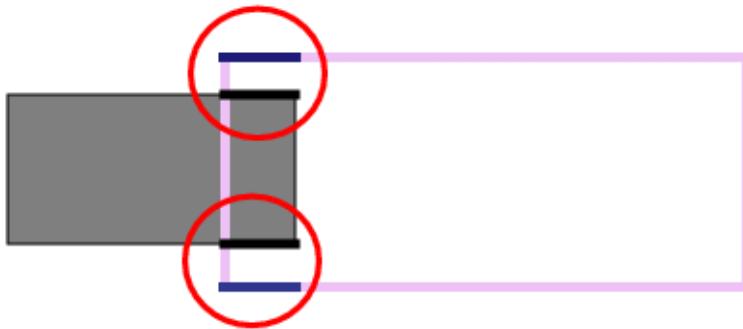
Arguments

None.

Description

Setting this variable turns off enclosure checking in corner fragments in situations where a corner on the block mask overlaps a corner on the phase-shifted mask.

Figure 6-51. Turning Off Enclosure Checking for Corner Fragments



Related Topics

[OPC_ENCLOSURE_MIN_LENGTH](#)

[MATRIX_OP](#)

OPC_LOW_MEEF_DEFAULT

Type: [Litho Variables](#). Used by Matrix OPC.

Specifies the default movement to apply to a low-MEEF fragment.

Usage

sse OPC_LOW_MEEF_DEFAULT *value*

Arguments

- ***value***

A required argument that specifies how far to move a low-MEEF fragment by default.

Description

The default amount of movement to apply to a low-MEEF fragment is specified by this value. It controls handling of all fragments that haven't been tagged by [OPC_LOW_MEEF_POS_TAG](#), [OPC_LOW_MEEF_NEG_TAG](#), or [OPC_LOW_MEEF_FREEZE_TAG](#). When this variable is not set, the default is 1.0, which indicates one full step in the positive direction.

Examples

```
sse OPC_LOW_MEEF_DEFAULT 0.5
```

This example changes the default step size to one half step in the positive direction.

OPC_LOW_MEEF_FREEZE_TAG

Type: [Litho Variables](#). Used by Matrix OPC.

Tags low-MEEF fragments that should not be moved.

Usage

sse **OPC_LOW_MEEF_FREEZE_TAG *tagname***

Arguments

- ***tagname***

A required argument specifying a tag to assign to low-MEEF fragments which should be frozen in place.

Description

The default movement of a low-MEEF fragment is in the positive direction. If you are having difficulty getting the system to converge to a solution with low-MEEF fragments, the positive direction may not be correct. These fragments can be marked with a tag to indicate that they should be handled differently.

When a fragment receives ***tagname***, it triggers special handling. If that fragment has a low-MEEF value, the movement value is set to zero. If the MEEF does not meet the minimum requirement, ***tagname*** fragments move as usual.

Low-MEEF Tagging Precedence

In the event that multiple low-MEEF tags are assigned to the same fragment, the movement value for a low-MEEF fragment is chosen in this order of precedence:

1. positive tag
2. freeze tag
3. negative tag
4. default (no tag at all)

For example, if a low-MEEF fragment were to have all three tags present, the positive movement value would be used. If only the freeze tag and negative tag were present, freeze would take precedence. If no low-MEEF handling tag were found at all, the default value would be used.

Related Topics

[OPC_LOW_MEEF_LIMIT](#)

OPC_LOW_MEEF_LIMIT

Type: [Litho Variables](#). Used by Matrix OPC.

Sets a limit for low-MEEF handling to occur.

Usage

sse OPC_LOW_MEEF_LIMIT *value*

Arguments

- ***value***

A required argument that specifies the value below which a fragment is considered to have low MEEF. The default value is 1.0e-6.

Description

The absolute value of a fragment's MEEF must be less than or equal to this value for low-MEEF handling to occur. Default is 1.0e-6, which indicates a range of -1.0*10-6 to 1.0*10^-6.

Examples

sse OPC_LOW_MEEF_LIMIT 1.0e-5

This sets the new low-MEEF range to -1.0*10-5 <= MEEF value <= 1.0*10-5.

Related Topics

[OPC_LOW_MEEF_DEFAULT](#)

[OPC_LOW_MEEF_FREEZE_TAG](#)

[OPC_LOW_MEEF_NEG_SCALE](#)

[OPC_LOW_MEEF_POS_SCALE](#)

OPC_LOW_MEEF_NEG_SCALE

Type: [Litho Variables](#). Used by Matrix OPC.

Scales the movement to apply to a low-MEEF fragment marked with the tag **OPC_LOW_MEEF_NEG_TAG**.

Usage

sse OPC_LOW_MEEF_NEG_SCALE *value*

Arguments

- ***value***

A required argument that specifies the amount of movement to apply to a low-MEEF fragment marked with the [OPC_LOW_MEEF_NEG_TAG](#) tag. The default is -1.0, which indicates one full step in the negative direction.

Examples

sse OPC_LOW_MEEF_NEG_SCALE -0.5

This example causes all fragments tagged with the negative low-MEEF tag name to move by only one half their normal value in the negative direction.

Related Topics

[OPC_LOW_MEEF_DEFAULT](#)

[OPC_LOW_MEEF_LIMIT](#)

OPC_LOW_MEEF_NEG_TAG

Type: [Litho Variables](#). Used by Matrix OPC.

Tags low-MEEF fragments that should receive negative edge movements.

Usage

sse OPC_LOW_MEEF_NEG_TAG *tagname*

Arguments

- *tagname*

A required argument that specifies a tag name to assign to low-MEEF fragments that should receive negative movements.

Description

The default movement of a low-MEEF fragment is in the positive direction. If you are having difficulty getting the system to converge to a solution with low-MEEF fragments, the positive direction may not be correct. These fragments can be marked with a tag to indicate that they should be handled differently.

When a fragment receives *tagname*, it moves in the negative direction rather than the positive direction if it has a low-MEEF value. This tag has no effect if the MEEF does not meet the minimum requirement.

Low-MEEF Tagging Precedence

In the event that multiple low-MEEF tags are assigned to the same fragment, the movement value for a low-MEEF fragment is chosen in this order of precedence:

1. positive tag
2. freeze tag
3. negative tag
4. default (no tag at all)

For example, if a low-MEEF fragment were to have all three tags present, the positive movement value would be used. If only the freeze tag and negative tag were present, freeze would take precedence. If no low-MEEF handling tag were found at all, the default value would be used.

Related Topics

[OPC_LOW_MEEF_LIMIT](#)

[OPC_LOW_MEEF_NEG_SCALE](#)

[OPC_LOW_MEEF_POS_TAG](#)

OPC_LOW_MEEF_FREEZE_TAG

OPC_LOW_MEEF_POS_SCALE

Type: [Litho Variables](#). Used by Matrix OPC.

Scales the movement to apply to a low-MEEF fragment marked with the tag **OPC_LOW_MEEF_POS_TAG**.

Usage

sse OPC_LOW_MEEF_POS_SCALE *value*

Arguments

- ***value***

A required argument that specifies the amount of movement to apply to a low-MEEF fragment marked with the [OPC_LOW_MEEF_POS_TAG](#) tag. The default is [**1.0**](#), which indicates one full step in the positive direction.

Examples

sse OPC_LOW_MEEF_POS_SCALE 0.5

This example causes all fragments tagged with the positive low-MEEF tag name to move by only one half their normal value.

Related Topics

[OPC_LOW_MEEF_POS_TAG](#)

[OPC_LOW_MEEF_LIMIT](#)

OPC_LOW_MEEF_POS_TAG

Type: [Litho Variables](#). Used by Matrix OPC.

Tags low-MEEF fragments that should receive positive edge movements.

Usage

sse **OPC_LOW_MEEF_POS_TAG** *tagname*

Arguments

- *tagname*

A required argument that specifies a tag name to assign to low-MEEF fragments that should receive positive movements.

Description

The default movement of a low-MEEF fragment is in the positive direction. If you are having difficulty getting the system to converge to a solution with low-MEEF fragments, the positive direction may not be correct. These fragments can be marked with a tag to indicate that they should be handled differently.

Low-MEEF Tagging Precedence

In the event that multiple low-MEEF tags are assigned to the same fragment, the movement value for a low-MEEF fragment is chosen in this order of precedence:

1. positive tag
2. freeze tag
3. negative tag
4. default (no tag at all)

For example, if a low-MEEF fragment were to have all three tags present, the positive movement value would be used. If only the freeze tag and negative tag were present, freeze would take precedence. If no low-MEEF handling tag were found at all, the default value would be used.

Related Topics

[OPC_LOW_MEEF_LIMIT](#)

[OPC_LOW_MEEF_POS_SCALE](#)

[OPC_LOW_MEEF_NEG_TAG](#)

[OPC_LOW_MEEF_FREEZE_TAG](#)

OPC_MAX_ITER_MOVEMENT

Type: [Litho Variables](#). Used by Calibre OPCpro.

Defines the maximum fragment movement per iteration.

Usage

sse OPC_MAX_ITER_MOVEMENT *dist*

Arguments

- ***dist***

A required argument that specifies the maximum movement allowed per iteration, expressed in microns. This value must be ≥ 0 . The default value is equal to $((\text{LambdaNA})/8)$.

Note

 The default value is set to the largest reasonable value. You should set a smaller value for most applications to ensure convergence in your results. For Calibre versions beyond 2007.2, “default” no longer needs to be explicitly specified.

Description

This variable defines the maximum movement allowed per iteration. Note that this is different from [movelimit](#) and [setMoveLimitForTag](#), which sets the overall movement bounds. Use this keyword to improve OPC stability and control convergence.

OPC_MIN_ENCLOSURE_01

Type: [Litho Variables](#). Used by Matrix OPC.

Turns on enclosure checking between Mask 0 and Mask 1 (that is, for fragments such that the Mask 0 fragments enclose the Mask 1 fragments)

Usage

sse OPC_MIN_ENCLOSURE_01 *val metric*

Arguments

- ***val***

A required argument that defines the minimum allowed spacing.

- ***metric***

A required argument that defines the metric to be used when performing the checks.
Currently, only the Opposite metric is supported, and it must be specified.

Description

Enclosure rule checking checks the distances between fragments when a feature on one mask encloses a feature on the other mask. Enclosure rules used with Matrix OPC ensure that when the enclosure relationship is met by two edges, those edges maintain a minimum separation after movement by OPC. Here, “enclosure” is defined according to SVRF usage, comparing an interior-facing edge on one layer to an exterior-facing edge on another layer. As with the SVRF ENCclosure operation, order matters and results differ between when checking for Mask 0 enclosing Mask 1, compared to checking for Mask 1 enclosing Mask 0.

Enclosure rules are checked during each iteration of OPC. They are not enforced for jogs created by Calibre OPCpro.

Figure 6-52. Enclosure Check Between Mask 0 and Mask 1



Related Topics

[OPC_ENCLOSURE_MIN_LENGTH](#)
[OPC_ENFORCE_FRAG_LAYER_ORDER](#)
[OPC_IGNORE_ENCLOSURE_AT_CORNER](#)
[OPC_MIN_ENCLOSURE_10](#)
[LITHO_MASK_PRIORITY_MOVEMENT](#)

OPC_MIN_ENCLOSURE_10

Type: [Litho Variables](#). Used by Matrix OPC.

Turns on enclosure checking between Mask 1 and Mask 0 (that is, for fragments such that the Mask 1 fragments enclose the Mask 0 fragments).

Usage

sse OPC_MIN_ENCLOSURE_10 *val metric*

Arguments

- ***val***

A required argument that defines the minimum allowed spacing.

- ***metric***

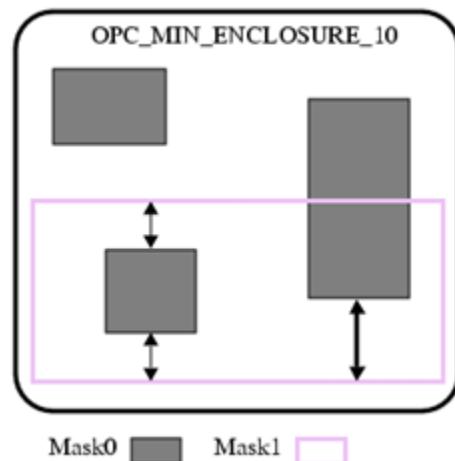
A required argument that defines the metric to be used when performing the checks. Currently, only the Opposite metric is supported, and it must be specified.

Description

Enclosure rule checking checks the distances between fragments when a feature on one mask encloses a feature on the other mask. Enclosure rules used with Matrix OPC ensure that when the enclosure relationship is met by two edges, those edges maintain a minimum separation after movement by OPC. Here, “enclosure” is defined according to SVRF usage, comparing an interior-facing edge on one layer to an exterior-facing edge on another layer. As with the SVRF ENCclosure operation, order matters and results differ between when checking for Mask 0 enclosing Mask 1, compared to checking for Mask 1 enclosing Mask 0.

Enclosure rules are checked during each iteration of OPC. They are not enforced for jogs created by Calibre OPCpro.

Figure 6-53. Enclosure Check Between Mask 1 and Mask 0



Related Topics

[OPC_ENCLOSURE_MIN_LENGTH](#)
[OPC_ENFORCE_FRAG_LAYER_ORDER](#)
[OPC_IGNORE_ENCLOSURE_AT_CORNER](#)
[OPC_MIN_ENCLOSURE_01](#)
[LITHO_MASK_PRIORITY_MOVEMENT](#)

OPC_MIN_SITE_ANGLE

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Ignores control sites between 0 and specified angle.

Usage

sse OPC_MIN_SITE_ANGLE *angle*

Arguments

- ***angle***

A required argument that represents the minimum acceptable angle between a site's normal vector and the site's fragment. It may be expressed as an angle between 0 and 90 degrees. The default value is **0.0**. (No detection.)

Description

When this variable is set, any control site with a rotation angle between 0 and the specified angle value is ignored. Sites with angles between the specified value and 90 degrees are used normally. When a control site is not rotated in any way, the angle it forms with the fragment is 90 degrees. Angles of 0 degrees mean the control site is parallel, possibly even coincident with, the fragment. This can happen because sites can be rotated prior to OPC.

Performing OPC with sites that are not perpendicular to the fragment may yield undesirable results, as the site's intensity values are invalid at extreme rotations. Setting this variable allows these sites to be detected, and no EPE estimations are made based on these sites. When this variable is not set, even a site that is fully parallel to a fragment would be accepted.

Caution

 While a value up to 90 degrees may be entered, it is not recommended. Entering large minimum angles could result in most or all sites being eliminated from OPC correction. Smaller values, such as 10 to 30 degrees, are more useful.

OPC_NEAREST_ADJ

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Controls which type of feature is given priority when placing sites on a fragment.

Usage

sse OPC_NEAREST_ADJ {[0](#) | [1](#)}

Arguments

- [0](#)

A required argument that places control sites by checking for the following corners in order: line end, convex, concave, no adjacent corner. For non-adjacent fragments, the site is placed in the default location. This is the default behavior.

- [1](#)

A required argument that places control sites on a fragment based on the nearest corner.

Description

This variable makes control sites easier to position by giving corners a higher priority. Previously, site placement near convex corners fluctuated and appeared in locations close to SITES_ON_EDGE. By default, a fragment may be adjacent to line ends, convex corners, and concave corners simultaneously. This could occur, for example, when a large value is specified for adjacent distances such as lineEndAdjDist and small polygons are present.

Setting this variable to 1 changes behavior so that the corner closest to a fragment takes precedence in determining site placement. Cases where adjacency conflicts occur are unusual, so the default value is [0](#).

Examples

Consider a fragment that is simultaneously line-end adjacent, convex adjacent, and concave adjacent.

The default behavior places the site using the settings for line-end adjacent sites.

When sse OPC_NEAREST_ADJ 1 is in the rule file, the nearest corner's settings are used.

Related Topics

[cornerSiteStyle](#)

OPC_NEW_FRAG_ORDER

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Changes the default order of fragmentation.

Usage

sse OPC_NEW_FRAG_ORDER {false | true}

Arguments

- **false**

A required argument that uses typical fragmentation order.

- **true**

A required argument that enables a newer order of fragmentation that is the same for both global and [fragmentLayer](#).

Description

This variable changes the default order of fragmentation. By default, it is off, and the following fragmentation orders are used:

Default Order of Fragmentation

| Global | Inside a fragmentLayer block |
|----------------|-------------------------------------|
| islandFragment | visible layer |
| visible layer | asraf layer |
| asraf layer | islandFragment |
| intrafeature | intrafeature |
| interfeature | interfeature |
| maxedgelength | maxedgelength |

Order of Fragmentation if OPC_NEW_FRAG_ORDER Is Set to True

[intrafeature](#)

[interfeature](#)

[maxedgelength](#)

[visible layers](#)

[asraf layer](#)

[islandFragment](#)

OPC_USE_LE_DEF

Type: [Litho Variables](#). Used by Calibre WORKbench and Calibre OPCpro.

Allows line-end fragmentation using the intrafeature command.

Usage

```
sse OPC_USE_LE_DEF {0 | 1}
```

Arguments

- **0**

A required argument that allows line ends to be fragmented with the [intrafeature](#) command.
(Does not apply to intrafeature fragmentation created with the [concavcorn](#) and [cornedge](#) commands.)

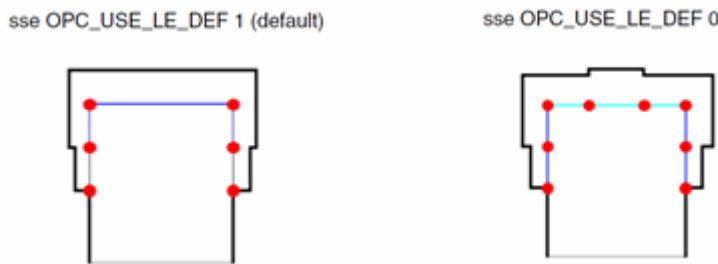
- **1**

A required argument that prevents intrafeature fragmentation from being applied to line ends. This is the default.

Description

Controls whether OPC uses the line end definition. This can affect intrafeature fragmentation.

Figure 6-54. OPC_USE_LE_DEF Effects



Intrafeature fragmentation is performed two different ways: with concavcorn and cornedge commands, or with the intrafeature command. Even with seriftype 0, ends are only fragmented by concavcorn and cornedge keywords. Line ends, and short edges which meet the definition of a line end, are not fragmented by the intrafeature command by default (setting 1).

Setting this variable to 0 disables the line end definition and allows a short edge to be fragmented via the intrafeature command. Edges are not tagged as [tLineEnd](#) or [tLineEndAdjacent](#). (The basic [line_end](#) and [line_end_adjacent](#) tags are still used.) The [seriftype](#) keyword has no effect.

Note

 When OPC_USE_LE_DEF is 0, you must use intrafeature fragmentation instead of concavcorn and cornedge.

Related Topics

[lineEndLength](#)

SEM_CONSIDER_SITE_OFFSET

Type: [Litho Variables](#). Used by Calibre PRINTImage.

Calculates a site offset along the direction perpendicular to the edge that gives a correct EPE result.

Usage

sse SEM_CONSIDER_SITE_OFFSET {0 | 1}

Arguments

- **0**
A required argument that calculates displacement along the direction of the site.
- **1**
A required argument that calculates displacement normal to the edge. This setting should be used when running Calibre PRINTImage.

Description

Edge placement error (EPE) is typically calculated along the control site. This may or may not be normal to the edge, particularly around corners. Sites that are not near normal can cause false errors during Calibre PRINTImage operations.

When the SEM_CONSIDER_SITE_OFFSET variable is set to **1**, the offset is calculated in the direction perpendicular to the edge. This gives an accurate result.

When [SEM_USE_SUF_SITES](#) is set to 0, this variable is automatically set to **1**. Do not set it to **0** while using SEM_USE_SUF_SITES.

SEM_DO_MORPH

Type: [Litho Variables](#). Used by Calibre PRINTImage.

Enables morphological over/under sizing of the Calibre PRINTImage result.

Usage

sse SEM_DO_MORPH {false | true}

Arguments

- **false**

A required argument that disables overunder sizing of the results.

- **true**

A required argument that enables overunder sizing of the results. This is the default.

Description

By default, Calibre PRINTImage performs a morphological overunder sizing of its results before PIECEWISE LINEAR. The amount of overunder sizing is controlled by the variable [SEM_MORPH_SIZE](#).

Related Topics

[SEM_MAX_NONPRINT_MOVE](#)

SEM_MAX_NONPRINT_MOVE

Type: [Litho Variables](#). Used by Calibre PRINTimage.

Controls the handling of non-printable edges.

Usage

sse SEM_MAX_NONPRINT_MOVE *val*

Arguments

- *val*

A required argument that limits how far nonprintable edges on features can move. Legal values are whole numbers in dbus or -1 (to disable). The default is (lambda/NA)/3.

Description

This setup variable is used with Calibre PRINTimage to control the handling of non-printable edges.

Non-printable edges can occur on features which are not intended to be present in the final printed result. This can include features such as very small notches or negative scattering bars. These features can cause problems with PRINTimage, which may assume that such features are printable. The result is that these edges may be incorrectly moved a large distance from their origin, and cause spikes to appear in the PRINTimage results. These spikes are not present in a real print. This variable prevents these edges from being moved a large distance, thereby preventing spikes from forming.

The default value for SEM_MAX_NONPRINT_MOVE is (lambda/NA)/3. If a different value is needed, it can be specified in database units as part of the setup file. Comparing the results from an aerial image is recommended before attempting to change this value.

If necessary, this feature can be turned off completely by setting the distance to “-1”. This removes all limits on edge movement for PRINTimage operations. For example:

```
sse SEM_MAX_NONPRINT_MOVE -1
```

This shuts off all limits on unprintable edges and causes Calibre PRINTimage to behave as it did in versions prior to 2006.4.

SEM_MORPH_SIZE

Type: [Litho Variables](#). Used by Calibre PRINTImage.

Sets the amount of the morphological overunder sizing operation.

Usage

sse SEM_MORPH_SIZE *floating_point_number*

Arguments

- *floating_point_number*

A required argument that sets the amount of morphological overunder sizing in microns.
The default is [0.01](#).

Description

The SEM_MORPH_SIZE setup variable sets the amount of the morphological overunder sizing operation. The variable is only used when the [SEM_DO_MORPH](#) is set to true.

When SEM_MORPH_SIZE is positive, Calibre PRINTImage first undersizes a shape and then oversizes it.

When SEM_MORPH_SIZE is negative, the shape is first oversized and then undersized.

The morphological sizing is similar to the SVRF SIZE UNDEROVER or SIZE OVERUNDER operation.

Related Topics

[Size \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

SEM_PWL_JOT_SIZE

Type: [Litho Variables](#). Used by Calibre PRINTimage.

Controls filtering on edges which are excluded from piecewise linear conversion.

Usage

sse SEM_PWL_JOT_SIZE *floating_point_number*

Arguments

- *floating_point_number*

A required argument that specifies a minimum size in microns for edges to receive piecewise linear conversion. Any edge which is shorter than *floating_point_number* is excluded from the piecewise linear “connect the dots” operation.

Related Topics

[SEM_USE_PWL](#)

SEM_USE_PWL

Type: [Litho Variables](#). Used by PRINTimage.

Controls whether a rectilinear or piecewise linear image is drawn.

Usage

sse SEM_USE_PWL {false | true}

Arguments

- **false**

A required argument that sets the output to rectilinear. This is the default.

- **true**

A required argument that sets the output to piecewise linear.

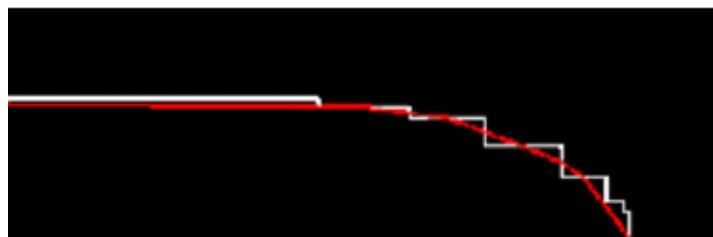
Description

The Calibre PRINTimage batch tool supports two modes, rectilinear and piecewise linear, for drawing contours for SEM images. They use the same method for calculating the image data. The difference between the two drawing modes lies in how the tool connects up the many print locations:

- In rectilinear mode, the Calibre PRINTimage batch tool draws contours by moving each fragment to line up with the print location, then connecting the vertices of adjacent fragments. The resulting image appears somewhat jagged. This is the default.
- In piecewise linear mode, the Calibre PRINTimage batch tool draws contours by drawing straight lines connecting the print locations on adjacent control sites. To use this mode, set SEM_USE_PWL to **true**.

Figure 6-55 shows the rectilinear image in white and the piecewise linear image in magenta.

Figure 6-55. Piecewise Linear vs. Rectilinear SEM



Related Topics

[SEM_PWL_JOT_SIZE](#)

SEM_USE_SUF_FRAG

Type: [Litho Variables](#). Used by Calibre PRINTimage.

Controls the use of fragmentation from the associated setup controls for simulated SEMs.

Usage

```
sse SEM_USE_SUF_FRAG {false | true}
```

Arguments

- **false**

A required argument that specifies Calibre PRINTimage should use the PRINTimage heuristics, which are optimized for SEM simulation, rather than the fragmentation settings in the litho setup file.

- **true**

A required argument that causes the Calibre PRINTimage operation to use the OPC fragmentation settings.

Description

This setup variable specifies whether the Calibre PRINTimage application uses the fragmentation in the associated setup controls to generate SEM images or uses special heuristics optimized for SEM simulation.

Setting this variable to true means that the fragmentation specified using the cornedge, concavecorn, minfeature, maxedgelength, minedgelength and interfeature commands is used during SEM computation. The default value is **false**, which causes values calculated by the PRINTimage operation to override those in the setup controls.

The fragmentation for simulated SEMs is usually much finer than for OPC or ORC, in order to produce a smoother, more visibly pleasing image. This variable allows one to specify coarser or finer fragmentations to control the computational complexity of the operation.

Related Topics

[SEM_USE_PWL](#)

[SEM_USE_SUF_SITES](#)

SEM_USE_SUF_SITES

Type: [Litho Variables](#). Used by Calibre PRINTimage.

Controls the position and orientation of sites when generating a SEM image with Calibre PRINTimage.

Usage

```
sse SEM_USE_SUF_SITES {false | true}
```

Arguments

- **false**

A required argument that causes control sites to be placed as SITE_ON_ARC and orthogonal for the Calibre PRINTimage run. This is the default.

- **true**

A required argument that causes Calibre PRINTimage to use the settings in the litho setup file.

Related Topics

[cornerSiteStyle](#)

VERBOSITY

Type: [Litho Variables](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Controls how errors, warnings, and status messages are reported.

Usage

`sse VERBOSITY integer`

Arguments

- *integer*

A required argument that sets the verbosity level. Use -1 to turn off logging. The maximum value is 100. The default is 10.

Description

The verbosity level controls how errors, warnings, and status messages are reported. A low verbosity causes fewer such messages to be printed to stdout. A verbosity of “-1” turns off logging. A verbosity of “100” logs all possible messages. The verbosity level can also be set at program start by using the -verbosity switch.

Table 6-16. Verbosity Levels

| | |
|-----|--|
| -1 | Nothing is logged |
| 0 | Minimal logging |
| 10 | More command results are logged |
| 20 | Command intermediate results such as EPE values are logged |
| 100 | Full logging |

Tagging Commands

Tagging commands work on specific subsets of fragments. These subsets are “tag sets.” You can define custom tag sets with the newTag commands and then modify the fragments in the sets using other tagging commands.

Tagging commands should appear at the end of the setup file. Unlike litho setup file keywords, tagging commands can appear multiple times in the file.

You can also use the tag sets that Calibre uses for its fragmentation methods. These tag sets are listed in “[Pre-Defined Tags](#)” on page 541.

Table 6-17. Tagging Commands Summary

| Tag Command | Description |
|---|--|
| bindModelToTag | Controls which model is used for performing simulations on a tag-by-tag basis. |
| clearTagScript | Clears all previous tagging commands. |
| dump | Outputs a formatted text file containing information about fragments during OPC. |
| epeToleranceTag | Defines an EPE tolerance to be used for specified tags. |
| fastIter | Provides a direct method of executing fast iterations in the tagging commands. |
| fragalign | Aligns fragment breaks on horizontal and vertical edges to reduce shot count. |
| fragcoinc | Copies coincident fragmentation from one layer to another. |
| fragmentTag | Re-fragments edge fragments based on tags. |
| fragmentTag ... interfeature | Re-fragments interfeature fragmentation by projecting corners from the source(s) onto fragments. |
| fragmentTag ... intersection | Re-fragments island-layer type fragmentation on fragments meeting certain criteria. |
| fragmentTag ... maxedgelength | Re-fragments island-layer type fragmentation on fragments meeting certain criteria. |
| fragmentTag ... projection | Re-fragments island-layer type fragmentation on fragments meeting certain criteria. |

Table 6-17. Tagging Commands Summary (cont.)

| Tag Command | Description |
|--|---|
| <code>fragmentTag ... split</code> | Re-fragments island-layer type fragmentation on fragments meeting certain criteria. |
| <code>histogram</code> | Counts the number of edge fragments with the given tags in the form of a histogram. |
| <code>jogsmooth</code> | Makes fragments collinear to smooth jogs. |
| <code>MATRIX_OP</code> | Runs Matrix OPC, a feature for multiple mask OPC. |
| <code>newTag</code> | Creates new, user-defined tags using the methods described in subsequent reference pages. |
| <code>newTag ... -how ANDTAGS</code> | Creates a new tag <i>tagName</i> containing everything that is tagged with <i>tag1</i> AND <i>tag2</i> AND ... <i>tagN</i> . This identifies the intersection of two or more sets of fragments. |
| <code>newTag ... -how COINCIDENTEDGE</code> | Tags all fragments on a source layer that are coincident with a marker layer. |
| <code>newTag ... -how COINCIDENTINSIDEEDGE</code> | Tags all fragments on the source layer that are inside coincident with a marker layer. |
| <code>newTag ... -how COINCIDENTOUTSIDEEDGE</code> | Tags all fragments on the source layer that are outside coincident with the marker layer. |
| <code>newTag ... -how DISPLACEMENT</code> | Creates a new tag after model-based OPC based on how far the fragment moved (was displaced) during correction. |
| <code>newTag ... -how EDGE</code> | Tags all fragments on an edge that meets constraints. There are two mutually-exclusive tagging modes: FROMCORNER; or TYPE, LENGTH and ANGLE. |
| <code>newTag ... -how EPE</code> | Tags all fragments with an Edge Placement Error (EPE) within a desired range. |
| <code>newTag ... -how EPESENSITIVITY</code> | Checks the EPE tolerances as a function of varying the threshold of the resist model. |
| <code>newTag ... -how EXTERNAL</code> | Tags fragments based on the DRC External dimension check. |
| <code>newTag ... -how fragAfterEdge</code> | Tags all fragments after the edge tagged with tag1. |

Table 6-17. Tagging Commands Summary (cont.)

| Tag Command | Description |
|---|---|
| <code>newTag ... -how fragAfterFrag</code> | Tags all fragments after the fragment tagged with tag1. |
| <code>newTag ... -how fragBeforeEdge</code> | Tags all fragments before the edge tagged with tag1. |
| <code>newTag ... -how fragBeforeFrag</code> | Tags all fragments before the fragment tagged with tag1. |
| <code>newTag ... -how FRAGMENT</code> | Tags fragments based on their lengths and the types of corners at either end point. |
| <code>newTag ... -how fragNextToEdge</code> | Tags all fragments next to the edge tagged with tag1. |
| <code>newTag ... -how fragNextToFrag</code> | Tags all fragments next to the fragment tagged with tag1. |
| <code>newTag ... -how HAS_SITE</code> | Creates a subgroup based on whether a site was generated on tagged fragments. |
| <code>newTag ... -how IMAGE</code> | Tags fragments based on aerial image properties. |
| <code>newTag ... -how INSIDEEDGE</code> | Tags all source layer fragments that lie completely inside the marker layer polygons. |
| <code>newTag ... -how INTERNAL</code> | Tags fragments based on the DRC INTERNAL dimension check. |
| <code>newTag ... -how LAYERAND</code> | Tags fragments on opc layers that intersect polygons on island layers. |
| <code>newTag ... -how LAYERANDNOT</code> | Tags fragments on opc layers that do not interact with polygons on island layers. |
| <code>newTag ... -how loopWithFrag</code> | Creates a new tag containing all the loops tagged with fragmentTag. |
| <code>newTag ... -how MRC</code> | Tags fragments that had their movements restricted during the previous iteration. |
| <code>newTag ... -how NOTCOINCIDENTEDGE</code> | Tags all fragments on the source layer that are not tagged by COINCIDENTEDGE. |
| <code>newTag ... -how NOTCOINCIDENTINSIDEEDGE</code> | Tags all fragments on the source layer that are not tagged by COINCIDENTINSIDEEDGE. |
| <code>newTag ... -how NOTCOINCIDENTOUTSIDEEDGE</code> | Tags all fragments on layer1 that are not tagged by COINCIDENTOUTSIDEEDGE. |

Table 6-17. Tagging Commands Summary (cont.)

| Tag Command | Description |
|--|---|
| <code>newTag ... -how NOTINSIDEEDGE</code> | Tags all fragments on layer1 that are not tagged by INSIDEEDGE. |
| <code>newTag ... -how NOTOUTSIDEEDGE</code> | Tags all fragments on layer1 that are not tagged by OUTSIDEEDGE. |
| <code>newTag ... -how NOTTOUCHEDGE</code> | Tags all fragments not tagged by the corresponding TOUCHEDGE. |
| <code>newTag ... -how NOTTOUCHINSIDEEDGE</code> | Tags all layer1 fragments not tagged by the corresponding TOUCHINSIDEEDGE. |
| <code>newTag ... -how NOTTOUCHOUTSIDEEDGE</code> | Tags all fragments not tagged by the corresponding TOUCHOUTSIDEEDGE. |
| <code>newTag ... -how ORTAGS</code> | Creates a new tag everything that is tagged with either tag1 OR tag2 OR ... tagN. This allows you to find the union of two or more sets of fragments. |
| <code>newTag ... -how OUTSIDEEDGE</code> | Tags source layer fragments that lie completely outside marker layer polygons. |
| <code>newTag ... -how SEQUENCE</code> | Identifies a sequence of edges (multiple fragments per edge) and places all fragments that make up the sequence into the output tag set. |
| <code>newTag ... -how SUBTRACTTAGS</code> | Creates a new tag by subtracting everything in tag2 to tagN from tag1. |
| <code>newTag ... -how TOUCHEDGE</code> | Tags all layer1 fragments that are completely or partially coincidental with an edge on layer2. |
| <code>newTag ... -how TOUCHINSIDEEDGE</code> | Tags all layer1 fragments that are completely or partially inside coincidental with an edge on layer2. |
| <code>newTag ... -how TOUCHOUTSIDEEDGE</code> | Tags all layer1 fragments that are completely or partially outside coincidental with an edge on layer2. |
| <code>opcTag</code> | Controls the OPC method by tag. |
| <code>setMoveLimitForTag</code> | Limits movements for all edge fragments with the given tag. |
| <code>siteControl DELETE</code> | Deletes sites on all fragments in tag. |

Table 6-17. Tagging Commands Summary (cont.)

| Tag Command | Description |
|--|---|
| siteControl OFFSET_FROM | Moves sites on all tagged fragments that already have a site, and creates new sites if they do not exist. |
| siteControl OFFSET_PROP | This command is identical to the siteControl OFFSET_FROM command, except that its arguments are expressed as a fraction of the fragment's length, rather than distance. |
| siteControl OFFSET_ROTATED | A command that defines tag-specific site placement for each fragment using the site's coordinate system. |
| siteControl SHIFT_PROP | Moves sites on tagged fragments that already have a site toward a particular corner type, and creates new sites when they do not exist. This command is similar to siteControl SHIFT_TOWARD. |
| siteControl SHIFT_TOWARD | Shifts a control site towards a specified corner type. |
| siteControl SMOOTH | Moves existing sites on fragments using a 2D Gaussian. |
| sitemodify | Trims the control sites for specific tagged fragments. |
| tags2boxes | Creates boxes around tags or sites. |
| Pre-Defined Tags | The built-in tags are used to globally select specific types of fragments, such as mid-edge segments or fragments at the initial fragmentation points. They can be used in any command calling for a tag. |

bindModelToTag

Type: [Tagging Commands](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.
Controls which model is used for performing simulations on a tag-by-tag basis.

Usage

bindModelToTag *modelName tagName*

Arguments

- *modelName*

A model whose name has been defined by the [processModel](#) *modelname modelfile* setting.

- *tagName*

The tag that identifies the fragments for which the specified model applies.

Description

This command controls which model is used for performing simulations on a tag by tag basis.

The model must have the same reference threshold as the default resist model or the run exits with the following error:

SETUP FILE ERROR: The model named **modelName** and default resist model have different referenceThreshold settings.

Examples

```
resistpolyfile mymodel.mod          # primary model
processModel  thresh25 C > 0.3      # 0.25 model for line end
processModel  thresh30 C < -0.2     # 0.30 model for inv line end
...
bindModelToTag thresh25 line_end
bindModelToTag thresh30 tSpaceEnd
```

Related Topics

[Pre-Defined Tags](#)

clearTagScript

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Clears all previous tagging commands.

Usage

clearTagScript

Arguments

None.

Description

This tagging command clears all previous tagging commands and should always be the first line in the tagging commands section.

dump

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Outputs a formatted text file containing information about fragments during OPC.

Usage

```
dump { {filename} [-iters on]} | -iters off}
```

Arguments

- ***filename***

A required argument specifying the name of the output file containing the dumped formatted text. When dumping within iterations, the ***filename*** becomes the prefix, and the files are written as *filename.0*, *filename.1*, *filename.2*, and so on, up to the number of iterations. This is a total of $N+1$ files for N iterations.

- **-iters on**

An optional argument that causes information to be output during iterations. The information is not output immediately, but rather during the next iteration and any subsequent iterations until turned off with **-iters off**.

- **-iters off**

A required argument that turns off information output. You can specify either ***filename*** or **-iters off** but not both.

Description

This keyword dumps a formatted text file containing information about the fragments during OPC. The current state of the fragments is included in the dumped information. A fragment typically contains the following information:

- Number of points
- Coordinates of the points
- Fragment endpoints
- Sites
- X-Y intensity values
- EPE at each of the site points
- Bias EPE
- Threshold EPE
- On any iteration, the desired movement and actual movement

The format of the information dump includes an XML-like separator that can be used to parse individual fragments from the overall file. The current implementation of this has no

hierarchical format, and overwrites the *filename* for each tile and for each cell during the processing. Therefore, it is best used for small (inside of one tile) flat cases.

This command can also be run from Calibre WORKbench, but only non-globally (only the last tile in a set of tiles is dumped).

If no **fastIter** commands are present, then the **iterations** keyword causes iterations to occur at the end of the tag script. To allow the dumping during those iterations, set -iters on as the last item in the tag script. When the dump file is written for iterations, the *filename* is the prefix, and the files are written as *<filename>.0*, *<filename>.1*, and so on, up to the number of iterations for a total of $N+1$ files for N iterations.

Example of Format

The following is a segment of the dump format for a single fragment.

```
<fragment>
coords      1545 1570      1545 1690
layer       6
disp        5
desired_disp 5
forced      0
invalid_hier 0
flag        67108864
image       20      3
n          0      0.084806  1645 1630
n          1      0.095556  1625 1630
n          2      0.099514  1605 1630
n          3      0.095555  1585 1630
n          4      0.084800  1565 1630
n          5      0.070148  1545 1630
n          6      0.055074  1525 1630
n          7      0.042378  1505 1630
n          8      0.033478  1485 1630
n          9      0.028354  1465 1630
n         10      0.026049  1445 1630
n         11      0.025345  1425 1630
n         12      0.025317  1405 1630
n         13      0.025546  1385 1630
n         14      0.025975  1365 1630
n         15      0.026602  1345 1630
n         16      0.027287  1325 1630
n         17      0.027816  1305 1630
n         18      0.028083  1285 1630
n         19      0.028173  1265 1630
t          0      0.051823  1645 1630
t          1      0.053728  1625 1630
t          2      0.051818  1605 1630
imin       0.025317
imax       0.099514
slope      0.680986
C          -0.381576
epe_r      6.111522
th         0.048258
epe_th     30.737381
epe_bias   0.000000
epe        30.737381
density_eval 1523 1630
d          0      0.573529
d          1      0.870570
d          2      1.000000
</fragment>
```

Examples

In this example, the fragment information is dumped before and after calling the image tagging operation. Because the image tagging operation invokes a simulation pass, the values of the simulation properties on the fragments in *before.txt* are uninitialized. The values of the simulation properties in *after.txt* contain valid simulation results.

```
# Dump before calling image tagging
dump before.txt
newTag t1 -how image all slope > 2

# Dump again, after doing simulation
dump after.txt

# Turn on dumping during "fastIter" iterations
dump dump.txt -iters on
fastIter 4

# Turn off the dumping for future iterations
dump -iters off
```

This produces the following files:

- *before.txt*
Contains the state at the beginning of the tag script.
- *after.txt*
Contains the state after the simulation caused by the “image” tagging operation.
- *dump.txt.0, dump.txt.1, dump.txt.2, dump.txt.3, dump.txt.4*
These files contain the state during the iterations. The first N files, where N is the number of iterations, contain the state just after simulation but just before edge movement is performed. The final file contains the state just after the final iteration movement.

epiToleranceTag

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Defines an EPE tolerance to be used for specified tags.

Usage

epiToleranceTag tag *d1* [*d2*]

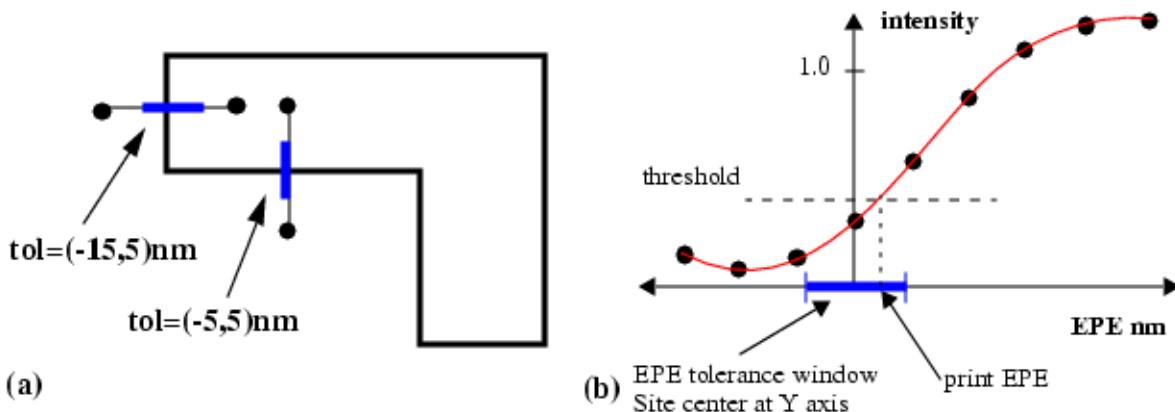
Arguments

- **tag**
A required argument specifying the name of the tag to which this tolerance applies.
- ***d1***
A required argument specifying the outer tolerance, in microns.
- ***d2***
An optional argument specifying the inward tolerance, in microns. Because inward EPEs are expressed as negative values, this value must also be negative. When not specified, this value defaults to -*d1*.

Description

This tagging operation allows users to define an EPE tolerance to be used for specified tags during OPC. During a given iteration, when performing OPC on a tag for which `epiToleranceTag` is set, OPCpro only moves those edges having an EPE outside the tolerance range. Up to 20 `epiToleranceTag` operations can be specified. Drawing (a) in [Figure 6-56](#) shows different EPE tolerance windows on neighboring simulation sites on the layout.

Figure 6-56. Tolerance Windows



Drawing (b) in [Figure 6-56](#) shows a sample cutline at a simulation site. Note the EPE tolerance window on the x-axis. Negative direction is toward the polygon inside and positive is the outside.

If more than one epeToleranceTag is applied to the same *tag*, only the first one is used. Keep in mind that a fragment may have more than one tag. In particular, built-in tags such as line_end are automatically applied. Multiple tags can result in seemingly inconsistent fragment behavior. If a fragment does not belong to any epeToleranceTag, the application assigns it the default tolerance, which is [OPC_EPE_TOLERANCE_FRAC * stepsize](#).

This operation provides functionality that is similar to that provided by the OPC_EPE_TOLERANCE_FRAC variable, but much more flexible. The differences are:

- How the values are used
 - OPC_EPE_TOLERANCE_FRAC defines a *tolerance factor* that is multiplied by the stepsize to calculate the minimum distance EPE receiving correction.
 - This tagging operation defines the actual tolerance range in microns. (The value is not multiplied by the stepsize.)
- Tolerance symmetry
 - With OPC_EPE_TOLERANCE_FRAC the same tolerance is used for outward (that is, positive) and inward (negative) EPEs.
 - This tagging operation lets you specify a different tolerance for outward EPEs than for inward EPEs.
- Scope
 - OPC_EPE_TOLERANCE_FRAC is used to define the default tolerance for all fragments.
 - This tagging operation defines a tolerance for a specific tag. You can issue this command multiple times to define different tolerances for different tags.

Examples

In this example, multiple tolerances are applied for EPE and for slope.

```
stepsize 0.001
sse OPC_EPE_TOLERANCE_FRAC 5
newTag t1 -how edge LENGTH < 0.12
epeToleranceTag line_end 0.005 -0.015 # tol window (-15,5)nm
epeToleranceTag t1 0.005 # tol window (-5,5)nm
MATRIX_OPC {
    METHOD CLOSEST
    OPTSLOPE YES
    SEARCH 0.1
    METRIC EUCLIDEAN
    MEEF_MODE 0.7
    DELTA 0.001
    slopeToleranceTag line_end 2.7 # min slope for line ends
    slopeToleranceTag t1 3.1      # min slope for short fragments
    MINSLOPE 3.0                  # default min slope
}
```

A line end which is less than 0.12 microns long matches both “line_end” and “t1”. However, line_end is first in the setup file, so the (-15,5) nm tolerance window applies to any such line end shorter than 0.12 microns. Fragments which are not “t1” or “line_end” automatically have a (-5,5)nm tolerance window defined. The diagram in [Figure 6-56](#) shows a case of two different sites, which have different EPE tolerance windows defined. This also shows the intensity cutline and how the EPE tolerance is viewed from the cutline perspective.

Related Topics

[newTag ... -how EPE](#)

[newTag ... -how EPESENSITIVITY](#)

[setMoveLimitForTag](#)

[Pre-Defined Tags](#)

fastIter

Type: [Tagging Commands](#). Used by Calibre WORKbench and Calibre OPCpro.

Provides a direct method of executing fast iterations in the tagging commands.

Usage

```
fastIter tag number [-stepby step_num]  
[-mrc_cleanup cleanup_limit]
```

Arguments

- **tag**
A required string specifying the tag receiving the corrections.
- **number**
A required integer stipulating the number of fast iterations to perform. This overrides any other iteration setting.
- **-stepby *step_num***
An optional argument specifying a movement increment. The default, if not specified, is to move in increments of 1 step. The *step_num* value must be a positive integer. The effect of specifying this argument is to multiply the stepsize by the specified *step_num* value.
The fastIter command does *not* use the auto stepby algorithm controlled by the environment variable [OPC_AUTO_STEPBY_THRESHOLD](#).
- **-mrc_cleanup *cleanup_limit***
An optional argument specifying that cleanup of MRC constraints should be done following the iterations. During a fastIter process, MRC enforcement is done by restricting movement. This option specifies that additional enforcement can be performed at the end, even if it results in an increase in the bias for a fragment. The *cleanup_limit* argument specifies how much the bias is allowed to increase on any one fragment during this process. The value is specified in user units (microns).
The cleanup is performed after all iterations are complete. If you wish to perform OPC after this cleanup, an additional fastIter command can be used.

Description

The fastIter command provides a direct method of executing fast iterations in the tagging commands.

Calibre OPCpro corrects all edges simultaneously at specified edges (using tag numbers). You can use fastIter for a more controlled and customized iteration process, particularly when you want to do the following:

- Control the processing order of edge corrections by specifying tags.
- Control the number of iterations of corrections on specified edges.

- Create iterative corrections based on previous corrections (such as image properties or edge placements).

When the fastIter command is reached during a tagging command evaluation, the specified number of fast iterations (not the number specified in [iterations](#)) is performed on the fragments with the user-supplied tag.

This command is commonly used with the [opcTag](#) -hintOffset tagging command, which calculates the fragment position by adding the offset to the original position. After this operation, you typically use the fastIter command to instruct the Calibre OPCpro batch tool to “process” the hints into final bias offsets.

Note

 Do not specify fast iterations in both the setup file and the tagging commands. Whenever iterations are specified in the tagging script, the iterations keyword is ignored (0 iterations).

Examples

The following example performs several hintOffset operations, followed by EPE checking, and then more hintOffsets on top of the previous hintOffsets.

```
newTag t1 -how EPE all -0.02 -0.01      // find things printing to small
opcTag t1 -hintOffset 0.01                  // bias them out a little bit
fastIter all                                // process all hintOffsets, not just t1 tag
newTag t2 -how t1 all -0.02 -0.01          // find things STILL too small
opcTag t1 -incrHintOffset 0.01              // bias them out a little more
fastIter all                                // process all hintOffsets
```

Related Topics

[clearTagScript](#)

[fragmentTag](#)

[iterations](#)

[newTag](#)

[opcIter](#)

[opcTag](#)

[LITHO_MRC_CLEANUP_LIMIT](#)

fragalign

Type: [Tagging Commands](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.
Aligns fragment breaks on horizontal and vertical edges to reduce shot count.

Usage

fragalign layer tag

maxmove max_move maxpolywidth max_width [alignToCorners] [copy [allowDelete]]
[minedgelen minedgelength] [priorityTag *inTagName*] [unalignedTag *outTagName*]

Arguments

- ***layer***

A required argument specifying the name of a layer containing fragments to align.
Fragments on other layers are not adjusted.

- ***tag***

A required argument that specifies the tag set of fragments to adjust. Use “all” to align all fragments.

Because all of the fragments must be contiguous, it is best to specify all of the tags to be smoothed at one time and not to use multiple calls to fragalign. Multiple tag sets should be combined first, as shown in “[Example 1 - Combining Tags](#)” on page 386.

- ***maxmove max_move***

A required argument specifying the maximum distance in microns that a fragment break may move from its original location. The ***max_move*** must be greater than or equal to 0.

- ***maxpolywidth max_width***

A required argument specifying the maximum distance across a polygon for which two edges should have their fragments aligned. The ***max_width*** must be greater than or equal to 0. To prevent unintended alignment, this should be set to the minimum acceptable value.

- **alignToCorners**

An optional argument that indicates fragments opposite a corner should also be aligned.

By default, fragments opposite a corner are not aligned to a corner because that is done by [interfeature](#) fragmentation. However, sometimes this does not occur or fragments are adjusted so that corners are not aligned. With alignToCorners, fragalign attempts to align fragments opposite corners also.

If priorityTag is set and a fragment opposite a corner is tagged *inTagName*, priorityTag takes priority and the fragment is not adjusted.

- **copy [allowDelete]**

An optional argument that copies fragment breaks from one Manhattan edge of a polygon to the parallel edge. The two edges must be separated by no more than ***max_width***. Fragment

breaks that violate *minedgelength* are not copied. Only fragments of type **tag** on **layer** have their breaks copied.

The allowDelete keyword can only be used with copy. The keyword forces alignment when it is not otherwise possible by allowing a fragment to be deleted if and only if the deletion results in correct alignment. An example of where this might occur is when one side of a polygon has promoted fragments that cannot be moved. This means that only the opposite side can be adjusted.

- **minedgelen** *minedgelength*

An optional argument specifying the shortest edge that may be created during alignment. Lengths must be greater than 0. Although this value is optional, it is recommended for clarity.

The default value for **minedgelen** is the **minedgelength** value specified in the **fragmentLayer** block. If **minedgelength** was not specified in a **fragmentLayer** block, it defaults to the global default value. If more than one **fragmentLayer** block specifies a **minedgelength** value the first one is used.

- **priorityTag** *inTagName*

An optional argument specifying the tag set of fragments that should not be changed. Other fragments are aligned to *inTagName* fragments. The *inTagName* set should be a subset of the **tag** set.

- **unalignedTag** *outTagName*

An optional argument specifying that non-aligned fragments in **tag** be tagged as *outTagName*. By default, unaligned fragments are not marked.

For a fragment to be tagged *outTagName*, it must be able to be aligned within the constraints of **maxmove** and **maxpolywidth** but has at least one endpoint that does not align with the fragment across the polygon.

Description

The optional **fragalign** keyword aligns specified fragments so that the fragment breaks across a feature are in line as shown in [Figure 6-57](#). It does not affect fragments on edges that are not strictly horizontal or vertical. A fragment break is adjusted to mirror its neighbor across the polygon if all of the following conditions are met:

- The two edges are parallel and on the same polygon.
- The distance between the two edges does not exceed **maxpolywidth**.
- The fragment break of one or both does not need to be moved more than **maxmove**.
- The adjusted fragments do not violate **minedgelen** (the argument specified in **fragalign**, not the **minedgelength** command).

The breakpoints to be aligned are generally adjusted to the midpoint between the two breaks. In some cases, this may violate minedgelen, so other points are tried. If no suitable points are found, neither fragment's break point is moved.

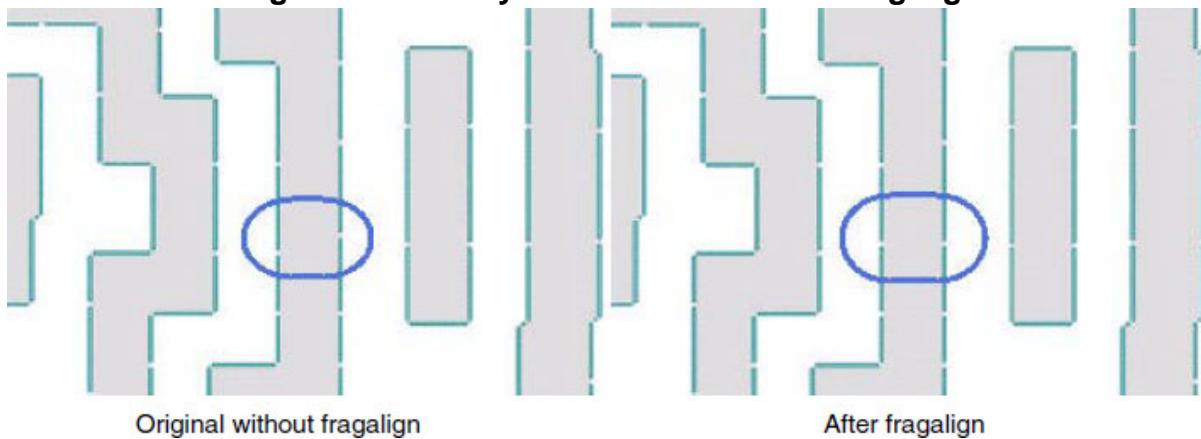
In some situations, you may have fragments that should not change but still want to reduce shot count through fragalight. Other fragments can be aligned to the fragments that should not change. To handle this, create a tag to identify the fragments that should not be moved and use the priorityTag argument.

Exceptions

The following types of fragments do not get aligned:

- Fragments on non-Manhattan edges do not get aligned. Only vertical and horizontal edges are considered.
- By default, fragments opposite a corner are not aligned. You can enable this using the alignToCorners argument.
- Fragments whose adjustment would violate the minedgelen argument are not aligned.

Figure 6-57. A Layout With and Without fragalight



Examples

Example 1 - Combining Tags

This example finds fragments on the opc layer that are on lines greater than 65 nm wide and further than 65 nm from other features. All fragments which meet both requirements are aligned. This approach runs more quickly than trying to align twice, and is more effective at identifying long fragment spans which can be aligned.

```
newTag wideLayer -how INTERNAL > 0.065 OPPOSITE INPUT1 opc_layer all
newTag wideGap -how EXTERNAL > 0.065 OPPOSITE INPUT1 opc_layer all
newTag alignTags -how andTags wideLayer wideGap

fragalign opc_layer alignTags maxmove 0.005 maxpolywidth 0.040 \
minedgelen 0.025
```

Example 2 - Preserving Corner Fragments With priorityTag

Fragments near corners are sometimes very sensitive to adjustments and should not be lengthened or shortened during fragalign. Preserve these fragments by creating an identifying tag set and using priorityTag. The protected fragments serve as the starting point for aligning other fragments.

```
newTag corners -how ORTAGS convex_corner concave_corner  
fragalign opcLayer all maxmove 0.005 maxpolywidth 0.040 \  
priorityTag corners
```

The fragalign command in this example adjusts all fragments on the opcLayer except for the fragments in the corners tag set. This permits most of the fragments to be aligned to the midpoint between fragments. The fragments on corners are not changed but the fragment across the polygon from them may be lengthened or shortened to line up with the end of a corner fragment.

fragcoinc

Type: [Tagging Commands](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.
Copies coincident fragmentation from one layer to another.

Usage

fragcoinc *coincLayer* *layerToFrag*

Arguments

- ***coincLayer***
A required argument specifying the layer name of the layer providing fragmentation.
- ***layerToFrag***
A required argument specifying the layer name of the layer receiving the fragmentation.

Description

The fragcoinc command applies the [coincidentLayer](#) -overlay 1 operation. The fragmentation on layer ***coincLayer*** is copied to layer ***layerToFrag***. If ***layerToFrag*** has no previous fragmentation on it, it is a copy of ***coincLayer***.

There are no limitations on the layer types for the layers involved.

Examples

```
fragcoinc L1 L2
```

fragmentTag

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Re-fragments edge fragments based on tags.

Usage

```
fragmentTag tag [-first {df1 df2 dfn ...} -last {dl1 dl2 dln ...}] [-rem r]
[-unfragment] [-force] [-ripplestyle 0 | 1]
```

Arguments

- **tag**
The name of the tag to be fragmented. Each of the input tag sets are unfragmented concurrently and separately. This functionality is not equivalent to multiple calls to unfragment. It is recommended to unfragment multiple tag sets concurrently whenever performing multiple unfragments for better run-time performance.
- **-first**
A switch specifying the first point to tag. The *dfn* parameters are the distances in microns of each successive fragment vertex from the previous vertex. *df1* is the first distance specified in microns.
- **-last**
A switch specifying the last point to tag. The *dln* parameters are the distances in microns of each successive fragment vertex from the previous vertex. *dl1* is the last distance specified in microns.

Note

 When determining the values of the -first and -last switches, note that polygons are always traversed in a clockwise direction based on the cell origin, not considering the transforms that might occur to instantiations of the cell. For example, a cell with an "L" shape polygon that is mirrored does not have a different order from that in the unmirrored cell.

- **-rem r**
An optional switch specifying the minimum remainder size after fragmentation. The default is equal to [minedgelength](#).
- **-unfragment**
An optional flag instructing the operation to merge consecutive fragments of the given tag name into a single fragment before performing the specified fragmentation. Multiple tag set inputs can be specified when using the -unfragment option, as in the following example:

```
fragmentTag tag1 [tag2 ...] -unfragment
```

When using multiple tags as input to fragmentTag ... -unfragment, each tag is processed by a fragmentTag operation in the order by which the tags are declared. For example:

```
fragmentTag t1 t2 -unfragment
```

is exactly equivalent to

```
fragmentTag t1 -unfragment  
fragmentTag t2 -unfragment
```

Because of this, the order is important. In the following example:

```
fragmentTag t1 t2 -unfragment
```

can have different results from

```
fragmentTag t2 t1 -unfragment
```

if there are fragments that are in both t1 and t2.

To unfragment between multiple tag sets, OR the tags into a single tag set sets prior to using -unfragment.

```
newTag t3 -how orTags t1 t2  
fragmentTag t3 -unfragment
```

- **-force**

An optional switch used to force fragmentation even if it would cause a minedgelength violation.

- **-ripplestyle 0 | 1**

An optional argument used to control ripple generation. Choose one of the following:

0 — All conflicts between fragmentation points are resolved according to the relevant -distancePriority argument. Fragmentation points that were blocked by the -rem constraint do not generate ripples.

1 — Ripples are generated by *all* fragmentation points, whether successfully inserted or blocked by -rem constraints. This is the default.

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 is 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 is 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 their distance from the first point in the edge.

Description

This command re-fragments edge fragments based on tags. When the command is reached during the tagging commands' evaluation, the fragments labeled **tag** are re-fragmented, if possible, based on distances specified by the -first and -last specifiers.

After the command is processed in the running state, it is removed from the tag and the tagging commands are re-evaluated from the beginning. This ensures that previous tagging commands

are applied to any new fragments. You can optimize your run time by setting the order of fragmentTag commands, working on progressively smaller sets of tags (see “Optimizing fragmentTag Execution” for a description). All distances are in units of microns.

Note

 The fragmentTag keyword is not supported by PRINTImage. PRINTImage simulations do not reflect any fragmentTag operations.

Optimizing fragmentTag Execution

All tagging commands are executed in the order presented in the setup file and processed up to the first fragmentTag command encountered. After the fragmentTag is processed, it is removed from an internal copy of the tagging script. The tagging commands are then re-executed, from the beginning, until the next fragmentTag encountered. The fragmentTag command is executed, removed, and the process begins once again. This procedure is repeated until no further fragmentTag commands are encountered in the setup file.

For example, if you had a setup file with the following tagging commands:

```
newTag t1
newTag t2
newTag t3
fragmentTag t1
fragmentTag t2
```

When Calibre is run, the setup file commands are executed in the following order:

Table 6-18. Setup File Command Execution — Case 1

| First Pass | Second Pass | Third Pass |
|---|---|-------------------------------------|
| newTag t1 newTag t2 newTag t3 fragmentTag t1 | newTag t1 newTag t2 newTag t3 fragmentTag t2 | newTag t1 newTag t2 newTag t3 |

Note the repeated execution of newTag commands. You can optimize the performance of your tagging scripts by placing the fragmentTag commands earlier in the sequence. For instance, if the setup file commands were re-arranged in the following order:

```
newTag t1
fragmentTag t1
newTag t2
fragmentTag t2
newTag t3
```

When Calibre is run, the setup file commands are executed in the following order:

Table 6-19. Setup File Command Execution — Case 2

| First Pass | Second Pass | Third Pass |
|-----------------------------|--|-------------------------------------|
| newTag t1 fragmentTag t1 | newTag t1 newTag t2 fragmentTag t2 | newTag t1 newTag t2 newTag t3 |

You can compare how many times the newTag commands are repeated in the two cases:

Table 6-20. Command Execution Comparisons

| Command | Case 1 | Case 2 |
|-----------|--------|--------|
| newTag t1 | 3 | 3 |
| newTag t2 | 3 | 2 |
| newTag t3 | 3 | 1 |

In the second version of the setup file, certain tagging operations are performed less often, resulting in a faster run time.

In general, for best optimized performance for setup files using fragmentTag:

- Place each fragmentTag command as close to the beginning of the tagging script as possible.
- Place all tagging operations that aren't needed by fragmentTag after all tagging operations that are needed as inputs to fragmentTag commands.
- You can also use [LITHO_TAG_TIMER](#) in order to tag the amount of time spent in various tagging operations. However, the LITHO_TAG_TIMER only works in single-CPU mode.

Interaction Distance and Removing Fragmentation

The -unfragment parameter causes tagged, consecutive fragments to be merged into a single fragment. The fragment can be larger than the interaction distance assumed by typical checks: care should be taken in the subsequent tagging operations. It is important that long fragments not be used to propagate information beyond the interaction distance.

One example of this is corner-type interactions. Normally long fragments do not have corners due to intrafeature fragmentation. If long fragments have been created at corners, using built-in tags based on corner type such as [concave_corner](#) may cause problems.

When you create long fragments, assume that they will be split by tile or hierarchy boundaries and that information about neighbors or corners might not be available.

Examples

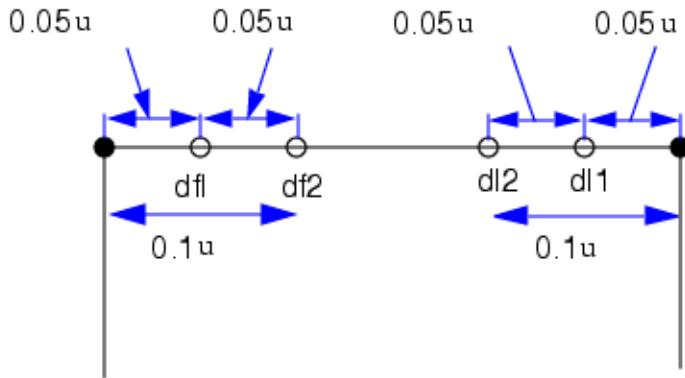
Example 1

In this example, two 50 nanometer fragmentation points are added at the end of all line ends:

```
fragmentTag line_end -first 0.05 0.05 -last 0.05 0.05
```

The results are shown in [Figure 6-58](#).

Figure 6-58. Re-Fragmenting a Line End to Create a Hammerhead



Example 2

This example unfragments everything on the mask:

```
fragmentTag all -unfragment
```

Example 3

This example unfragments those edges that meet specified criteria:

```
newTag t1 -how EDGE ANGLE1 == 90 ANGLE2 == 270 LENGTH <= 0.12 ORDERED
fragmentTag t1 -first 0.04 -last 0.04 -rem 0.01 -unfragment
```

Related Topics

[clearTagScript](#)

[opcTag](#)

[newTag](#)

fragmentTag ... interfeature

Type: [Tagging Commands](#). Used by Calibre OPCpro and Calibre ORC.

Re-fragments interfeature fragmentation by projecting corners from the source(s) onto fragments.

Usage

```
fragmentTag tagName interfeature from src1 {src2 ...}  
  [-interdistance dist] [-shield n]  
  [-ripples n]  
  [-ripplelen len]  
  [-rem rlen]  
  [-distancePriority {0|1}] [-cornerDist w]
```

Arguments

- ***tagName***

Fragments with this tag may be further fragmented.

- **from *src1* {*src2* ...}**

Corners from one or more sources are projected onto the fragments with tag ***tagName***; each source can be a tag or layer name.

Note

 For a description of the remaining arguments, refer to the [interfeature](#) keyword. The default values for any unspecified arguments are taken from the interfeature keyword.

Description

This command performs interfeature fragmentation by projecting corners from the source(s) onto fragments with the tag ***tagName*** which are within the specified interdistance.

After the command is processed in the running state, it is removed from the tag and the tagging commands are re-evaluated from the beginning. This ensures that previous tagging commands are applied to any new fragments. All distances are in units of microns.

Examples

In this example, corners from an external layer MOPC are projected onto the opc layer:

```
fragmentTag all interfeature from MOPC -interdistance 0.5 -shield 1 \  
  -ripples 0 -distancePriority 1
```

Related Topics

[clearTagScript](#)

[opcTag](#)

[newTag](#)

[interfeature](#)

fragmentTag ... intersection

Type: [Tagging Commands](#). Used by Calibre OPCpro and Calibre ORC.

Re-fragments island-layer type fragmentation on fragments meeting certain criteria.

Usage

```
fragmentTag tagName intersection from src1 {src2 ...} [-ripples n] [-ripplelen len]  
[-rem rlen] [-enforce {0 | 1}] [-rippleinside | -rippleboth] [-dontcross | -allowcross]  
[-offset offset_value]
```

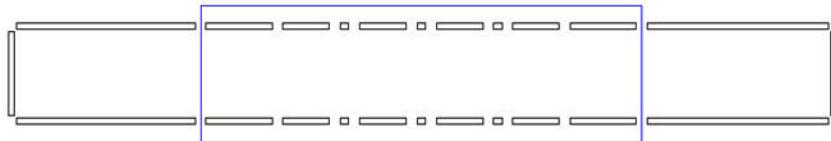
Arguments

- ***tagName***
Fragments with this tag may be further fragmented.
- **from *src1* {*src2* ...}**
Edges from one or more sources are intersected with the fragments with tag ***tagName***; each source can be a tag or layer name.
- **-ripples *n***
An optional argument specifying the count of ripples from each endpoint when the ***tagName*** fragments are coincident with the source edge. Default ripple count is zero; no ripples are created.
- **-ripplelen *len***
An optional argument specifying the length of a single ripple edge in microns. The default for ripplelen is [minedgelength](#).
- **-rem *rlen***
An optional argument specifying the minimum allowed remainder during ripple generation. If not specified, the default value used is [minedgelength](#).
- **-enforce {0 | 1}**
When set to 1, a fragment is broken at intersection points only if no fragments of lengths less than [minedgelength](#) are created. When not present, fragmentation can result in [minedgelength](#) violations. The default is 0.
- **-rippleinside**
Generates ripples toward the inside of the intersection layer polygons at any intersection, not just where coincident edges are present. Either -rippleinside or -rippleboth may be specified, but not both.
- **-rippleboth**
Generates ripples toward both the inside and the outside of the intersection layer polygons at any intersection, not just where coincident edges are present. Either -rippleinside or -rippleboth may be specified, but not both.

- -dontcross

This option prevents ripples from crossing an island layer, or crossing one another. [Figure 6-59](#) shows an example of what happens when this option is not specified.

Figure 6-59. Without the -dontcross Option



In the example above, the island layer shown in blue has fragmented the layer shown in black with these options:

```
fragmentTag all intersection from island_layer -ripples 4 \
    -ripplelen 0.07 -rem 0.0005 -enforce 0 -rippleinside
```

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 6-60](#) shows an example of what happens when this option is specified.

Figure 6-60. With the -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.

- -allowcross

This is the opposite of the -dontcross option. Ripples that appear at the center of an edge are allowed, even if that is undesirable. This is the default behavior.

- -offset offset_value

An optional argument that specifies an offset for all ripples. Normally, an intersection break occurs at exactly the point where the island layer crosses the OPC layer. There are occasions

when it's desirable to have the break offset from the crossing point (this might permit a fragment's site to be placed exactly on the intersection, for example).

This option causes all ripples to be offset from the intersection by specifying an *offset_value* in microns. All produced ripples occur at $(len * n) + offset_value$ locations.

If an offset is specified, the initial break at the intersection does not occur. Instead, the first break occurs at the specified *offset_value* away from the intersection. The second break occurs at $(offset_value + len)$ microns from the intersection, with all further breaks occurring at *len* intervals. This means that the total number of ripples is reduced by one ripple when an *offset_value* is specified versus when an offset is not specified.

Description

This command performs island-layer type fragmentation on the fragments tagged with *tagName*. Refer to the [islandFragment](#) command for a detailed description.

Fragmentation points due to *src1* are inserted first, followed by *src2* and so on.

After the command is processed in the running state, it is removed from the tag and the tagging commands are re-evaluated from the beginning. This ensures that previous tagging commands are applied to any new fragments. All distances are in units of microns.

Incorrect Use Cases

The following examples describe several typical incorrect usage cases for fragmentTag intersection as well as the correct use cases.

Case 1

```
minedgelength 0.09
fragmentTag all intersection from mark -ripples 2 -ripplelen 0.09 \
-enforce 0
```

On the surface, this example appears to generate a remaining fragment of less than 0.09 in length, because “-enforce 0” has been specified. However, careful examination of the behavior of “intersection from” indicates that the default -rem value is minedgelength, or 0.09. This means that the remainder does not violate minedgelength, even though “-enforce 0” was specified. Although the syntax is technically correct, it is still counterintuitive behavior under the specification. A more efficient means to ignore minedgelength under these conditions is to set -rem to 0, as in the following example:

```
minedgelength 0.09
fragmentTag all intersection from mark -ripples 2 -ripplelen 0.09 \
-rem 0.0
```

Case 2

```
minedgelength 0.09
fragmentTag all intersection from mark -ripples 2 -ripplelen 0.08 \
-enforce 0
```

This example appears to be technically correct, but it does not work. The failure is due to the fact that the default -rem is 0.09 (minedgelength), which is greater than the length of the ripples, 0.08. This is an error, because -rem must be greater than or equal to -ripplelen.

Case 3

```
minedgelength 0.09
fragmentTag all intersection from mark -ripples 2 -ripplelen 0.08 \
-enforce 1
```

This example fails because the ripple length is less than minedgelength, and minedgelength enforcement is on.

Case 4

```
minedgelength 0.09
fragmentTag all intersection from mark -ripples 2 -ripplelen 0.08 \
-rem 0.08 -enforce 0
```

This example generates ripples of 0.08 microns in length with a remainder of at least 0.08 microns while ignoring the minedgelength limitation of 0.09 microns.

Examples

Example 1

The following 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 are generated, and no fragments less than minedgelength in length are created.

```
fragmentTag POLY intersection from DIFF -ripples 2 -ripplelen 0.04 \
-enforce 1 -rippleinside
```

The following 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 are generated, two ripples of length 0.04 microns towards the outside of the DIFF layer polygon are generated, and no fragments less than minedgelength in length are created in either direction.

```
fragmentTag POLY intersection from DIFF -ripples 2 -ripplelen 0.04 \
-enforce 1 -rippleboth
```

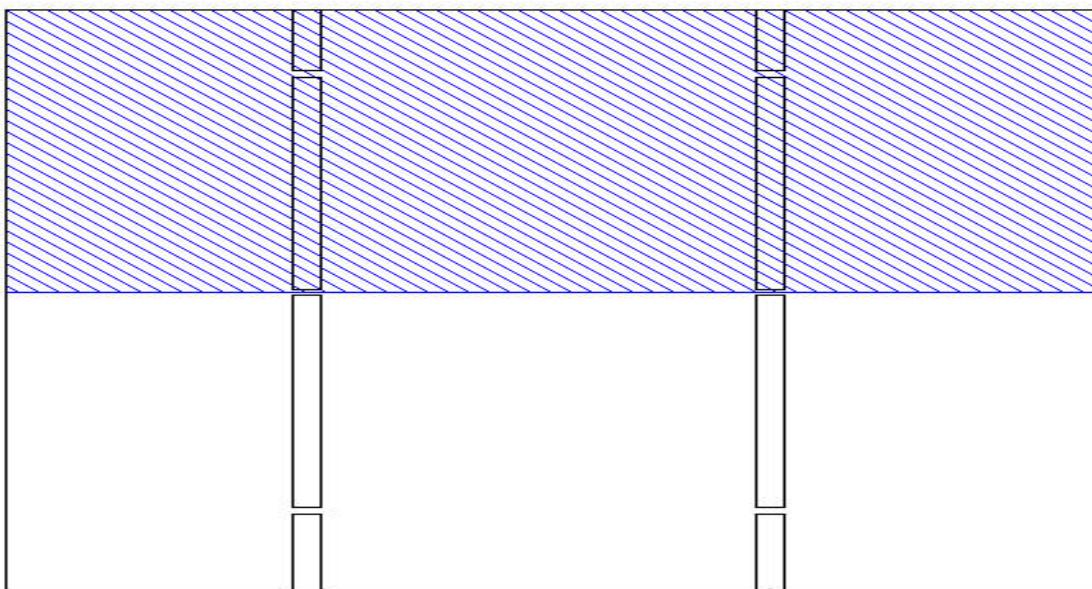
Example 2

This an example of fragmentTag ... intersection from without an offset. [Figure 6-61](#) illustrates the following command:

```
fragmentTag all intersection from island_layer -ripples 1 \
-ripplelen 0.07 -rippleboth
```

The intersection layer is marked by the blue hatched region while the resulting fragment breaks on the OPC layer are shown in black.

Figure 6-61. FragmentTag ... Intersection Without an Offset



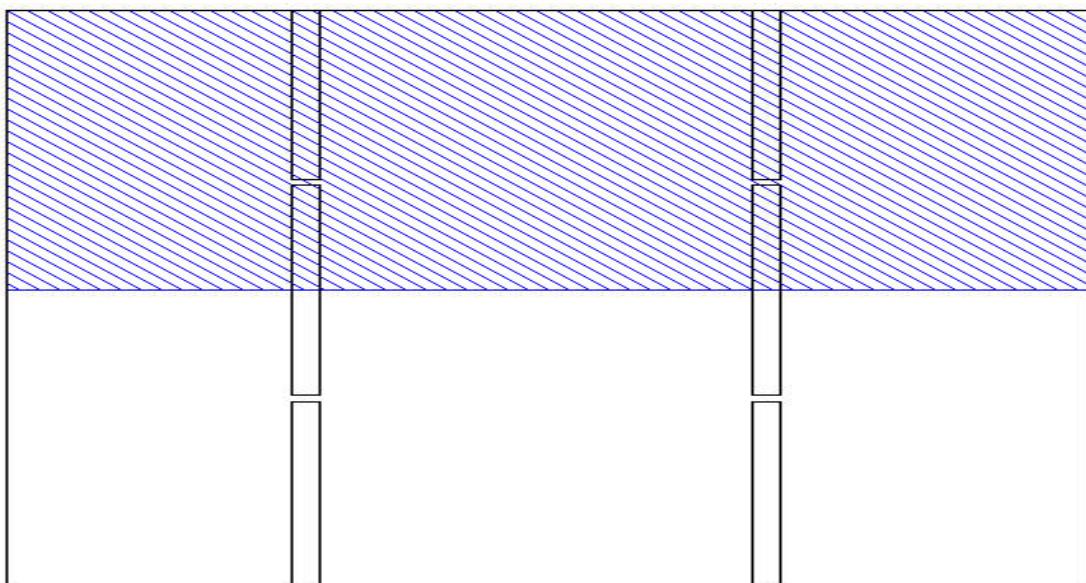
Note that two ripples were created in the OPC layer, one on either side of the intersection layer crossing, and there is a break centered on the crossing.

The next command shows the effect of fragmentTag ... intersection using an offset. It is illustrated in [Figure 6-62](#).

```
fragmentTag all intersection from island_layer -ripples 1 \
    -ripplelen 0.07 -rippleboth -offset 0.035
```

Note that only one ripple was created, and there is no break at the intersection layer crossing. Instead, there is just one break on either side of the intersection layer crossing offset 0.035 microns. A break occurs on both sides of the intersection layer crossing as specified by “-rippleboth”.

Figure 6-62. FragmentTag ... Intersection With an Offset



Related Topics

[clearTagScript](#)

[opcTag](#)

[newTag](#)

[islandFragment](#)

fragmentTag ... maxedgelength

Type: [Tagging Commands](#). Used by Calibre OPCpro and Calibre ORC.

Re-fragments island-layer type fragmentation on fragments meeting certain criteria.

Usage

```
fragmentTag tagName maxedgelength maxedge [-split {0 | 1}]
```

Arguments

- ***tagName***

Fragments with this tag may be further fragmented.

- **maxedgelength *maxedge***

A required argument specifying the maximum length of fragments generated by maxedgelength fragmentation, specified in microns. The value must be greater than or equal to minedgelength.

- **-split {0 | 1}**

0 — The maxedgelength fragmentation algorithm attempts to break all edges longer than $2 * \text{maxedge} + \text{minedgelength}$ into fragments of length ***maxedge***, with a centered remainder fragment with length greater than minedgelength.

1 — The maxedgelength fragmentation algorithm attempts to break all edges longer than $2 * \text{maxedge}$ into edges as long as possible with none of the resulting fragments longer than ***maxedge***, and none shorter than minedgelength.

The default value for split is the value from the maxedgelength keyword.

See the -split {0 | 1} argument to the [maxedgelength](#) command for the detailed description of the impact of this argument.

Description

This command performs maxedgelength fragmentation upon the ***tagName*** fragments.

After the command is processed in the running state, it is removed from the tag and the tagging commands are re-evaluated from the beginning. This ensures that previous tagging commands are applied to any new fragments. All distances are in units of microns.

Note

 FragmentTag ... maxedgelength can potentially propagate fragmentation for an unlimited distance. This creates problems because OPCpro's hierarchical processing is based on a limited interaction distance. It is better to use interfeature and intrafeature fragmentation whenever possible and limit the use of fragmentTag with maxedgelength to only short fragments. Additionally, using maxedgelength as an alternative to interfeature fragmentation is not recommended because fragment points and site locations may shift depending on where the tile boundaries fall.

fragmentTag ... projection

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.

Re-fragments island-layer type fragmentation on fragments meeting certain criteria.

Usage

```
fragmentTag tagName projection from source1 {source2 ...}  
[-interdistance dist] [-shield n] [-distancePriority {0 | 1}]
```

Arguments

- ***tagName***

Fragments with this tag name may be further fragmented.

- **from *source1* {*source2* ...}**

Endpoints of tags from one or more sources are projected onto the fragments with tag ***tagName***; each source can be a tag or layer name.

- **-interdistance *dist***

An optional argument that specifies the maximum distance at which ***source*** endpoints are projected.

The default value for -interdistance is the value from the [interfeature](#) statement outside of any fragmentLayer blocks.

- **-shield *n***

Similar to interfeature command's shield parameter, this optional argument used to ensure that projection 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 fragmentTag projection. The default value for -shield is 0.

- **-distancePriority {0 | 1}**

When this value is set to 1, then the priority insertion of fragmentTag projection points is decided by the distance from the projecting endpoint. When -distancePriority 1 is set, the closer of the two endpoints "wins". When two endpoints have the same distance, the one with a smaller shielding count "wins". With -distancePriority 0, the "winner" depends on the processing order and cannot be determined ahead of time. The default value for -distancePriority is 0.

Description

This command performs fragmentation by projecting endpoints from the source(s) onto fragments with the tag ***tagName*** which are within the specified interdistance.

After the command is processed in the running state, it is removed from the tag script and the tagging commands are re-evaluated from the beginning. This ensures that previous tagging commands are applied to any new fragments. All distances are in units of microns.

Examples

In this example, fragments from the opc layer TARGET are projected onto the correction layers:

```
...
layer 1 TARGET 17 0 opc dark 0 1
layer 2 PHA000 17 0 correction clear 1 1.1
layer 3 PHA180 17 0 correction phase180 1 1.1
layer 4 TRIM    17 0 correction dark 0 1
...
fragmentTag PHA000 projection from TARGET -interdistance 0.160 \
             -distancePriority 1
fragmentTag PHA180 projection from TARGET -interdistance 0.160 \
             -distancePriority 1
fragmentTag TRIM projection from TARGET -interdistance 0.160 \
             -distancePriority 1
...
...
```

Related Topics

[clearTagScript](#)

[newTag](#)

fragmentTag ... split

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Re-fragments island-layer type fragmentation on fragments meeting certain criteria.

Usage

```
fragmentTag tagName split {2|3} [-enforce {0|1}]
```

Arguments

- *tagName*

Fragments with this tag may be further fragmented, depending on their length and the values of the **split** and -enforce parameters.

- **split {2 | 3}**

This required parameter indicates whether fragments with *tagName* should be split evenly into two or three fragments. There is no default value.

- -enforce {0 | 1}

This optional parameter indicates whether [minedgelength](#) should be enforced. When -enforce 1 is specified, fragments must be longer than 2*minedgelength to be split by “fragmentTag split 2”. The default value for -enforce is 0.

Description

This command splits fragments with *tagName* into two or three fragments of equal length. Fragments created by this option must not be shorter than [minedgelength](#) when -enforce 1 is specified.

After the command is processed in the running state, it is removed from the tag script and the tagging commands are re-evaluated from the beginning. This ensures that previous tagging commands are applied to any new fragments.

Examples

```
fragmentTag bottoms split 3 -enforce 1
// split fragments at the bottoms of U shapes into thirds.
```

Related Topics

[clearTagScript](#)

[newTag](#)

histogram

Type: [Tagging Commands](#). Used by Calibre OPCpro and Calibre ORC.

Counts the number of edge fragments with the given tags in the form of a histogram.

Usage

```
histogram histogram_name -tags TAG1 TAG2 ... -bins BIN1 BIN2 ...  
[-percent] [-EPE_stats] [-flat_count]
```

Arguments

- ***histogram_name***
A required string containing the histogram's name.
- **-tags *TAG1 TAG2 ...***
A required list of the names of tags for which to compile statistics. The ordered list of tag names must match the ordered list of bins. You must supply at least one tag name.
- **-bins *BIN1 BIN2 ...***
A required list of the bin names for which the compiled statistics. The ordered list of bin names must match the ordered list of tag names. You must supply at least one bin name.
- **-percent**
An optional argument specifying to print the percentage of the total histogram covered by each bin.
- **-EPE_stats**
An optional argument specifying to print the mean and standard deviation of objects in the histogram.
- **-flat_count**
An optional argument that provides an approximate flat count for the histogram when working with cells. When this is specified, if a fragment belongs to a cell the histogram tag counter adds 1 * total cell placement count. The default behavior is for a fragment to only be counted once no matter how often it is repeated.

Description

This command counts the number of edge fragments with the given tags and stores that data in the form of a histogram. The arguments are pairs of bin names and tags. The number of tags and bins is limited to 4096.

Use [LITHO_HISTOGRAM_OUTPUT_FILE](#) to define the name of the output file for the histogram data. If no file is defined, the output is written in the session transcript.

Examples

In this example, the line ends are counted using the following commands:

```
clearTagScript
newTag badEPE0 -how EPE all -5 -1
newTag badEPE1 -how EPE all -1 -0.01
newTag badEPE3 -how EPE all 0.01 1
newTag badEPE4 -how EPE all 1 5

histogram MY_HIST -tags badEPE0 badEPE1 badEPE3 badEPE4 \
    -bins b1 b2 b3 b4
histogram LINE_ENDS -tags convex_corner concave_corner line_end\
    -bins cv cc le
```

A sample histogram output from this example is illustrated below.

```
oooooooooooooooooooooooooooooooooooooooooooo
>>> Histograms created for SVRF command <<<
oooooooooooooooooooooooooooooooooooooooooooo
Histogram name: LINE_ENDS
cv    1001
cc    506
le    155 Reference
Histogram name: MY_HIST
b1    0
b2    827
b3    518
b4    0
oooooooooooooooooooooooooooooooooooo
```

You can compute histograms for any tag. For instance, you can compute EPE histograms using several EPE tags.

Related Topics

[clearTagScript](#)

[newTag](#)

[Pre-Defined Tags](#)

jogsmooth

Type: [Tagging Commands](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTImage.
Makes fragments collinear to smooth jogs.

Usage

[exec] **jogsmooth layer tag max_length max_displacement**

Note

 The “exec” portion of the command is optional and does not explicitly need to be specified in the setup file for Calibre versions 2010.2 and later.

Arguments

- **layer**

A required argument that specifies the name of the OPC layer on which to smooth jogs.

- **tag**

A required argument that specifies the tag name of fragments to smooth. Use “all” to smooth all fragments.

Since all of the fragments that are smoothed must be contiguous, it is best to specify all of the tags to be smoothed at one time and not to use multiple calls to jogsmooth. Multiple tag sets should be combined before calling jogsmooth.

- **max_length**

A required argument that specifies the total length of a span of fragments that can be smoothed and not the length of an individual fragment. If there is a span of contiguous fragments longer than this length, jogsmooth only considers the first set of fragments that lie within the **max_length** limit for smoothing. Setting **max_length** to 1000 microns forces jogsmooth to consider all lengths of fragments in your tag set.

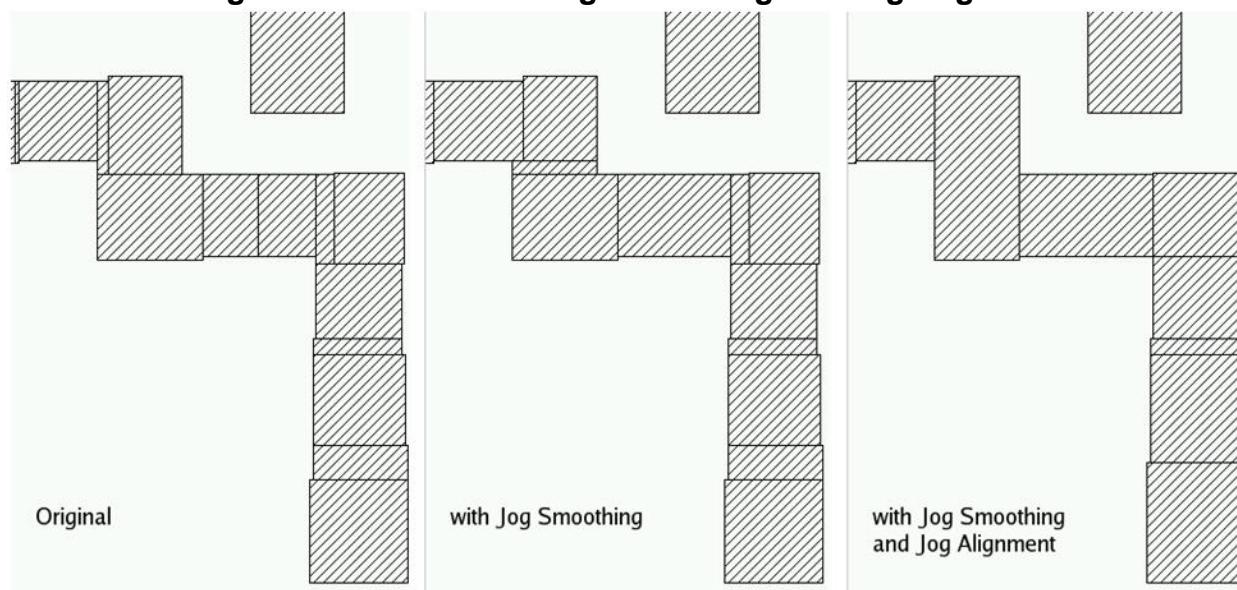
- **max_displacement**

The maximum displacement over which smoothing can occur. For two or more fragments to be smoothed into one edge, the maximum displacement from one fragment to another fragment must be less than or equal to this value.

Description

The jogsmooth command smooths out jogs if they fall within the maximum length and maximum displacement limit values. The jog must have fragments in the **tag** set on both sides. Smoothed jogs are two or more fragments that have biases corrected so that they are collinear and appear as one edge before fracturing. [Figure 6-63](#) shows the effects of jog smoothing and LITHO MASKOPT (jog alignment).

Figure 6-63. Effects of Jog Smoothing and Jog Alignment



Jog smoothing can be used with LITHO MASKOPT in Calibre mask data preparation to reduce the mask shot count in mask writers. Reducing shot count can save mask writers time and help reduce the cost of making masks. For more information on using MASKOPT, see the [Calibre Mask Data Preparation User's and Reference Manual](#).

Using **jogsmooth**

Jog smoothing operates on the specified tag set in the layer used for invocation. It begins by searching for fragments in the specified tag set with measurements smaller than the **max_length** value. All of the fragments in the group have their offsets adjusted so that the edges are aligned. Fragments are not moved more than half the **max_displacement** value from their current location.

It is best to start out with small amounts of jog smoothing to see what impact it has on image quality. Use Calibre PRINTimage or Calibre OPCverify to check the image quality results.

Tagging should be used to control which fragments receive jog smoothing. Setting **max_length** to a large value (1000 microns) ensures that jogsmooth looks at all lengths of fragments. Starting values for **max_displacement** are shown in [Table 6-21](#). These suggested values should be tested and fine tuned before using in your own processes.

Table 6-21. Starting Values for max_displacement

| Technology node | Suggested max_displacement start value |
|-----------------|---|
| 180 or 130 nm | 0.002 um |
| 90 nm | 0.0015 um |
| 65 nm | 0.001 um |
| 45 nm | <= 0.001 um |

The smallest usable amount for **max_displacement** is a single OPC step size. Appropriate values of **max_displacement** are between one and two OPC step sizes.

Tagging is the best way to control jog smoothing. Using the settings in [Table 6-21](#) with “all” tags is a good starting point for initial test runs. Regions that have received too much smoothing can be identified by pinching or bridging problems, and should be removed from the tag set as needed.

Jog smoothing should be applied only to low-MEEF and less critical areas of a design. In addition, [DEANGLE](#) should be used before calling jogsmooth to remove non-Manhattan edges. The setup file grid size and database unit setting of the layout file must match before jogsmooth is successfully called.

Examples

This example finds fragments on the OPC layer which are on lines greater than 65 nm wide and farther than 65 nm away from other features. All fragments which meet both of these requirements are smoothed, as long as the maximum displacement in a group of fragments is not greater than 1 nm (the 1000.0 micron length is effectively infinity). This approach uses less CPU time than trying to smooth twice and is more effective at identifying long fragment spans to smooth.

```
newTag wideLayer -how INTERNAL > 0.065 OPPOSITE INPUT1 opc_layer all
newTag wideGap      -how EXTERNAL > 0.065 OPPOSITE INPUT1 opc_layer all
newTag smoothTags -how andTags wideLayer wideGap
jogsmooth opc_layer smoothTags 1000.0 0.001
```

MATRIX_OPC

Type: [Tagging Commands](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.
Runs Matrix OPC, a feature for multiple mask OPC.

Usage

```
MATRIX_OPC {'  
    METHOD {CLOSEST | AVERAGE n | NON_MATRIX}  
    SEARCH dist  
    METRIC { MIDPOINT| CLOSEPOINT |  
        WEIGHTED_EUCLIDEAN w |  
        WEIGHTED_PROJECTING w |  
        EUCLIDEAN| PROJECTING }  
    [PARALLEL_PROJ_MULTIPLIER w] [PARALLEL_ONLY {YES | NO}]  
    [MAP_PRIORITY layer tag priority]  
    [MAP_DISTANCE_MULTIPLIER layer tag mult]  
    MEEF_MODE {thresh | MODEL [NORECYCLE |  
        RECYCLE INITIAL_THRESHOLD {t | REFERENCE_THRESHOLD}] }  
    DELTA delta  
    [DISTANCE_ADDITION value]  
    [OPTSLOPE {YES | NO}]  
    [SLOPESTEPSIZE dist]  
    [MINSLOPE val] [slopeToleranceTag tag low_slope [high_slope]]  
}'
```

Arguments

- ‘{’ and ‘}’

Braces must surround all keywords that apply solely to Matrix OPC. In addition,

- The opening brace must be on the same line as the MATRIX_OPC keyword.
- ONLY those keywords that apply solely to Matrix OPC can be included within the braces.
- Matrix OPC keywords cannot be on the same line as the braces.

- **METHOD** {CLOSEST| AVERAGE *n* | NON_MATRIX}

A required argument defining the calculation for the MEEF matrix. Options are:

CLOSEST — Each movable fragment is controlled by control at most ONE site, whose EPE drives its movement. The site chosen is the closest one, using the defined METRIC.

AVERAGE *n* — Picks the *n* closest sites to each movable fragment and makes the EPE at all *n* of those sites responsible for corrections to the movable fragment. The *n* closest sites are determined using the METRIC of choice.

NON_MATRIX — Used with binary masks (no correction layers are present) to prohibit corrections. When specified, the MATRIX_OPC operation only calculates the MEEF for fragments, which can then be used for MEEF tagging.

- **SEARCH *dist***

A required argument defining the maximum search distance, in microns, when mapping fragments on the correction layer to a fragment on the opc (target) layer. Note here that a fragment is said to be within this search distance if and only if the actual (non-weighted) distance from midpoint to the closest point on the fragment is less than *dist*.

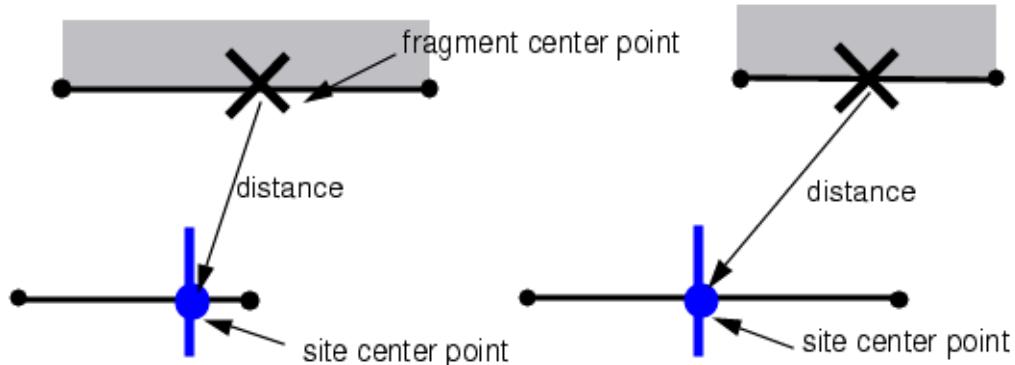
- **METRIC {WEIGHTED_EUCLIDEAN *w* | WEIGHTED_PROJECTING *w* | MIDPOINT | EUCLIDEAN | CLOSEPOINT | PROJECTING}**

A required argument defining which edges to include in the mapping and how the distance between a site on the target layer and a movable fragment on the correction layer is to be measured.

You define how distances are measured and how they are weighted by specifying one of the following measurement styles:

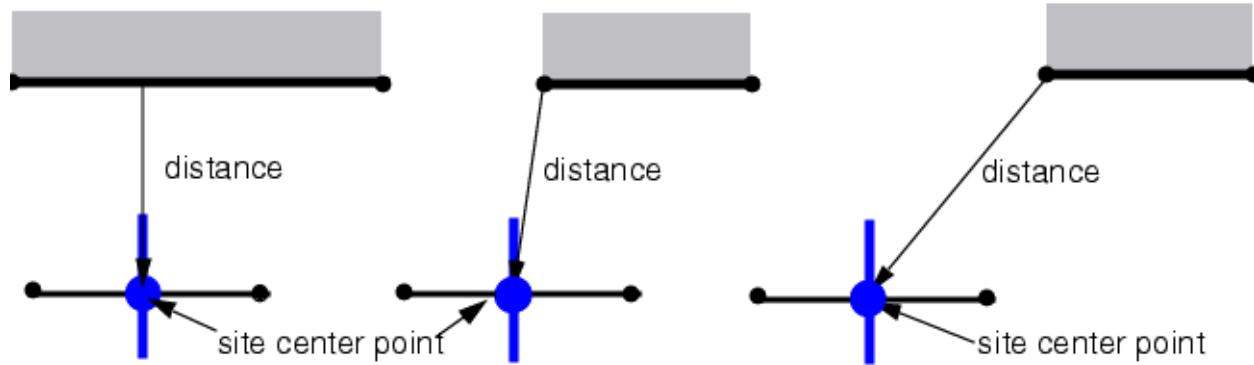
MIDPOINT(alternatively **EUCLIDEAN**) — Calculates the distance as the distance from the site center point to the midpoint of the movable fragment.

Figure 6-64. How Distance is Measured Using MIDPOINT



CLOSEPOINT (alternatively **PROJECTING**) — Calculates the distance as the distance between the site center point and the closest point on the fragment, as shown. In addition, it applies a multiplier of 0.01 to all projecting edges.

Figure 6-65. How Distance is Measured Using CLOSEPOINT



Note here that the built in weighting of 0.01 makes this **METRIC** behave as though fragments are only factored into the mapping if they actually project onto the target layer fragment containing the site. In reality, this metric does allow non-projecting edges to be mapped, but only if there are no projecting edges within the SEARCH dist.

To qualify as projecting, the fragment which is mapping and the fragment containing the site which is mapped to, must be parallel and have overlapping spans, as shown in Figure 6-66.

WEIGHTED_EUCLIDEAN w — Measures the distance from the site center point to the movable fragment midpoint, and discounts the distance for parallel projecting edges by the weight factor, w .

Note here that all the same edges are factored into the mapping as when using **EUCLIDEAN**. However, the distance ranking of fragments is affected by the weighting factor, w . If you have 4 potential site fragments to map to, from a moving fragment, suppose they are as follows:

```
e1    distance = 0.09 projecting status = 1
e2    distance = 0.1   projecting status = 1
e3    distance = 0.2   projecting status = 0
e4    distance = 0.25  projecting status = 1
```

For **METHOD AVERAGE 3**, the fragment maps only to three sites.

With **EUCLIDEAN**, the 3 winners are e1,e2,e3.

With **WEIGHTED_EUCLIDEAN 0.5**, the three winners are e1, e2, e4.

That is because the effective distance of e4 becomes $0.25 * 0.5 = 0.125$ and the effective distance of e3 is not modified due to the fact that it is not projecting.

WEIGHTED_PROJECTING w — Measures the distance from the site center point to the closest point on the movable fragment, and discounts the distance for parallel projecting edges by the weight factor, w .

- PARALLEL_PROJ_MULTIPLIER

An optional argument that allows you to give parallel projecting edges preferred treatment relative to other edges by discounting the distance for parallel projecting edges. [Figure 6-66](#) shows what it means for an edge to be projecting.

Discounting refers to calculating an effective distance by multiplying the actual distance by a reduction factor.

For parallel projecting edges:

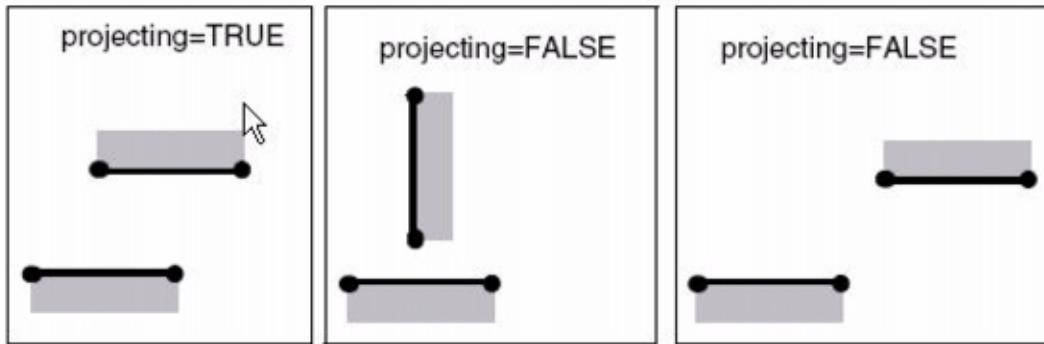
```
effective distance = distance * multiplier
```

For all other edges:

```
effective distance = distance
```

The default value is 1, which means that parallel projecting edges do not receive special treatment.

Figure 6-66. Projecting



For example, assume you are using METHOD AVERAGE 3 and from one moving fragment have four potential site fragments to map to. Suppose they are as shown:

Table 6-22. Potential Mappings

| | | |
|--------|------------------------|-----------------------|
| site 1 | actual distance = 0.09 | projecting status = 1 |
| site 2 | actual distance = 0.1 | projecting status = 1 |
| site 3 | actual distance = 0.2 | projecting status = 0 |
| site 4 | actual distance = 0.25 | projecting status = 1 |

When PARALLEL_PROJ_MULTIPLIER is used, the distances you compare are effective distances, not actual distances. You calculate the effective distance by multiplying the actual distance by the multiplier, if the projecting status is 1. Assume a PARALLEL_PROJ_MULTIPLIER of 0.5.

Table 6-23. Distances

| | | | |
|--------|------------------------|-----------------------|----------------------------|
| site 1 | actual distance = 0.09 | projecting status = 1 | effective distance = 0.045 |
|--------|------------------------|-----------------------|----------------------------|

Table 6-23. Distances (cont.)

| | | | |
|--------|------------------------|-----------------------|----------------------------|
| site 2 | actual distance = 0.1 | projecting status = 1 | effective distance = 0.05 |
| site 3 | actual distance = 0.2 | projecting status = 0 | effective distance = 0.2 |
| site 4 | actual distance = 0.25 | projecting status = 1 | effective distance = 0.125 |

Without PARALLEL_PROJ_MULTIPLIER, the 3 winners would be sites 1, 2, and 3.

With PARALLEL_PROJ_MULTIPLIER 0.5, the three winners are sites 1, 2, and 4. That is, the effective distance of site 4 becomes $0.25 * 0.5 = 0.125$ and the effective distance of site 3 is not modified due to the fact that it is not projecting.

- **PARALLEL_ONLY {YES | NO}**

Instructs the application to ignore all non-parallel edges when mapping fragments on a correction layer to sites on the target layer. Allowed values are YES or NO.

- **MAP_PRIORITY *layer tag priority***

An optional argument that assigns a priority to a mapping between a fragment on the specified correction layer and a site on the target layer. The highest priority mapping identifies the fragment to be moved to improve the EPE.

layer — The correction layer the containing movable fragments to which this priority applies.

tag — The name of the tag that identifies the fragments on the opc (target) layer that contain the sites to which this priority applies.

priority — The priority to assign to this mapping. A value ≥ 0 sets the priority of the mapping. The higher priority wins. The priority can be raised or lowered with respect to the base value of 10 for specific fragment layer-tag combinations.

A priority of -1 blocks any mapping from happening.

- **MAP_DISTANCE_MULTIPLIER *layer tag mult***

An optional argument that defines a value by which to multiply the measurement computed by the METRIC of choice in order to favor or impede certain mappings.

layer — The correction layer the containing movable fragments to which this multiplier applies.

tag — The name of the tag that identifies the fragments on the opc (target) layer that contain the sites to which this multiplier applies.

mult — The multiplier factor.

- **MEEF_MODE *thresh* | {**MODEL** [NORECYCLE | RECYCLE INITIAL_THRESHOLD {*t* | REFERENCE_THRESHOLD}]}{}**

A required argument defining how the EPE is evaluated when calculating the MEEF, using the formula dEPE/dEdge.

- When you provide a threshold, the MEEF is calculated using a constant threshold of the aerial image. In this case, the VT5-determined threshold is ignored.

- When you specify **MODEL**, the MEEF is calculated using the threshold predicted by the resist model (CTR or VT5).

When using the **MODEL** to predict the threshold to use in MEEF calculations, if the model is type VT5 that uses densities, you must also supply the RECYCLE or NORECYCLE keyword.

- NORECYCLE** — This causes the simulation to require 2X more performance. *It is very slow.* That is because the simulation is done twice to ensure valid densities on each simulation pass.
- RECYCLE INITIAL_THRESHOLD {t | REFERENCE_THRESHOLD}** — Each MEEF simulation pass uses the density values which were computed on the previous simulation pass. The first simulation pass of MEEF uses a threshold value to calculate MEEF, specified by the INITIAL_THRESHOLD keyword. INITIAL_THRESHOLD defines how to calculate EPE when the VT5 model cannot be evaluated due to densities not being valid at the time.

- DELTA delta**

A required argument used in calculating the MEEF. The **delta**, specified in microns, can be a positive or a negative number. A negative value corresponds to shifting the fragment inwards into the polygon on the correction layer, and a positive value corresponds to shifting the fragment outwards from the polygon.

$$MEEF = \frac{\delta EPE}{\delta \text{delta}}$$

Note that this is different from the standard definition of MEEF which is:

$$MEEF_{lithography} = \frac{\delta CD_{wafer}}{\delta CD_{mask}}$$

You can define a value for **delta** that is different than the MEEF delta used in the [newTag ... -how IMAGE](#) command. For a complete description of how **DELTA** affects movement, refer to “[How Matrix OPC Calculates Edge Movement](#)” on page 421.

- DISTANCE_ADDITION value**

An optional argument defining a distance, in microns, to use in weight calculation for the AVERAGE method.

For the average method, the movement is computed by the following method:

$$\text{desired_displacement} = \frac{\sum_N (W_i \cdot \text{FEEDBACK}_i \cdot EPE_i)}{\sum_N W_i}$$

where N is the number of sites averaged, and W is a weight which is determined as follows:

$$W = \frac{1.0}{(D + \text{DISTANCE_ADDITION})}$$

- **OPTSLOPE {YES | NO}**

An optional argument that activates “full-image OPC”, in which case the image slope and EPE are both optimized. (See Examples following or “[Example 2: Matrix OPC Followed by Calibre ORC](#)” on page 582.) The default is NO.

- **SLOPESTEPSIZE *dist***

An optional argument used only when OPTSLOPE is set to YES.

This argument defines the amount to move a fragment in the direction of the gradient when the minimum slope is violated, causing slope to be optimized with a higher priority than EPE. When this condition is met, each iteration only moves by one SLOPESTEPSIZE at a time in order to improve slope. Default = 5 nm.

- **MINSLOPE *val***

An optional argument used most often when OPTSLOPE is set to YES.

This argument defines a global minimum slope to allow. This causes the slope to be optimized instead of EPE when the slope is below this cutoff. If set to 0, then slope is always secondary to EPE. Default = 0.

If OPTSLOPE is NO, you can supply a non-zero value for MINSLOPE to force Matrix OPC to perform slope optimization until the minimum slope is achieved.

- **slopeToleranceTag *tag* *low_slope* [*high_slope*]**

An optional argument used only when OPTSLOPE is set to YES. It defines the slope tolerance range for fragments with the specific tag; in other words, when you are willing to sacrifice EPE correction for slope improvement. This argument defines a range of acceptable image intensity slopes for the tagged fragments.

Fragments that match the specified tag use the *low_slope* value instead of MINSLOPE for determining when to optimize slope at a higher priority than EPE. The *high_slope* value lets the algorithm know that slopes above the tolerance do not need to be optimized further.

You can define specific slope tolerance ranges for as many tags as needed by including this argument multiple times. If multiple slope tolerance ranges apply to a fragment, then the first slopeToleranceTag that matches is used. If no slope tolerance range applies to the fragment, then the MINSLOPE value is used.

tag — The tag identifying the set of fragments to which this slope tolerance range applies.

low_slope — Lower bound on slope tolerance range.

high_slope — Upper bound on slope tolerance range. Default = infinity.

After evaluating the slope and the EPE for the fragment, the application moves the edge as follows:

- If $\text{slope} < \text{low_slope}$, always shift the edge in the direction that increases the slope by a distance of SLOPESTEPSIZE.
- If $\text{slope} > \text{high_slope}$ and EPE is outside the tolerance window, shift the edge in the direction that improves the EPE.
- If $\text{low_slope} < \text{slope} < \text{high_slope}$ and EPE is inside the tolerance window, shift the edge in the direction that increases the slope by a distance of SLOPESTEPSIZE.
- If $\text{slope} > \text{high_slope}$ and EPE is inside the tolerance window, do not shift the edge.

Figure 6-67. OPC Corrections Related to Parameter Space

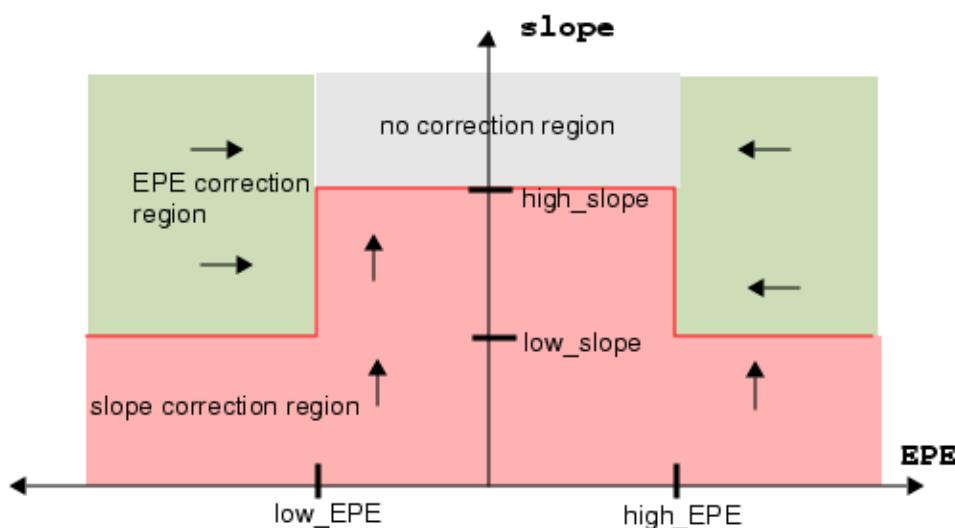


Figure 6-67 provides a graphical explanation of the rules described above. The arrows indicate the desired improvement (EPE or slope) in the parameter space as a function of current location in the space.

Description

MATRIX_OPC supports advanced fragment mapping between correction layers and opc layers. It supports complex functionality that allows basic double patterning or simple PSM masks. Note that it is no longer the recommended tool for these purposes, as Siemens EDA has developed more advanced tools since the MATRIX_OPC command was added.

How Weighting Impacts Mapping

You can use weightings to discount distances for certain pairings. Discounting distances works by multiplying the measured distance by a constant to calculate the effective distance.

```
effective_distance = measured_distance * weighting
```

To decide which sites are closest to a specific fragment, Matrix OPC compares effective distances, not measured distances. You control the weighting through the choice of METRIC:

- MIDPOINT (EUCLIDEAN) — No discounting while measuring to the midpoint.
- WEIGHTED_EUCLIDEAN w — Discounts parallel projecting edges by the multiplier w while measuring to the midpoint.
- CLOSEPOINT (PROJECTING) — Discounts parallel projecting fragments by a multiplier of 0.01 while measuring to the closest point. This is equivalent to WEIGHTED_PROJECTING 0.01.
- WEIGHTED_PROJECTING w — Discounts parallel projecting edges by the multiplier w while measuring to the closest point.

Table 6-24. Discounting Distances for Parallel Projecting Edges

| To measure to... | And discount Parallel Projecting Edges by... | Use... |
|--|--|--|
| the midpoint of the correction layer fragment | no discounting | MIDPOINT or WEIGHTED_EUCLIDEAN 1 |
| the closest point on the correction layer fragment | no discounting | WEIGHTED_PROJECTING 1 |
| the closest point on the correction layer fragment | 0.01 | CLOSEPOINT or WEIGHTED_PROJECTING 0.01 |
| the midpoint of the correction layer fragment | 0.01 | WEIGHTED_EUCLIDEAN 0.01 |
| the closest point on the correction layer fragment | 0.5 | WEIGHTED_PROJECTING 0.5 |

How Priority Impacts Mappings

By default, pairings between all fragments and all sites are of equal priority. You can define some pairings to have a higher priority using the MAP_PRIORITY keyword, which assigns priorities on a tag-by-tag basis. You must create these tags with tagging commands that appear before the Matrix OPC block.

When some fragment-to-site pairings are higher priority than others, Matrix OPC evaluates the highest priority pairings first, and only looks at lower priority pairings if it has not identified enough pairings to satisfy the method.

Example 1: Assume METHOD CLOSEST

If there is one highest priority pairing within the search distance, that pairing is used as the official mapping, even if there are closer non-prioritized pairings.

If there are several highest priority pairings within the search distance, the closest is used as the official mapping.

Example 2: Assume METHOD AVERAGE 4

If there are four highest priority pairings within the search distance, all four are used in the mapping, regardless of how many lower priority pairings are within the search distance and regardless of how close those other pairings are.

If there are three highest priority pairings within the search distance, these three are used in the mapping. The fourth pairing in the mapping is the closest of the lower priority pairings.

How Matrix OPC Compares Equally Close Fragments

When two sites are equally close to a fragment, Matrix OPC resolves the conflict by comparing a secondary measurement. The secondary measurement also varies according to the METRIC you have selected:

Table 6-25. Primary and Secondary Distance Comparisons

| Metric | Primary Measurement | Secondary Measurement |
|------------|-------------------------------|--|
| MIDPOINT | measures to the midpoint | measures to the closest point |
| CLOSEPOINT | measures to the closest point | calculates an expanded distance as: $a^2 + b^2$ where: a = distance to one endpoint of fragment b = distance to the other endpoint of fragment |

How Matrix OPC Calculates Edge Movement

The following describes the algorithm Matrix OPC uses for calculating edge movement:

1. Simulate at sites, compute EPE.
2. Check to see if EPE is within all tolerances defined for the fragment:
 - o global tolerance: $|EPE| < OPC_EPE_TOLERANCE_FRAC * stepsize$
 - o EPE satisfies all epeToleranceTag statements defined for the fragment

If not, move the fragment using the method specified by the chosen OPC style ([Table 6-26](#)).

Table 6-26. Edge Movements

| OPC Style | Calculation |
|---------------------------------|--|
| Matrix OPC using METHOD AVERAGE | <p>If the EPE is not within any one of the defined tolerances, the desired displacement is set to according to the formula:</p> $\text{desired_displacement} = \frac{\sum_{N} (W_i \cdot \text{FEEDBACK}_i \cdot EPE_i)}{\sum_{N} W_i}$ <p>where N is the number of sites averaged, and W is a weight which is determined as follows:</p> $W = \frac{1.0}{(D + \text{DISTANCE_ADDITION})}$ <p>where:</p> <ul style="list-style-type: none"> • D = distance from the site center point to the edge that is moving. Note that this is the undiscounted distance between the center point of the site and the closest point on the fragment. • FEEDBACK = is determined for each site using the method described previously for the CLOSEST method. • DISTANCE_ADDITION = a user supplied value in microns, default 0.0. |
| Matrix OPC using OPTSLOPE | <p>If the application determines that the goal of edge movement is to improve the slope, then displacement is a movement equal to one SLOPESTEP SIZE in the direction of the gradient.</p> <p>If the application determines that the goal of edge movement is to improve the EPE, then it computes the displacement using the method described previously for the specified METHOD.</p> |

Table 6-26. Edge Movements (cont.)

| OPC Style | Calculation |
|---------------------------------|--|
| Matrix OPC using METHOD CLOSEST | <p>If the EPE is not within any one of the defined tolerances, the desired displacement is set to according to the formula:</p> $T1 = -\frac{1}{MEEF}$ <p>if ($T1 < OPC_FEEDBACK$) then FEEDBACK = $T1^1$ else if ($MEEF < 0$), FEEDBACK = $-OPC_FEEDBACK$ else FEEDBACK = $OPC_FEEDBACK$</p> <p>Then compute the desired displacement using the new (temporary) feedback:</p> $\text{desired_displacement} = \text{EPE} * \text{FEEDBACK}$ |
| Standard OPC (non-matrix) | <p>If the EPE is not within any one of the defined tolerances, the desired displacement is set to according to the formula:</p> $\text{desired_displacement} = \text{EPE} * \text{OPC_FEEDBACK}$ |

1. $|T1|$ is rarely less than $|OPC_FEEDBACK|$, so in most cases the MEEF is only used to calculate the direction of the edge movement. While it is possible to increase the value of $OPC_FEEDBACK$ to force Matrix OPC to use the $|T1|$ as the FEEDBACK, this is not recommended.

Low-MEEF Tagging Precedence

In the event that multiple low-MEEF tags are assigned to the same fragment, the movement value for a low-MEEF fragment is chosen in this order of precedence:

1. positive tag
2. freeze tag
3. negative tag
4. default (no tag at all)

For example, if a low-MEEF fragment were to have all three tags present, the positive movement value would be used. If only the freeze tag and negative tag were present, freeze would take precedence. If no low-MEEF handling tag were found at all, the default value would be used.

Examples

In this example, we look at the performance of matrix-OPC with slope optimization on a section of layout and also use ORC to analyze its effectiveness in improving slope, compared to standard OPC.

Lithography settings:

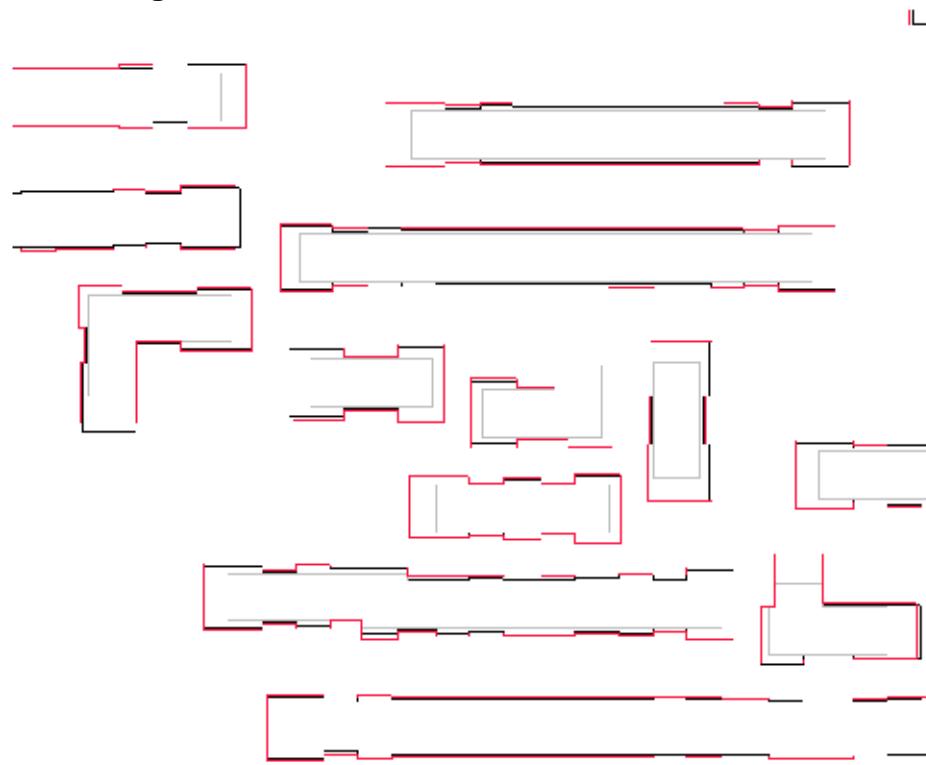
```
mask: attenuated 6% background, clear features
illumination: STANDARD, sigma = 0.75
lambda=193, NA = 0.7
```

OPC settings:

```
epeToleranceTag line_end      0.005 -0.010
epeToleranceTag line_end_adjacent 0.010 -0.005
epeToleranceTag all           0.005 -0.005
MATRIX_OPC {
    SEARCH 0.1
    OPTSLOPE YES
    MINSLOPE 3.0
    slopeToleranceTag line_end 2.7
    slopeToleranceTag line_end_adjacent 2.7
}
```

Figure 6-68 shows the results of Matrix OPC (black), superimposed with the standard OPC (red) and the wafer target (gray). The slope-optimized OPC output has slightly less correction on line ends in order to improve slope. Also, the slope-optimized OPC output has slightly wider corrections on isolated edges, which results in higher image intensity hence better slope.

Figure 6-68. Standard OPC Versus Matrix OPC



Related Topics

[epeToleranceTag](#)

[Using Matrix OPC in Calibre OPCpro](#)

newTag

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Creates new, user-defined tags using the methods described in subsequent reference pages.

Usage

newTag *tagName* -how *method arguments*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- **-how *method arguments***

A required secondary keyword followed by one of the tagging keyword as described in the following pages of this section. The **-how *method*** describes the conditions a fragment must meet to be included in the output tag set.

Most **-how** method keywords have multiple options, switches, and parameters. As there is no standard set, each method keyword is described in its own reference topic. See Related Topics for a full list of ***method*** choices.

Description

The newTag command instructs the application to create a new tag containing all the fragments that meet the conditions described by the **-how *method***.

Caution

 Tag names cannot be reused. The same tag name in multiple newTag commands results in a fatal setup file error.

Examples

Example 1

```
newTag dense -how external >0.001 <=0.5 INPUT1 opc_layer
newTag iso -how subtractTags all dense
fragmentTag iso
```

This example defines two new tag groups, “dense” and “iso”, and then refragments the fragments in the iso group. The iso group applies to all fragments that are not in dense.

Example 2

The following example uses newTag commands to find “bad” EPEs that are within a certain distance of line ends. The fragments with this tag are then frozen.

```
-----Arbitrary Commands Can Follow This Line. Don't delete this line! -----
newTag badEPE -how EPE all -0.002 -0.002 -not
newTag endcap -how edge -TYPE1 CONVEX -TYPE2 CONVEX -LENGTH < 0.3
newTag exceptions -how EXTERNAL >-0.3 < 0.9 \
    INPUT1 all endcap INPUT2 all badEPE
opcTag exceptions -freeze
```

Related Topics

[newTag ... -how ANDTAGS](#)
[newTag ... -how COINCIDENTEDGE](#)
[newTag ... -how COINCIDENTINSIDEEDGE](#)
[newTag ... -how COINCIDENTOUTSIDEEDGE](#)
[newTag ... -how DISPLACEMENT](#)
[newTag ... -how EDGE](#)
[newTag ... -how EPE](#)
[newTag ... -how EPESENSITIVITY](#)
[newTag ... -how EXTERNAL](#)
[newTag ... -how fragAfterEdge](#)
[newTag ... -how fragAfterFrag](#)
[newTag ... -how fragBeforeEdge](#)
[newTag ... -how fragBeforeFrag](#)
[newTag ... -how FRAGMENT](#)
[newTag ... -how fragNextToEdge](#)
[newTag ... -how fragNextToFrag](#)
[newTag ... -how HAS_SITE](#)
[newTag ... -how IMAGE](#)
[newTag ... -how INSIDEEDGE](#)
[newTag ... -how INTERNAL](#)
[newTag ... -how LAYERAND](#)
[newTag ... -how LAYERANDNOT](#)
[newTag ... -how loopWithFrag](#)
[newTag ... -how MRC](#)
[newTag ... -how NOTCOINCIDENTEDGE](#)
[newTag ... -how NOTCOINCIDENTINSIDEEDGE](#)
[newTag ... -how NOTCOINCIDENTOUTSIDEEDGE](#)

newTag ... -how NOTINSIDEEDGE
newTag ... -how NOTOUTSIDEEDGE
newTag ... -how NOTTOUCHEDGE
newTag ... -how NOTTOUCHINSIDEEDGE
newTag ... -how NOTTOUCHOUTSIDEEDGE
newTag ... -how ORTAGS
newTag ... -how OUTSIDEEDGE
newTag ... -how SEQUENCE
newTag ... -how SUBTRACTTAGS
newTag ... -how TOUCHEDGE
newTag ... -how TOUCHINSIDEEDGE
newTag ... -how TOUCHOUTSIDEEDGE

newTag ... -how ANDTAGS

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Creates a new tag *tagName* containing everything that is tagged with *tag1* AND *tag2* AND ... *tagN*. This identifies the intersection of two or more sets of fragments.

Usage

```
newTag tagName -how ANDTAGS tag1 tag2 [tagN ...]
```

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***tag1***
A required argument specifying the name of the first tag being evaluated.
- ***tag2***
A required argument specifying the name of the second tag being evaluated.
- ***tagN***
Optional arguments specifying the names of additional tags to be evaluated.

Examples

```
newTag 45DegLE -how andTags line_end angle_45_line
```

The example finds all 45-degree line ends by selecting fragments that are in both the [line_end](#) group and the [angle_45_line](#) group. (These two groups are predefined.)

Related Topics

[Pre-Defined Tags](#)

[newTag ... -how ORTAGS](#)

newTag ... -how COINCIDENTEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on a source layer that are coincident with a marker layer.

Usage

`newTag tagName -how COINCIDENTEDGE markerLayer srcLayer [-tag filterTag]`

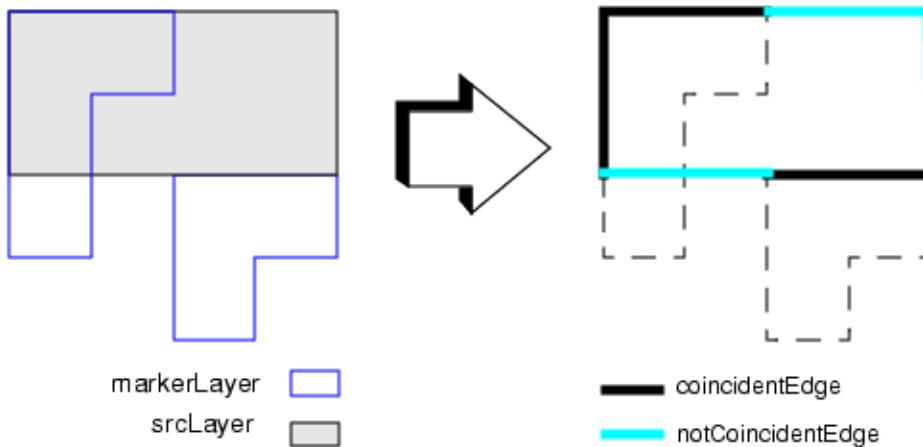
Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***markerLayer***
A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.
- ***srcLayer***
A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.
- **-tag *filterTag***
An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation [COINCIDENT EDGE](#).

Figure 6-69. Coincident Edges



Examples

```
newTag calibre_line_end -how coincidentEdge end_cap POLY
```

This example creates a new tag group, `calibre_line_end`, from fragments on the `POLY` layer that are on the boundaries of the shapes in the `end_cap` layer.

Related Topics

[newTag ... -how COINCIDENTINSIDEEDGE](#)

[newTag ... -how COINCIDENTOUTSIDEEDGE](#)

[newTag ... -how NOTCOINCIDENTEDGE](#)

newTag ... -how COINCIDENTINSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on the source layer that are inside coincident with a marker layer.

Usage

```
newTag tagName -how COINCIDENTINSIDEEDGE markerLayer srcLayer  
[-tag filterTag]
```

Arguments

- *tagName*

A required argument specifying the name for the new tag.

- *markerLayer*

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- *srcLayer*

A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.

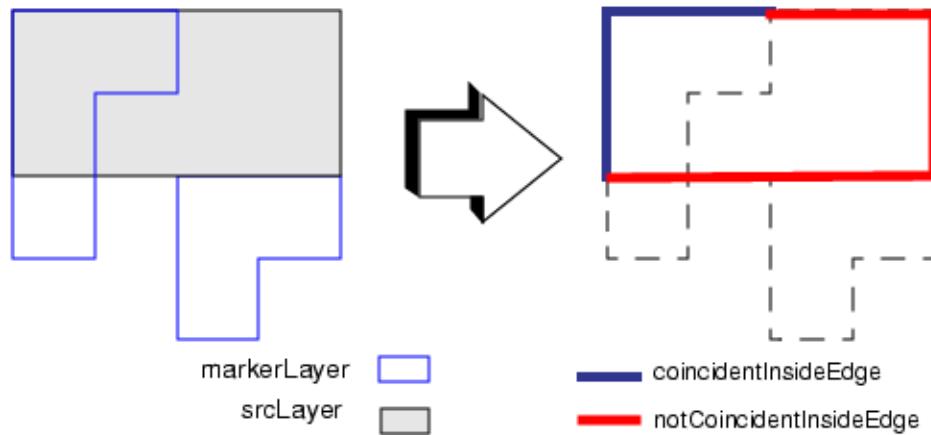
- -tag *filterTag*

An optional argument specifying filtering for a subset of the fragments in the *markerLayer* to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation [COINCIDENT INSIDE EDGE](#).

Figure 6-70. Coincident Inside Edge



Related Topics

[newTag ... -how NOTCOINCIDENTINSIDEEDGE](#)

[newTag ... -how COINCIDENTEDGE](#)

[newTag ... -how COINCIDENTOUTSIDEEDGE](#)

newTag ... -how COINCIDENTOUTSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on the source layer that are outside coincident with the marker layer.

Usage

```
newTag tagName -how COINCIDENTOUTSIDEEDGE markerLayer srcLayer [-tag filterTag]
```

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***markerLayer***

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- ***srcLayer***

A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.

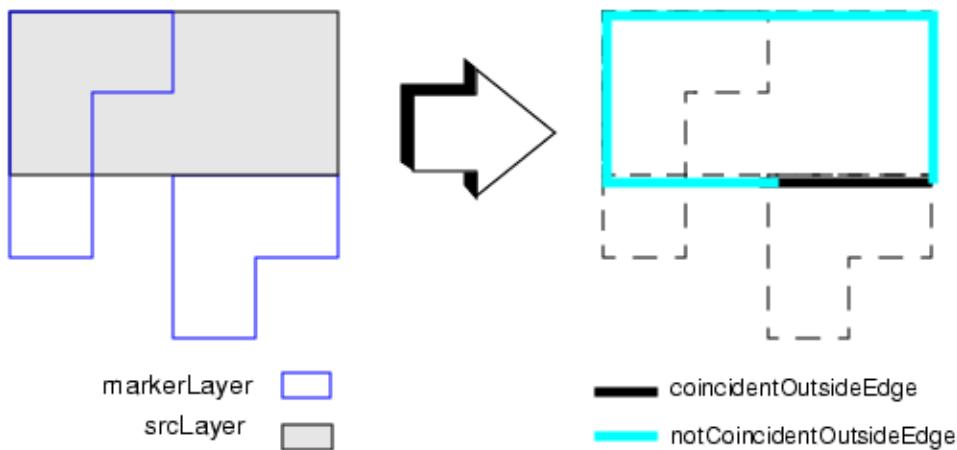
- **-tag *filterTag***

An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation [COINCIDENT OUTSIDE EDGE](#) as shown in [Figure 6-71](#) following.

Figure 6-71. Coincident Outside Edge



Related Topics

[newTag ... -how COINCIDENTEDGE](#)
[newTag ... -how COINCIDENTINSIDEEDGE](#)
[newTag ... -how NOTCOINCIDENTOUTSIDEEDGE](#)

newTag ... -how DISPLACEMENT

Type: [Tagging Commands](#). Used by Calibre ORC.

Creates a new tag after model-based OPC based on how far the fragment moved (was displaced) during correction.

Usage

newTag *tagName* -how DISPLACEMENT *filterTag* *disp_constraint*

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***filterTag***
A required argument specifying the tag name for the set of fragments for examining displacement.
- ***disp_constraint***
A required argument specifying constraints for the displacement in microns.

Description

This newTag command tags fragments after model-based OPC based on how far the fragment moved (was displaced) during correction. One use for this tagging method is to create a histogram showing the distribution of OPC correction amounts. You must use this tag in conjunction with the [fastIter](#) tag after the iterations are complete.

Examples

The following example finds sites with displacement with a magnitude smaller than 0.020 um and put them into tag t.

```
newTag t -how DISPLACEMENT all >-0.02 <0.02
```

Related Topics

[fastIter](#)

[Pre-Defined Tags](#)

newTag ... -how EDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on an edge that meets constraints. There are two mutually-exclusive tagging modes: FROMCORNER; or TYPE, LENGTH and ANGLE.

Usage

FROMCORNER Syntax:

```
newTag tagName -how EDGE [filterTag] FROMCORNER D
```

TYPE, ANGLE, and LENGTH Syntax:

```
newTag tagName -how EDGE [[ONEWITH | ALLWITH] filterTag]
    [TYPE1 type1 | ANGLE1 angle1_constraint] [TYPE2 type2 | ANGLE2 angle2_constraint]
    [LENGTH len_constraint] [LENGTH1 len1_constraint] [LENGTH2 len2_constraint]
    [WITHJOG jog_constraint] [WITHJOG1 jog_constraint1] [WITHJOG2 jog_constraint2]
    [ORDERED]
```

Arguments

- *tagName*

A required argument specifying the name for the new tag.

- *filterTag*

An optional argument specifying the tag name for the set of fragments from which distances are measured. If *filterTag* is not specified, then all fragments are tested.

FROMCORNER Mode

- **FROMCORNER *D***

A required argument that tags all fragments within a certain distance from a corner. You must specify the distance constraint, *D*, as follows:

D — A distance constraint without a lower bound, as shown in [Table 6-27](#):

Table 6-27. Constraints Without Lower Bounds

| Constraint Notation | Represents |
|---------------------|------------|
| $< a$ | $x < a$ |
| $\leq a$ | $x \leq a$ |

Note

 The **FROMCORNER** option does not allow a lower bounded constraint. For example, the following statement is incorrect:

```
newTag tag1 -how edge FROMCORNER > 0.1 < 0.3
```

TYPE, ANGLE and LENGTH Mode

- ONEWITH

An optional argument that selects any edge with at least one fragment matching the named *filterTag*. If you use ONEWITH you must also use *filterTag*. Cannot be used with ALLWITH, WITHJOG, WITHJOG1, or WITHJOG2. The default (neither ALLWITH or ONEWITH is specified) is to only tag edges in which the first fragment on the edge is in the filterTag group.

- ALLWITH

An optional argument that selects any edge with all fragments matching the named *filterTag*. If you use ALLWITH you must also use *filterTag*. Cannot be used with ONEWITH, WITHJOG, WITHJOG1, or WITHJOG2. The default (neither ALLWITH or ONEWITH is specified) is to only tag edges in which the first fragment on the edge is in the filterTag group.

- TYPE1 *type1*

An argument that stipulates the first edge corner's type. Note that you can specify either the TYPE1 or the ANGLE1, but not both. The default value is "TYPE1 ANY".

Valid values for type1 are:

ANY

CONCAVE

CONVEX

- ANGLE1 *angle1_constraint*

An argument that defines the constraint for the first edge corner's angle. The value of angle1_constraint must be a constraint within the 0 to 360 range. Note that you can specify either the TYPE1 or the ANGLE1, but not both.

- TYPE2 *type2*

An argument that stipulates the second edge corner's type. Note that you can specify either the TYPE2 or the ANGLE2, but not both. The default value is "TYPE2 ANY".

Valid values for type2 are:

ANY

CONCAVE

CONVEX

- ANGLE2 *angle2_constraint*

An argument that defines the constraint for the first edge corner's angle. The value of angle2_constraint must be a constraint within the 0 to 360 range. Note that you can specify either the TYPE2 or the ANGLE2, but not both.

- LENGTH *len_constraint*

An argument that tags all fragments of a certain length. You must specify the length constraint, *len_constraint*, using one of the keywords (operators) shown in [Table 6-28](#).

Table 6-28. LENGTH Constraints

| Constraint Notation |
|------------------------|
| < a |
| <= a |
| == a |
| != a |
| > a |
| >= a |
| > a < b or < b > a |
| >= a < b or < b >= a |
| > a <= b |
| >= a <= b or <= b >= a |

The following are LENGTH constraint examples:

```
LENGTH >= 0.1 < 0.2
LENGTH == 1.0
LENGTH != 1.0
LENGTH < 0.4
```

- LENGTH1 *len1_constraint*

An optional argument that defines length constraints that adjacent edges must meet in order for an edge to be tagged. When ORDERED is specified, it applies only to the first adjacent edge, which is defined as the edge counter-clockwise from the edge under consideration. (When ORDERED is not specified, it applies to either adjacent edge.) The constraint can contain inequalities using one of the six keywords (operators) shown in [Table 6-28](#).

- LENGTH2 *len2_constraint*

An optional argument that defines length constraints that adjacent edges must meet in order for an edge to be tagged. When ORDERED is specified, it applies only to the second adjacent edge, which is defined as the edge clockwise from the edge under consideration. (When ORDERED is not specified, it applies to either adjacent edge and cannot be specified without also specifying LENGTH1.) The constraint can contain inequalities using one of the six keywords (operators) shown in [Table 6-28](#).

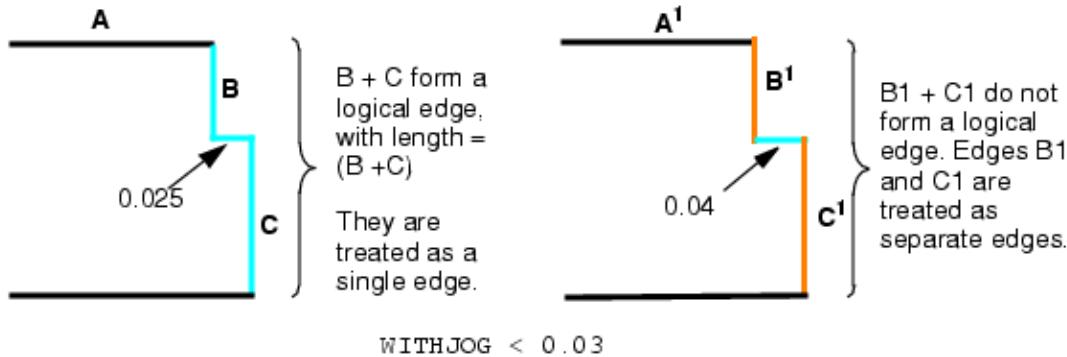
- WITHJOG *jog_constraint*

An optional argument used to permit tagging of edges containing jogs. The value *jog_constraint* defines the maximum length of the jog segment. When used, the method tags

fragments that meet the TYPE or ANGLE and LENGTH constraints plus the fragments comprising any logical edge meets the TYPE or ANGLE and LENGTH constraints.

The jog_constraint must be a qualifier plus a length, and the qualifier must either be “`<`” or “`<=`”. In other words, the size qualifiers cannot use “greater than.”

Figure 6-72. Edges and Logical Edges



- **WITHJOG1 jog_constraint1**

Controls how the length of the first adjacent edge is to be measured. If the first adjacent edge meets the TYPE1 or ANGLE1 constraint, then its length is checked to see if it meets the LENGTH1 constraint. By default, the length is measured from vertex to vertex. When WITHJOG1 is set, the length is calculated by measuring the edge plus any edges to which it is connected by a jog that meets *jog_constraint1*.

For example, in [Figure 6-72](#), if WITHJOG1 is set to < 0.03 , the length of the edge adjacent to A is $(B+C)$, while the length of the edge adjacent to A1 is B1, because the jog between B1 and C1 does not meet the *jog_constraint1*.

The size qualifier *jog_constraint1* must be a qualifier plus a length, and the qualifier must either be “`<`” or “`<=`”. In other words, the size qualifiers cannot use “greater than”.

- **WITHJOG2 jog_constraint2**

Controls how the length of the first adjacent edge is to be measured. If the first adjacent edge meets the TYPE2 or ANGLE2 constraint, then its length is checked to see if it meets the LENGTH2 constraint. By default, the length is measured from vertex to vertex. When WITHJOG2 is set, the length is calculated by measuring the edge plus any edges to which it is connected by a jog that meets *jog_constraint2*.

The size qualifier *jog_constraint2* must be a qualifier plus a length, and the qualifier must either be “`<`” or “`<=`”. In other words, the size qualifiers cannot use “greater than”.

- **ORDERED**

Forces the EDGE tagging command to consider orientation when tagging:

- When you use ORDERED, ANGLE1, TYPE1, LENGTH1, and WITHJOG1 apply to the first endpoint encountered when traversing the polygon clockwise.

- When you do not use ORDERED, the EDGE tags all fragments such that one end meets the ANGLE1, TYPE1, LENGTH1, and WITHJOG1 constraints and the other end meets the ANGLE2, TYPE2, LENGTH2, and WITHJOG2 constraints.

For example, the following code re-fragments any jogs which are less than 0.12 in length, putting the fragmentation point 0.05 away from the 90 degree corner specified by ANGLE1.

```
newTag t1 -how EDGE ANGLE1 == 90 ANGLE2 == 270 LENGTH <= 0.12 ORDERED
newTag t2 -how EDGE ANGLE2 == 90 ANGLE1 == 270 LENGTH <= 0.12 ORDERED
fragmentTag t1 -first 0.05 -rem 0.05 -unfragment
fragmentTag t2 -last 0.05 -rem 0.05 -unfragment
```

Description

The EDGE command provides two mutually-exclusive tagging modes:

- FROMCORNER (tags fragments based on proximity to a corner)
- TYPE or ANGLE and LENGTH (tags fragments based on angle or length)

It tags all fragments in an edge when the entire edge meets the constraints. Note that if the FROMCORNER option is not used, then the command is analogous to the [CONVEX EDGE](#) command in SVRF.

An EDGE tagging operation checks whether the operations require an interaction distance greater than or equal to what is set by the program automatically, based on the size of the optical radius. The interaction distance is calculated as:

```
0.5 * hoodpix + (numsites - sitecenter - 1) * sampleSpacing
```

If the operation requires a larger interaction range, then the tools generate an error during the setup file compilation stage. You can disable this check by using the following environment variable:

```
sse LITHO_SUPPRESS_INTERACTION_ERROR 1
```

Disabling this error is not recommended.

Tip

i Best Practice: During length-based tagging, set an upper bound on length checks to keep the tags out of invalid regions. The upper bound value should be less than or equal to the interaction distance. Allowing tags to apply to fragments longer than the interaction distance can cause false tags, which gives bad OPC results.

FROMCORNER Mode

You specify FROMCORNER mode by including the FROMCORNER option in the command. In this mode, the method tags all fragments within the specified range of distances from a corner, using the *filterTag* as seed, or, if no *filterTag* is used, using all corners.

For most other newTag methods, the results are a subset of the *filterTag*. This is not the case with the FROMCORNER mode of the EDGE command. In FROMCORNER mode the *filterTag* defines the fragments against which other edges are measured. The results include all fragments within the FROMCORNER specified range of distances of any corners in the *filterTag* set.

You can filter the results of the FROMCORNER operation, limiting them to subset of a specific tag, by following the EDGE command with a [newTag ... -how ANDTAGS](#) operation as illustrated in the example that follows. In this example the tag tle is used in the EDGE command as the *filterTag* (or seed) from which distances are measured and is then used with ANDTAGS to weed out all results that are not also in tle.

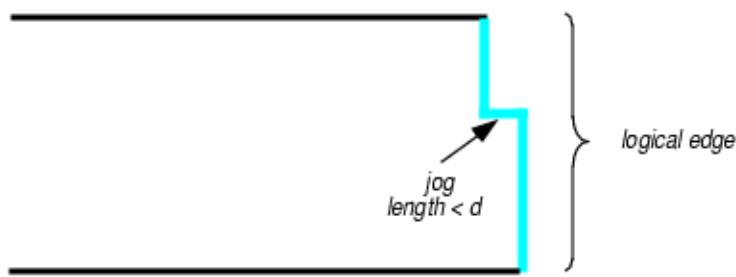
```
newTag tag1 -how edge tle FROMCORNER < 0.12
newTag tag2 -how andTags tag1 tle
```

TYPE, ANGLE, and LENGTH Mode

You specify TYPE, ANGLE, and LENGTH mode by omitting the FROMCORNER option and including the TYPE and LENGTH or ANGLE and LENGTH options in the command. This method tags fragments based on edge type, length, and angle. When using the TYPE, LENGTH, and ANGLE mode, TYPE and ANGLE are mutually-exclusive.

Note that here the length refers to either the edge itself or to a “logical edge,” which is formed by consecutive parallel edges separated by jogs. The jog is defined as an edge that meets the size qualifier *d* and has two ninety-degree corners, one convex and one concave. Each logical edge is then treated as if it were a single edge. If a logical edge matches the edge constraints, then all fragments in the logical edge are tagged.

Figure 6-73. A Logical Edge



Examples

Example 1: FROMCORNER

Using the FROMCORNER mode to tag all edges up to 0.2 microns from all corners:

```
newTag cc -how edge FROMCORNER <=0.2
```

Tagging all edges up to 0.2 microns from convex corners:

```
newTag cc -how edge convex FROMCORNER <=0.2
```

Example 2: TYPE, LENGTH, and ANGLE

Using the TYPE, LENGTH, and ANGLE mode to tag a variety of different elements:

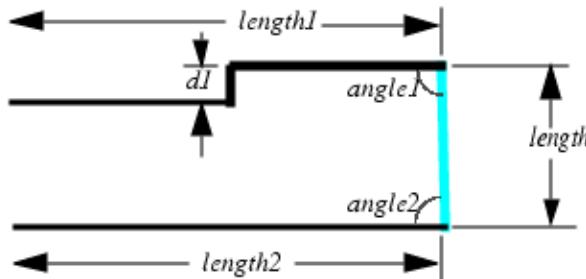
```
newTag le -how edge TYPE1 CONVEX TYPE2 CONVEX LENGTH1 \
    >0.1 LENGTH2 >0.1 LENGTH <0.3
newTag se -how edge TYPE1 CONCAVE TYPE2 CONCAVE \
    LENGTH1 >0.1 LENGTH2 >0.1 LENGTH <0.3
newTag jog -how edge TYPE1 CONVEX TYPE2 CONCAVE \
    LENGTH1 >0.1 LENGTH2 >0.1 LENGTH <0.3
newTag cx -how subtractTags convex_corner le lea jog jogadj
newTag cc -how subtractTags concave_corner se sea jog jogadj
```

Example 3: Contrasting WITHJOG and WITHJOG1

Tagging all edges that are less than 0.24 microns, if they also form a 90 degree angle with both adjacent edges and if the adjacent edges are longer than 0.24 microns. The first adjacent edge may contain a jog, as long as the jog is less than or equal to 0.08 microns and the length of the logical edge formed is still less than 0.24 microns. The blue fragment is output to t1.

```
newTag t1 -how edge LENGTH < 0.24 LENGTH1 >0.24 LENGTH2 >0.24 \
    ANGLE1 == 90 ANGLE2 == 90 WITHJOG1 <= 0.08
```

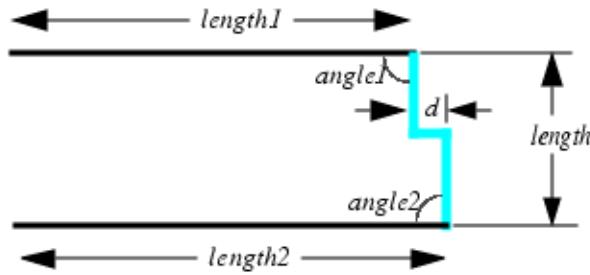
Figure 6-74. WITHJOG1 Tagging



To also tag all logical edges, use WITHJOG instead of WITHJOG1. The tagged edge may contain a jog, as long as the jog is less than or equal to 0.08 microns. The blue fragments show a logical edge; each of the blue fragments are output to t1.

```
newTag t1 -how edge LENGTH < 0.24 LENGTH1 >0.24 LENGTH2 >0.24 \
    ANGLE1 == 90 ANGLE2 == 90 WITHJOG <= 0.08
```

Figure 6-75. WITHJOG Tagging

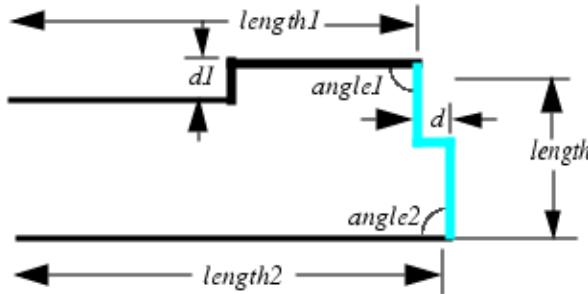


Example 4: Using WITHJOG and WITHJOG1 Together

Using WITHJOG and WITHJOG1 together. This example tags all edges and logical edges that are less than 0.24 microns, if they also form a 90 degree angle with both adjacent edges and if the adjacent edges are longer than 0.24 microns. The tagged edge may contain a jog, as long as the jog is less than or equal to 0.08 microns. The first adjacent edge may contain a jog, as long as the jog is less than or equal to 0.08 microns and the length of the logical edge formed is still less than 0.24 microns. The blue fragments are output to t1.

```
newTag t1 -how edge LENGTH < 0.24 LENGTH1 >0.24 LENGTH2 >0.24
ANGLE1 ==90 ANGLE2 == 90 WITHJOG <= 0.08 WITHJOG1 <=0.08
```

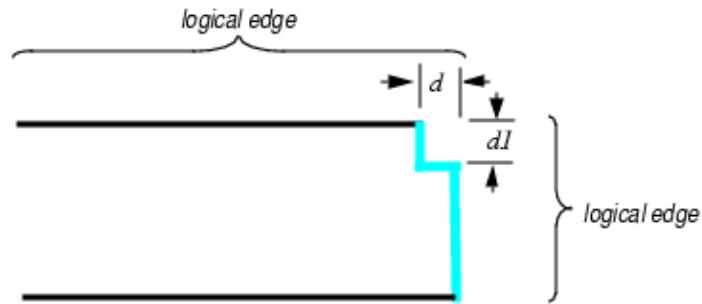
Figure 6-76. WITHJOG and WITHJOG1 Tagging



A special case with two adjacent jogs at a corner. In this case, two logical edges are formed: one is the first adjacent edge (top horizontal), the other is the tagged edge (blue). Note how both jogs belong to both logical edges.

```
newTag t1 -how edge LENGTH < 0.24 LENGTH1 >0.24 LENGTH2 >0.24
ANGLE1 ==90 ANGLE2 == 90 WITHJOG <= 0.08 WITHJOG1 <=0.08
```

Figure 6-77. Adjacent Jogs



Related Topics

[Pre-Defined Tags](#)

[newTag ... -how SEQUENCE](#)

newTag ... -how EPE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments with an Edge Placement Error (EPE) within a desired range.

Usage

```
newTag tagName -how EPE filterTag lowEPE highEPE [NOT]  
[YVAL thresh]
```

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***filterTag***
A required argument specifying the tag name for the set of fragments for which edge placement errors (EPE) are to be computed and examined.
- ***lowEPE***
A required argument specifying the minimum EPE for ***tagName*** in microns.
- ***highEPE***
A required argument specifying the maximum EPE for ***tagName*** in microns.
- NOT
An optional argument that causes the fragments that have defined EPEs lying outside the range to be tagged. That is, edges are tagged if EPE is less than ***lowEPE*** OR EPE is greater than or equal to ***highEPE***.
- **YVAL *thresh***
An optional switch that lets you specify the threshold at which to calculate the EPE. If this argument is not supplied, the threshold from the model is used.

Description

This newTag method tags all fragments with an Edge Placement Error (EPE) within a desired range. That is, edges are tagged if ***lowEPE*** <= EPE < ***highEPE***. Tagging fragments based on their placement error is one of the most useful tagging methods available. Of the many uses for this feature, the most common is to turn off model based OPC for fragments with small but acceptable edge placement errors.

Examples

Example 1

Tag gates with bad EPE, outside the range -0.005 micron to 0.005 micron.

```
newTag tag1 -how EPE gate -0.005 0.005 NOT
```

Example 2

The following commands turn off model-based OPC for all fragments with an edge placement error between -0.02 and 0.02 um:

```
newTag smallEPE -how EPE all -0.02 0.02
opcTag smallEPE -fixedOffset 0
```

Positive edge placement errors mean an edge has moved outward making a polygon larger.
Negative edge placement errors mean an edge has moved inward making a polygon smaller.

Tip

 When calculating edge placement error, this method takes into account all previous fixed offsets, so this method can be used to test the accuracy of rule-based OPC.

Example 3

The following series of commands turn on model-based OPC for all line ends that still have an EPE greater than -10 nm after the rule-based fixed offset has been applied¹. The commands begin by identifying those line ends to receive the fixed offset, apply the offset, then return all line ends that still have an EPE greater than -10 nm. The final command turns on model-based OPC for the results.

```
newTag shortLE -how EPE line_end -100 -0.010
opcTag shortLE -fixedOffset 0.020
newTag stillShortLE -how EPE shortLE -100 -0.010
opcTag stillShortLE
```

In the above example the `line_end` tag is used as a *filterTag* in the first newTag command and the `shortLE` tag is used as a *filterTag* in the second newTag command. The *filterTag* argument must be the name of any currently existing tag (user defined or built-in). Only fragments tagged with *filterTag* are examined when executing the tagging for the EPE method.

Example 4

The following commands first identify all line ends with an EPE of less than -0.010 um. Next, all of these line ends are given a rule-based fixed offset of +0.020 um. All line ends that have an EPE less than -0.010 microns (after the rule-based fixed offset) are then tagged for OPC. Finally, OPC is performed, and any line ends that still have an EPE less than -0.010 um have a box created around them on layer 100.

1. Because EPEs are distances, a smaller negative number implies a larger inward EPE. Thus the values between -100 and -0.01 are smaller than -0.01, but the EPEs with those values are greater than an EPE of -0.01.

```
newTag shortLE -how EPE line end -100 -0.010
opcTag shortLE -fixedOffset 0.020
newTag stillShortLE -how EPE shortLE -100 -0.010
opcTag stillShortLE
newTag shortAfterOPC -how EPE stillShortLE -100 -0.010
fastIter shortAfterOPC 8
tags2boxes -tags shortAfterOPC -layers 100
```

This usually occurs for fragments that are very small jogs. Small jogs are typically not given sites as the data is not critical to the simulation. As a result, the “everything” tag created by the following command is not necessarily equivalent to the “all” tag:

```
newTag everything -how EPE all -100 100
```

The fragments without sites can be identified with the following two commands:

```
newTag withSites -how EPE all -100 100
newTag withoutSites -how subtractTags all withSites
```

Related Topics

[Pre-Defined Tags](#)

newTag ... -how EPESENSITIVITY

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Checks the EPE tolerances as a function of varying the threshold of the resist model.

Usage

newTag *tagName* -how EPESENSITIVITY *filterTag* RANGE *lower upper model*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***filterTag***

A required argument specifying the tag name for the set of fragments to be examined.

- **RANGE *lower upper***

A required argument defining the range of sensitivity values that is of interest. The command tags all edges whose sensitivity falls in the interval between ***lower*** and ***upper***.

lower — The lower bound EPE sensitivity rang, expressed in microns. This value must be ≥ 0 and $< \text{upper}$.

upper — The upper bound of the EPE sensitivity range, expressed in microns. This value must be $> \text{lower}$ and ≤ 2 .

- ***model***

A required argument set specifying the resist model. You must specify either variable (VTR) or constant (CTR) threshold model in the following form:

VTR *decrease increase*

The VTR form is for fractional changes for the threshold that would be found using the current resist model. The ***decrease*** and ***increase*** values are fractional; that is, the two new thresholds would be computed as

- threshold1 = Tmodel * (1 - ***decrease***)
- threshold2 = Tmodel * (1 + ***increase***)

CTR *t1 t2*

The CTR form is for explicit thresholds. The ***t1*** and ***t2*** arguments are the actual thresholds to be used.

Description

The EPE sensitivity tagging method lets you identify fragments that have a sensitivity to perturbation in the resist model. Sensitivity is defined as the difference between two EPEs computed using different resist model thresholds. Sensitivity is a measure of process robustness and how well the structures print over process variations.

The sensitivity is computed as:

$$S = | \text{EPE}(\text{threshold1}) - \text{EPE}(\text{threshold2}) |$$

Any edge fragment whose sensitivity falls in the interval (*lower, upper*) specified by the RANGE argument is tagged.

You specify perturbations as changes to the threshold used to determine edge printing from resist profiles. Threshold perturbations are specified with the VTR or CTR form.

Examples

This example tags all edges which have a sensitivity range between 0.03 microns and 10.0 microns for a 10% variation of the VTR threshold result.

```
newTag T1 -how epeSensitivity all -VTR 0.10 0.10 -RANGE 0.03 0.10
```

Related Topics

[Pre-Defined Tags](#)

newTag ... -how EXTERNAL

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags fragments based on the DRC External dimension check.

Usage

```
newTag tagName -how EXTERNAL constraint [metric] [FULL]  
[[NOT] PROJECTING [constraint]]  
INPUT1 layer1 [tag1]  
[INPUT2 layer2 [tag2]]
```

Arguments

- ***tagName***

A required argument specifying the name for the new tag, which is a subset of tag1.

- ***constraint***

A required argument specifying the distance to check. The constraint must be specified in microns using non-negative real numbers.

Note



The maximum distance that can be checked inside litho tagging is the optical radius, as defined for the optical mode. The default value for the optical radius is 0.64.

- ***metric***

An optional argument specifying the shape of the measurement region for the check. Metric must be one of

OPPOSITE

SQUARE

EUCLIDEAN

OPPOSITE_EXTENDED *extension*

The default is OPPOSITE. Refer to “[Construct Measurement Regions](#)” on page 44 for more information on metrics.

- **FULL**

An optional keyword that causes the tag set to include only fragments in which the entire fragment satisfies ***constraint***. The default behavior is to include fragments that partially or fully meet ***constraint***.

- **NOT PROJECTING or PROJECTING [*constraint*]**

An optional keyword whose usage matches the PROJECTING option available in Calibre DRC checks. The constraint value cannot be used when NOT is specified. The default setting is that the projection is not calculated or used.

Although this keyword set is optional, when used it must come before INPUT1 in the command or the run exits with a setup file error, “PROJECTING must come before INPUT1 or INPUT2.”

- **INPUT1** *layer1* [*tag1*]

A required keyword and its arguments, used to specify the layer and tag to check.

The required *layer1* argument is the name of layer to check. This must be an opc layer.

The optional tag1 argument is the tag within *layer1* to check. Those fragments in *tag1* that meet the distance constraint are tagged with *tagName*. If *tag1* is omitted, this defaults to the built-in tag “all”.

- **INPUT2** *layer2* [*tag2*]

An optional keyword and its arguments, used when performing a two-layer dimensional check, to specify the layer and tag containing the edges against which distances are measured. If not specified, the *tagName* is created using a one-layer dimensional check.

The *layer2* argument is the name of second layer to check. This can be any layer type.

The *tag2* argument is the tag within *layer2* to check. If INPUT2 is specified but tag2 is omitted, tag2 defaults to the “all” built-in tag. If tag2 is specified, then layer2 must be opc or correction type, which are the only layer types that can be tagged. If the layer type is not opc or correction, then tag2 is not allowed.

Description

The **EXTERNAL** tagging method allow users to tag fragments based on the DRC EXTERNAL dimension check. A fragment selected by the DRC check in Calibre is placed into the results tag, as follows:

- 1-input operation — When only **INPUT1** is specified, the check is a 1-layer DRC check between the fragments of the same layer.
- 2-input operation on same layers — When **INPUT1** and INPUT2 are both specified, and *layer1* is the same as *layer2*, the check is a 2-layer check between the **INPUT1** tagged fragments and the INPUT2 tagged fragments, both of which happen to be on the same layer. The results are a subset of tag1, that is, the results ONLY include fragments that were in **INPUT1** tag1.
- 2-input operation on different layers — When **INPUT1** and INPUT2 are specified, and *layer1* differs from *layer2*, the check is a 2-layer check between the fragments of the first layer and the fragments of the second layer. The results are a subset of tag1, that is, the results ONLY include fragments that were in **INPUT1** tag1.

EXTERNAL uses the default orientation filters: PARALLEL ALSO, ACUTE ALSO, NOT PERPENDICULAR, and NOT OBTUSE. No other DRC dimension check filter tags or other options are allowed.

An EXTERNAL tagging operation checks whether the operations require an interaction distance greater than or equal to what is set by the program automatically, based on the size of the optical radius. The interaction distance is calculated as:

$$0.5 * \text{hoodpix} + (\text{numsites} - \text{sitecenter} - 1) * \text{sampleSpacing}$$

If the operation requires a larger interaction range, then the tools generate an error during the setup file compilation stage. You can disable this check by using the [LITHO_SUPPRESS_INTERACTION_ERROR](#) environment variable:

```
sse LITHO_SUPPRESS_INTERACTION_ERROR 1
```

Disabling this error is not recommended.

Tip  **Best Practice:** During length-based tagging, set an upper bound on length checks to keep the tags out of invalid regions. The upper bound value should be less than or equal to the interaction distance. Allowing tags to apply to fragments longer than the interaction distance can cause false tags, which gives bad OPC results.

Examples

```
newTag EPE_near_endcap -how EXTERNAL > 0.3 < 0.6 \
INPUT1 m1 badEPE INPUT2 m1 endcap
```

This example finds all fragments in badEPE within 0.3 to 0.6 microns of endcap on layer m1.

Related Topics

[Pre-Defined Tags](#)

newTag ... -how fragAfterEdge

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments after the edge tagged with tag1.

Usage

newTag *tagName* -how fragAfterEdge *tag1*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***tag1***

A required argument specifying the name of tag from which to start. Only one tag may be specified per command. Additional tags generate the error “Excess arguments specified.”

Description

The fragAfterEdge tagging method tags all fragments after the edge tagged with ***tag1***. It is sometimes useful to be able to tag fragments touching an edge in a detailed way. All fragments are oriented in a direction such that the interior is always to the right of the fragment. Thus to get to the fragment “after” a given fragment, move along the fragment such that the interior is to the right. “Before” is defined as the opposite of “after,” and “next to” is defined as “before or after.”

Related Topics

[newTag ... -how EDGE](#)

[newTag ... -how SEQUENCE](#)

newTag ... -how fragAfterFrag

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments after the fragment tagged with tag1.

Usage

newTag *tagName* -how fragAfterFrag *tag1*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***tag1***

A required argument specifying the name of tag from which to start. Only one tag may be specified per command. Additional tags generate the error “Excess arguments specified.”

Description

This method tags all the fragments after any fragment tagged with ***tag1***. All fragments are oriented in a direction such that the interior is always to the right of the fragment. Thus to get to the fragment “after” a given fragment, move along the fragment such that the interior is to the right. “Before” is defined as the opposite of “after,” and “next to” is defined as “before or after.”

Related Topics

[newTag ... -how fragBeforeFrag](#)

[newTag ... -how fragNextToFrag](#)

newTag ... -how fragBeforeEdge

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments before the edge tagged with *tag1*.

Usage

newTag *tagName* -how fragBeforeEdge *tag1*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***tag1***

A required argument specifying the name of tag from which to start. Only one tag may be specified per command. Additional tags generate the error “Excess arguments specified.”

Description

This newTag method tags all fragments before an edge tagged with *tag1*. All fragments are oriented in a direction such that the interior is always to the right of the fragment. Thus to get to the fragment “after” a given fragment, move along the fragment such that the interior is to the right. “Before” is defined as the opposite of “after,” and “next to” is defined as “before or after.”

Related Topics

[newTag ... -how EDGE](#)

[newTag ... -how SEQUENCE](#)

newTag ... -how fragBeforeFrag

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments before the fragment tagged with *tag1*.

Usage

newTag *tagName* -how fragBeforeFrag *tag1*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***tag1***

A required argument specifying the name of tag from which to start. Only one tag may be specified per command. Additional tags generate the error “Excess arguments specified.”

Description

This method tags all the fragments before any fragment tagged with *tag1*. All fragments are oriented in a direction such that the interior is always to the right of the fragment. To get to the fragment after a given fragment, move along the fragment such that the interior is to the right. Thus to get to the fragment “after” a given fragment, move along the fragment such that the interior is to the right. “Before” is defined as the opposite of “after,” and “next to” is defined as “before or after.”

Related Topics

[newTag ... -how fragAfterFrag](#)

[newTag ... -how fragNextToFrag](#)

newTag ... -how FRAGMENT

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags fragments based on their lengths and the types of corners at either end point.

Usage

newTag *tagName* -how FRAGMENT *filterTag*

[TYPE1 *type1* | ANGLE1 *angle1_constraint*]
[TYPE2 *type2* | ANGLE2 *angle2_constraint*]
[BIAS *b_constraint* [BIAS1 *b1_constraint*] [BIAS2 *b2_constraint*] [RELATIVE]]
[LENGTH *len_constraint*] [ORDERED]

Arguments

- ***tagName***

A required argument specifying the new tag's name.

- ***filterTag***

A required argument specifying an initial tag set to which a fragment must belong in order to be considered for this new tag.

- **TYPE1 *type1***

An argument that stipulates that at least one of the fragment's corners must be of the specified type if the fragment is to be tagged by this method. Note that you can specify either the TYPE1 or the ANGLE1, but not both.

Valid values for type1 are:

ANY

CONCAVE

CONVEX

The default is ANY.

- **ANGLE1 *angle1_constraint***

The angle constraint placed on the corner at the fragment's first endpoint. The value of angle1_constraint must be a constraint within the 0 to 360 range. Note that you can specify either the TYPE1 or the ANGLE1, but not both.

- **TYPE2 *type2***

An argument that stipulates the type of corner at the fragment's second endpoint, if the fragment is to be tagged by this method. Note that you can specify either the TYPE2 or the ANGLE2, but not both.

Valid values for type2 are:

ANY

CONCAVE

CONVEX

The default is ANY.

- ANGLE2 *angle2_constraint*

The angle constraint placed on the corner at the fragment's second endpoint. The value of *angle2_constraint* must be a constraint within the 0 to 360 range. Note that you can specify either the TYPE2 or the ANGLE2, but not both.

- BIAS *b_constraint*

An optional argument that filters the tags based on the bias of the fragment. The value of *b_constraint* must be a constraint in microns; for example, BIAS > -0.05 <= 0.05.

- BIAS1 *b1_constraint*
BIAS2 *b2_constraint*

Optional arguments that filter the tags based on the bias of the neighboring fragments. The value must be a constraint in microns.

The specific neighbor referred to by BIAS1 and BIAS2 is determined by the keyword ORDERED.

- When ORDERED is part of the command, BIAS1 refers to the first endpoint and BIAS2 to the last endpoint.
- When ORDERED is not used, the constraints are applied to either neighbor.

- RELATIVE

An optional argument used with BIAS1 or BIAS2 to indicate the values are relative to the bias of the current fragment. The bias of the fragment is subtracted from the bias of the neighbor. This causes the constraint to be equivalent to the change in bias from one fragment to another along the edge.

The default (not RELATIVE) is to use the fragment's isolated bias value.

- LENGTH *len_constraint*

A constraint on the lengths of fragments to be tagged. You must specify the length constraint, *len_constraint*.

- ORDERED

Used for tagging the desired edge fragments according to orientation. This is useful for integrating to fragmentTag. When using ORDERED, the value of ANGLE1, TYPE1, or BIAS1 is equivalent to the first endpoint (for -first in fragmentTag) and the value of ANGLE2, TYPE2, or BIAS2 are equivalent to the last endpoint (for -last in fragmentTag). This allows non-symmetric fragmentation of an edge using fragmentTag.

Although the default (unordered) condition allows the constraint conditions to be applied to either endpoint, they are applied consistently: If BIAS1 is used with ANGLE1 or TYPE1, both must be true for the same endpoint.

Description

This newTag method tags fragments based on their lengths and the types of corners at either end. This method is related to the [newTag ... -how EDGE](#) tagging method, but differs in that the EDGE method tags all fragments in an edge when the entire edge meets the constraints. This method tags those specific fragments that themselves meet the constraints.

A FRAGMENT tagging operation checks whether the operations require an interaction distance greater than or equal to what is set by the program automatically, based on the size of the optical radius. The interaction distance is calculated as:

$$0.5 * \text{hoodpix} + (\text{numsites} - \text{sitecenter} - 1) * \text{sampleSpacing}$$

If the operation requires a larger interaction range, then the tools generate an error during the setup file compilation stage. You can disable this check by using the environment variable [LITHO_SUPPRESS_INTERACTION_ERROR](#):

```
sse LITHO_SUPPRESS_INTERACTION_ERROR 1
```

Disabling this error is not recommended.

Tip

 **Best Practice:** During length-based tagging, set an upper bound on length checks to keep the tags out of invalid regions. The upper bound value should be less than or equal to the interaction distance. Allowing tags to apply to fragments longer than the interaction distance can cause false tags, which gives bad OPC results.

Constraints

Valid constraints can be written using either an upper bound or a lower bound, or both, and can use any of the standard operators. The following are examples of valid LENGTH constraints:

```
LENGTH >= 0.1 < 0.2
LENGTH == 1.0
LENGTH != 1.0
LENGTH < 0.4
```

Examples

ORDERED Effects

The following newTag command selects all fragments less than 0.06 microns that are collinear (in line) with any neighboring fragment.

```
newTag short_phase1 -how FRAGMENT phase_edges \
LENGTH < 0.06 ANGLE1 == 0
```

When ORDERED is part of the command, however, the fragment must be in line specifically with the fragment preceding it on the edge.

```
newTag short_phase1 -how FRAGMENT phase_edges \
LENGTH < 0.06 ANGLE1 == 0 ORDERED
```

To indicate the angle formed by the fragment and the next one in line, use ORDERED and ANGLE2 instead of ANGLE1.

Related Topics

[newTag ... -how EDGE](#)

[fragmentTag](#)

newTag ... -how fragNextToEdge

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments next to the edge tagged with tag1.

Usage

newTag *tagName* -how fragNextToEdge *tag1*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***tag1***

A required argument specifying the name of tag from which to start. Only one tag may be specified per command. Additional tags generate the error “Excess arguments specified.”

Related Topics

[newTag ... -how EDGE](#)

[newTag ... -how SEQUENCE](#)

newTag ... -how fragNextToFrag

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments next to the fragment tagged with tag1.

Usage

newTag *tagName* -how fragNextToFrag *tag1*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***tag1***

A required argument specifying the name of tag from which to start. Only one tag may be specified per command. Additional tags generate the error “Excess arguments specified.”

Related Topics

[newTag ... -how fragAfterFrag](#)

[newTag ... -how fragBeforeFrag](#)

newTag ... -how HAS_SITE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Creates a subgroup based on whether a site was generated on tagged fragments.

Usage

`newTag tagName -how HAS_SITE filter`

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***filter***
A required argument specifying the name of tag for whose possession of a site is checked.

Description

This newTag command creates a new tag, ***tagName***, each of which is from the ***filter*** tag and has a control site.

Examples

Put line ends with sites into tag t.

```
newTag t -how HAS_SITE line_end
```

Related Topics

[Pre-Defined Tags](#)

[Control Sites](#)

newTag ... -how IMAGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags fragments based on aerial image properties.

Usage

```
newTag tagName -how IMAGE filterTag [NOT]
[OUTOFCOMMITS]
[IMAGE intensity_constraint DISTANCE distance_constraint]
[SLOPE slope_constraint [LOG | {YVAL thresh}]]
[C factor_constraint]
[IMAX imax] [IMIN imin] [CHECKMODEL]
[CONTRAST con_constraint]
[THRESHOLD threshold_constraint]
[EXTREMA_DISTANCE ex_constraint]
[GRADIENT_ANGLE constraint]
[GRADIENT_ABSANGLE constraint]
[GRADIENT_MAG constraint]
[USE_MODEL_DEFINITIONS | UMD]
[MEEF constr [DELTA delta]]
[QUAD_INTERP]
[DENSITY kernel constraint]
```

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***filterTag***

A required argument specifying the tag name for the set of fragments whose aerial image is evaluated against the specified constraints.

- NOT

An optional keyword complementing all subsequent constraints, instructing the application to tag those fragments that do not meet the specified constraints.

- OUTOFCOMMITS

This optional argument determines when OPC moves outside the parameter space of the resist model. If this keyword is specified, then it is a user error to specify any constraint on the image properties through this newTag command.

The resist model has the capability to bound the image properties which by default are unbounded. After OPC, the fragments that do not lie within these constraints are tagged by the newTag command.

- IMAGE *intensity_constraint* DISTANCE *distance_constraint*

A set of optional keyword-value pairs used to tag those fragments in the *filterTag* whose aerial image intensity is within the given *intensity_constraint* over the entire range of the *distance_constraint*.

- The IMAGE keyword defines the constraint on allowed aerial image intensity values. The user-supplied constraint, *intensity_constraint*, must be specified as an interval of intensity values expressed as non-negative real numbers. Values are in units of “normalized aerial image”, in which the full field exposure yields an image value of one.
- The DISTANCE keyword defines the constraint on the spatial range to check. The user-supplied constraint, *distance_constraint*, must be specified in microns as an interval of real numbers. Negative values represent distances inside the polygon, positive values represent distances outside the polygon.

- SLOPE *slope_constraint* [LOG | {YVAL *thresh*}]

A set of optional keyword-value pairs used to tag those fragments with an aerial image slope that satisfies *slope_constraint*. The slope being evaluated is the slope along the cutline for the fragment.

- The SLOPE keyword defines the constraint on allowed slope values, and is written as an interval of Dintensity-per-micron values.
- The LOG keyword instructs the application to use the LOG slope, and cannot be used with YVAL.
- The YVAL keyword instructs the application to evaluate the slope at the position where the image intensity matches the specified threshold. If YVAL *thresh* is omitted, the application evaluates the maximum slope along the cutline.

- C *factor_constraint*

An optional keyword-value pair used to tag those fragments having an aerial image factor that satisfies the constraint *factor_constraint*.

- IMAX *imax*

An optional keyword-value pair used to tag those fragments having a maximum intensity value that satisfies the constraint *imax*.

- IMIN *imin*

An optional keyword-value pair used to tag those fragments having a minimum intensity value for that satisfies the constraint *imin*.

- CHECKMODEL

An optional keyword that checks whether a constrained parameter for IMAGE is defined in the VT5 model. The parameters which can be checked are IMIN, IMAX, SLOPE, and C. If the parameter is not defined, the run halts with an error.

If this keyword is not used, the run continues even if the constrained parameter is not defined. Instead, the parameter is treated as having a value of 0 for all fragments.

- **CONTRAST** *con_constraint*

An optional keyword-value pair used for tagging based on contrast.

You compute the *con_constraint* for each site using the following formula:

$$\text{contrast} = \{\max(I) - \min(I)\} / \{\max(I) + \min(I)\}$$

For example:

```
newTag low_contrast_ends -how IMAGE line_end CONTRAST < 0.7
```

- **THRESHOLD** *threshold_constraint*

An optional keyword-value pair you use for tagging fragments having an image threshold within the specified constraint. You obtain the constraint by evaluating the resist model at the given fragment and observing its predicted threshold.

- **EXTREMA_DISTANCE** *ex_constraint*

An optional keyword-value pair you use with IMAX, IMIN, or CONTRAST to define a search window for the calculations. This option requires a closed distance interval as an argument, such as the following:

```
> -0.1 < 0.1
```

It is required under the following circumstances:

- tagging based on CONTRAST
- tagging based on intensity (IMIN or IMAX) using a resist model that is version 3 or later (VT5).

It is recommended under the following circumstance:

- tagging based on IMIN or IMAX using a resist model that is version 2 or earlier.

For example:

```
newTag tagName -how IMAGE IMAX imax_constraint ... \
EXTREMA_DISTANCE ex_constraint
```

- **GRADIENT_MAG** *constraint*

An optional argument specifying the image gradient magnitude in units of 1/user units (microns). It can only be used with VT5 version 3 resist models having curvature (even if the coefficient is 0).

- **GRADIENT_ANGLE** *constraint*

An optional argument specifying the image gradient angle with respect to the fragment which owns it. Valid values are from 0 to 360. It can only be used with VT5 version 3 resist models having curvature (even if the coefficient is 0).

- GRADIENT_ABSANGLE *constraint*

An optional argument specifying the absolute value of the angle, in the range 0 to 180. It can only be used with VT5 version 3 resist models having curvature (even if the coefficient is 0).

- USE_MODEL_DEFINITIONS

Used for tagging image properties consistent with the VT5 definitions currently in use. May also be specified as UMD.

When specified, the definitions of the IMAX, IMIN, C, SLOPE, and, consequently, CONTRAST (which uses IMAX and IMIN), are used exactly as the VT5 model would use them.

USE_MODEL_DEFINITIONS cannot be used with QUAD_INTERP, LOG, YVAL, or EXTREMA_DISTANCE.

- MEEF *constr*

An optional argument, required when using MEEF tagging, defining a range of MEEF values. All fragments in *filterTag* meeting all other IMAGE conditions and having a MEEF within this range are tagged as *tagName*.

- DELTA *delta*

An optional argument used in calculating the MEEF for the specified tag. The *delta*, specified in user units, can be a positive or a negative number. A negative value corresponds to shifting the fragment inwards into the polygon, and a positive value corresponds to shifting the fragment outwards from the polygon.

When specified, this value overrides the global DELTA set in the MATRIX_OPCT block.

When not specified, this argument defaults to the global DELTA set in the MATRIX_OPCT block.

- QUAD_INTERP

An optional keyword for calculating IMAX, IMIN, and CONTRAST using a 3-point quadratic interpolation of the sample image values. By default, the Litho tools compute IMIN, IMAX, and CONTRAST using the discrete sampled points, with no interpolation.

Note

 If you use the QUAD_INTERP option, you must specify it last.

- DENSITY *kernel constraint*

An optional keyword for the IMAGE tagging method that tags fragments based on the density kernels. Any fragments from the tag set *filterTag* that match the constraint for density kernels are selected and added to *tagName*. For example:

```
newTag d1tag -how IMAGE all density D1 >= 0.5
```

This example tags all fragments with a D1 density greater than or equal to 0.5 and is placed in the tag set d1tag.

Description

This newTag method tags fragments in the *filterTag* based on image intensity, IMAX, IMIN, aerial image factor, or slope. At least one constraint must be specified. Any unspecified constraint defaults to UNCONSTRAINED.

For VT5 models, the valid arguments are IMAX, IMIN, SLOPE, and Curvature.

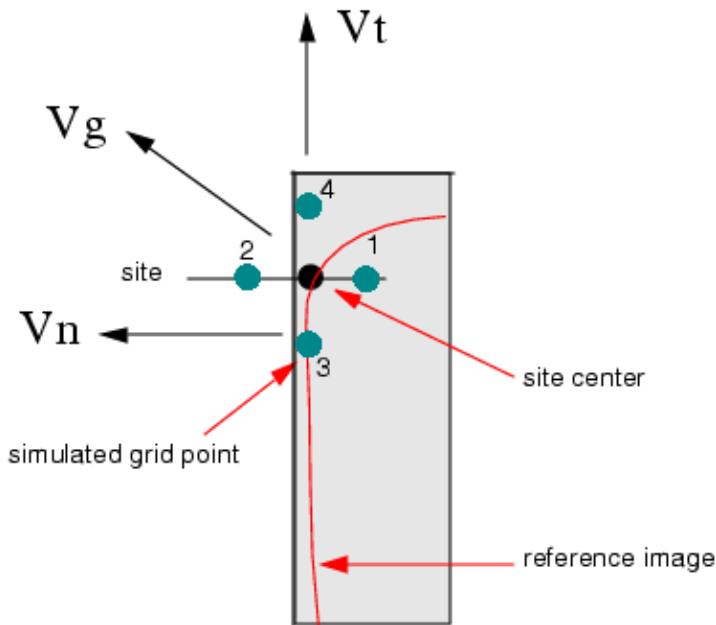
GRADIENT Options

Image gradient tagging allows tagging for the gradient of the aerial image on each site. Gradient tagging produces the gradient vector, gradient angle, absolute gradient angle and magnitude.

An analogy for a gradient magnitude is a contour-line topographical map, where contour lines that are closer together indicate a steeper incline or gradient than contour lines that are spread further apart. In this case, the contour lines indicate intensity at different points.

The gradient is calculated using a mini-grid centered at the site center value with four simulation points (Figure 6-78 illustrates the four simulation points). The image gradient is a vector sum, V_g , derived from the Vector Normal (the direction of the site), V_n , and Vector Tangent (the tangential direction of the site), V_t .

Figure 6-78. Image Gradient Tagging



The spacing between the simulated grid points is equal to the sampleSpacing value of the resist model. The center location of the four points is along the contour created by the referenceThreshold in the VT5 model.

Based on this information, the gradient vector \vec{V}_g is calculated as follows:

$$\vec{V}_g = (((I_2 - I_1)/d_{21}) \bullet \vec{V}_n)^2 + (((I_4 - I_3)/d_{43}) \bullet \vec{V}_t)^2$$

where I is the Intensity and d is the distance between two points on the four-point grid (for example, d21 is distance between grid point 2 to point 1 as represented in [Figure 6-78](#)).

The image gradient magnitude, as represented in units of 1/user units (microns), is derived as follows:

$$magnitude = \sqrt{(((I_2 - I_1)/d_{21}) \bullet \vec{V}_n)^2 + (((I_4 - I_3)/d_{43}) \bullet \vec{V}_t)^2}$$

MEEF Options

You can define as many MEEF related tags as needed. However, each MEEF tagging command in the setup file that uses a different DELTA triggers a full intensity calculation pass. You can minimize the impact of using multiple DELTA values by grouping all MEEF tagging commands with the same DELTA together. This speeds up simulation.

For example:

```
newTag t1 -how IMAGE all MEEF > 1 DELTA 0.01
newTag t2 -how IMAGE all MEEF > 1 DELTA -0.01
newTag t3 -how IMAGE all MEEF < 2 DELTA 0.01
```

This causes the intensity to be calculated for all sites 3 times. The following modification results in only 2 simulation passes.

```
newTag t1 -how IMAGE all MEEF > 1 DELTA 0.01
newTag t3 -how IMAGE all MEEF < 2 DELTA 0.01
newTag t2 -how IMAGE all MEEF > 1 DELTA -0.01
```

The reason this arrangement is more efficient is that the tagging operations are executed in the order seen in the setup file.

Examples

Example 1

Check for line-end bridging by observing if the aerial image is above the threshold of 0.3 for the entire range of 0 to 0.1 microns from the line end.

```
newTag tag2 -how image line_end IMAGE > 0.3 DISTANCE >0 <=0.1
```

Example 2

This example tags all the fragments that are outside the bounds of slope and Imin.

```
newTag t3 -how IMAGE all OUTOFCOMMITS SLOPE IMIN
```

No constraints on the other keywords may be specified if you are using the OUTOFCOMMITS keyword. See also the next example.

Example 3

If bounds are specified in the resist file as:

```
bound SLOPE 0 3.23801
```

There are two ways of displaying the fragments that are outside the bounds of the resist file:

```
newTag t3 -how IMAGE all NOT slope >= 0 <= 3.23801  
newTag t3 -how IMAGE all OUTOFCOMMITS SLOPE
```

newTag ... -how INSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all source layer fragments that lie completely inside the marker layer polygons.

Usage

```
newTag tagName -how INSIDEEDGE markerLayer srcLayer [-tag filterTag]
```

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***markerLayer***

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- ***srcLayer***

A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.

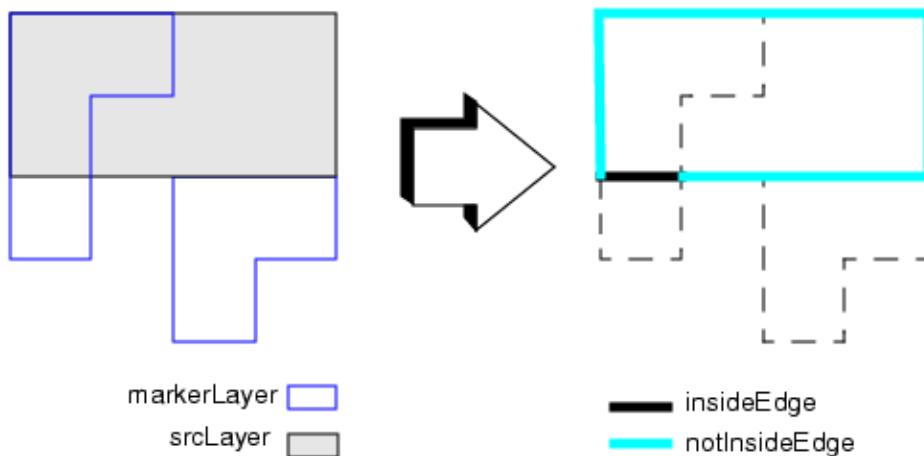
- **-tag *filterTag***

An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation [INSIDE EDGE](#). Fragments that only touch the border are not included.

Figure 6-79. INSIDEEDGE



Related Topics

[newTag ... -how LAYERAND](#)
[newTag ... -how NOTINSIDEEDGE](#)
[newTag ... -how OUTSIDEEDGE](#)
[newTag ... -how NOTOUTSIDEEDGE](#)
[newTag ... -how COINCIDENTEDGE](#)
[newTag ... -how TOUCHEDGE](#)

newTag ... -how INTERNAL

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags fragments based on the DRC INTERNAL dimension check.

Usage

```
newTag tagName -how INTERNAL constraint [metric] [FULL]  
[[NOT] PROJECTING [constraint]]  
INPUT1 layer1 [tag1] INPUT2 layer2 [tag2]  
[MEASURE all | coincident]
```

Arguments

- ***tagName***

A required argument specifying the name for the new tag, which is a subset of tag1.

- ***constraint***

A required argument specifying the distance to check. The constraint must be specified in microns using non-negative real numbers.

Note



The maximum distance that can be checked inside litho tagging is the optical radius, as defined for the optical mode. The default value for the optical radius is 0.64.

- ***metric***

An optional argument specifying the shape of the measurement region for the check. Metric must be one of

OPPOSITE

SQUARE

EUCLIDEAN

OPPOSITE_EXTENDED *extension*

The default is OPPOSITE.

- **FULL**

An optional keyword that causes the tag set to include only fragments in which the entire fragment satisfies ***constraint***. The default behavior is to include fragments that partially or fully meet ***constraint***.

- **INPUT1 *layer1* [*tag1*]**

A required keyword and its arguments, used to specify the layer and tag to check.

The ***layer1*** argument is the name of layer to check. This must be an “opc” layer.

The tag1 argument is the tag within layer1 to check. Those fragments in tag1 that meet the distance constraint are tagged with **tagName**. If omitted, this defaults to the built-in tag “all”.

- INPUT2 *layer2* [*tag2*]

An optional keyword and its arguments, used when performing a two-layer dimensional check, to specify the layer and tag containing the edges against which distances are measured. If not specified, the **newTag** is created using a one-layer dimensional check.

The *layer2* argument is the name of second layer to check. This can be any layer type.

The *tag2* argument is the tag within *layer2* to check. If INPUT2 is specified but tag2 is omitted, tag2 defaults to the “all” built-in tag. If tag2 is specified, then layer2 must be of type opc or correction, the only layer types that can be tagged. If the layer type is not opc or correction, then tag2 is not allowed.

- NOT PROJECTING or PROJECTING [*constr*]

An optional keyword whose usage matches the PROJECTING option available in Calibre DRC checks. The *constr* value cannot be used when NOT is specified. The default setting is that the projection is not calculated or used.

Although this keyword set is optional, when used it must come before INPUT1 in the command or the run exits with a setup file error, “PROJECTING must come before INPUT1 or INPUT2.”

- MEASURE all | coincident

An optional keyword which is relevant only for 2-layer INTERNAL checks and works as in the [Standard Verification Rule Format \(SVRF\) Manual](#) description for this modifier to the DRC operation. The default setting is that the default Calibre containment criteria is used.

Description

The INTERNAL tagging method allows users to tag fragments based on the SVRF [INTERNAL](#) dimension check. A fragment that would have been selected by the DRC check in Calibre is placed into the results tag, as follows:

- 1-input operation — When only INPUT1 is specified, the check is a 1-layer DRC check between the fragments of the same layer.
- 2-input operation on same layers — When INPUT1 and INPUT2 are both specified, and layer1 is identical to layer2, the check is a “2-layer” check between the INPUT1 tagged fragments and the INPUT2 tagged fragments, both of which happen to be on the same layer. The results are a subset of tag1, that is, the results ONLY include fragments that were in the INPUT1 tag1.
- 2-input operation on different layers — When INPUT1 and INPUT2 are specified, and layer1 != layer2, the check is a 2-layer check between the fragments of the first layer and the fragments of the second layer. The results are a subset of tag1, that is, the results ONLY include fragments that were in the INPUT1 tag1.

INTERNAL uses the default orientation filters PARALLEL ALSO, ACUTE ALSO, NOT PERPENDICULAR, and NOT OBTUSE. No other DRC dimension check filters or other options are allowed at this time.

An INTERNAL tagging operation checks whether the operations require an interaction distance greater than or equal to what is set by the program automatically, based on the size of the optical radius. The interaction distance is calculated as:

$$0.5 * \text{hoodpix} + (\text{numsites} - \text{sitecenter} - 1) * \text{sampleSpacing}$$

If the operation requires a larger interaction range, then the tools generate an error during the setup file compilation stage. You can disable this check by using the environment variable [LITHO_SUPPRESS_INTERACTION_ERROR](#):

sse LITHO_SUPPRESS_INTERACTION_ERROR 1

Disabling this error is not recommended.

Tip  **Best Practice:** During length-based tagging, set an upper bound on length checks to keep the tags out of invalid regions. The upper bound value should be less than or equal to the interaction distance. Allowing tags to apply to fragments longer than the interaction distance can cause false tags, which gives bad OPC results.

Examples

```
newTag wideLayer -how INTERNAL > 0.065 < 0.1 OPPOSITE INPUT1 opc_layer all
```

The example finds all fragments that are more than 65 nm away from the fragments of the other side of the polygon.

Related Topics

[newTag ... -how EXTERNAL](#)

[newTag ... -how EPE](#)

newTag ... -how LAYERAND

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags fragments on opc layers that intersect polygons on island layers.

Usage

`newTag tagName -how LAYERAND islandLayer opcLayer`

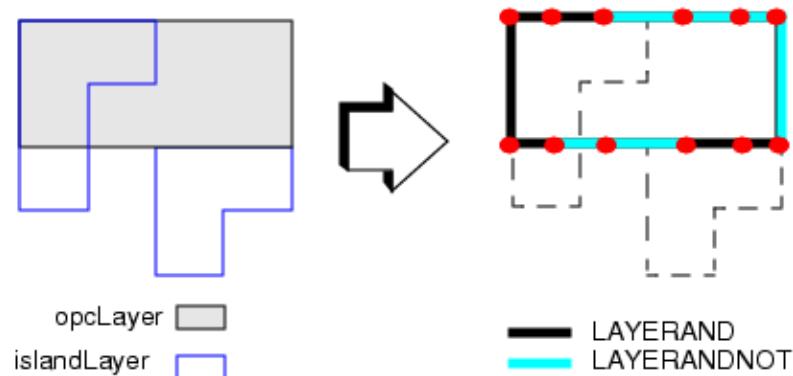
Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***islandLayer***
A required argument specifying the name of the island-type layer containing the polygons. The island layer must be defined in the setup file as type island.
- ***opcLayer***
A required argument specifying the name of the opc-type or correction-type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file as type opc.

Description

This tagging keyword creates a new tag set with handle ***tagName*** that includes all fragments from ***opcLayer*** that lie inside or along the boundaries of ***islandLayer*** polygons. It is the equivalent of selecting fragments with newTag -how INSIDEEDGE or COINCIDENTEDGE.

Figure 6-80. LAYERAND (Island Fragmentation Off)



Note

This operation does not perform fragmentation; it tags existing fragments. Consequently, if a fragment lies partially within the *islandLayer* polygon border, then the entire fragment is not tagged by this command. By default, island layer fragmentation is ON, which causes vertices to be added wherever the island layer intersects the opc layer. You can turn OFF island layer fragmentation by setting the frag option in the [layer](#) definition for the island layer to 32.

Examples

The LAYERAND keyword is useful to identify and tag a certain region of a layout underneath a layer. One use of this feature is to flag an area for this operation. For example, if you want to put a box on layer gate around a critical region of polysilicon. Assuming the polysilicon layer was layer poly, the following command would tag this critical region:

```
newTag criticalRegion -how layerAnd gate poly
```

Related Topics

[newTag ... -how COINCIDENTEDGE](#)

[newTag ... -how INSIDEEDGE](#)

[newTag ... -how LAYERANDNOT](#)

[newTag ... -how TOUCHEDGE](#)

[Layer Types](#)

newTag ... -how LAYERANDNOT

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags fragments on opc layers that do not interact with polygons on island layers.

Usage

`newTag tagName -how LAYERANDNOT islandLayer opcLayer`

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***islandLayer***
A required argument specifying the name of the island-type layer containing the polygons. The island layer must be defined in the setup file as type island.
- ***opcLayer***
A required argument specifying the name of the opc-type or correction-type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file as type opc.

Description

The layerAndNot command tags everything in layer ***opcLayer*** that was not selected by the [newTag ... -how LAYERAND](#) command. This command does not tag anything in the ***islandLayer*** layer.

This operation does not perform fragmentation, but tags existing fragments. The final result is the complement of LAYERAND. By default, island layer fragmentation is ON, which causes vertices to be added wherever the island layer intersects the opc layer, so no fragment extends outside the layer. You can turn OFF island layer fragmentation by setting the frag option in the [layer](#) definition for the island layer to 32.

The primary difference between [newTag ... -how NOTINSIDEEDGE](#) and LAYERANDNOT is the special case of fragments lying entirely on a coincident edge, or shared border, of the two layers. These are included in NOTINSIDEEDGE, but not included in LAYERANDNOT.

Related Topics

[Layer Types](#)

newTag ... -how loopWithFrag

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Creates a new tag containing all the loops tagged with fragmentTag.

Usage

newTag *tagName* -how loopWithFrag *fragmentTag*

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***fragmentTag***
The name of the tag for fragments to start from.

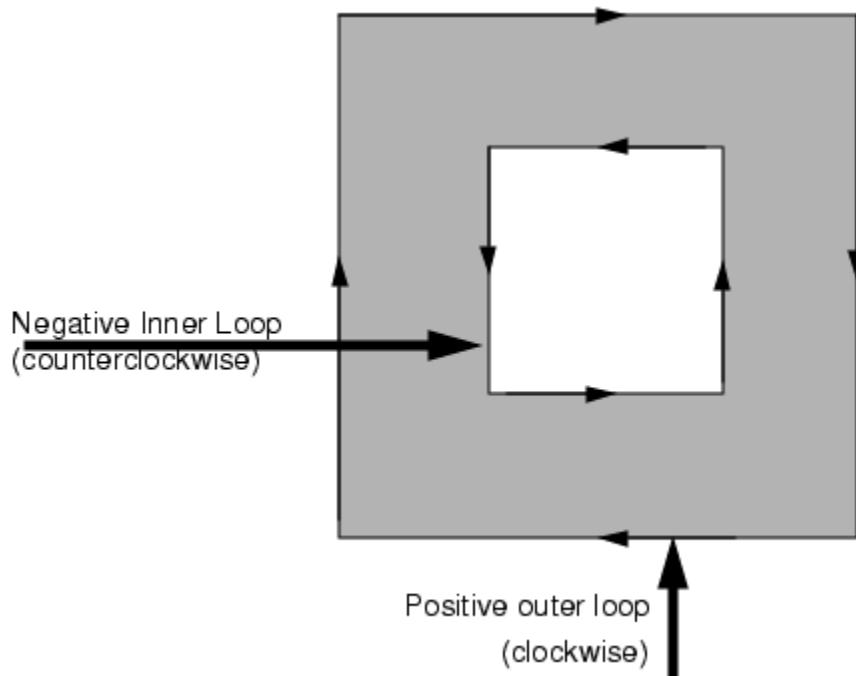
Description

This method creates a new tag called ***tagName*** containing all the loops tagged with ***fragmentTag***. The **loopWithFrag** method goes through all the fragments tagged with ***fragmentTag***. For each of these fragments, the entire loop with ***tagName*** is tagged.

Initially, a layout is a collection of polygons. When you perform fragmentation, the layout becomes a collection of loops. A loop is a closed chain of edge fragments that does not intersect any other loop, including itself. Loops are either positive or negative. If positive, then the loop is like a regular polygon. If negative, then the loop is a hole.

[Figure 6-81](#) shows a positive outer loop containing a negative inner loop.

Figure 6-81. Positive and Negative Loops



Examples

This example creates a new tag, “concaveLoop,” that refers to every fragment in a concave loop:

```
newTag concaveLoop -how loopWithFrag concave_corner
```

Related Topics

[Pre-Defined Tags](#)

[newTag ... -how SEQUENCE](#)

newTag ... -how MRC

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags fragments that had their movements restricted during the previous iteration.

Usage

newTag *tagName* -how *mrc_type* *filterTag*

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***mrc_type***

A required argument that indicates the type of MRC rule violation. Select one of the following:

MRC

Indicates that a fragment's movement was impeded due to *any* MRC rule violation.

MRCEXT

Indicates that a fragment's movement was impeded due to an external MRC rule violation.

MRCINT

Indicates that a fragment's movement was impeded due to an internal MRC rule violation.

MRCENCL

Indicates that a fragment's movement was impeded due to an MRC enclosure rule violation.

MRCOTHER

Indicates that a fragment's movement was impeded due to a rule MRC rule violation other than external, internal, or enclosure rule violations.

- ***filterTag***

A required argument specifying the tag name for the set of fragments to check. There is no default for this command. The option “all” must be specified in order to obtain tags of all restricted fragments.

Examples

Example 1

This example creates a new tag, “blocked_fragments,” that applies to any movement-restricted fragment in the set “large_epe”.

```
newTag blocked_fragments -how MRC large_epe
```

Example 2

This example tags all fragments in the design whose movement was restricted during the previous iteration due to an MRC internal rule.

```
newTag int_blocked -how MRCINT all
```

newTag ... -how NOTCOINCIDENTEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on the source layer that are not tagged by COINCIDENTEDGE.

Usage

`newTag tagName -how NOTCOINCIDENTEDGE markerLayer srcLayer [-tag filterTag]`

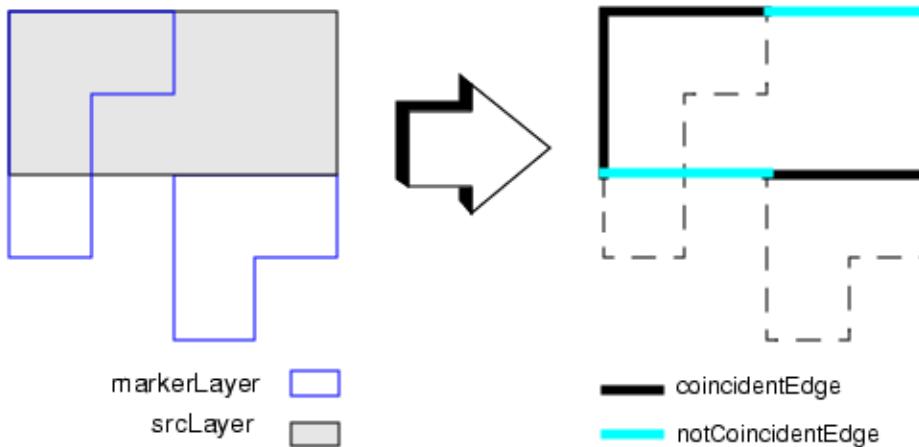
Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***markerLayer***
A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.
- ***srcLayer***
A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.
- **-tag *filterTag***
An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation NOT COINCIDENT EDGE.

Figure 6-82. Not Coincident Edges



Related Topics

[newTag ... -how COINCIDENTEDGE](#)
[newTag ... -how NOTCOINCIDENTINSIDEEDGE](#)
[newTag ... -how NOTCOINCIDENTOUTSIDEEDGE](#)

newTag ... -how NOTCOINCIDENTINSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on the source layer that are not tagged by COINCIDENTINSIDEEDGE.

Usage

```
newTag tagName -how NOTCOINCIDENTINSIDEEDGE markerLayer srcLayer
[-tag filterTag]
```

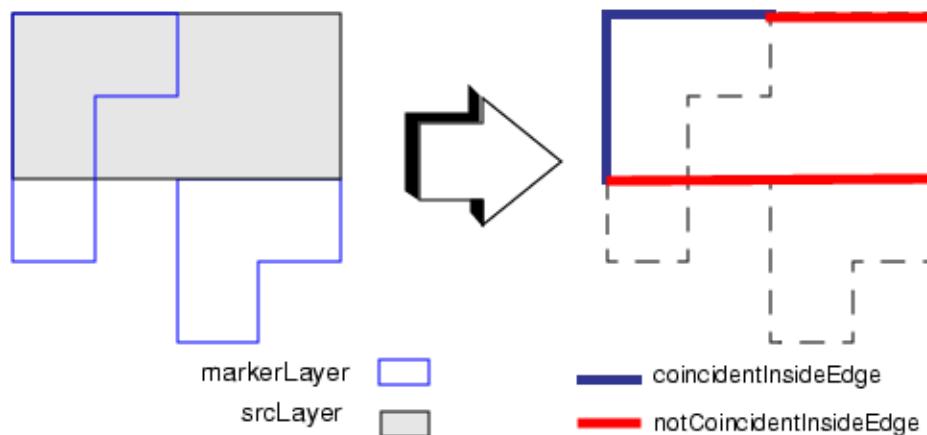
Arguments

- **tagName**
A required argument specifying the name for the new tag.
- **markerLayer**
A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.
- **srcLayer**
A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.
- **-tag filterTag**
An optional argument specifying filtering for a subset of the fragments in the **markerLayer** to be used as the marking fragments.

Description

This method tags fragments that are selected by the SVRF operation [NOT COINCIDENT INSIDE EDGE](#).

Figure 6-83. Not Coincident Inside Edge



Related Topics

[newTag ... -how COINCIDENTINSIDEEDGE](#)
[newTag ... -how NOTCOINCIDENTEDGE](#)
[newTag ... -how NOTCOINCIDENTOUTSIDEEDGE](#)

newTag ... -how NOTCOINCIDENTOUTSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on layer1 that are not tagged by COINCIDENTOUTSIDEEDGE.

Usage

```
newTag tagName -how NOTCOINCIDENTOUTSIDEEDGE markerLayer srcLayer [-tag filterTag]
```

Arguments

- *tagName*

A required argument specifying the name for the new tag.

- *markerLayer*

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- *srcLayer*

A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.

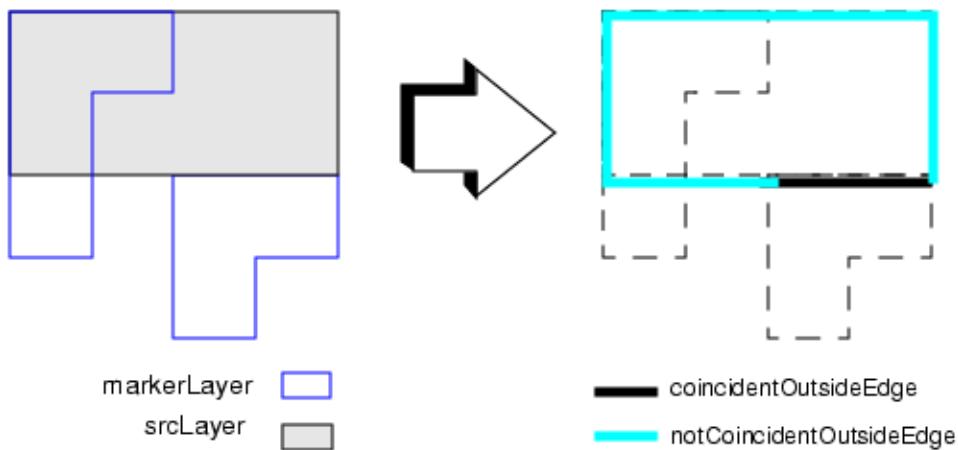
- -tag *filterTag*

An optional argument specifying filtering for a subset of the fragments in the *markerLayer* to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation [NOT COINCIDENT OUTSIDE EDGE](#).

Figure 6-84. Not Coincident Outside Edge



Related Topics

[newTag ... -how COINCIDENTOUTSIDEEDGE](#)

[newTag ... -how NOTOUTSIDEEDGE](#)

newTag ... -how NOTINSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on layer1 that are not tagged by INSIDEEDGE.

Usage

`newTag tagName -how NOTINSIDEEDGE markerLayer srcLayer [-tag filterTag]`

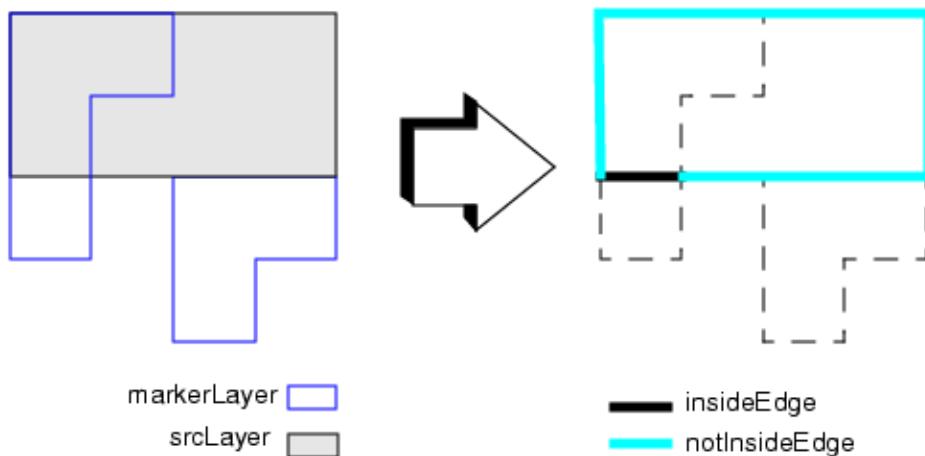
Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***markerLayer***
A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.
- ***srcLayer***
A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.
- **-tag *filterTag***
An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation NOT INSIDE EDGE.

Figure 6-85. NOTINSIDEEDGE



Related Topics

[newTag ... -how INSIDEEDGE](#)

[newTag ... -how OUTSIDEEDGE](#)

[newTag ... -how LAYERANDNOT](#)

newTag ... -how NOTOUTSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments on layer1 that are not tagged by OUTSIDEEDGE.

Usage

newTag *tagName* -how NOTOUTSIDEEDGE *markerLayer* *srcLayer* [-tag *filterTag*]

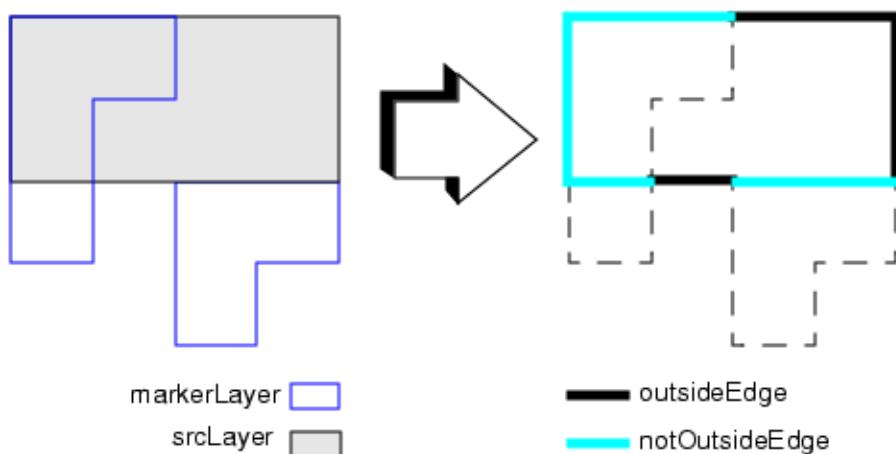
Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***markerLayer***
A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.
- ***srcLayer***
A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.
- **-tag *filterTag***
An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation **NOT OUTSIDE EDGE**.

Figure 6-86. NOTOUTSIDEEDGE



Related Topics

[newTag ... -how INSIDEEDGE](#)

[newTag ... -how OUTSIDEEDGE](#)

newTag ... -how NOTTOUCHEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments not tagged by the corresponding TOUCHEDGE.

Usage

`newTag tagName -how NOTTOUCHEDGE markerLayer sourceLayer [-tag filterTag]`

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***markerLayer***

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- ***sourceLayer***

A required argument specifying the name of the opc or correction layer containing the fragments being evaluated. The new tag is a subset of this layer.

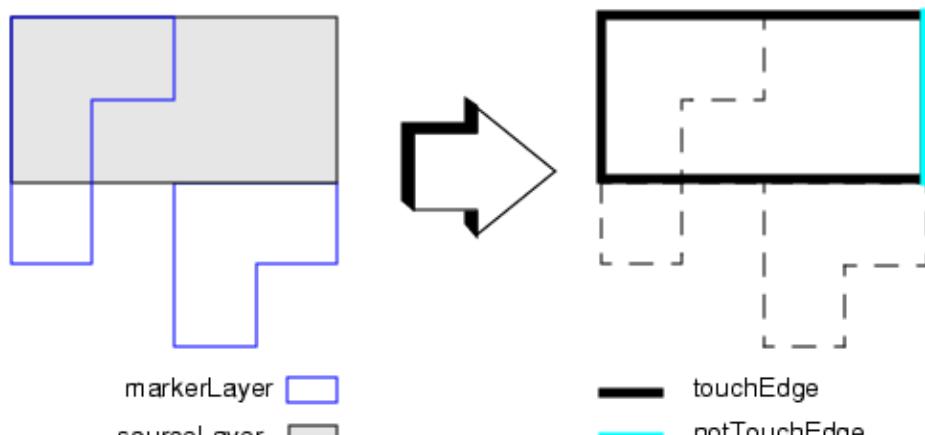
- **-tag *filterTag***

An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags all fragments in *filterTag* on *sourceLayer* that would not be tagged by [newTag ... -how TOUCHEDGE](#).

Figure 6-87. NOTTOUCHEDGE



Assume that the polygon on the sourceLayer has no fragmentation other than at the corners.

Related Topics

[newTag ... -how TOUCHEdge](#)
[newTag ... -how NOTCOINCIDENTEDGE](#)
[newTag ... -how NOTTOUCHINSIDEEDGE](#)
[newTag ... -how NOTTOUCHOUTSIDEEDGE](#)

newTag ... -how NOTTOUCHINSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all layer1 fragments not tagged by the corresponding TOUCHINSIDEEDGE.

Usage

newTag *tagName* -how NOTTOUCHINSIDEEDGE *markerLayer* *sourceLayer* [-tag *filterTag*]

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***markerLayer***

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- ***sourceLayer***

A required argument specifying the name of the opc or correction layer containing the fragments being evaluated. The new tag is a subset of this layer.

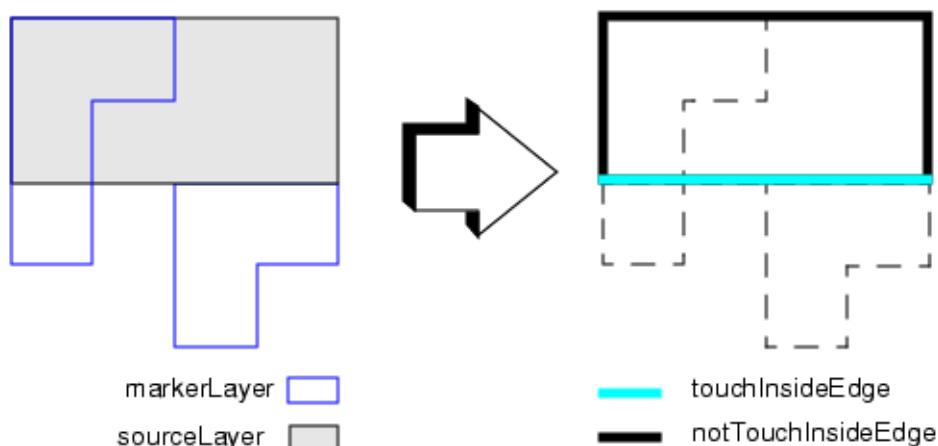
- **-tag *filterTag***

An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags all fragments in *filterTag* on *sourceLayer* that would not be tagged by [newTag ... -how TOUCHINSIDEEDGE](#), as shown in [Figure 6-88](#) following.

Figure 6-88. NOTTOUCHINSIDEEDGE



Assume that the polygon on the sourceLayer has no fragmentation other than at the corners.

Related Topics

- [newTag ... -how TOUCHINSIDEEDGE](#)
- [newTag ... -how TOUCHOUTSIDEEDGE](#)
- [newTag ... -how NOTINSIDEEDGE](#)
- [newTag ... -how NOTCOINCIDENTINSIDEEDGE](#)

newTag ... -how NOTTOUCHOUTSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all fragments not tagged by the corresponding TOUCHOUTSIDEEDGE.

Usage

```
newTag tagName -how NOTTOUCHOUTSIDEEDGE markerLayer sourceLayer [-tag filterTag]
```

Arguments

- **tagName**

A required argument specifying the name for the new tag.

- **markerLayer**

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- **sourceLayer**

A required argument specifying the name of the opc or correction layer containing the fragments being evaluated. The new tag is a subset of this layer.

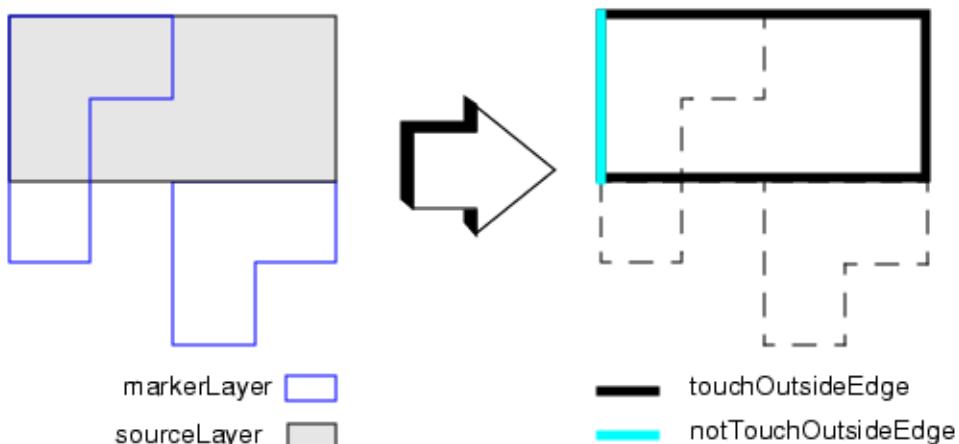
- **-tag filterTag**

An optional argument specifying filtering for a subset of the fragments in the **markerLayer** to be used as the marking fragments.

Description

This method tags all fragments in **filterTag** on **sourceLayer** that would not be tagged by [newTag ... -how TOUCHOUTSIDEEDGE](#), as shown in [Figure 6-89](#) following.

Figure 6-89. NOTTOUCHOUTSIDEEDGE



Assume that the polygon on the sourceLayer has no fragmentation other than at the corners.

Related Topics

[newTag ... -how TOUCHOUTSIDEEDGE](#)

[newTag ... -how TOUCHINSIDEEDGE](#)

[newTag ... -how NOTCOINCIDENTOUTSIDEEDGE](#)

[newTag ... -how NOTOUTSIDEEDGE](#)

newTag ... -how ORTAGS

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Creates a new tag everything that is tagged with either tag1 OR tag2 OR ... tagN. This allows you to find the union of two or more sets of fragments.

Usage

newTag *tagName* -how ORTAGS *tag1 tag2 [tag3...]*

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***tag1***
A required argument specifying the name of the first tag to include in the new tag.
- ***tag2***
A required argument specifying the name of a second tag to include in the new tag.
- ***tag3***
An optional list of arguments specifying the names of additional tags to include in ***tagName***.

Examples

The following command creates a new tag, corner, which applies to any fragment which is either a convex or a concave corner:

```
newTag corner -how orTags convex_corner concave_corner
```

Related Topics

[Pre-Defined Tags](#)

[newTag ... -how ANDTAGS](#)

newTag ... -how OUTSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags source layer fragments that lie completely outside marker layer polygons.

Usage

```
newTag tagName -how OUTSIDEEDGE markerLayer srcLayer [-tag filterTag]
```

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***markerLayer***

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- ***srcLayer***

A required argument specifying the name of the opc or correction type layer containing the edges being evaluated. The new tag is a subset of this layer. This layer must be defined in the setup file.

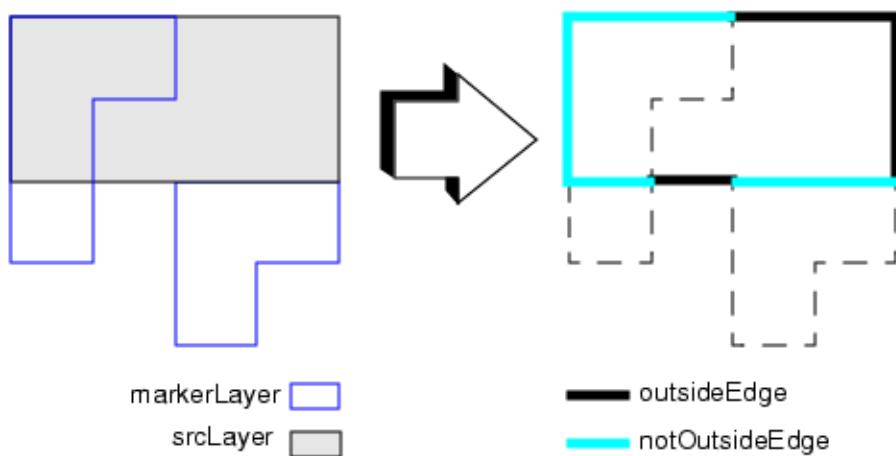
- **-tag *filterTag***

An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation **OUTSIDE EDGE**. Fragments that only touch the border are not included.

Figure 6-90. OUTSIDEEDGE



Related Topics

[newTag ... -how INSIDEEDGE](#)
[newTag ... -how NOTOUTSIDEEDGE](#)
[newTag ... -how COINCIDENTOUTSIDEEDGE](#)
[newTag ... -how TOUCHOUTSIDEEDGE](#)

newTag ... -how SEQUENCE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Identifies a sequence of edges (multiple fragments per edge) and places all fragments that make up the sequence into the output tag set.

Usage

```
newTag tagName -how SEQUENCE filterTag
  {LENGTH length_constraint ANGLE angle_constraint}...
  LENGTH length_constraint [OUTPUT ALL]
```

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***filterTag***
A required argument specifying the tag name for the set of fragments to which the entire sequence of fragments must belong in order for its fragments to be tagged.
- **LENGTH *length_constraint* ANGLE *angle_constraint***
Pairs of **LENGTH** and **ANGLE** constraints that define the sequence edges. You must specify at least one length-angle pair and end the sequence with a final **LENGTH**; the angle following the sequence is irrelevant.
- **OUTPUT_ALL**
An optional argument that instructs the tagging operation to output each fragment in the sequence into a second output tag indicating which edge of the sequence the fragment belongs to. The tag name for these additional output tags are derived from the ***tagName*** by appending *_n*, where *n* is an integer.

Description

The newTag method creates a new tag ***tagName*** identifying a sequence of edges (multiple fragments per edge) and places all fragments that make up the sequence into the output tag set.

A SEQUENCE tagging operation checks whether the operations require an interaction distance greater than or equal to what is set by the program automatically, based on the size of the optical radius. The interaction distance is calculated as:

$$0.5 * \text{hoodpix} + (\text{numsites} - \text{sitecenter} - 1) * \text{sampleSpacing}$$

If the operation requires a larger interaction range, then the tools generate an error during the setup file compilation stage. Use the setup variable declaration [LITHO_SUPPRESS_INTERACTION_ERROR](#) to suppress the setup file error.

```
sse LITHO_SUPPRESS_INTERACTION_ERROR 1
```

Disabling this error is not recommended.

Tip

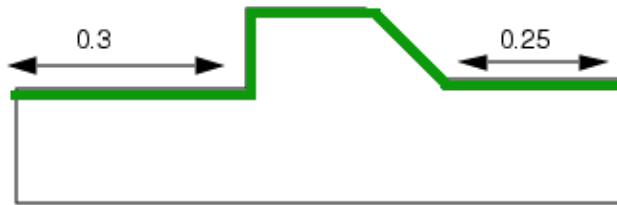
i Best Practice: During length-based tagging, set an upper bound on length checks to keep the tags out of invalid regions. The upper bound value should be less than or equal to the interaction distance. Allowing tags to apply to fragments longer than the interaction distance can cause false tags, which gives bad OPC results.

Examples

Example 1

Tag all fragments shown in the 5-edge sequence shown in green.

Figure 6-91. Sequence Tagging Example



```
newTag t1 -how SEQUENCE all LENGTH == 0.3 ANGLE == 270 \
LENGTH == 0.12 ANGLE = 90 \
LENGTH < 0.2 ANGLE == 135 \
LENGTH < 0.2 ANGLE = 225 LENGTH == 0.25
```

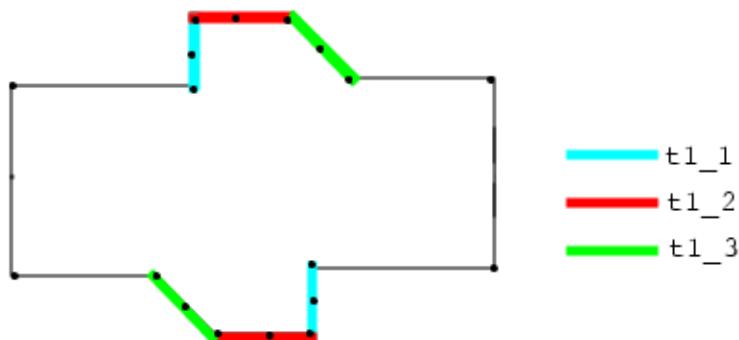
Example 2

Tag all fragments in the specified 3-edge sequence with t1, and also tags all the fragments making up the first edge with t1_1, all the fragments making up the second edge with t1_2, and all the fragments making up the third edge in the sequence with t1_3.

```
newTag t1 -how SEQUENCE all LENGTH == 0.12 ANGLE = 90 \
LENGTH < 0.2 ANGLE == 135 \
LENGTH < 0.2 OUTPUT_ALL
```

This produces the tags shown in [Figure 6-92](#).

Figure 6-92. Sequence Tagging With OUTPUT_ALL



Related Topics

[Pre-Defined Tags](#)

[newTag ... -how EDGE](#)

newTag ... -how SUBTRACTTAGS

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Creates a new tag by subtracting everything in tag2 to tagN from tag1.

Usage

newTag *tagName* -how SUBTRACTTAGS *tag1* *tag2* [*tagN* ...]

Arguments

- ***tagName***
A required argument specifying the name for the new tag.
- ***tag1***
The name of the tag to subtract from.
- ***tag2***
The name of the tag to subtract.
- ***tagN***
Optional arguments specifying additional tag names.

Description

This newTag method creates a tag *tagName* by subtracting everything in *tag2* to *tagN* from *tag1*. Being able to subtract one or more tags from another is useful in creating mutually exclusive tags.

Examples

The following commands creates a new tag called onlyCorners that applies to all convex or concave corners that are not line ends:

```
newTag corner -how orTags convex_corner concave_corner
newTag onlyCorners -how subtractTags corner line_end
```

Another (less efficient) way to create the onlyCorners tag would be the following:

```
newTag corner -how orTags convex_corner concave_corner
newTag notLE -how subtractTags all line_end
newTag onlyCorners -how andTags notLE corner
```

Related Topics

[Pre-Defined Tags](#)

newTag ... -how TOUCHEdge

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all layer1 fragments that are completely or partially coincidental with an edge on layer2.

Usage

newTag *tagName* -how TOUCHEdge *markerLayer* *sourceLayer* [-tag *filterTag*]

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***markerLayer***

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- ***sourceLayer***

A required argument specifying the name of the opc or correction layer containing the fragments being evaluated. The new tag is a subset of this layer.

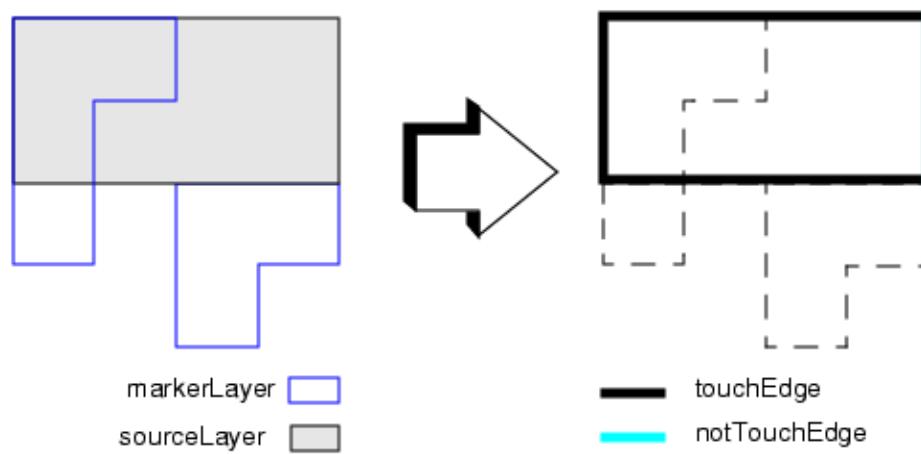
- **-tag *filterTag***

An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation [TOUCH EDGE](#). It tags all ***sourceLayer*** fragments that are completely or partially coincidental with an edge on ***markerLayer*** as shown in [Figure 6-93](#) following.

Figure 6-93. TOUCHEdge



Related Topics

- [newTag ... -how NOTTOUCHEdge](#)
- [newTag ... -how INSIDEEDGE](#)
- [newTag ... -how TOUCHINSIDEEDGE](#)
- [newTag ... -how TOUCHOUTSIDEEDGE](#)

newTag ... -how TOUCHINSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all layer1 fragments that are completely or partially inside coincidental with an edge on layer2.

Usage

`newTag tagName -how TOUCHINSIDEEDGE markerLayer sourceLayer [-tag filterTag]`

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***markerLayer***

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- ***sourceLayer***

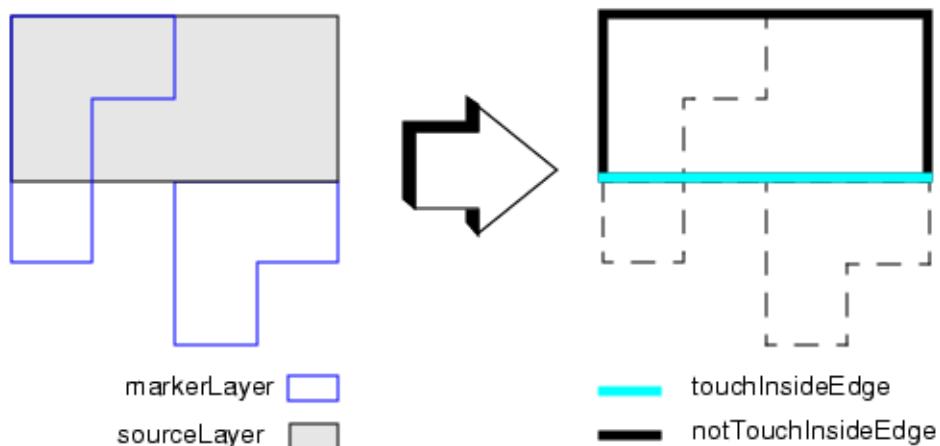
A required argument specifying the name of the opc or correction layer containing the fragments being evaluated. The new tag is a subset of this layer.

- **-tag *filterTag***

An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation **TOUCH INSIDE EDGE**. It tags all ***sourceLayer*** fragments that are completely or partially inside coincidental with an edge on ***markerLayer*** as shown in [Figure 6-94](#) following.

Figure 6-94. TOUCHINSIDEEDGE

Assume that the polygon on the sourceLayer has no fragmentation other than at the corners.

Related Topics

[newTag ... -how TOUCHOUTSIDEEDGE](#)

[newTag ... -how COINCIDENTINSIDEEDGE](#)

[newTag ... -how NOTTOUCHINSIDEEDGE](#)

[newTag ... -how INSIDEEDGE](#)

newTag ... -how TOUCHOUTSIDEEDGE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.

Tags all layer1 fragments that are completely or partially outside coincidental with an edge on layer2.

Usage

`newTag tagName -how TOUCHOUTSIDEEDGE markerLayer sourceLayer [-tag filterTag]`

Arguments

- ***tagName***

A required argument specifying the name for the new tag.

- ***markerLayer***

A required argument specifying the name of the layer containing the polygons against which the fragments are evaluated.

- ***sourceLayer***

A required argument specifying the name of the opc or correction layer containing the fragments being evaluated. The new tag is a subset of this layer.

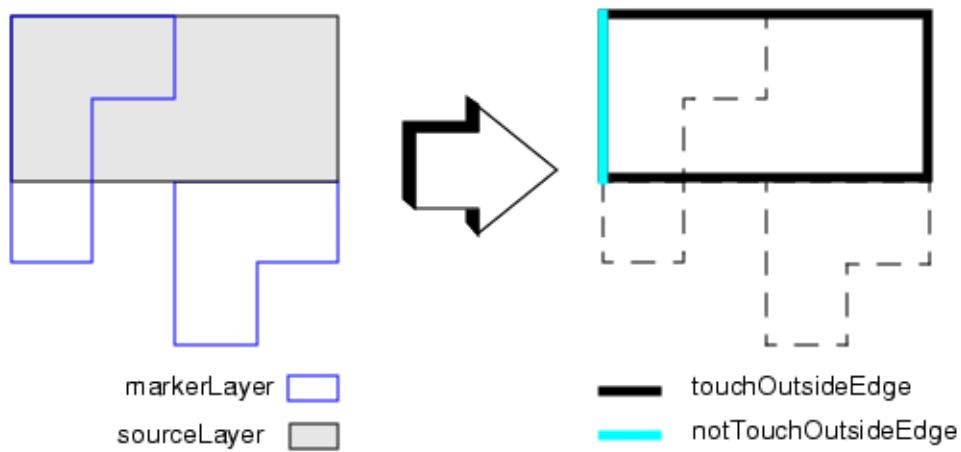
- **-tag *filterTag***

An optional argument specifying filtering for a subset of the fragments in the ***markerLayer*** to be used as the marking fragments.

Description

This method tags fragments that are completely selected by the SVRF operation [TOUCH OUTSIDE EDGE](#). It tags all ***sourceLayer*** fragments that are completely or partially outside coincidental with an edge on ***markerLayer*** as shown in [Figure 6-95](#) following. Notice that if a fragment is both outside and inside coincident, as with the bottom and upper edges of the ***sourceLayer*** in the figure, it is tagged.

Figure 6-95. TOUCHOUTSIDEEDGE



Assume that the polygon on the sourceLayer has no fragmentation other than at the corners.

Related Topics

- [newTag ... -how TOUCHINSIDEEDGE](#)
- [newTag ... -how NOTTOUCHOUTSIDEEDGE](#)
- [newTag ... -how COINCIDENTOUTSIDEEDGE](#)
- [newTag ... -how OUTSIDEEDGE](#)

opcTag

Type: [Tagging Commands](#). Used by Calibre WORKbench and Calibre OPCpro.

Controls the OPC method by tag.

Usage

```
opcTag tag [-unfreeze | -freeze | {-feedback val [val ...]} |  
           {-hintOffset hintDist} | {-fixedOffset offsetDist} |  
           {-hintOffsetIncr incrHintDist} | {-fixedOffsetIncr incrOffsetDist}]
```

Arguments

- ***tag***
A required argument specifying the tag to control with this command.
- **-unfreeze**
An optional argument indicating that fragments in tag can be moved by OPC operations.
This option is equivalent to specifying “opcTag ***tag***” with no other options.
- **-freeze**
An optional switch that prohibits further changes to edges with tag.
- **-feedback *val***
An optional argument specifying feedback on a per tag and per iteration basis. This option takes a varying number of feedback values.

Note

 The -feedback option’s settings override values you specify using the [OPC_FEEDBACK](#) setup variable.

- Example 1

```
# do 3 iterations with high feedback  
opcTag all -feedback -0.9 -0.9 -0.7  
fastIter all 3  
...
```

- Example 2

```
# do 7 iterations with a feedback of -0.3  
opcTag all -feedback -0.3  
fastIter all 7
```

You can also specify different feedback values for different tags:

```
opcTag all      -feedback -0.5 -0.3  
opcTag line_end -feedback -0.7 -0.4  
opcTag serif    -feedback -0.9 -0.1
```

If a fragment is in multiple tag sets, then the most-recently specified feedback takes priority. In the previous example, serif would take priority over line_end and both serif and line_end take priority over all.

- **-hintOffset *hintDist***

An optional argument-value pair defining an offset to apply before performing model-based OPC. As the application moves fragments towards this offset, it checks them against the width and spacing rules defined by [MRC RULE](#) and all related controls. The *hintDist* is expressed in microns. The default is 0.

-hintOffsets are not applied immediately; they are delayed until any operations that require the offset to be known. This allows multiple -hintOffsets to be present and for them to move symmetrically towards the goals without violating constraints.

- **-fixedOffset *offsetDist***

An optional argument-value pair defining a fixed offset to apply to the tag. This argument also turns off model-based OPC for this tag. That is, the fragment is not moved even if it violates an MRC rule. The *offsetDist* is expressed in microns.

The -fixedOffset biases are applied immediately. This is different than -hintOffsets, because constraint checking is not performed.

- **-hintOffsetIncr *incrHintDist***

An optional argument-value pair analogous to -hintOffset except that it adds the *hintDist* amount to whatever displacement was applied at the time this operation is performed. Because this operation also obeys all constraints, the actual offset applied may be less than the specified distance.

After this operation, you typically use the fastIter command to instruct OPCpro to “process” the hints into final bias offsets.

- **-fixedOffsetIncr *incrOffsetDist***

An optional argument-value pair analogous to -fixedOffset except that it adds the *offsetDist* amount to whatever displacement applied at the time this operation is performed. The offset amounts are applied immediately, and can violate constraints, similar to fixedOffset.

Description

This tagging command controls the OPC method for **tag**. You can use this command to:

- Turn model-based OPC off.
- Turn model-based OPC on (after it was turned off).
- Perform biasing as a precursor to model-based OPC.
- Perform rule-based OPC.
- Provide a rule-based hint to model-based OPC.

The command's most common use is to assign a fixed offset to a tag. This is useful for biasing and other rule-based OPC.

Note

 By default, any non-jog fragment on the layer being corrected is subject to correction. You can override this behavior and turn model-based OPC off for a tag using either the -fixedOffset or -freeze options to this tagging command.

Model-based corrections are applied in iterations based on the [iterations](#) keyword in the setup file. The application applies the offsets specified by the opcTag command before any iterations occur. If the iterations keyword is 0, then all fixed offsets are applied, but no model-based OPC is performed.

Depending on the arguments, opcTag offsets are applied to either the position of the fragments before movement or after movement:

- With hintOffset or fixedOffset, the application calculates the fragment position by adding the offset to the original position.
- With hintOffsetIncr or fixedOffsetIncr, the application calculates the fragment position by adding the offset after movement by OPCpro when iterations are specified using [fastIter](#).

Successive opcTag settings for the same *tag* are applied in sequence as the setup file is executed.

Examples

If no options are present, tagged fragments are enabled for OPC. For example, the following example turns off OPC for some fragments, then re-enables them:

```
opcTag t1 -freeze //freezes the fragments
opcTag t1 //unfreezes the fragments
```

One of the most common uses of this command is to assign a fixed offset to a tag. This is useful both for biasing and rule-based OPC. For example, the following command gives a positive fixed offset of 0.02 microns to all line ends to stop line end shortening:

```
opcTag line_end -fixedOffset 0.02
```

Any fragment that is assigned a fixed offset is not moved by model-based OPC. Thus, assigning a fixed offset of 0 to a tag turns off OPC for that tag as in the example below:

```
opcTag no_OPc -fixedOffset 0
```

A similar option is -freeze. Using this stops any further corrections to a tagged fragment. This may be used to stop further corrections for edges which have achieved a sufficiently low EPE,

for instance. Because these edges may have been corrected in the past, the “-fixedOffset 0” option would be inappropriate.

```
opcTag good_enough -freeze
```

Since the `opcTag` command can turn off model-based OPC for a tag using fixed offsets or freezing, there must be some way to turn model-based OPC back on. Using the `opcTag` with no arguments, as shown below, turns model-based OPC back on for *tag*:

```
opcTag tagName
```

The above usage is meant to reverse the effect of a fixed offset which turned off model-based OPC. It does not turn on OPC for *tag* if it is not an opc layer. One occasion where such a command helps is when OPC is performed with two passes, rule-based then model-based. The first `opcTag` command gives a fixed offset to line ends to combat line end shortening with rule-based OPC. Next the `newTag` command tags all line ends which are shortened by more than 0.02 microns after the rule-based fixed offset. The second `opcTag` command then turns model-based OPC back on for the fragments where rule-based OPC was not sufficient.

```
opcTag line_end -fixedOffset 0.02
newTag needModelBasedCorrection -how EPE line_end -10 -0.02
opcTag needModelBasedCorrection
```

Another use of the `opcTag` command is to provide a rule-based hint to model-based OPC. This is useful in speeding up the OPC algorithm. For example, an engineer might discover that for a certain process, almost all line ends are shorter than desired by 0.02 microns. In this case the engineer might want model-based OPC to start with an initial guess of a 0.02 micron positive offset for line ends. This can be achieved with the command below:

```
opcTag line_end -hintOffset 0.02
```

The above command causes model-based OPC to start line ends with a positive correction of 0.02 microns as the initial guess.

Related Topics

[fastIter](#)
[iterations](#)
[fragmentTag](#)
[newTag](#)
[clearTagScript](#)

setMoveLimitForTag

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Limits movements for all edge fragments with the given tag.

Usage

setMoveLimitForTag *tag* -high *max* -low *min*

Arguments

- ***tag***
The specified tag to limit movement.
- **-high *max***
The maximum outward movement distance in microns.
- **-low *min***
The maximum inward movement distance in microns.

Description

Sets movement limits in microns for all edge fragments with a specified ***tag***.

When this command is reached in the evaluation of the tagging commands, the user-specified tag is restricted to an offset specified by the **-high** and **-low** interval. If this command is set multiple times for the same ***tag***, the most restrictive settings are retained.

Examples

The following example uses the setMoveLimitForTag command to limit the movement of the [line_end](#) tag:

```
setMoveLimitForTag line_end -high 0.02 -low -0.01
```

Related Topics

- [fragmentTag](#)
- [opcTag](#)
- [newTag](#)

siteControl DELETE

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Deletes sites on all fragments in tag.

Usage

siteControl *tag* DELETE

Arguments

- *tag*

The specified tag to delete sites from.

Description

Deletes sites on all fragments in a tag.

Note

 The siteControl commands are executed in the sequence specified in the setup file. The final site location is the cumulative result of all applicable siteControl commands, keeping in mind that fragments may belong to multiple tag groups.

Examples

The following examples deletes sites on the line_end tag:

```
siteControl line_end DELETE
```

Related Topics

[newTag ... -how HAS_SITE](#)

siteControl OFFSET_FROM

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Moves sites on all tagged fragments that already have a site, and creates new sites if they do not exist.

Usage

```
siteControl tag OFFSET_FROM {CENTER | LAST | FIRST | CURRENT}  
[site_location [site_center]] [ROTATE angle_from_normal]
```

Arguments

- **tag**

The tag that identifies the fragments on which to move or create sites.

- **OFFSET_FROM**

A required argument defining the reference point used to determine site's new position and orientation. It is used in conjunction with *site_location* and *site_center* arguments. Allowed values are:

CENTER — The center of the fragment. This is the default value.

FIRST — The first endpoint encountered when traversing the polygon clockwise.

LAST — The second endpoint encountered when traversing the polygon clockwise.

CURRENT — Allows the movement to be relative to the existing site position without reference to end of the fragment or center. If CURRENT is used, the movement is incremental. The rotation of the site is not changed and the ROTATE specification is not allowed.

- *site_location*

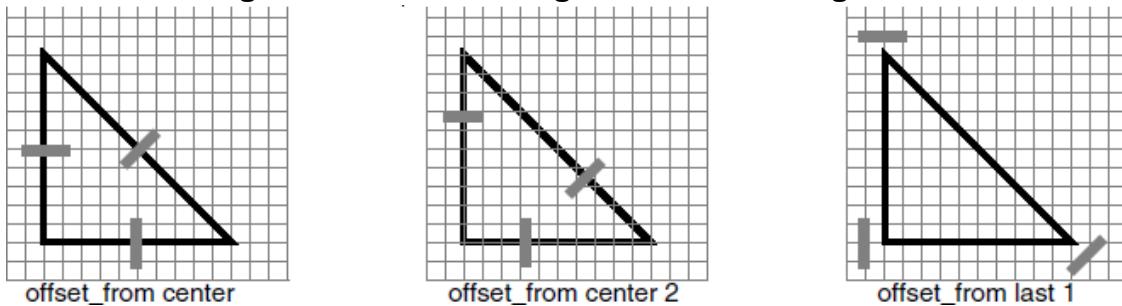
The site location, specified as a distance from the reference point defined by **OFFSET_FROM**, expressed in user units. The default is **0**.

Positive values indicate that the distance is measured from the reference point toward the last vertex of the fragment, negative values indicate that the distance is measured from the reference point toward the first vertex of the fragment.

It is possible to position a site such that it is not actually on the fragment itself. This happens when:

- *site_location* is positive and **OFFSET_FROM** is LAST
- *site_location* is negative. and **OFFSET_FROM** is FIRST
- *site_location* is greater than the distance between the reference point and the vertex.

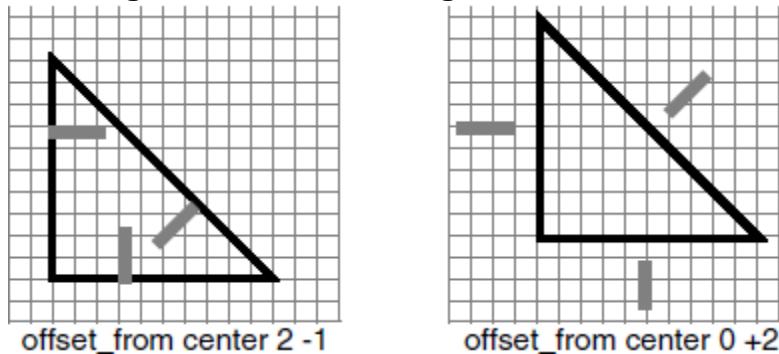
Figure 6-96. Positioning the Site on a Fragment



- *site_center*

The location of the site's center point defined by the `siteinfo` keyword, expressed as a distance perpendicular to the fragment, in user units. Positive values move the center of a site toward outside the polygon, negative values move the center of a site toward the inside of the polygon. The default is `0`.

Figure 6-97. Positioning the Site Center



- *ROTATE angle_from_normal*

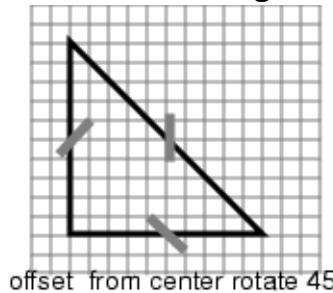
The orientation of the site, expressed as the angle of rotation around site's center point as defined by the `siteinfo` command. The default is `0`, which equates to perpendicular to the fragment.

- Positive values mean counter-clockwise rotation around the site's center point.
- Negative values mean clockwise rotation.

Note

If the setup file contains “`cornerSiteStyle orthog all`”, then sites are still rotated to be perpendicular to the aerial image during the opc operation.

Figure 6-98. Orienting the Site



Description

Defines a placement and orientation for the sites on each of the fragments in the tag. If the fragment already had a site, this moves it. If the fragment did not have a site, it creates one.

The OFFSET_FROM option causes cloning on all transformed cells, thus affecting performance. This cloning is done to achieve siteControl results for normal and mirrored placement of a cell that are consistent with the flat view of the data. The cloning behavior of siteControl in a given setup file can be overridden with the variable [LITHO_ASYM_SOURCE_CELL_CLONE](#).

Note

 siteControl SMOOTH should always be declared prior to siteControl OFFSET_FROM.

The siteControl commands are executed in the sequence specified in the setup file. The final site location is the cumulative result of all applicable siteControl commands, keeping in mind that fragments may belong to multiple tag groups.

Examples

The following is an example of OFFSET_FROM CURRENT:

```
siteControl my_tag OFFSET_FROM CURRENT 0 -0.01
siteControl my_tag OFFSET_FROM CURRENT 0 -0.01
```

These declarations are equivalent to the following:

```
siteControl my_tag OFFSET_FROM CURRENT 0 -0.02
```

This would also be the same as declaring:

```
siteControl my_tag OFFSET_FROM CENTER 0 -0.02
```

This applies only for those edges tagged with my_tag (edges that had the site already located at the center prior to the command).

An example of fragments where that is usually not the case is concave_corner, convex_corner and line_end_adjacent. For these corner types, the CURRENT method allows movement relative to the placement induced by cornerSiteStyle.

Related Topics

[siteControl OFFSET_PROP](#)
[siteControl OFFSET_ROTATED](#)
[sitemodify](#)
[siteinfo](#)
[siteControl SMOOTH](#)
[newTag ... -how HAS_SITE](#)

siteControl OFFSET_PROP

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.

This command is identical to the siteControl OFFSET_FROM command, except that its arguments are expressed as a fraction of the fragment's length, rather than distance.

Usage

```
siteControl tag OFFSET_PROP {CENTER | LAST | FIRST | CURRENT}  
[p_along [p_perp]] [ROTATE angle_from_normal]
```

Arguments

- ***tag***
The specified tag on which to move or create sites.
- **OFFSET_PROP**
OFFSET_PROP arguments define the reference point used to determine a site's new position in conjunction with *p_along* and *p_perp* arguments. **OFFSET_PROP** is **CENTER** by default.
 - CENTER** — The center of the fragment.
 - LAST** — The last point of the fragment.
 - FIRST** — The first point of the fragment.
 - CURRENT** — The current position of the fragment.
- ***p_along***
Distance along the fragment as a fraction of the fragment's length. For example, a value of 0.1 would indicate that a site on a 0.1 micron-long fragment should be moved 0.01 microns. The default value is 0.
Positive values move the center of a site toward the last point of the fragment, negative values move the center of a site toward the first point of the fragment.
If *p_along* is positive and **OFFSET_PROP** is **LAST**, or if *p_along* is negative and **OFFSET_PROP** is **FIRST**, the site is moved past the fragment's endpoints.
- ***p_perp***
Distance perpendicular to the fragment as a fraction of the fragment's length. For example, a value of 0.1 would indicate that a site on a 0.1 micron-long fragment should be moved 0.01 microns perpendicular to the fragment. 0 by default.
Positive values move the center of a site towards the exterior of the polygon, negative values move the center of a site towards the interior of the polygon.
- **ROTATE *angle_from_normal***
An optional argument that specifies how many degrees sites should be rotated. The rotation is around the site's center point as defined by the siteinfo command. 0 by default. Positive

values mean counter clockwise rotation around the site's center point, negative values signify clockwise rotation.

Note

-  If “cornerSiteStyle orthog all” is specified in the setup file, sites are still rotated to be perpendicular to the aerial image during the OPC operation.
-

Description

Moves sites on all fragments in tag that already have a site, and creates new sites on fragments in tag that do not have a site. This command is identical to the siteControl OFFSET_FROM command, except that it's arguments are expressed as a fraction of the fragment's length, rather than in microns.

This command causes cloning of all transformed cells, thus degrading performance. The cloning is done to achieve siteControl results for normal and mirrored placement of a cell that are consistent with the flat view of the data. The cloning behavior of siteControl in a given setup file can be overridden with the variable [LITHO_ASYM_SOURCE_CELL_CLONE](#).

If OFFSET_PROP CURRENT is used, the movement is incremental. The rotation of the site is not changed and the ROTATE specification is not allowed.

The siteControl commands are executed in the sequence specified in the setup file. The final site location is the cumulative result of all applicable siteControl commands, keeping in mind that fragments may belong to multiple tag groups.

Related Topics

- [siteControl OFFSET_FROM](#)
- [siteControl OFFSET_ROTATED](#)
- [sitemodify](#)
- [siteinfo](#)
- [siteControl SMOOTH](#)
- [newTag ... -how HAS_SITE](#)

siteControl OFFSET_ROTATED

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
A command that defines tag-specific site placement for each fragment using the site's coordinate system.

Usage

```
siteControl tag OFFSET_ROTATED {CENTER | LAST | FIRST | CURRENT}  
[d_along [d_perp]] [ROTATE angle_from_normal]
```

Arguments

- **tag**
The specified tag on which to move or create sites.
- **OFFSET_ROTATED**
The **OFFSET_ROTATED** argument defines the reference point used to determine the site's new position in conjunction with *d_along* and *d_perp* arguments. The default is **CENTER**.

CENTER — The center of the fragment.

LAST — The last point of the fragment.

FIRST — The first point of the fragment.

CURRENT — The current position of the fragment.

- *d_along*
A specified distance along the fragment in user units to move the site by. The default is 0. Positive values move the center of a site towards the last point of the fragment, negative values move the center of a site towards the first point of the fragment.

Note

 If *d_along* is positive and **OFFSET_ROTATED** is **LAST**, or if *d_along* is negative and **OFFSET_ROTATED** is **FIRST**, then the site is moved past the fragment's endpoints.

- *d_perp*
A specified distance perpendicular to the fragment in user units to move the site by. The default is 0. Positive values move the center of a site outside of a fragment, negative values move the center of a site toward the inside of a fragment.
- **ROTATE angle_from_normal**
Angle in degrees to rotate around the site's center point as defined in the [siteinfo](#) command. 0 by default.

Positive values mean counter clockwise rotation around the site's center point, negative values mean clockwise rotation.

Note

-  If “cornerSiteStyle orthog all” is specified in the setup file, sites are still rotated until they are perpendicular to the aerial image during the OPC operation.
-

Description

This option is identical to [siteControl OFFSET_FROM](#), except that the site is moved relative to the rotated coordinate system of the site, not relative to the coordinate system of the fragment. While this should only be used with the **CURRENT** (incremental) option, all other options are still accessible.

This operation moves sites on all fragments in *tag* that already have a site, and creates new sites on fragments in *tag* that do not have a site. If the fragment does not have a site, the motion is relative to the coordinate system of the fragment, because the site does not exist. This command behaves like [OFFSET_FROM](#) when there is no preexisting site.

[OFFSET_ROTATED](#) causes cloning of transformed cells and can increase run times. This cloning is done to achieve siteControl results for normal and mirrored placements of a cell that are consistent with the flat view of the data. The cloning behavior of siteControl in a given setup file can be overridden with the [LITHO_ASYM_SOURCE_CELL_CLONE](#) variable.

If **OFFSET_FROM CURRENT** is used, the movement is incremental. The rotation of the site is not changed and the ROTATE specification is not allowed.

The siteControl commands are executed in the sequence specified in the setup file. The final site location is the cumulative result of all applicable siteControl commands, keeping in mind that fragments may belong to multiple tag groups.

Related Topics

[siteControl OFFSET_FROM](#)

[siteControl OFFSET_PROP](#)

[sitemodify](#)

[siteinfo](#)

[siteControl SMOOTH](#)

[newTag ... -how HAS_SITE](#)

siteControl SHIFT_PROP

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.

Moves sites on tagged fragments that already have a site toward a particular corner type, and creates new sites when they do not exist. This command is similar to siteControl SHIFT_TOWARD.

Usage

```
siteControl tag SHIFT_PROP {CORNER | CONVEX | CONCAVE}  
    p_along [p_perp] [LIMIT p_limit_dist]
```

Arguments

- ***tag***
The specified tag on which to move or create sites.
- **SHIFT_PROP**

The **SHIFT_PROP** argument defines the reference point used to determine a site's new position in conjunction with the *p_along* and *p_perp* arguments. **SHIFT_PROP** has no default direction, so one of the following must be specified:

CORNER shifts the site toward the corner on the fragment. If there are no corners or two corners, the site is not shifted.

CONVEX shifts the site toward the convex corner on the fragment. If there is no convex corner on the fragment, the site is not shifted.

CONCAVE shifts the site toward the concave corner on the fragment. If there is no concave corner on the fragment, the site is not shifted.

- ***p_along***
A required argument specifying the distance to move the site as a fraction of the fragment's length along the fragment. For example, a value of 0.1 would indicate that a site on a 0.1 micron-long fragment should be moved 0.01 microns. A positive real number indicates toward the corner, along the fragment's edge. Conversely, a negative real number indicates away from the specified corner. If a site is not actually on the edge, it is moved the specified distance in a direction parallel to the edge. The *p_along* value has no default, and must be specified.
- ***p_perp***
An optional argument specifying the distance to move the site as a fraction of the fragment's length perpendicular to the fragment. A positive real number indicates outward from the fragment while a negative number moves the site inward. If this value is not specified, it defaults to 0, which results in motion parallel to the edge with no perpendicular component.
- **LIMIT *p_limit_dist***
An optional argument that limits site movement to *p_limit_dist*. The distance is given as a fraction of the fragment's length. A value of zero indicates that the site should not move past

a fragment endpoint. A positive real number means that the site can move past the corner up to the limit value. A negative limit distance keeps the site inside the fragment by at least that distance. If this value is not specified, it defaults to zero and the site is prevented from moving past the fragment endpoint. This value restricts only the motion parallel to the edge, and is not considered for perpendicular movements.

Caution

 Negative *p_along* values should be used with care, as the *p_limit_dist* is checked against the specified endpoint. Since the site is being moved away from the specified endpoint rather than toward it, the *p_limit_dist* provides little restriction over site movement. This makes it easy to move a site completely off a fragment to an undesired location.

Description

Moves sites on all fragments in tag that already have a site, and creates new sites on fragments in tag that do not have a site.

This command shifts a site toward the specified corner (assuming that only one corner fits the criteria). If more than one end of an fragment meets the criteria or if neither meets the criteria, the site is not moved at all.

This command is identical to the [siteControl SHIFT_TOWARD](#) command except that its arguments are in fractions of the site's fragment length rather than in microns.

The siteControl commands are executed in the sequence specified in the setup file. The final site location is the cumulative result of all applicable siteControl commands, keeping in mind that fragments may belong to multiple tag groups.

Related Topics

[siteControl SHIFT_TOWARD](#)

[siteControl OFFSET_PROP](#)

[newTag ... -how HAS_SITE](#)

[sitemodify](#)

siteControl SHIFT_TOWARD

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Shifts a control site towards a specified corner type.

Usage

```
siteControl tag SHIFT_TOWARD {CORNER | CONVEX | CONCAVE}  
    d_along [d_perp] [LIMIT limit_dist]
```

Arguments

- ***tag***

The specified tag that identifies fragments on which to move or create sites.

- **SHIFT_TOWARD {CORNER | CONVEX | CONCAVE}**

Specifies the corner type to shift towards:

CORNER — Any corner. If only one corner is present, the site is moved toward the corner. If more than one corner is present or no corners are present, the site is not moved.

CONVEX — Convex corners only. If a convex corner is present, the site is moved toward that corner. If no convex corner or more than one convex corner is present, the site is not moved.

CONCAVE — If a concave corner is present, the site is moved toward that corner. If no concave corner or more than one concave corner is present, the site is not moved.

- ***d_along***

Indicates the distance which the site is to be moved along the fragment. A positive real number indicates the distance that the site is to be moved toward the corner along the fragment's edge. Conversely, a negative real number indicates the distance that the site is to be moved away from the specified corner. If a site is not actually on the edge, it is moved the specified distance in a direction parallel to the edge.

Note

 Negative ***d_along*** values should be used with care, as the *limit_dist* is checked against the specified endpoint. Since the site is moved away from the specified endpoint rather than toward it, the *limit_dist* provides little restriction over site movement.

- ***d_perp***

Indicates the distance that the site is to be moved perpendicular to the fragment. A positive real number indicates that the site is to be moved outward from the fragment while a negative number moves the site inward. If this value is not specified, it defaults to zero, which results in motion parallel to the edge with no perpendicular component.

- **LIMIT *limit_dist***

Indicates a limit to the distance that the site can be moved. A value of zero indicates that the site should not move past a fragment endpoint. A positive real number means that the site can move past the corner up to the distance specified by the limit value. A negative limit distance keeps the site inside the fragment by at least that distance. If this value is not specified, it defaults to zero and the site is prevented from moving past the fragment end point. This value restricts only the motion parallel to the edge, and is not considered for perpendicular movement.

Description

Shifts a site toward the specified corner (assuming that only one corner fits the specified corner type of CORNER, CONVEX, or CONCAVE). If more than one end of a fragment meets the specified corner type, or if neither meets the criteria, the site is not moved.

The siteControl commands are executed in the sequence specified in the setup file. The final site location is the cumulative result of all applicable siteControl commands, keeping in mind that fragments may belong to multiple tag groups.

Examples

The following example shifts the site towards a convex corner (if there is only one convex corner present) by a distance 0.040 nanometers.

```
siteControl my_tag SHIFT_TOWARD CONVEX 0.040
```

Related Topics

[siteControl SHIFT_PROP](#)
[siteControl OFFSET_PROP](#)
[newTag ... -how HAS_SITE](#)
[sitemodify](#)

siteControl SMOOTH

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Moves existing sites on fragments using a 2D Gaussian.

Usage

```
siteControl tag SMOOTH sigma KERNEL_GRID k_g [LAYER L]
[SAMPLE_GRID s_g]
[REJECT_ANGLE r_a]
[ERROR_TAG_ANGLE r_a_tagset]
[ERROR_TAG_DISTANCE s_g_tagset]
[NO_MOVE] [NO_SHIFT] [NO_ROTATE]
```

Arguments

- **tag**
A required argument specifying the tag to control with this command. Only sites on fragments in this tag set are moved.
- **SMOOTH *sigma***
A required argument specifying the standard deviation of the smoothing Gaussian in user units.
- **KERNEL_GRID *k_g***
A required argument specifying the distance between samples of Gaussian kernel in user units. This value is rounded to a positive multiple of [gridsize](#) parameter in the setup file.
- **LAYER *L***
An optional argument specifying the layer to smooth by a Gaussian to determine sites' new positions and orientations. *L* is set to the OPC layer by default.
- **SAMPLE_GRID *s_g***
An optional argument setting the distance in user units between the site center and the four sample points at which convolution is evaluated. If the smoothed location is further than *s_g* away from the original center, the site is not moved, and the corresponding fragment is added to *s_g_tagset*, if [ERROR_TAG_DISTANCE](#) is specified. The *s_g* value is set to *sigma* by default.
- **REJECT_ANGLE *r_a***
An optional argument specifying the maximum angle that is acceptable for the rotation. If the absolute value of the angle between the original and smoothed sites is greater than *r_a* the site is not moved, and the corresponding fragment is added to *r_a_tagset*, if [ERROR_TAG_ANGLE](#) is specified. The value of *r_a* is 30 degrees by default.
- **ERROR_TAG_ANGLE *r_a_tagset***
An optional argument specifying a tag to identify fragments whose sites would have been rotated greater than [REJECT_ANGLE *r_a*](#), and therefore were not rotated at all.

- ERROR_TAG_DISTANCE *s_g_tagset*

An optional argument specifying a tag to identify fragments that fail the SAMPLE_GRID limit and are not moved.

Note



If “cornerSiteStyle orthog all” is specified in the setup file, sites are still rotated to be perpendicular to the aerial image during the opc operation.

- NO_MOVE

An optional argument that causes smoothing to be calculated, but sites remain in place. If ERROR_TAG_ANGLE or ERROR_TAG_DISTANCE is specified, the appropriate fragments are tagged. NO_MOVE is equivalent to specifying both NO_SHIFT and NO_ROTATE.

- NO_SHIFT

An optional argument that prevents the movement of all sites but allows them to be rotated.

- NO_ROTATE

An optional argument that prevents the rotation of all sites but still allows them to be shifted to the 0.5 contour of the SMOOTHING function.

Description

This command moves existing sites on fragments in *tag*. Geometry on layer *L* (opc layer by default) is convolved with a normalized 2D Gaussian with standard deviation *sigma*, and the center point of each site (as defined in [siteinfo](#) commands) on fragments in *tag* is moved to the closest point on a 0.5 contour of the convolution, and rotated so that the new site is perpendicular to the contour. In other words layer *L* is smoothed with a 2D Gaussian and the sites are moved onto the smoothed contour, and rotated to be perpendicular to it.

The siteControl commands are executed in the sequence specified in the setup file. The final site location is the cumulative result of all applicable siteControl commands, keeping in mind that fragments may belong to multiple tag groups.

Detailed Algorithm Description

Two dimensional Gaussian kernel with standard deviation *sigma* is constructed on a grid *k_g* (i.e. the distance between samples of the two dimensional Gaussian is **KERNEL_GRID**).

Kernel’s region of support is truncated to $6 * \sigma$, and the resulting kernel is then normalized to 1.

For each site corresponding to a fragment in *tag*, convolution of geometry on layer *L* and the kernel from the previous step is computed at 5 points: at the center of the site as defined by the [siteinfo](#) command, and 4 points *s_g* distance away from the center point. These 5 convolution values are used in interpolation of the closest point on the 0.5 convolution contour to the site’s center, and the normal direction of the 0.5 convolution contour at that point. Subsequently, the site is moved and rotated accordingly unless one of the following two conditions is encountered.

If the distance between the computed closest point and the site's center exceeds s_g , the site is not moved, and the corresponding fragment is added to s_g_tagset if it is specified.

Otherwise, if the absolute value of the angle between new direction and old direction of the site exceeds r_a , the site is not moved, and the corresponding fragment is added to r_a_tagset if it is specified.

Examples

Example 1 — Smoothing Target

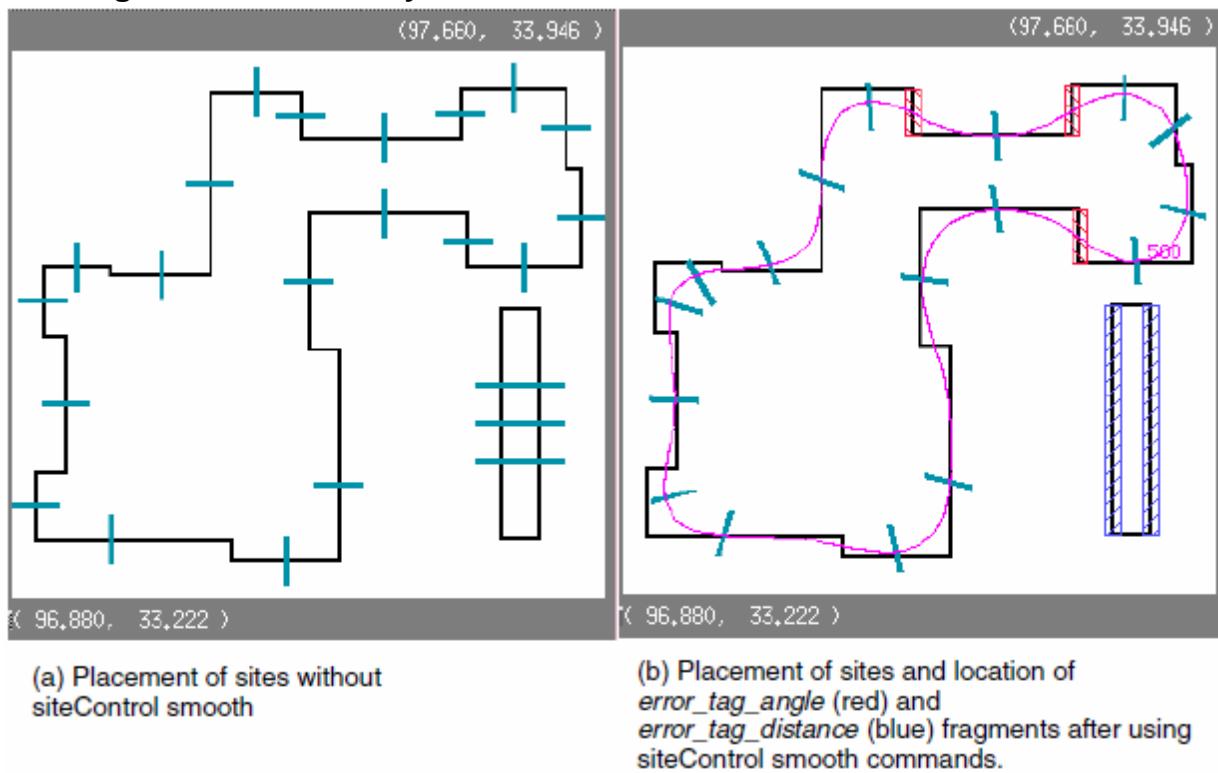
Figure 6-99(a) shows the placement of sites before the siteControl smooth command is applied. Figure 6-99(b) shows the placement of sites and location of r_a_tagset tagged fragments (three red shaded fragments in the upper right) and s_g_tagset tagged fragments (two blue shaded fragments on the sides of the rectangle in the bottom right) after executing the following tagging script.

```
clearTagScript
siteControl all smooth 0.05 kernel_grid 0.01 \
    sample_grid 0.1 reject_angle 50 \
    error_tag_angle eta error_tag_distance etd
siteControl eta delete
siteControl etd delete
tags2boxes -tags eta etd -layers 400 401 -halfWidths 0.01 0.01
tags2boxes -sites -siteLength 0.06 -tags all -layers 500 -halfWidths 0.001
```

Note that sites have moved towards smoothed contour of the geometry. Also note that sites on small jog-like fragments in the upper right satisfy the r_a criteria and are subsequently deleted.

The width of the rectangle in the lower right is 0.05 — the same as the value of ***sigma*** and half the value of s_g . Because the size of the rectangle is on the order of ***sigma*** and s_g , linear interpolation used in determining smoothed positions of the sites predicted smoothed locations to be further than s_g away from the non-smoothed locations. The fragments comprising the sides of the rectangle were therefore added to ***etd*** tag and the sites were subsequently deleted from these fragments.

Figure 6-99. Geometry and Sites With and Without siteControl smooth



(a) Placement of sites without siteControl smooth

(b) Placement of sites and location of *error_tag_angle* (red) and *error_tag_distance* (blue) fragments after using siteControl smooth commands.

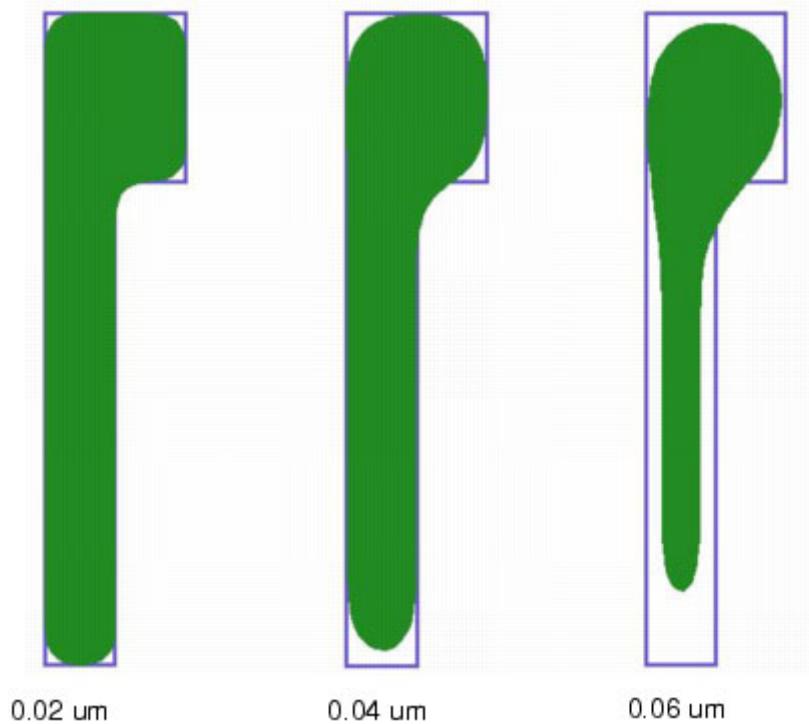
Example 2 — Choosing the Gaussian Sigma Size

It is a critical decision when choosing the sigma for site smoothing and requires an understanding of the effects the Gaussian sigma has. The following example shows the effects of the sigma through the convolution of a mask layer with a Gaussian kernel (filter).

A contour is created at the 0.5 level, after convolving a layer with the Gaussian sigma. Different results depend on the size of the sigma:

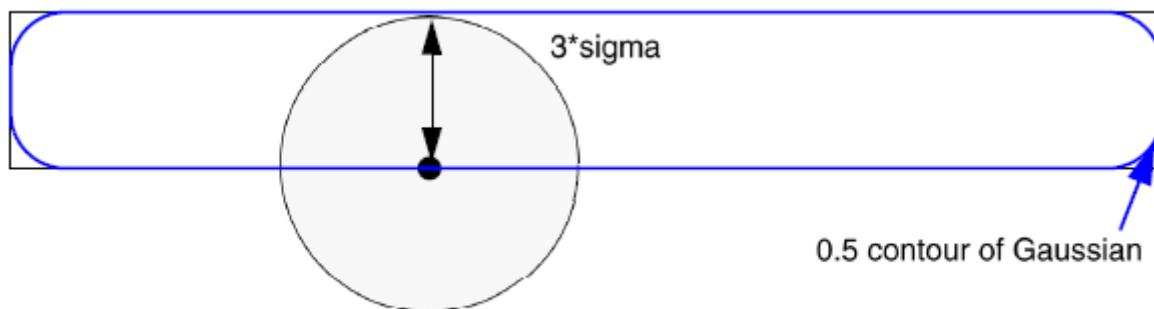
- The larger the Gaussian sigma, the more “blurred” the convolution result.
- The smaller the Gaussian sigma, the more the 0.5 level contour matches the layer itself.

This results from the fact that when the Gaussian sigma approaches 0, the Gaussian filter looks more like an impulse function. [Figure 6-100](#) shows the effect of convolution of Gaussian kernels with 3 different sigmas. The sigma can thus be treated as a control “knob,” allowing you to increase or decrease value to approximate how close to the smooth contour is to the input mask.

Figure 6-100. Convolution of Mask Shape With Gaussians for 3 Sigma Values

An important question to ask is: What is the biggest sigma we can choose that results in the CD of the contour to remain equal to the original layer CD, for 1D structures?

To find the answer, you can define an effective radius of the Gaussian kernel by 3σ . In this case, the convolution of the layer with the filter is almost exactly 0.5 when evaluated on the edges of the layer. This means the 0.5 contour does not shift edge placement (as shown in [Figure 6-101](#)). The value 3σ is a conservative number. You can experiment to find a factor K smaller than 3, such $K\sigma < \min$ feature is sufficient to avoid shifting of 1D edges.

Figure 6-101. Defining Gaussian Kernel Radius

When the minimum feature size is equal to 3σ , the 0.5 contour pattern lies on the 1D edges of the pattern.

sitemodify

Type: [Tagging Commands](#). Used by Calibre OPCpro, Calibre ORC, and Calibre PRINTimage.
Trims the control sites for specific tagged fragments.

Usage

```
[exec] sitemodify layer tag [numnormal value] [numtangent value] [center index]  
[spacing distance]
```

Note

 The “exec” portion of the command is optional and does not explicitly need to be specified in the setup file for Calibre versions 2010.2 and later.

Arguments

- ***layer***
A required argument that specifies the name of the OPC layer on which to modify sites.
- ***tag***
A required argument that specifies the tag name of fragments to modify. Use “all” to modify the sites of all fragments.
- **numnormal *value***
An optional argument setting the number of normal points on the new site. The default is the value selected by [siteinfo](#). The *value* may not exceed the default value.
- **numtangent *value***
An optional argument setting the number of tangent control points on the new site. Control sites usually have 0 or 2 tangent points, but *value* can be larger so long as it does not exceed (siteinfo – numnormal). The default is the value set by siteinfo.
- **center *index***
An optional argument specifying which control point to place on the fragment itself. Points are numbered from 0 to numnormal-1, beginning on the inside of the polygon. The default is the value set by siteinfo.
- **spacing *distance***
An optional argument specifying the spacing between the control points in microns. The default is the value set by siteinfo.

Description

The optional sitemodify keyword changes the control sites of a specific tag set without affecting other control sites. You can change the number of control points, how much of the site is inside or outside the polygon, and how far apart the control points within the site are spaced. This keyword cannot create new control sites.

Note that (numnormal + numtangent) <= (siteinfo num). If “siteinfo RESIST -num 10” is specified, then 10 normal control points and 2 tangent control points are allocated for 12 control points total on any site in the design. All sitemodify commands must allocate no more than 12 control points in that case.

Before applying sitemodify, use siteinfo to set the sites to the largest size needed in the entire design. The sitemodify command can only be used to reduce the number of controls points in a site, not increase them. This may require sites start with extra control points for later adjustment. (This could potentially mean applying sitemodify to all sites in a design, if unusual sites are needed.)

When you modify the number of control points, also modify the center index. The spacing value, however, is set by the model, and generally doesn't need to be modified.

If sitemodify is applied after a siteControl operation which rotates or moves a site, the rotation or movement is preserved. (Sites are modified in place.)

Examples

Example 1 - Trimming Sites

This example shows the combination of siteinfo and sitemodify. It tags regions known to have little curvature, which means sites can be shorter. The name of the opc layer is “target”.

```
# Set the site size to a fairly large value that's known to work. (This
# reserves 14 control points: 12 normal and 2 tangent:)
siteinfo RESIST -spacing 0.020 -num 12 -center 4

# Now create some tags to mark areas that don't need large sites:
newTag straightaway -how SUBTRACTTAGS all convex_corner concave_corner

# Now trim the sites.
# The straight regions are known to have little curvature, so no tangent
# control points are needed. (Experiment with PRINTimage to find
# good values for the number of normal control points and the center
# index value.)
sitemodify target straightaway numnormal 9 numtangent 0 center 2
```

Example 2 - Trimming Sites Near Sensitive Geometry

In this example, the sitemodify command trims sites on a layer that is not critical but adjacent to geometry that is important.

```
# Because the results do not matter too much, using aerial sites
# on the visible layer. The OPC layer receives its normal sites.
sitemodify visible_layer all numnormal 3 center 1
```

Related Topics

[siteControl OFFSET_FROM](#)

[siteControl OFFSET_PROP](#)

[siteControl OFFSET_ROTATED](#)
[siteControl SHIFT_PROP](#)
[siteControl SHIFT_TOWARD](#)
[siteinfo](#)

tags2boxes

Type: [Tagging Commands](#). Used by Calibre WORKbench, Calibre OPCpro, and Calibre ORC.
Creates boxes around tags or sites.

Usage

Tags

```
tags2boxes -tags tag [tag...] -layers layer [layer...] [-halfWidths hw [hw...]]  
[-offsets {off [off...] | epe | final_position}] [-trimEnds trimD] [-markend]
```

Sites

```
tags2boxes -sites -tags tag [tag...] -layers layer [layer...] [-halfWidths hw [hw...]]  
[-siteLength length]
```

Arguments

- **-sites**

When this option is specified, boxes are drawn around the sites that correspond to the fragments in tags specified by **-tags** option.

- **-tags tag**

The tags to save. You may specify multiple tags.

- **-layers layer**

Numerical values used to control the assignment of output data to a specific layer:

- When running in batch mode, you use this value to control which data gets written to which Calibre layer by supplying it as the argument to the MAPNUMBER keyword in the LITHO command.

Note



The tools disallow placing tags2boxes outputs on the same output layer as the OPC geometry in LITHO statements. When MAPNUMBER is not present, an error is flagged if there are tags2boxes in the setup file. You can suppress this error using the variable [LITHO_SUPPRESS_MAPNUMBER_ERROR](#); however, it is not recommended.

- When running in Calibre WORKbench, the application adds this value to the OPC layer bump value to define the layer number for the output data.
- When using MAP in concurrent operations to return a layer name, the -layers option is not required.

- **-halfWidths hw**

The half width of boxes in microns. A value of 10 nanometers is the default value.

- **-siteLength *length***

The length of the box, expressed in microns. By default, the actual length of the site is used. Most sites are created such that some of the site is inside the polygon and some is outside the polygon. When the length of the box does not equal the length of the site, the command positions the box such that the ratio of inside portion to outside portion is the same as for the actual site.

-siteLength can only be used with the **-sites** option.

- **-offsets {*off1* ... *offN* | epe | final_position}**

Choose one:

- **No -offsets specified** — Draws boxes centered over the fragment on the opc layer.
- **-offsets *off1* ... *offN*** — Defines a fixed distance offset specified in microns relative to the outside of the fragment on the opc layer.
- **-offsets epe** — Shifts the drawn boxes based on an offset equal to the EPE for the associated fragment, and filters out boxes for fragments that do not have a site (fragments that do not move do not have a site). The offset is relative to the fragment on the opc layer.
- **-offsets final_position** — Draws boxes centered over the final position of the fragment after OPC. Using -offsets final_position also impacts the lengths of the highlighting boxes. When using final_position, the operation calculates the length based on the length of the fragment after OPC. Thus any adjustments defined through the -trimEnds option apply to this post-OPC length, not the length of the original fragment on the opc layer.

This option cannot be used with the **-sites** option.

- **-trimEnds *trimD***

The amount to trim from the endpoints when creating the boxes. This is useful for visualizing where fragmentation occurred. If the length of the fragment is less than $2 * trimD$, then the trimming is skipped for that fragment. This option cannot be used with the **-sites** option.

- **-markend**

This option is intended to be used with fragments that have been marked with the fragment identification built-in tags (i.e. fragtype_interfeature or fragtype_builtin). Use this option to create a box around the fragment impacted by the initial breakpoint. The size of the box is twice the halfWidths value in width and twice the trimEnds value in length. This option cannot be used with **-sites** option.

Description

The tags2boxes command instructs the application to create boxes to help inspect fragment data. It can operate in either of two modes:

- TAG — Draws boxes around the fragments in the specified tags. (default)
- SITE — Draws boxes around sites on fragments in the specified tags.

For each tag, you must define the layer on which the boxes are drawn. You can create up to 256 tags2boxes output layers.

Optional arguments give you tag-by-tag control over the width and offset used for drawing the boxes. When **-sites** option is specified, the length of the boxes can be set using an optional **-siteLength** argument.

To output these multiple tags2boxes layers, you can use concurrent operations. For information on concurrent operations, refer to “[Calibre Layer Output](#)” on page 606.

Examples

```
tags2boxes -tags fragtype_corner -layers 62 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

The example finds all fragments created by a corner and marks them with a box. This is then output on layer 62.

Related Topics

[Pre-Defined Tags](#)

Pre-Defined Tags

The built-in tags are used to globally select specific types of fragments, such as mid-edge segments or fragments at the initial fragmentation points. They can be used in any command calling for a tag.

To create user-defined tags use one of the [newTag](#) commands.

all

Selects every fragment. Use in conjunction with [newTag ... -how SUBTRACTTAGS](#) to locate fragments with no other tag.

all_angle_line

Selects all skew fragments. In other words, all_angle_line selects all fragments that are not horizontal, vertical, or at 45 degrees, 135 degrees, 225 degrees, or 315 degrees.

angle_45_line

Selects all fragments that are at 45 degrees, 135 degrees, 225 degrees, or 315 degrees.

concave_corner

Selects all fragments that touch a concave corner. The concave_corner tag ignores [concaveAdjDist](#). Fragments that touch a concave corner are selected even if they extend beyond the distance.

concave_corner_adjacent

Selects all fragments that are completely within [concaveAdjDist](#) of a concave corner. Fragments that have only one endpoint within the distance are not selected.

convex_corner

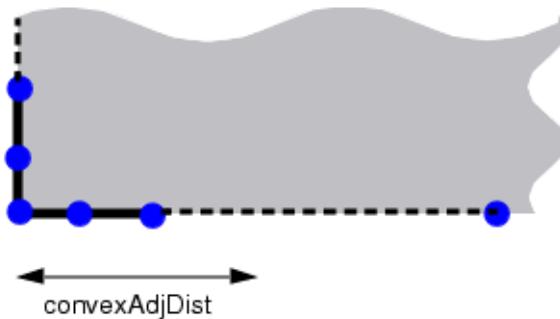
Selects all fragments that touch a convex corner. In [Figure 6-102](#), only the two inner fragments at the corner are selected by convex_corner.

The convex_corner tag ignores [convexAdjDist](#). Fragments that touch a convex corner are selected even if they extend beyond the distance.

convex_corner_adjacent

Selects all fragments that are completely within [convexAdjDist](#) distance of a convex corner. Fragments that have only one endpoint within the distance are not selected. In [Figure 6-102](#), all four fragments shown in thick black are selected, but not the fragments with a broken line as they extend past convexAdjDist.

Figure 6-102. Convex Corner Fragments



fragtype_builtin

Selects all fragments whose initial breakpoint was created by [Implicit Fragmentation](#). This includes fragments that were created to accommodate cell boundaries and tile boundaries, or

promoted fragments from cells lower in the hierarchy. There are no explicit commands to create these kinds of fragments; they are created by operations built in to Calibre.

For example, to find all fragments created by intrinsic fragmentation and mark them with a box on layer 61:

```
tags2boxes -tags fragtype_builtin -layers 61 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

fragtype_corner

Selects all fragments whose initial breakpoint was created by a [concavecorn](#), [cornedge](#), or [intrafeature](#) command.

For example, to find all fragments created by a corner and mark them with a box on layer 62:

```
tags2boxes -tags fragtype_corner -layers 62 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

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.

For example, to find all fragments with endpoints from the original edges and mark them with a box on layer 60:

```
tags2boxes -tags fragtype_endpt -layers 60 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

fragtype_interfeature

Selects all fragments whose initial breakpoint was created by an [interfeature](#) command.

For example, to find all fragments whose initial breakpoint was created by an interfeature fragmentation operation and mark them with a box on layer 63:

```
tags2boxes -tags fragtype_interfeature -layers 63 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

fragtype_layer

Selects all fragments whose initial breakpoint was created by an asraf, correction, external, island or visible layer, as well as layers declared with a command such as [coincidentLayer](#) or [fragmentLayer](#).

For example, to find all fragments whose initial breakpoint came from a layer operation and mark them with a box on layer 64:

```
tags2boxes -tags fragtype_layer -layers 64 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

fragtype_max

Selects all fragments initially created by [maxedgelength](#).

For example, to find all fragments whose initial breakpoint came from a maxedgelength operation and mark them with a box on layer 65:

```
tags2boxes -tags fragtype_max -layers 65 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

fragtype_other

Selects all fragments whose initial breakpoint was created by a custom operation such as LAPI. Use this command for newly created fragmentation operations that do not fit into other fragtype built-in tags.

For example, to find fragments that cannot be selected by other built-in tags and mark them with a box on layer 67:

```
tags2boxes -tags fragtype_other -layers 67 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

fragtype_tag

Selects all fragments whose initial breakpoint was created from a [fragmentTag](#) operation. (This includes the interfeature, intersection, and other specialized forms.)

For example, to find all fragments whose initial breakpoint was created by a fragmentTag operation and mark them with a 0.040 by 0.040 micron box and output the result on layer 66:

```
tags2boxes -tags fragtype_tag -layers 66 -halfWidths 0.020 \
-trimEnds 0.020 -markend
```

horizontal_line

Selects all horizontal fragments. For hierarchical designs, fragments in cell references are tagged based on the orientation within the cell, not in the design. Thus, if the cell reference is rotated, fragments are evaluated as if the angle of rotation were 0.

If the [LITHO_ASYM_SOURCE_CELL_CLONE](#) variable is set to FORCE, the tag is inconsistent with regard to the whole database, not just the cell. This also applies to the [vertical_line](#) tag.

line_end

Selects all fragments that are on a line end edge. Line ends are defined as edges less than or equal to [lineEndLength](#) between two convex corners, whose other sides are longer than [lineEndLength](#).

line_end_adjacent

Selects fragments that share a corner with a line end fragment and are on edges with a length greater than [lineEndLength](#).

opcable

Selects fragments that can be moved during OPC; that is, any fragment not belonging to a tag group frozen with [opcTag](#) -fixedOffset or -freeze.

space_end

Selects all fragments that are on a space end edge. Space ends are defined as a length less than or equal to [minfeature](#) with two concave corners, whose other sides have lengths greater than [minfeature](#).

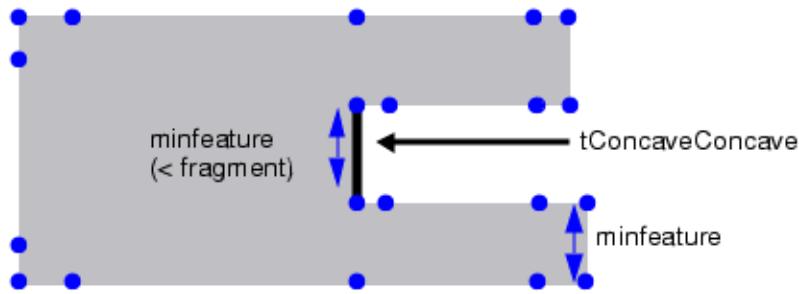
space_end_adjacent

Selects fragments that share a corner with a space end fragment, and are on edges with length greater than or equal to [minfeature](#).

tConcaveConcave

Selects fragments with two concave corners but with length greater than [minfeature](#) as shown in [Figure 6-103](#). These are fragments that look like space ends but are too wide.

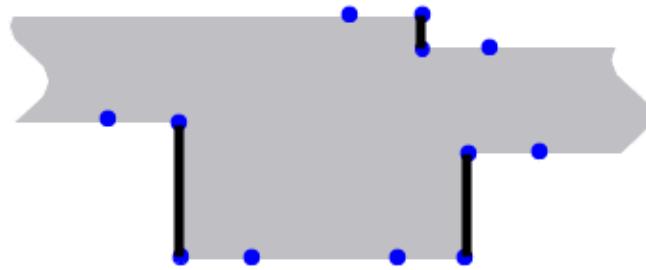
Figure 6-103. tConcaveConcave



tConcaveConvex

Selects fragments touching one concave corner and one convex corner as shown by the heavy black lines in [Figure 6-104](#). The corners can be in any order.

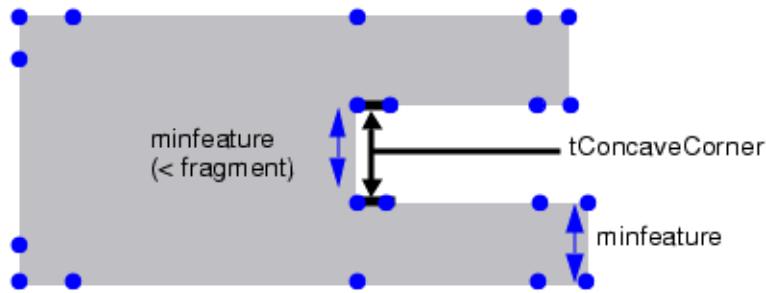
Figure 6-104. tConcaveConvex



tConcaveCorner

Selects fragments that touch a tConcaveConvex fragment as shown in Figure 6-105.

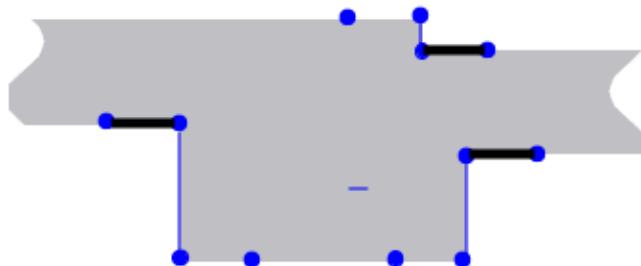
Figure 6-105. tConcaveCorner



tConcaveStepAdjacent

Selects fragments that touch a concave corner shared with a tConcaveConvex fragment. In Figure 6-106, tConcaveStepAdjacent fragments are in heavy black, and tConcaveConvex fragments are in thin blue. Compare with tConvexStepAdjacent.

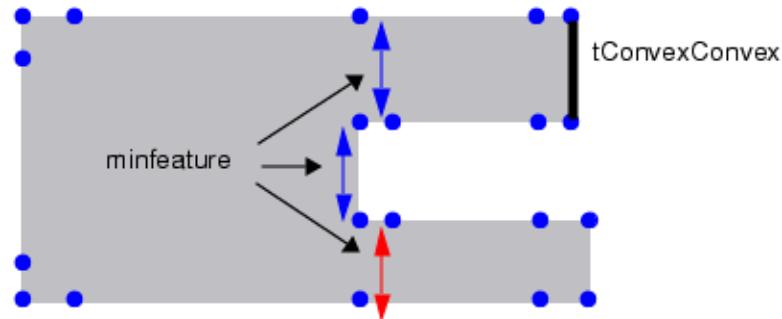
Figure 6-106. tConcaveStepAdjacent



tConvexConvex

Selects fragments with two convex corners but with length greater than `minfeature` or `lineEndLength`, if defined. These are fragments that look like line ends but are too wide or whose sides are too short.

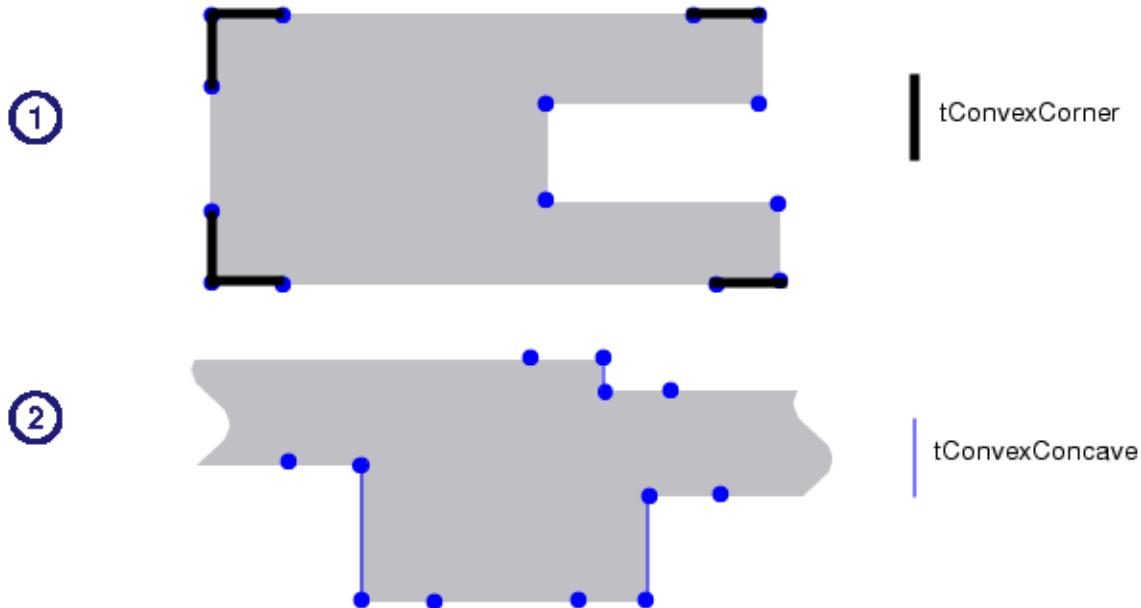
Figure 6-107. tConvexConvex



tConvexCorner

Selects fragments with only one convex corner that are not adjacent to `space_end`, `tSpaceEnd`, or `tConcaveConvex`. Notice that in Figure 6-108, line ends are not selected because line-end fragments have two convex corners. For shape 2 in the figure, no fragments are selected because a `tConcaveConvex` fragment (thin blue line) shares the convex corner.

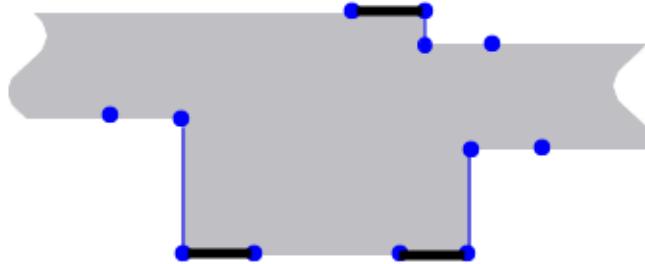
Figure 6-108. tConvexCorner



tConvexStepAdjacent

Selects fragments that share a convex corner with a [tConcaveConvex](#) fragment, as shown in [Figure 6-109](#). Thin blue lines are [tConcaveConvex](#) fragments and thick black lines are [tConvexStepAdjacent](#). Compare with [tConcaveStepAdjacent](#).

Figure 6-109. tConvexStepAdjacent



tLineEnd

Selects all fragments on a line end. This tag is identical to [line_end](#).

tLineEndAdjacent

Selects fragments that share a corner with a [tLineEnd](#) fragment. This tag is identical to [line_end_adjacent](#).

tSpaceEnd

Selects all fragments on a space end. This tag is identical to [space_end](#).

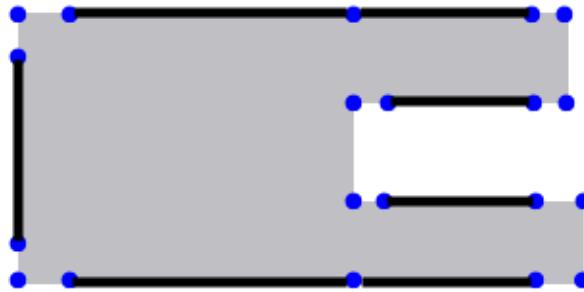
tSpaceEndAdjacent

Selects fragments that share a corner with a [tSpaceEnd](#) fragment. This tag is identical to [space_end_adjacent](#).

tStraight

Selects fragments with no corners, as shown by the thick black lines in [Figure 6-110](#).

Figure 6-110. tStraight



vertical_line

Selects all vertical fragments. For hierarchical designs, fragments in cell references are tagged based on the orientation within the cell, not in the design. Thus, if the cell reference is rotated, fragments are evaluated as if the angle of rotation were 0.

If the [LITHO_ASYM_SOURCE_CELL_CLONE](#) variable is set to FORCE, the tag is inconsistent with regard to the whole database, not just the cell. This also applies to the [horizontal_line](#) tag.

Chapter 7

Model-Based OPC Examples

The Calibre OPCpro batch tool performs optical process correction (OPC) on a mask layout to compensate for distortions introduced during the lithography process used to create a wafer. The resultant output is a mask layer, or set of mask layers, containing a modified version of the original data enhanced with features such as serifs, biasing, and line-end hammer heads.

Successful printing at 65nm and below requires advanced resolution enhancement techniques (RET) such as alternating PSM, multisource illumination systems (dipole or quadrupole), or process window based OPC. You can use Calibre nmOPC and Calibre OPCverify (“dense OPC”) tools, or use Matrix OPC in Calibre OPCpro. (Contact your Siemens representative if you have questions deciding which is more appropriate for your process technology.) Matrix OPC examples are in the last two sections of the chapter.

| | |
|---|------------|
| OPC Examples | 552 |
| Scenario 1: Basic OPC | 552 |
| Scenario 2: OPC With Sub-Resolution Assist Features (SRAFs) | 556 |
| Scenario 3: OPC With Printing Assist Features (PAFs)..... | 560 |
| Scenario 4: Etch Retargeting With Calibre TDopc and Calibre OPCpro..... | 565 |
| Scenario 5: OPCpro Concurrent Operations..... | 570 |
| Calibre OPCpro Example..... | 572 |
| Using Matrix OPC in Calibre OPCpro | 575 |
| Matrix OPC for Multiple Masks and Complex EPE | 575 |
| Methods of Matrix OPC Fragment Mapping | 577 |
| Matrix OPC Mapping Controls..... | 579 |
| Viewing Full-Chip Mappings With Visualize Matrix OPC | 580 |
| Matrix OPC Scenarios..... | 581 |
| Example 1: Multiple Tolerances..... | 581 |
| Example 2: Matrix OPC Followed by Calibre ORC | 582 |
| Example 3: Using Enclosure Checks With Matrix OPC | 585 |
| Example 4: Matrix OPC for Alternating Phase Shift Masks | 588 |

OPC Examples

There are a variety of methods by which Calibre OPCpro can be used to correct mask layouts, each depending on your design. The following sections describe five design and correction scenarios, including example setup and SVRF DRC files for illustration purposes.

| | |
|--|------------|
| Scenario 1: Basic OPC..... | 552 |
| Scenario 2: OPC With Sub-Resolution Assist Features (SRAFs) | 556 |
| Scenario 3: OPC With Printing Assist Features (PAFs) | 560 |
| Scenario 4: Etch Retargeting With Calibre TDopc and Calibre OPCpro..... | 565 |
| Scenario 5: OPCpro Concurrent Operations | 570 |

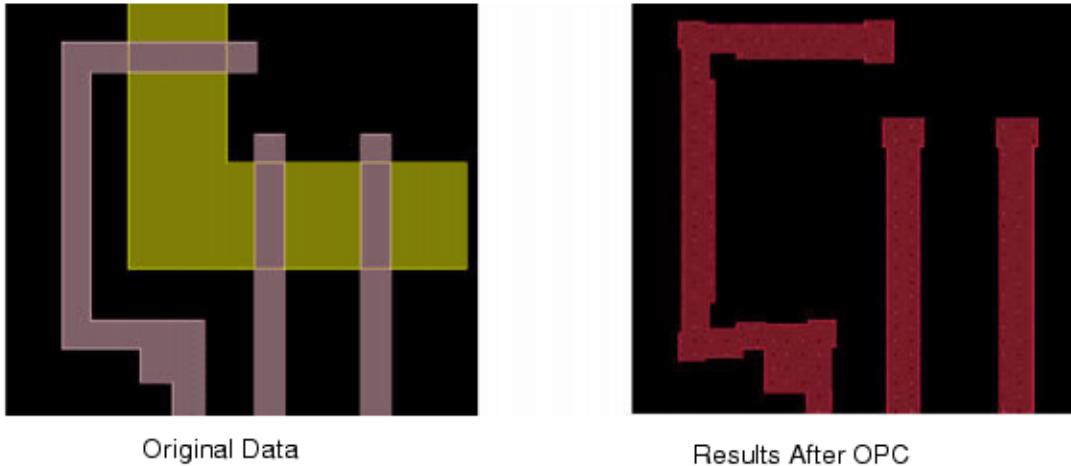
Scenario 1: Basic OPC

This scenario describes applying model-based OPC to the POLY layer of a 180-nanometer design.

Prerequisites

- GDS or OASIS layout database.
- A resist model. See “[VT5 Resist Model Creation](#)” in the *Calibre WORKbench User’s and Reference Manual* if you need to create a resist model.
- An optical model. See “[Optical Model Creation](#)” in the *Calibre WORKbench User’s and Reference Manual* if you need to create an optical model.

Figure 7-1. Before and After OPC



Method

1. Write a setup file ([Example 7-1](#)) in which you:
 - a. Define a layer containing data to be corrected. (The type must be set to opc.)

- b. Set the OPC algorithm parameters (such as gridsize and stepsize) as required for the process node you are using.
 - c. Define a fragmentation scheme appropriate for the process node you are using. (See “[Fragmentation Strategies](#)” on page 80.)
2. Write an SVRF rule file ([Example 7-2](#)) to:
 - a. Read in the layer to be corrected from the layout database.
 - b. Invoke Calibre OPCpro through the [LITHO OPC](#) operation, passing it the names of the layer to be corrected and the setup file.
 - c. Save the results as a layer in a new (output) layout database.
 3. Run Calibre using this SVRF rule file as input.
 4. View the results in Calibre WORKbench or your favorite viewer.

Example 7-1. Setup File for Basic OPC (setup.in)

```
# ----- Simulation models -----
modelpath ./models
opticalmodel lab.opt
resistpolyfile lab.mod

# ----- OPC algorithm -----
#
# Define parameters that control simulation and correction
#
iterations 4
tilemicrons 160.000000
stepsize 0.001
gridsize 0.001
siteinfo RESIST
cornerSiteStyle SITES_ON_ARC
lineEndAdjDist 0.190000
convexAdjDist 0.140000
concaveAdjDist 0.140000

# ----- Fragmentation -----
#
# Define the process for breaking edges into fragments
#
minfeature 0.180000
minedgelength 0.130000
maxedgelength 1000.000000
cornedge 0.130000
concavcorn 0.130000
interfeature -interdistance 0.290000 -ripplelen 0.130000 -num 0
seriftype 0
minjog 0.120000
lineEndLength 0.320000
```

```
# ----- Layer info -----
#
# Use the layer keyword to define the input layer.
#
background clear
# Layers
layer 4 POLY 17 0 opc dark

-- Arbitrary Commands Can Follow This Line. Don't delete this
line!--
sse LITHO_PROMOTION_TILING 1

MRC_RULE INTERNAL {USE 0.13}
MRC_RULE EXTERNAL {USE 0.16}
```

Example 7-2. SVRF Rule File for Basic OPC

```
LAYOUT SYSTEM GDS
LAYOUT PATH "gds/example.gds"
LAYOUT PRIMARY "*"
PRECISION 1000

DRC MAXIMUM RESULTS ALL
DRC SUMMARY REPORT "reports/example.rep"
DRC RESULTS DATABASE "gdsout/example_out.gds" GDS
LAYER POLY 4

MY_OPCT {
    LITHO OPC FILE "setup/setup.in" POLY
}

DRC CHECK MAP POLY 4
DRC CHECK MAP MY_OPCT 104
```

What Happens During Basic OPC

Figure 7-2. Fragmentation

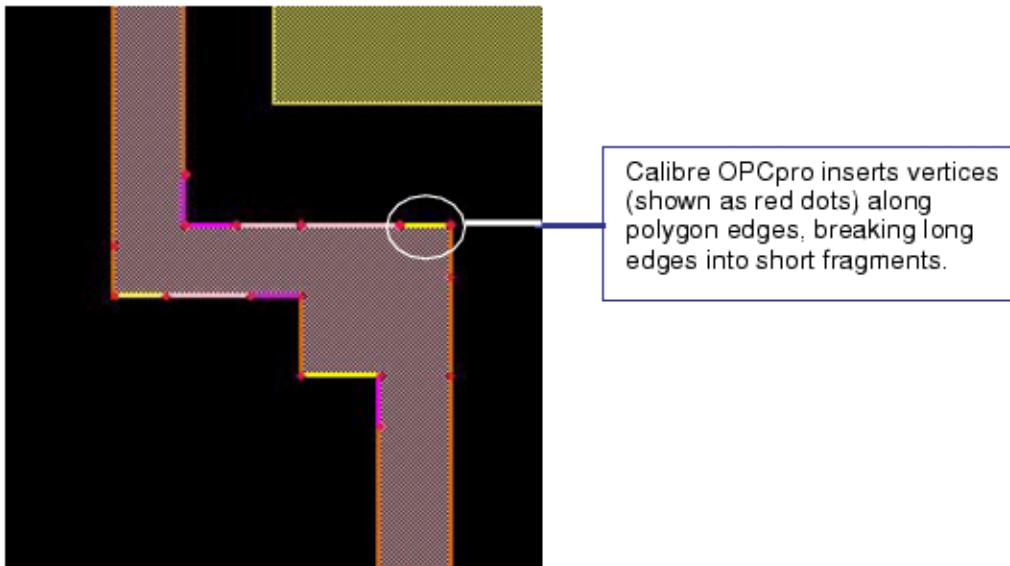


Figure 7-3. Simulation

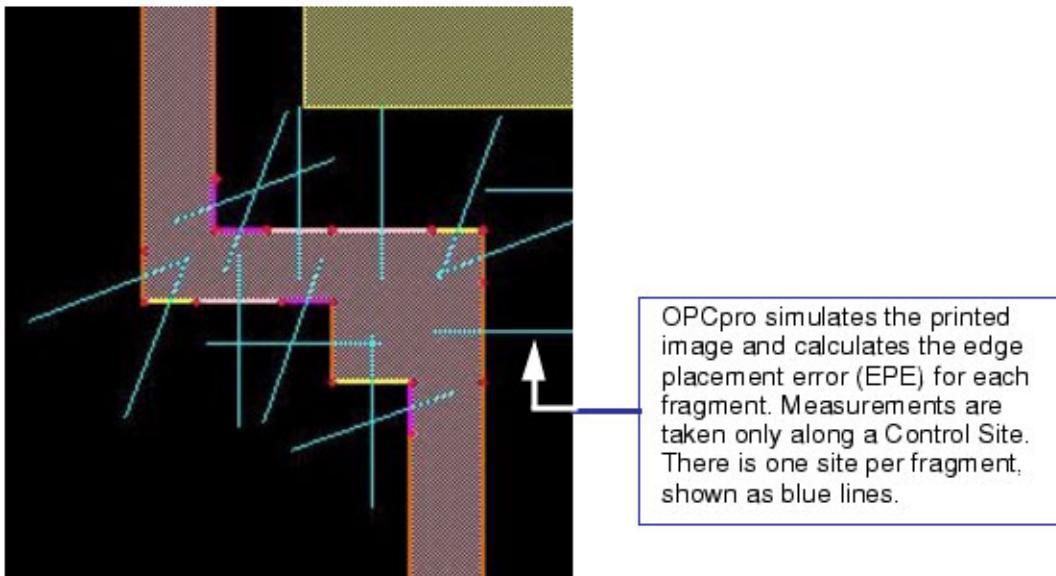
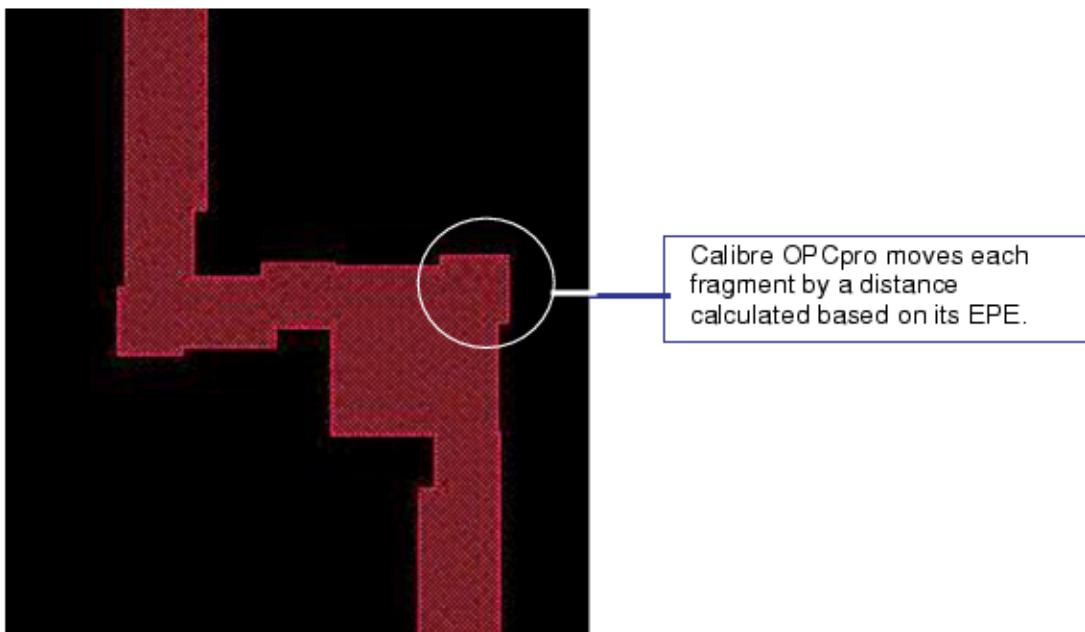
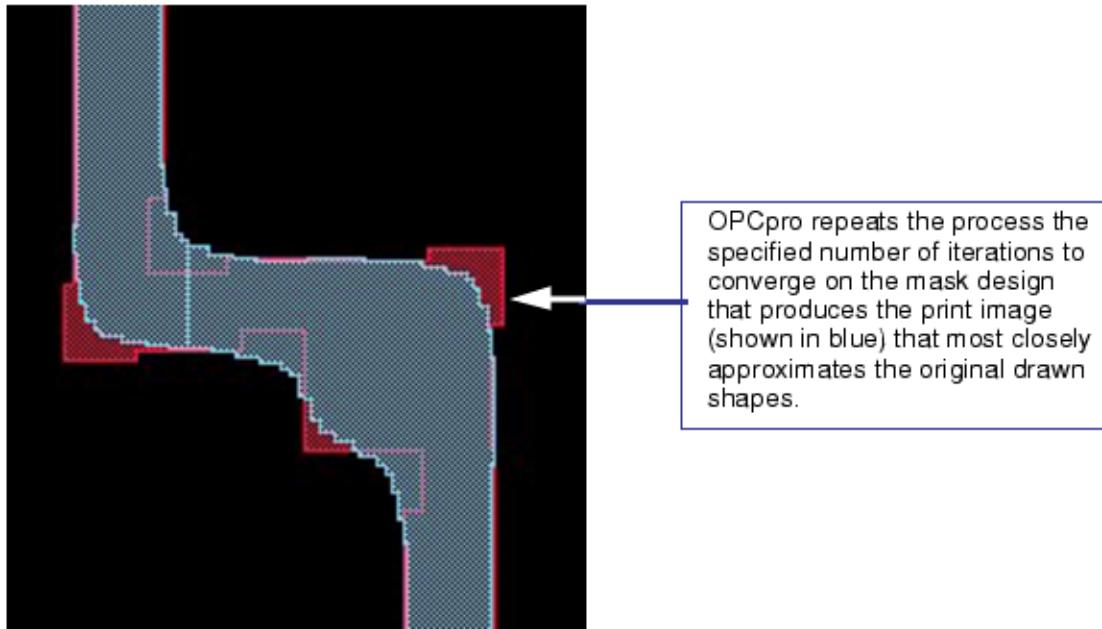


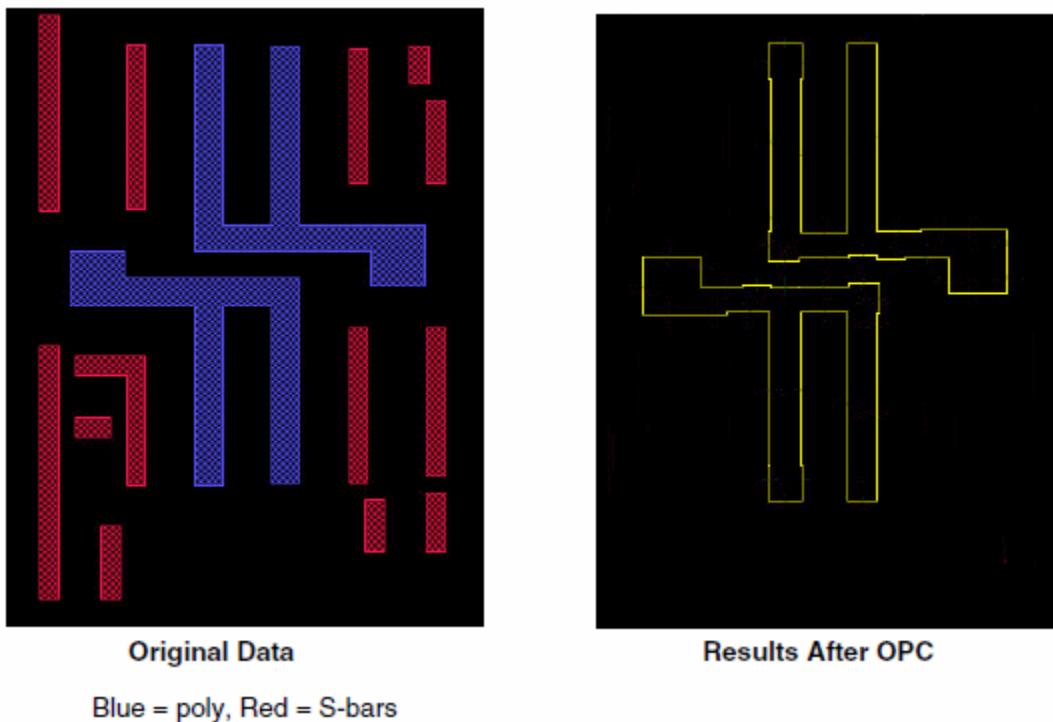
Figure 7-4. Correction**Figure 7-5. Final Results**

Scenario 2: OPC With Sub-Resolution Assist Features (SRAFs)

For this scenario, you are using sub-resolution assist features (SRAFs) such as scattering bars. SRAFs are not printed on the final mask.

Figure 7-6 shows the original data as well as the final print image simulation following the OPC with SRAFs process. The example used to illustrate this scenario is a 90-nanometer design.

Figure 7-6. Layer Data for OPC With SRAFs



Blue = poly, Red = S-bars

Method

1. Write a setup file ([Example 7-3](#)). In the setup file:
 - a. Define a fragmentation scheme. (See “[Fragmentation Strategies](#)” on page 80.)
 - b. Define the following layers in the setup file:
 - A layer containing mask data to be corrected (type opc).
 - A layer containing SRAFs such as scattering bars (type visible).
 - c. Set the OPC algorithm parameters.
2. Write the SVRF DRC rule file ([Example 7-4](#)) to:
 - a. Read in the mask layer.
 - b. Generate SRAFs using Calibre® OPCsbar™ or other tools.
 - c. Invoke Calibre OPCpro through the [LITHO OPC](#) operation, passing the OPC layer, the SRAF layer, and setup file.
 - d. Save the results to a new layer.
3. Run Calibre using this SVRF file as input.

-
4. View the results in Calibre® WORKbench or your favorite viewer.

Example 7-3. Setup File for OPC With SRAFs

```

# ----- Simulation models -----
modelpath models
opticalmodel opt
resistpolyfile res.mod

# ----- OPC algorithm -----
iterations 4
tilemicrons 100.000000
stepsize 0.001
gridsize 0.001
siteinfo RESIST -center 6 -numx 17
cornerSiteStyle CONSERVATIVE
lineEndAdjDist 0.220000
convexAdjDist 0.220000
concaveAdjDist 0.220000

# ----- Fragmentation -----
minfeature 0.090000
minedgelen 0.075000
maxedgelen 50.000000
cornedge 0.100000 0.100000 priority non90 0.120000 lea 0.080000 0.100000
concavecorn 0.100000 0.100000 non90 0.120000
interfeature -interdistance 0.480000 -ripplelen 0.100000 -num 1
seriftype 0
minjog 0.055000
lineEndLength 0.210000

# ----- Layer info -----
#
# Two layers are defined, the opc layer POLY and the SRAF layer
# SBAR (containing scattering bars). Placing the scattering bars on
# a visible layer allows them to be taken into account for the
# simulation, but no fragmentation or OPC is performed on them.
#
background clear
layer 1 POLY 17 0 opc atten 0.06
layer 3 SBAR 145 0 visible atten 0.06

-- Arbitrary Commands Can Follow This Line. Don't delete this line! --
sse LITHO_PROMOTION_TILING 1

MRC_RULE INTERNAL {USE 0.075 0.02}
MRC_RULE EXTERNAL {USE 0.065 0.02}

```

Example 7-4. SVRF DRC File for OPC With SRAFs

```
LAYOUT SYSTEM GDS
LAYOUT PATH "gds/lab1.gds"
LAYOUT PRIMARY "*"
PRECISION 1000

DRC MAXIMUM RESULTS ALL
DRC SUMMARY REPORT "reports/lab1.rep"
DRC RESULTS DATABASE "gdsout/lab1_out.gds" GDS
LAYER POLY 1
LAYER SBAR 3

MY_OPCT = LITHO OPC FILE "setup/setup.in" POLY SBAR

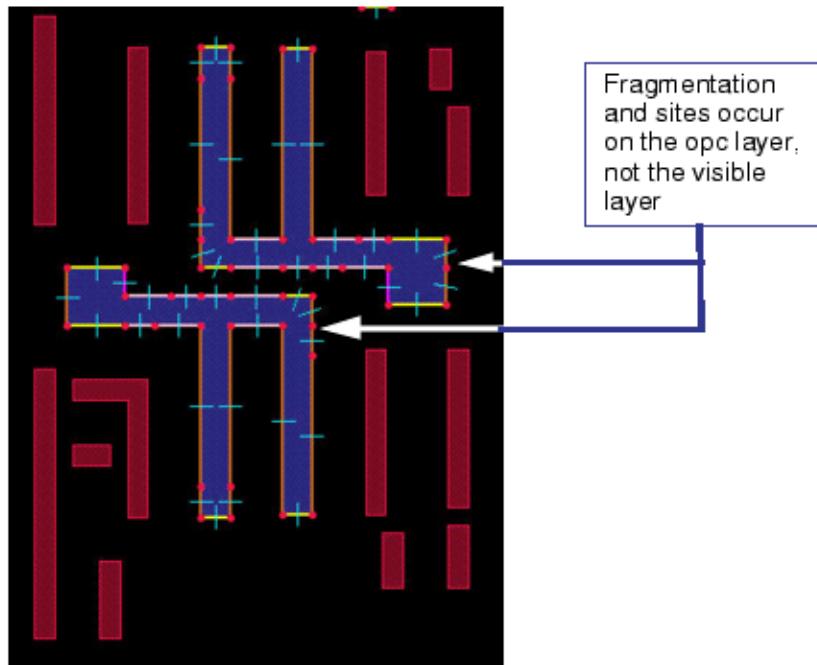
DRC CHECK MAP POLY 1
DRC CHECK MAP MY_OPCT 104
```

What Happens During OPC With SRAFs

Assist features such as scattering bars are created using tools such as Calibre® OPCsbar. The assist features are placed on a visible layer, which enables Calibre OPC to account for the presence of the features (for purposes of rule checking the original polygons and creating an accurate optical model), but not attempt to correct them.

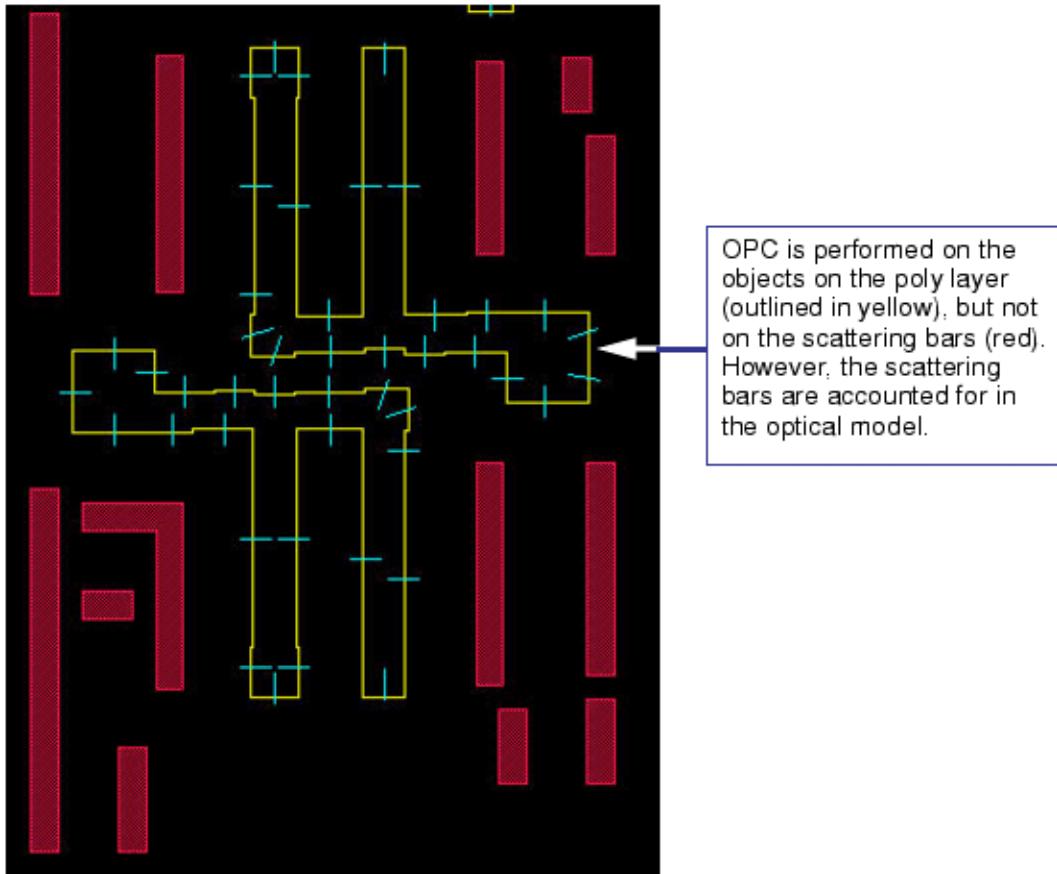
Only the shapes on the original polygon layer are fragmented (see [Figure 7-7](#)). The scattering bars on the visible layer are not corrected as they do not appear on the final printed mask. No sites are placed on features on a visible layer.

Figure 7-7. Fragmentation for Design With SRAFs



When OPC is run, the fragmented edges are moved for correction (Figure 7-8). The scattering bars are not adjusted, but do influence the outcome of the optical modeling. The final print image simulation is also performed on the polygon layer, not the visible layer.

Figure 7-8. OPC Results With SRAFs



Scenario 3: OPC With Printing Assist Features (PAFs)

This scenario uses assist features such as layer fill that print on the final mask but should not be corrected by the OPC process. It is possible to have both printing features and non-printing features (such as scattering bars) in the same design.

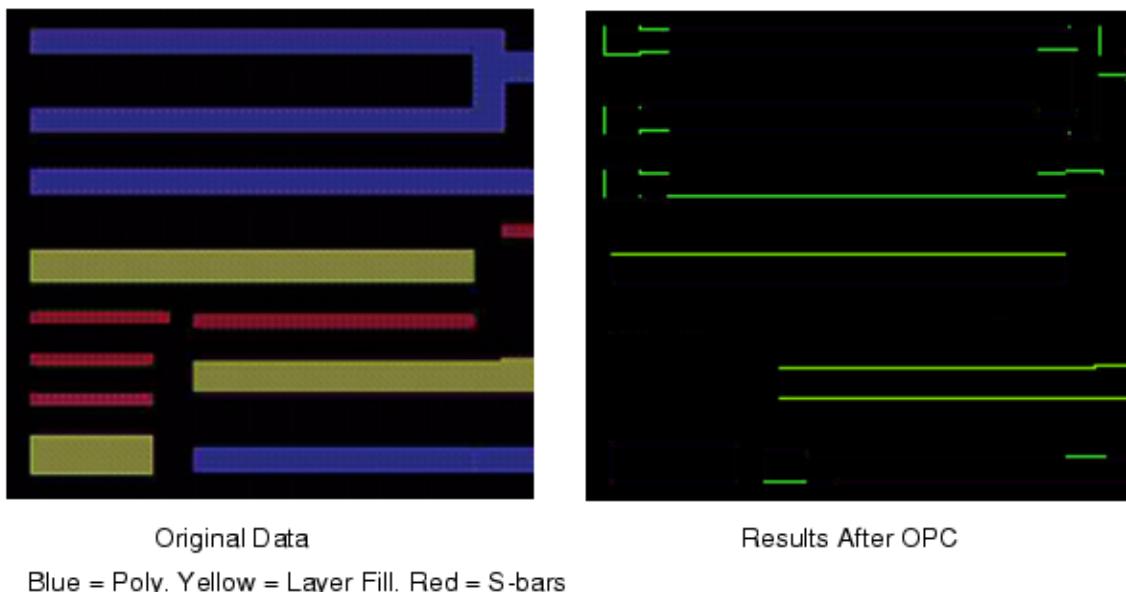
This scenario uses a VT5 model with density kernels. Density kernels (also known as convolution kernels) model the density of features having an aerial image intensity greater than or equal to the reference threshold.

A particular use of PAFs in a VT5 model is to deliberately increase the density of features (essentially adding extra non-functional polygons in empty spaces) to reduce the amount of resist etching in a particular area. The area of effect (calculated by a radius of $(\text{hoodpix} + \text{VT5})$

effect area)/2) must take into account the presence of PAFs in the OPC simulation. Any assist features beyond that area are considered “dummy fill” and have no effect.

[Figure 7-9](#) shows the original data as well as the final print image simulation of the OPC process with PAFs and SRAFs present.

Figure 7-9. Layer Data for OPC With PAFs



Method

1. Write a setup file ([Example 7-5](#)) in which you:
 - a. Define a fragmentation scheme. (See “[Fragmentation Strategies](#)” on page 80.)
 - b. Define layers in the setup file:
 - A new layer containing mask data to be corrected (type opc).
 - A PAF-only layer (type opc).
 - An SRAF layer, if needed (type visible).
 - c. Define the OPC algorithm parameters.
 - d. Freeze the edges on the PAF layer (`opcTag -freeze`).
2. Write an SVRF DRC rule file ([Example 7-6](#)) to:
 - a. Read in the mask layer.
 - b. Generate PAFs (for example, using SVRF [Rectangles](#) to generate layer fill).
 - c. Generate SRAFs, if needed.

- d. Invoke Calibre OPCpro through [LITHO OPC](#), passing the opc layers, SRAF layer (if needed), and setup file.
 - e. For each opc layer, save the results to a new layer.
3. Run Calibre using this SVRF file as input.
4. View the results in Calibre WORKbench or your favorite viewer.

Example 7-5. Setup File for OPC With PAFs

```

# ----- Simulation models -----
modelpath ./models
opticalmodel opt
resistpolyfile res.mod

# ----- OPC algorithm -----
iterations 6
stepsize 0.001
gridsize 0.001
siteinfo RESIST -num 40 -center 20
cornerSiteStyle CONSERVATIVE
cornerRadius 0.150

# ----- Fragmentation -----
minedgelength 0.100
maxedgelength 1
cornedge 0.12 0.12
concavecorn 0.12
interfeature -interdistance 1 -ripplelen 0.12 -num 1
minjog 0.03
lineEndLength 0.20

# ----- Layer info -----
# Create three layers: the opc layer, a PAFs only
# opc layer, and a layer for SRAFs (such as scattering bars), if
# required.
background clear

# Layers
layer 1 OPC 0 0 opc atten 0.06 0 1
layer 2 PAF 0 0 opc atten 0.06 0 1
layer 3 SRAF 0 0 visible atten 0.06 0 1

-----Arbitrary Commands Can Follow This Line. Don't delete this line! -----
sse LITHO_PROMOTION_TILING 1

MRC_RULE EXTERNAL {USE 0.110 EUCLIDEAN}
MRC_RULE INTERNAL {USE 0.085 EUCLIDEAN}

# Freeze edges on opc layers that are only visible (and simulated for
# density calculations)

opcTag PAF -freeze

```

Example 7-6. SVRF DRC File for OPC With PAFs

```
LAYOUT SYSTEM GDS
LAYOUT PATH "gds/lab3.gds"
LAYOUT PRIMARY "*"
PRECISION 1000

DRC MAXIMUM RESULTS ALL
DRC SUMMARY REPORT "reports/lab3.rep"
DRC RESULTS DATABASE "gdsout/lab3_out.gds" GDS
LAYER POLY 1
LAYER PAF 2
LAYER SRAF 3

/* There should be an output layer created for each opc layer,
/* including the poly layer and the PAFs layer */

OPC_OUT = LITHO OPC FILE "setup/setup.in" OPC PAF SRAF MAP OPC
DUMMY = LITHO OPC FILE "setup/setup.in" OPC PAF SRAF MAP PAF

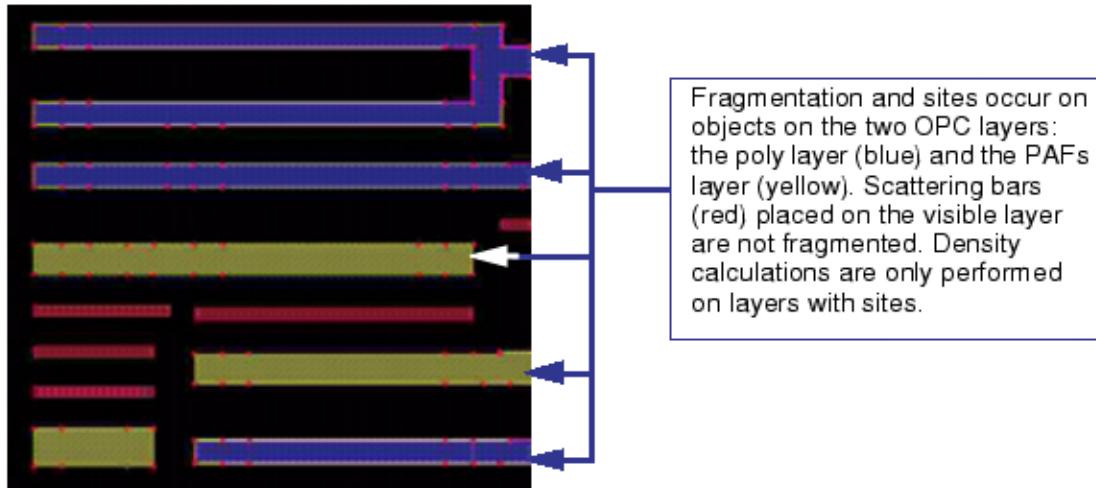
OPC_OUT { COPY OPC_OUT } DRC CHECK MAP OPC_OUT 1
DUMMY { COPY DUMMY } DRC CHECK MAP DUMMY 0 dummy.gds

DRC CHECK MAP OPC 1
DRC CHECK MAP PAF 2
DRC CHECK MAP OPC_OUT 104
DRC CHECK MAP DUMMY 105
```

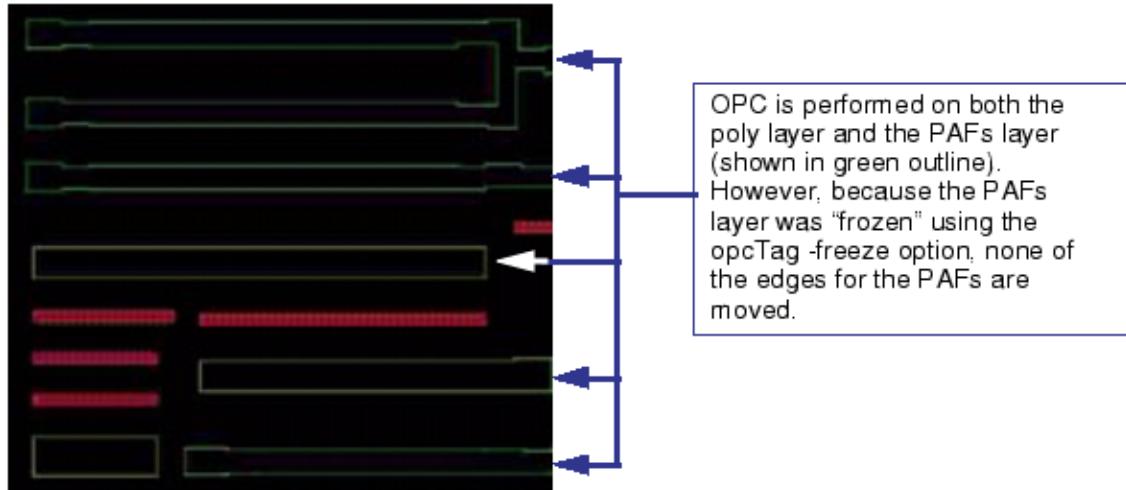
What Happens During OPC With PAFs

Density data is calculated at the simulation sites on the opc layer. Densities are not calculated for visible layers because no sites are present. If a printing assist feature (PAF) such as layer fill is placed on a visible layer (as in the case of scattering bars), the density calculations do not account for the feature and produce inaccurate OPC simulation results.

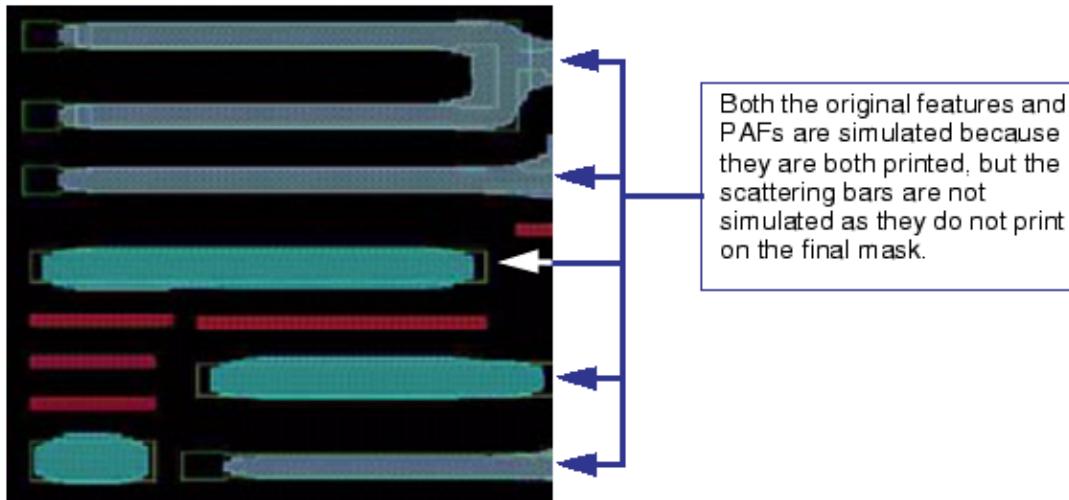
To have the PAFs be accounted for during the simulation, they must be placed on their own separate opc layer instead. SRAFs such as scattering bars can remain on the visible layer as they are not printed on the final mask. [Figure 7-10](#) shows the fragmentation performed on the poly and PAFs layer.

Figure 7-10. Fragmentation for Design With PAFs

To prevent OPC from being performed on the PAFs on this new layer, use the `opcTag -freeze` setup file keyword to “freeze” all the polygons on that layer. When OPC is performed, only the fragments for the poly layer are moved ([Figure 7-11](#)).

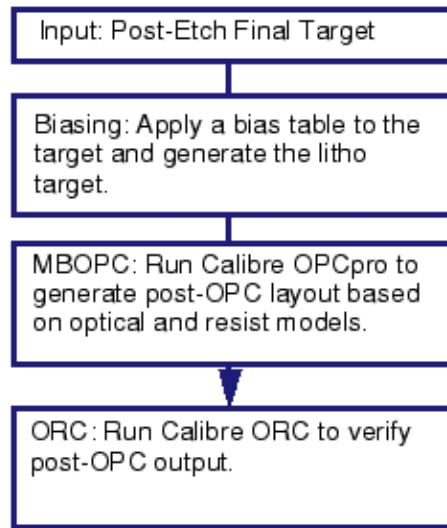
Figure 7-11. OPC With PAFs

When you create a simulation ([Figure 7-12](#)), both the original features and PAFs are simulated, because they are both printed on the final mask. However, the SRAFs remain unsimulated, because they are not intended to be printed.

Figure 7-12. Simulation Based on PAFs and SRAFs

Scenario 4: Etch Retargeting With Calibre TDopc and Calibre OPCpro

For technology nodes above 90 nm, etch bias is compensated by applying a rule table through Calibre® TDopc™. This flow is known as etch retargeting, and is summarized in the figure.

Figure 7-13. Etch Retargeting Flow

Two setup files are used in the etch retargeting flow: one to carry out OPC and another to carry out ORC simulations for checking the quality of the corrections. Two major differences between an OPC setup file ([Example 7-8](#)) and an ORC setup file ([Example 7-9](#)) are the iterations keyword and the types of layers that each one employs.

In an OPC setup file, the iterations keyword should be greater than one for corrections to take place. In an ORC setup file, corrections are not applied so the iterations keyword must be set to zero.

Another major difference between an OPC setup file and an ORC setup file is the way they use layers. An OPC setup file makes corrections on the opc layer specified in the layers section. An ORC setup file uses external layers for post-OPC simulations to check for the quality of edge movements made. An external layer must be used along with an opc layer for Calibre to define the quality of the end result. [Figure 7-14](#) shows the output of the etch retargeting flow.

[Example 7-7](#) shows a full SVRF rule file with the complete LITHO OPC and ORC calls.

Example 7-7. SVRF Rule File

```
LAYOUT SYSTEM GDSII
LAYOUT PATH "./gds/test.gds"
LAYOUT PRIMARY "*"

PRECISION    1000
RESOLUTION    1
UNIT LENGTH    u

DRC RESULTS DATABASE "./gdsout/test_opc_orc.oas" OASIS PSEUDO
DRC SUMMARY REPORT "./report/opc.rep"
DRC MAXIMUM RESULTS ALL

///////////////////////////////
//Original Layout I/O
/////////////////////////////
LAYER M1_orig 10

M1_orig {COPY M1_orig} DRC CHECK MAP M1_orig 1

///////////////////////////////
//Apply Etch Bias, generate the litho target for MBOPC
/////////////////////////////
M1_bias = OPCBIAS M1_orig MINBIASLENGTH 0.20

SPACE >= 0 < 0.16      WIDTH >= 0.20 <= 0.30 OPPOSITE EXTENDED 0.08
MOVE -0.002
SPACE >= 0.16 < 0.18 WIDTH >= 0.20 <= 0.30 OPPOSITE EXTENDED 0.08
MOVE -0.005
SPACE >= 0.18 < 0.20 WIDTH >= 0.20 <= 0.30 OPPOSITE EXTENDED 0.08
MOVE 0
SPACE >= 0.20 < 0.30 WIDTH >= 0.20 <= 0.30 OPPOSITE EXTENDED 0.08
MOVE 0.008
SPACE >= 0.30          WIDTH >= 0.20 <= 0.30 OPPOSITE EXTENDED 0.08
MOVE 0.010

/////////////////////////////
//Litho Target output after TDOPC
/////////////////////////////
M1_bias {COPY M1_bias} DRC CHECK MAP M1_bias 30
```

```
///////////
//Run OPCpro
///////////
M1_opc = LITHO OPC M1_bias FILE "./setup/opc.in" MAP M1_bias

///////////
//OPCpro Output
/////////
M1_opc {COPY M1_opc} DRC CHECK MAP M1_opc 110

///////////
//Run ORC to check EPE
/////////
negEPE1 = LITHO ORC M1_bias M1_opc MAPNUMBER 201 FILE "./setup/orc.in"
negEPE2 = LITHO ORC M1_bias M1_opc MAPNUMBER 202 FILE "./setup/orc.in"
posEPE1 = LITHO ORC M1_bias M1_opc MAPNUMBER 203 FILE "./setup/orc.in"
posEPE2 = LITHO ORC M1_bias M1_opc MAPNUMBER 204 FILE "./setup/orc.in"

///////////
//ORC errors Output
/////////
negEPE1 {COPY negEPE1} DRC CHECK MAP negEPE1 201
negEPE2 {COPY negEPE2} DRC CHECK MAP negEPE2 202
posEPE1 {COPY posEPE1} DRC CHECK MAP posEPE1 203
posEPE2 {COPY posEPE2} DRC CHECK MAP posEPE2 204
```

Example 7-8. OPC Setup File

```
# ----- Simulation models -----
## Model definitions for OPCpro
modelpath ./models
opticalmodel generic_mod
resistpolyfile VT5.mod

# ----- OPC algorithm -----
iterations 6
tilemicrons 100
stepsize 0.001
gridsize 0.001
siteinfo SAMPLE -numx 18 -center 8
cornerSiteStyle CONSERVATIVE
lineEndAdjDist 0.14
convexAdjDist 0.14
concaveAdjDist 0.14

# ----- Fragmentation -----
## Fragmentation settings for opc input layer
minfeature 0.12
minedgelength 0.1
maxedgelength 1
cornedge 0.14 0.12 0.12 0.12 rem 0.10
concavcorn 0.14 0.12 0.12 0.12 rem 0.10
interfeature -interdistance 0.55 -ripplelen 0.14 -num 2 -ripplestyle 1 \
-split 1 -shield 2 -distancePriority 1 -skipCorners
seriftype 0
minjog 0.08
lineEndLength 0.25
```

```

# ----- Layer info -----
## Etch biased target layer is now used as the OPC target layer.
background clear

# Layers
layer 30 M1_bias 1 0 opc dark

----- Arbitrary Commands Can Follow This Line. Don't delete this line!
sse LITHO_PROMOTION_TILING 1
sse OPC_MAX_ITER_MOVEMENT 0.02
sse OPC_FEEDBACK -0.3 -0.5 -0.6 -0.4 -0.3

#MRC Rules
## MRC restriction definitions
MRC_RULE EXTERNAL {
    USE 0.13 EUCLIDEAN
    NOT_PROJECTING
    USE 0.10 EUCLIDEAN
}
MRC_RULE INTERNAL {
    USE 0.12 EUCLIDEAN
    NOT_PROJECTING
    USE 0.10 EUCLIDEAN
}

```

Example 7-9. ORC Setup File

```

# ----- Simulation models -----
modelpath ./models
opticalmodel generic_mod
resistpolyfile VT5.mod

# ----- OPC algorithm -----
## Iterations must be set to 0 for ORC runs.
iterations 0
tilemicrons 100
stepsize 0.001
gridsize 0.001
siteinfo SAMPLE -numx 18 -center 8
cornerSiteStyle CONSERVATIVE
lineEndAdjDist 0.14
convexAdjDist 0.14
concaveAdjDist 0.14

# ----- Fragmentation -----
minfeature 0.12
minedgelength 0.05
maxedgelength 1
cornedge 0.07 0.07 0.06 0.06
concavecorn 0.07 0.07 0.06 0.06
interfeature -interdistance 0.55 -ripplelen 0.14 -num 2 \
    -ripplestyle 1 -split 1 -shield 2 -distancePriority 1 -skipCorners
seriftype 0
minjog 0.08
lineEndLength 0.25

```

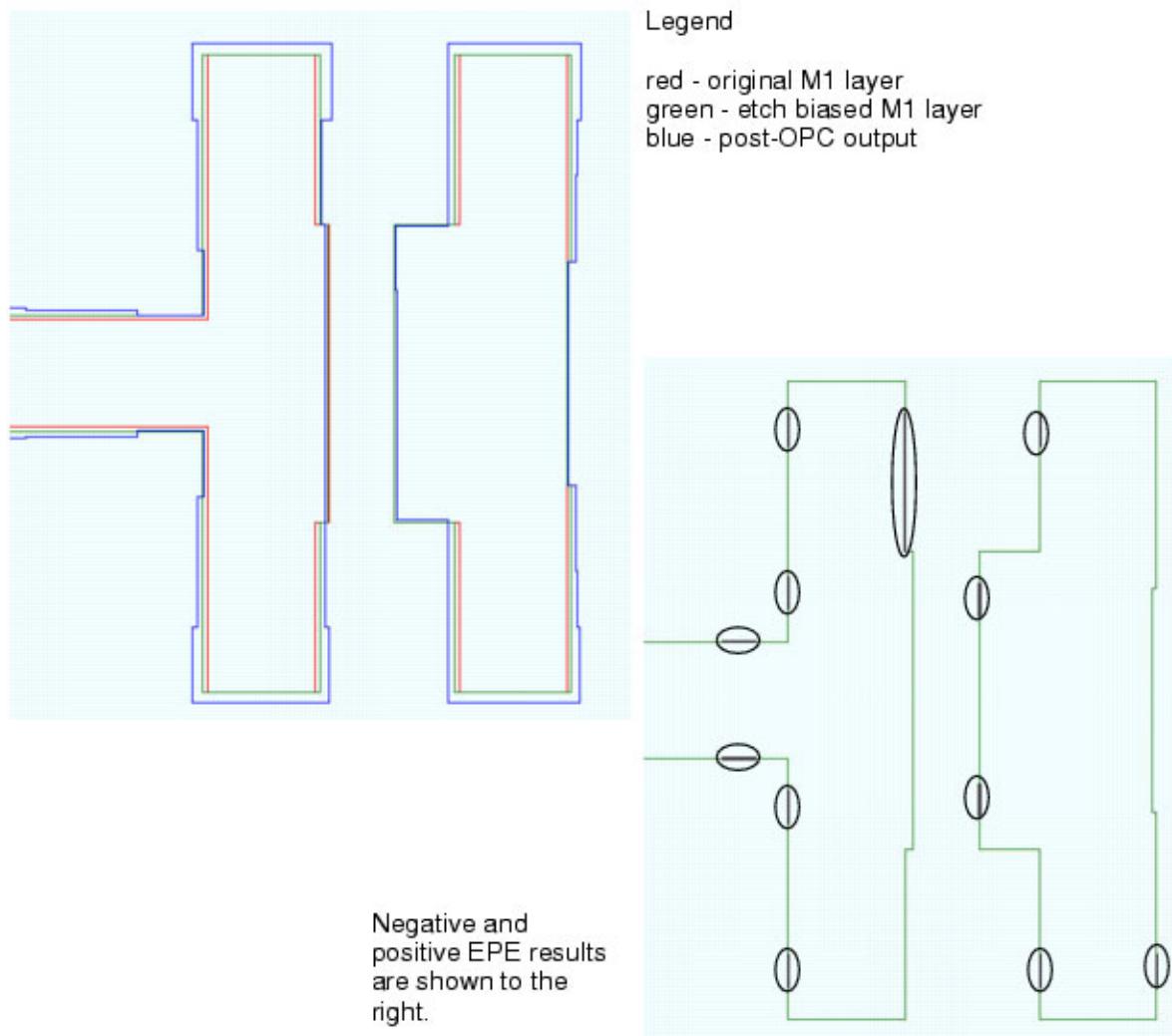
```
# ----- Layer info -----
background clear

# Layers
## LITHO target was previously set as the opc layer. Post-OPC output
## needs to be set as the external layer for ORC simulations.
layer 30 M1_bias 1 0 opc dark
layer 110 M1_opc 1 0 external dark

----- Arbitrary Commands Can Follow This Line. Don't delete this line!
clearTagScript

## Run ORC to highlight bad EPE after OPC corrections.
## EPE is defined as the distance between the simulated image
## and the drawn edge (OPC target edge).
newTag negEPE1 -how EPE all -0.005 -0.003
newTag negEPE2 -how EPE all -0.010 -0.005
newTag posEPE1 -how EPE all 0.003 0.005
newTag posEPE2 -how EPE all 0.005 0.010

## Tags2boxes highlights the bad EPE error markers on the
## output layer for inspection.
tags2boxes -tags negEPE1 -layers 201 -halfWidths 0.002
tags2boxes -tags negEPE2 -layers 202 -halfWidths 0.002
tags2boxes -tags posEPE1 -layers 203 -halfWidths 0.002
tags2boxes -tags posEPE2 -layers 204 -halfWidths 0.002
```

Figure 7-14. Etch Bias Output

Scenario 5: OPCpro Concurrent Operations

Calibre OPCpro can run LITHO operations concurrently if all the LITHO operations have the same arguments and the same order of inputs except for the map number. If the LITHO statements are different in any other way, they are not executed concurrently.

Concurrency is important for faster execution in Calibre. Calibre performs many of the layer operations concurrently. This means that they run simultaneously when present in a rule file. Whenever Calibre performs layer operations, it locates all required layer operations within the set that it can run concurrently and executes them as a single group. You can optimize your rule file by taking advantage of this feature. Concurrent operations can be seen in the log file shown in [Figure 7-15](#) before the execution of the setup file interpreter.

Figure 7-15. Log File Analysis of Concurrency

```
DIFF = OR DIFF
-----
DIFF (TYP=1 CFG=1 ECT=112 OCT=86 SRT=1 CMP=F MPN=37)
CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/1/2 OPS COMPLETE = 2 OF 62
ELAPSED TIME = 0

Original Layer DIFF DELETED -- LVHEAP = 1/1/2

GATE_ORC::<1> = LITHO ORC POLY DIFF MAPNUMBER 100 FILE ./setup/orc.in
POLY_ORC::<1> = LITHO ORC POLY DIFF MAPNUMBER 101 FILE ./setup/orc.in
epeneg::<1> = LITHO ORC POLY DIFF MAPNUMBER 200 FILE ./setup/orc.in
epen10::<1> = LITHO ORC POLY DIFF MAPNUMBER 201 FILE ./setup/orc.in
epen09::<1> = LITHO ORC POLY DIFF MAPNUMBER 202 FILE ./setup/orc.in
.
.
.
epen03::<1> = LITHO ORC POLY DIFF MAPNUMBE
epen02::<1> = LITHO ORC POLY DIFF MAPNUMBER 209 FILE ./setup/orc.in
epen01::<1> = LITHO ORC POLY DIFF MAPNUMBER 210 FILE ./setup/orc.in
-----
// Litho Libraries v2013.3_0.10 Wed Jun 11 18:03:03 2013
.

.

CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3 OPS COMPLETE = 60 OF 62
ELAPSED TIME = 1
```

LITHO calls are batched together before the setup file interpreter is called.

ELAPSED TIME shows how long the concurrent rules took to execute.

Example 1

The following example shows how a rule file is set up to use concurrency.

```
ORC1 = LITHO ORC POLY DIFF FILE setup.in MAPNUMBER 10
ORC2 = LITHO ORC POLY DIFF FILE setup.in MAPNUMBER 11
```

If MAPNUMBER is the only difference between each LITHO call, then the rule file is optimized for concurrency because these commands are executed simultaneously in Calibre.

Example 2

If any arguments in the LITHO statements are different from each other, then the Calibre run does not run concurrently. Non-concurrent operations can occur in Calibre under these conditions:

- When LITHO operations mismatch

```
ORC1 = LITHO OPC POLY DIFF FILE setup.in MAPNUMBER 10
ORC2 = LITHO ORC POLY DIFF FILE setup.in MAPNUMBER 11
```

- When the order of input layers mismatch

```
ORC1 = LITHO ORC POLY DIFF FILE setup.in MAPNUMBER 10
ORC2 = LITHO ORC DIFF POLY FILE setup.in MAPNUMBER 11
```

- When input setup files mismatch

```
ORC1 = LITHO ORC POLY DIFF FILE setup1.in MAPNUMBER 10
ORC2 = LITHO ORC POLY DIFF FILE setup2.in MAPNUMBER 11
```

Calibre OPCpro Example

This example performs limited OPC on a subset of edges (line ends) and leaves the rest of the edges untouched. The first step is to identify the line ends. There is some overlap in capabilities between SVRF and the tagging commands and keywords. In general, logical operations are faster using SVRF and tagging commands are more concise for controlling movement of which edge fragments.

For this example, you could identify line ends by using the built-in tag, line_end. However, in order to gain some experience using SVRF operations to prepare data for processing, the following steps detail creating an island layer containing boxes drawn around line ends. You then tag the fragments identified by this island layer and perform OPC on them.

Deriving the Layers

The SVRF operations used to identify the line ends and create the island layer are as follows:

```
w1 = CONVEX EDGE POLY
      ANGLE1 == 90 LENGTH1 > 0.3
      ANGLE2 == 90 LENGTH2 > 0.3
      WITH LENGTH <= 0.3
w2 = CONVEX EDGE POLY
      ANGLE1 == 90 LENGTH1 > 0.5
      ANGLE2 == 90 LENGTH2 > 0.5
      WITH LENGTH > 0.3 <= 0.5
w3 = CONVEX EDGE POLY
      ANGLE1 == 90 LENGTH1 > 0.7
      ANGLE2 == 90 LENGTH2 > 0.7
      WITH LENGTH > 0.5 <= 0.7
y1 = EXPAND EDGE w1 OUTSIDE BY 0.005
y2 = EXPAND EDGE w2 OUTSIDE BY 0.005
y3 = EXPAND EDGE w3 OUTSIDE BY 0.005
end_cap = y1 or (y2 or y3)
```

Tagging the Fragments

Pass the layer end_cap to the LITHO OPC operation for use as an island layer. Create a new tag, calibre_line_end, by ANDing the two input layers. The setup file that sets the layer type and uses the layer to tag the opc layer is called *setup.in*, as follows:

```
# ----- simulation models -----
modelpath ../models
opticalmodel DUV10tab
resistpolyfile aerialmodel.30.mod

# ----- OPC algorithm -----
iterations 0
tilemicrons 160.000
stepsize 0.0100
gridsize 0.001000
siteinfo AERIAL
cornerSiteStyle SITES_ON_ARC

# ----- fragmentation -----
minfeature 0.2800
minedgelength 0.100
maxedgelength 1000.000
cornedge 0.100 0.100
concavecorn 0.100 0.100
interfeature -interdistance 0.740 -ripplelen 0.200 -num 1
seriftype 1
minjog 0.070
lineEndLength 0.300

# ----- Layer info -----
background clear
# Layers
layer 2 POLY 0 0 opc dark
layer 10 end_cap 0 0 island dark

----- Arbitrary Commands Can Follow This Line -----
clearTagScript
newTag calibre_line_end -how coincidentEdge end_cap POLY
fastIter calibre_line_end 4
```

Note the first line in the OPC algorithm section, which sets the iterations keyword to 0. This turns off OPC for all fragments. The last line in the file is the fastIter command, which instructs the OPC algorithm to perform four iterations on the line ends only. (When fastIter is set, the iterations keyword is ignored.) The fastIter line performs OPC on the edges with the tag calibre_line_end.

Performing the LITHO Operation

The operations used in the SVRF file to invoke the Calibre OPCpro batch tool, then to output the data are as follows:

```
MY_OPCT
    LITHO OPC FILE opc.in POLY end_cap
}

POLY {COPY POLY}

DRC CHECK MAP POLY 2
DRC CHECK MAP end_cap 3
DRC CHECK MAP MY_OPCT 4
```

Notice that the output from the LITHO operation is stored as a rule check statement named MY_OPCT. After processing, the DRC CHECK MAP statements write all three layers (POLY, end_cap, and MY_OPCT) to the results database, each on a different layer.

Using Matrix OPC in Calibre OPCpro

This section begins with an extensive explanation of Matrix OPC and its uses for multiple masks and other advanced OPC situations.

| | |
|--|------------|
| Matrix OPC for Multiple Masks and Complex EPE | 575 |
| Methods of Matrix OPC Fragment Mapping | 577 |
| Matrix OPC Mapping Controls | 579 |
| Viewing Full-Chip Mappings With Visualize Matrix OPC..... | 580 |

Matrix OPC for Multiple Masks and Complex EPE

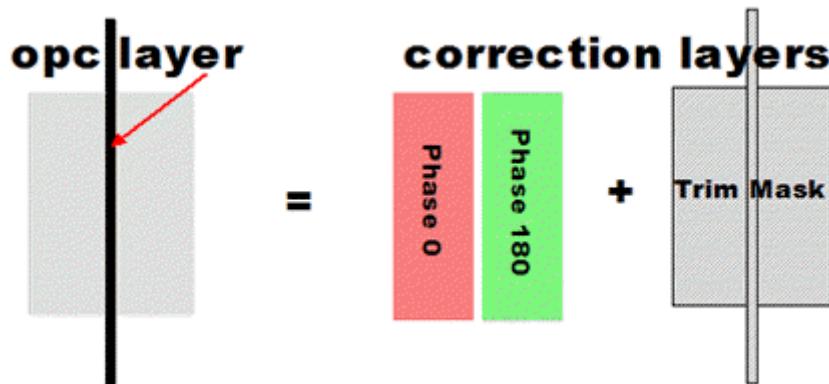
For most binary and attenuated PSM mask RET, the final image in resist is formed by a conventional mask exposure. For many advanced RET schemes, the mask must be decomposed into multiple masks, such that the final post-OPC masks are radically different from the designed, or target, layer.

Table 7-1 describes the layer decomposition when using the Calibre OPCpro and Matrix OPC tools.

Table 7-1. Layer Decomposition for Multimask Layers

| Layer | Contains | Layer Type |
|--------|--|------------|
| target | Non-movable fragments with simulation sites Desired silicon target (drawn mask) | opc |
| mask | Movable fragments, but no simulation sites Examples: <ul style="list-style-type: none">• phase and trim masks for Alt-PSM• horizontal and vertical masks for dipole exposure | correction |

Figure 7-16. Layer Decomposition for Alt PSM

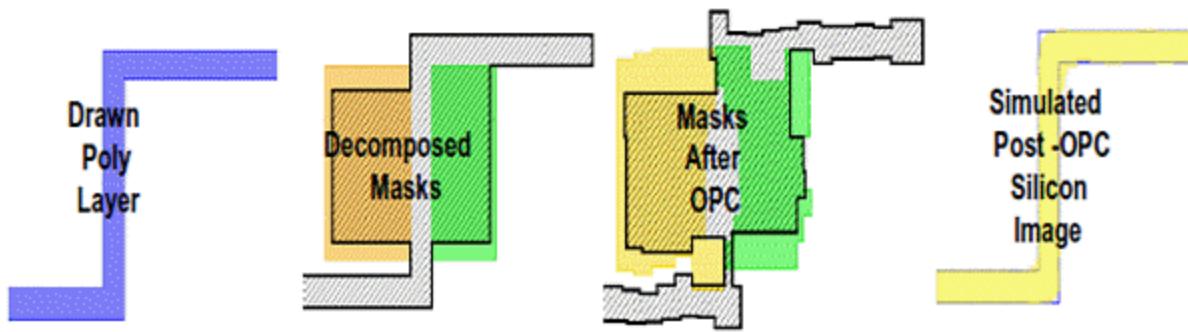


To support performing OPC on multimask layers, Matrix OPC maps correction (mask) layer fragments to opc (target) layer fragments. The EPE at the site on the opc layer fragment then

drives the corrective movement applied to the correction layer fragment. Matrix OPC support for multiple masks includes:

- **Many-to-One mapping** maps many correction layer fragments to one target layer fragment site.
- **One-to-Many mapping** maps one correction layer fragment to many target layer fragment sites.
- **Priority Mapping** prioritizes the influence each site has over the corrections.

Figure 7-17. OPC on Multimask Layers

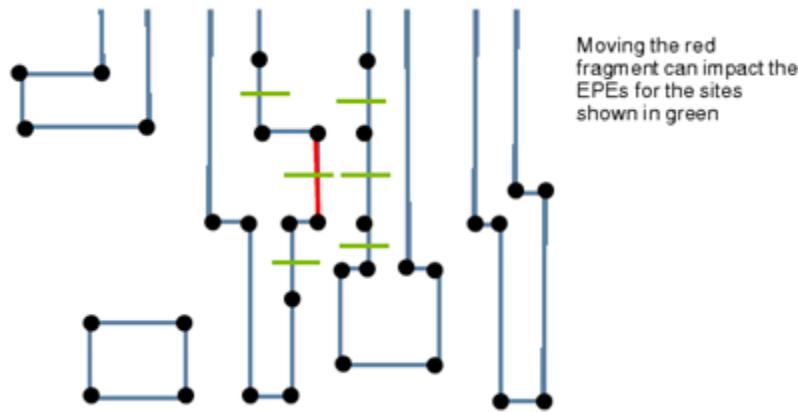


Using the functionality provided through Matrix OPC, you can design your own innovative OPC solution. For example, it is known that moving one edge affects EPEs at many positions. For masks at the 65 nm node and below, these interactions are significant enough that they must be factored into OPC.

Calibre Matrix OPC gives you the ability to:

- Average multiple EPEs into movement calculations.
- Select and weight the EPEs to factor in.

Figure 7-18. Moving One Edge Can Affect Multiple EPEs

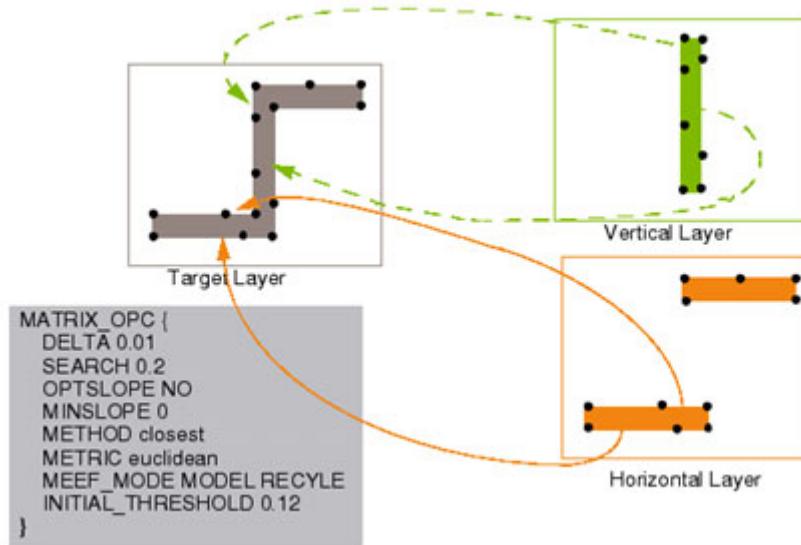


Methods of Matrix OPC Fragment Mapping

Matrix OPC provides several different methods for mapping correction layer fragments to the associated fragment on the opc layer.

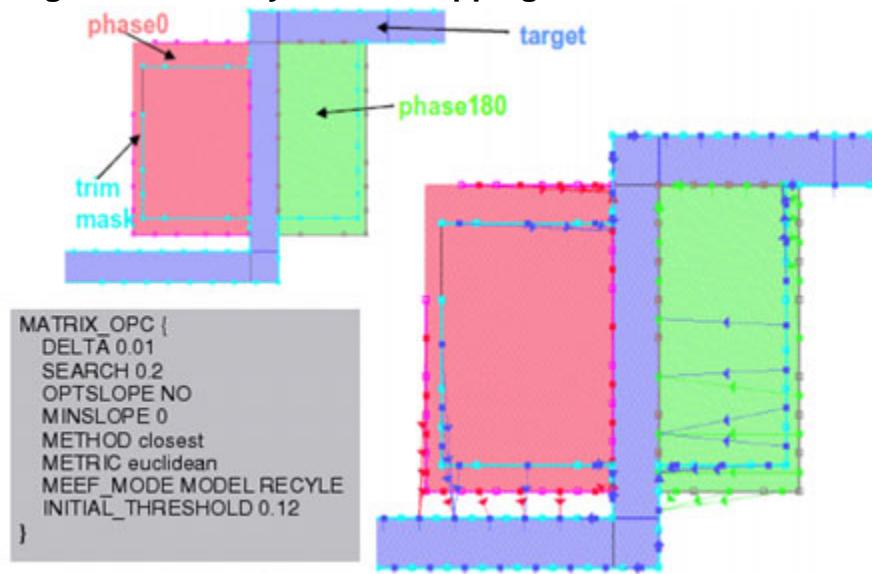
- **One-to-One** — Maps each correction layer fragment to the closest target fragment, as identified using the specified metric. [Figure 7-19](#) shows one-to-one mapping on a dipole mask.

Figure 7-19. One-to-One Mapping With Euclidean Metric



- **Many-to-One** — Maps many correction layer fragments to the site on a single target fragment. [Figure 7-20](#) shows many-to-one mapping on an Alt PSM mask. The image on the right shows the mapping of trim and phase layer fragments to the same site on the target layer fragment.

Figure 7-20. Many-to-One Mapping With Euclidean Metric



- **One-to-Many** — Maps one correction layer to many target fragments. Figure 7-21 shows how you can use one-to-many fragmentation to resolve areal image ripple problems.

Figure 7-21. One-to-Many Mapping With Euclidean Metric

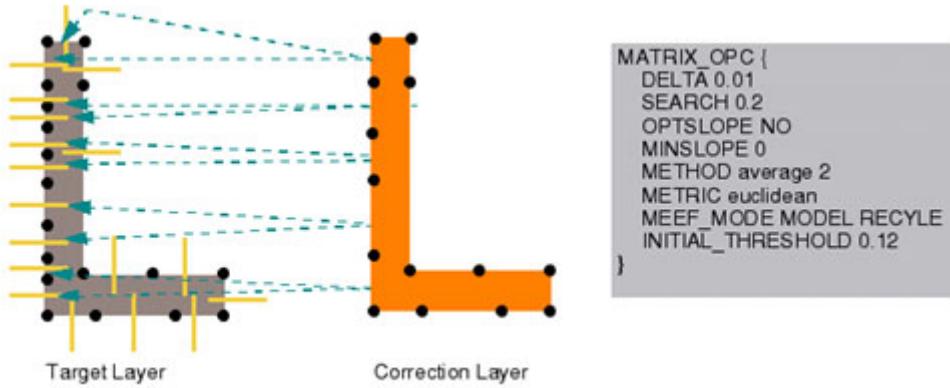
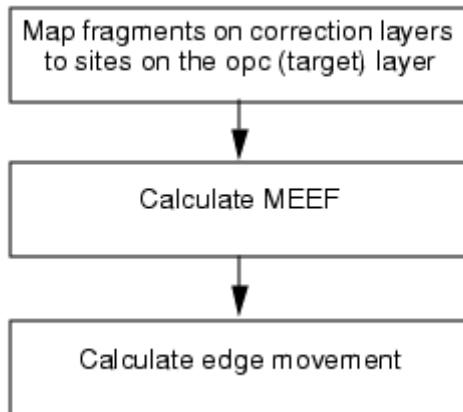


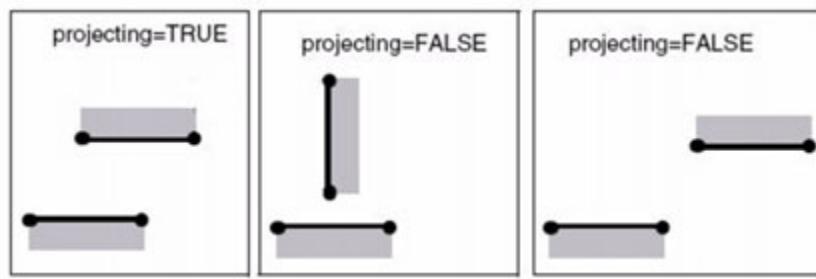
Figure 7-22 shows the basic flow of Matrix OPC.

Figure 7-22. The Basic Matrix OPC Flow

Matrix OPC Mapping Controls

To calculate edge movement, Matrix OPC begins by associating each movable fragment with one or more simulation sites. This association takes the form of a mapping. Some properties you can map by are:

- Layer and tag
- Orientation of the fragment to the edge on the opc layer
- Maximum distance between a fragment and the site(s) to which it is mapped
- Weighted distance between a fragment and its site(s)
- Projection as shown in [Figure 7-23](#)

Figure 7-23. Graphic Description of Projection

Viewing Full-Chip Mappings With Visualize Matrix OPC

To check that Matrix OPC is mapping fragments to sites as you expected, you can view the mappings using Calibre WORKbench. You can generate the mapping data for the full chip when running in batch mode or for a test area by running mini-opc in Calibre WORKbench.

Prerequisites

- GDS or OASIS layout file
- Litho setup file that includes a **MATRIX_OP**C command

Procedure

1. Set **OPC_VISUALIZE_MATRIX** to 1. This can be done either in the Linux environment or with the **sse** command.
2. Run in batch mode. Because of **OPC_VISUALIZE_MATRIX**, the application creates an additional Tcl file with a name beginning with the characters “vis”.
3. Invoke Calibre WORKbench and load the layout.
4. Select **Macros > EXECUTE TCL SCRIPT** to load the generated Tcl file.

The Tcl script contains layout commands that generate the mapping layers. After running the mapping layers are displayed along with the original layers.

Results

Visualize Matrix OPC lets you view the fragmentation, the mapping, and the MEEF calculations. Each type of data is on a separate layer, so you can hide or display as much data as you want.

Table 7-2. Full-Chip Layers Created With Visualize Matrix OPC

| Layer Number ¹ | Function |
|---------------------------|--|
| n | original layer |
| n + 100 | post-OPC layer |
| n + 1000 | original layer with fragmentation points |
| n + 2000 | arrows showing mappings |
| n + 3000 | text displaying MEEFs |

1. The numbers added to “n” are called layer “bump” numbers. The layer bump numbers for post-OPC layers (row 2) are user-definable. 100 is the default value. The layer bump numbers for Matrix OPC layers (rows 3-5) are not user-definable.

Matrix OPC Scenarios

The following scenarios show some techniques for using Matrix OPC.

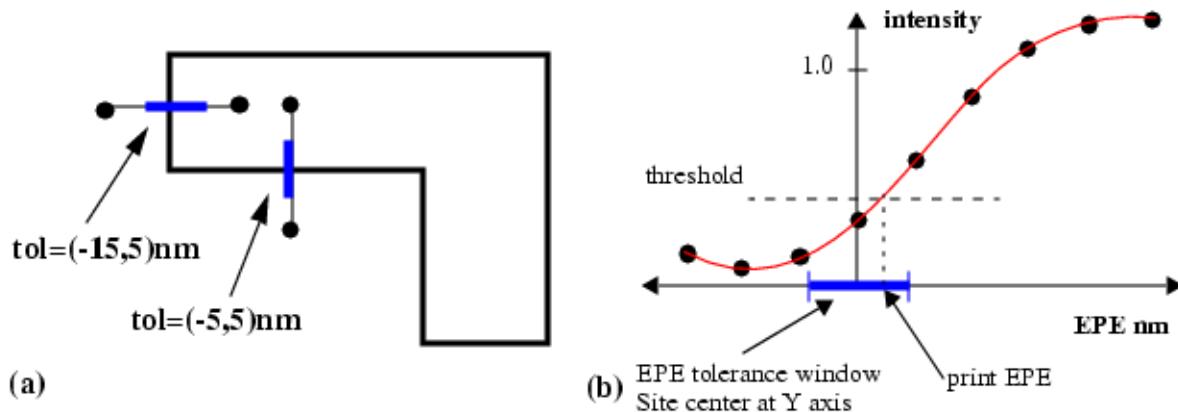
| | |
|--|------------|
| Example 1: Multiple Tolerances | 581 |
| Example 2: Matrix OPC Followed by Calibre ORC | 582 |
| Example 3: Using Enclosure Checks With Matrix OPC | 585 |
| Example 4: Matrix OPC for Alternating Phase Shift Masks | 588 |

Example 1: Multiple Tolerances

In this example, multiple tolerances are applied for EPE and for slope.

```
stepsize 0.001
sse OPC_EPE_TOLERANCE_FRAC 5
newTag t1 -how edge LENGTH < 0.12
epeToleranceTag line_end 0.005 -0.015 # tol window (-15,5)nm
epeToleranceTag t1 0.005 # tol window (-5,5)nm
MATRIX_OPIC {
    METHOD CLOSEST
    OPTSLOPE YES
    SEARCH 0.1
    METRIC EUCLIDEAN
    MEEF_MODE 0.7
    DELTA 0.001
    slopeToleranceTag line_end 2.7 # min slope for line ends
    slopeToleranceTag t1 3.1      # min slope for short fragments
    MINSLOPE 3.0                  # default min slope
}
```

A line end that is less than 0.12 microns long matches both “line_end” and “t1”. However, line_end is first in the setup file, so the (-15,5) nm tolerance window would apply to any such line end shorter than 0.12 microns. Fragments that are not “t1” or “line_end” automatically have a (-5,5) nm tolerance window defined. The diagram in [Figure 7-24](#) shows a case of two different sites, which have different EPE tolerance windows defined. This also shows the intensity cutline and how the EPE tolerance is viewed from the cutline perspective.

Figure 7-24. Multiple Tolerance Windows

Example 2: Matrix OPC Followed by Calibre ORC

This example looks at the performance of Matrix OPC with slope optimization on a section of layout and also uses ORC to analyze its effectiveness in improving slope, compared to standard OPC.

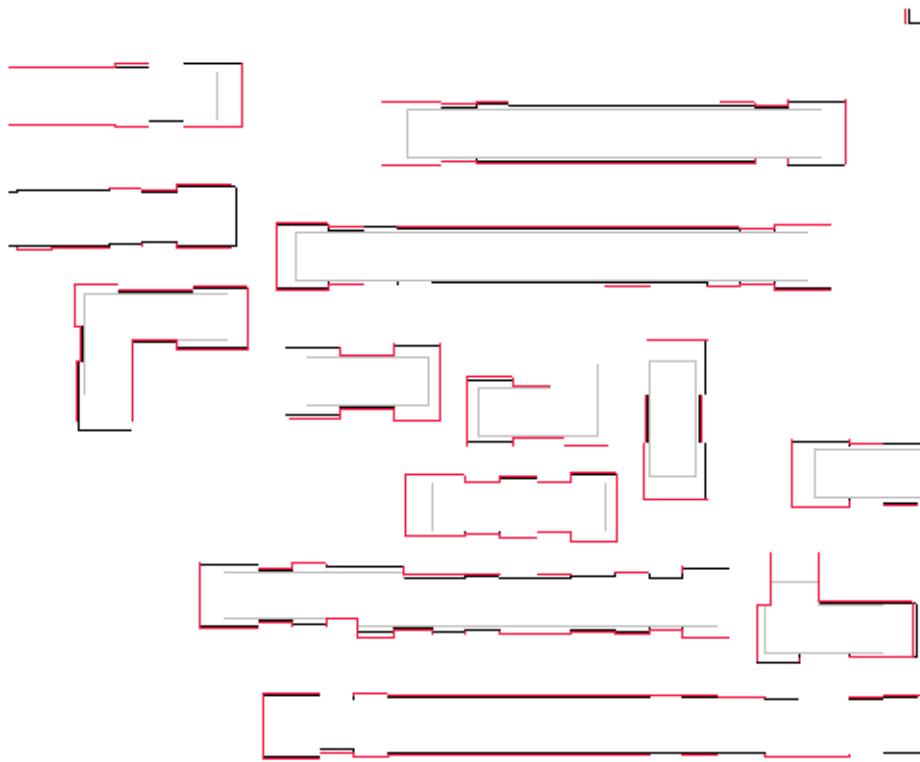
Lithography settings:

```
mask: attenuated 6% background, clear features
illumination: STANDARD, sigma = 0.75
lambda=193, NA = 0.7
```

OPC settings:

```
epeToleranceTag line_end          0.005 -0.010
epeToleranceTag line_end_adjacent 0.010 -0.005
epeToleranceTag all               0.005 -0.005
MATRIX_OPCTag {
    SEARCH 0.1
    OPTSLOPE YES
    MINSLOPE 3.0
    slopeToleranceTag line_end 2.7
    slopeToleranceTag line_end_adjacent 2.7
}
```

[Figure 7-25](#) shows the results of Matrix OPC (black), superimposed with the standard OPC (red) and the wafer target (gray). The slope-optimized OPC output has slightly less correction on line ends in order to improve slope. Also, the slope-optimized OPC output has slightly wider corrections on isolated edges, which results in higher image intensity hence better slope.

Figure 7-25. Overlay of Standard OPC, Matrix OPC, Target Layer

ORC settings:

```

background atten 0.6
layer 100 TARGET 17 0 opc clear
layer 101 MYOPC 17 0 external clear
sse LITHO_HISTOGRAM_OUTPUT_FILE histogram.txt
newTag t1 -how image all SLOPE >= 2 < 2.1
newTag t2 -how image all SLOPE >= 2.5 < 2.7
newTag t3 -how image all SLOPE >= 2.7 < 2.9
newTag t4 -how image all SLOPE >= 2.9 < 3.1
newTag t5 -how image all SLOPE >= 3.1 < 3.3
newTag t6 -how image all SLOPE >= 3.3 < 3.5
...
newTag t20 -how image all SLOPE >= 6.9 < 7.1
tags2boxes -tags t1 ... t20 -layers 1 ... 20

```

The ORC tagging output results are shown in [Figure 7-26](#).

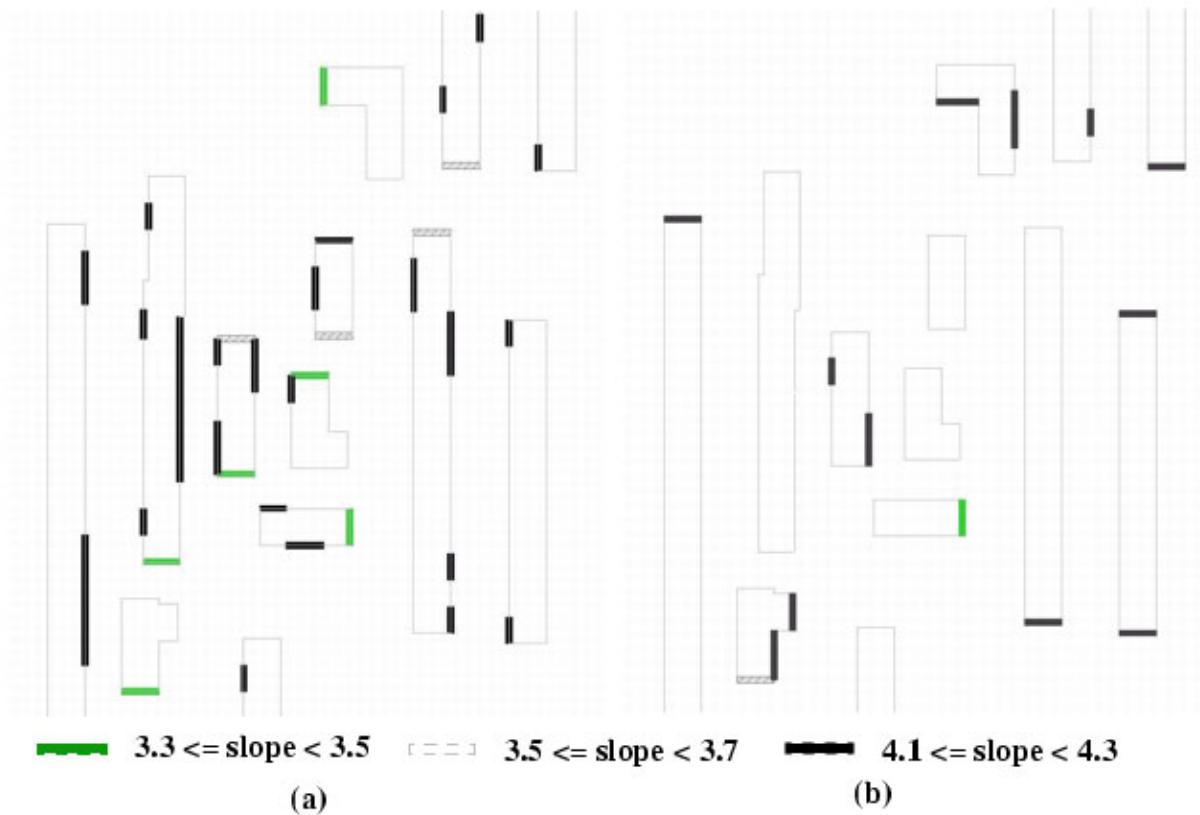
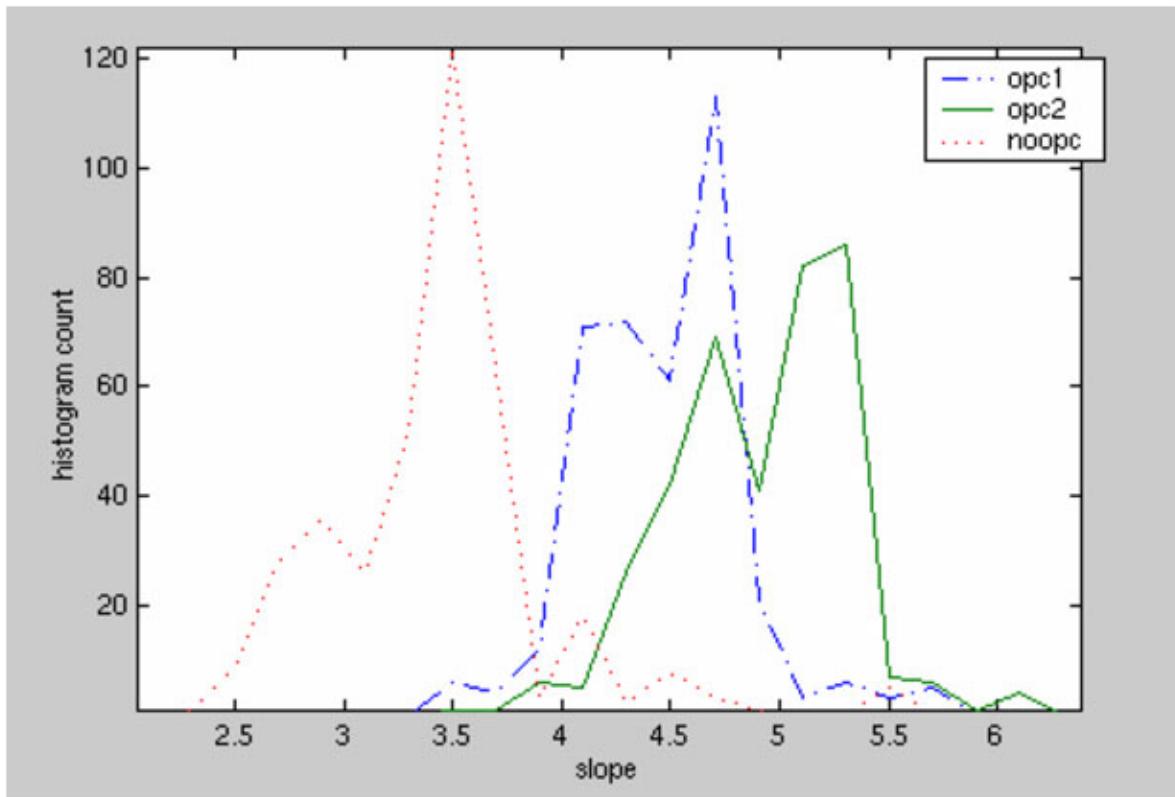
Figure 7-26. ORC Tagging Results From Example 2

Figure 7-26 shows the output from ORC as boxes drawn on different output layers. (a) The ORC results from the standard OPC corrections, shown with the target. (b) The ORC results from matrix OPC corrections, shown with the target.

The histogram data produced by ORC is plotted in Figure 7-27, where we can see that slope has been improved by 10% compared to standard OPC.

Figure 7-27. Histogram of Slope (Intensity per Micron)

In [Figure 7-27](#), the OPC1 results are for standard OPC and OPC2 results are for Matrix OPC with slope optimization.

Example 3: Using Enclosure Checks With Matrix OPC

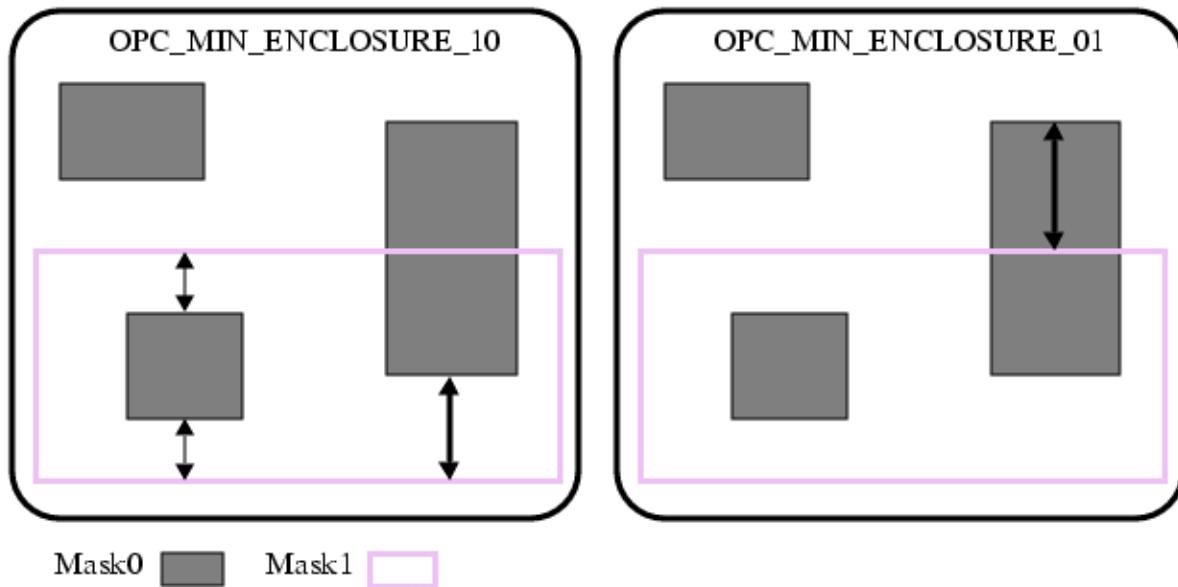
When performing OPC on multiple masks used to create a single physical layer (for example, one created when using alternating Phase Shift Masks), it is sometimes necessary for checks to be performed between the different mask layers. The rule checking functionality provided through Calibre OPCpro, such as `OPC_MIN_EXTERNAL` and `OPC_MIN_INTERNAL` checks, checks only between features on the same mask. Enclosure rule checking, which is supported by six environment variables available through Matrix OPC, allows you to perform limited enclosure rule checks between different masks.

Enclosure rule checking checks the distances between fragments when a feature on one mask encloses a feature on the other mask. Enclosure rules used with Matrix OPC ensure that when the enclosure relationship is met by two edges, those edges maintain a minimum separation after movement by OPC. Here, “enclosure” is defined according to SVRF usage, comparing an interior-facing edge on one layer to an exterior-facing edge on another layer. As with the SVRF

ENClosure operation, order matters and results differ between when checking for Mask 0 enclosing Mask 1, compared to checking for Mask 1 enclosing Mask 0.

Enclosure rules are checked during each iteration of OPC. They are not enforced for jogs created by Calibre OPCpro.

Figure 7-28. Enclosure Check With Multiple Masks



Note that enclosure checking is between *masks*, not *layers*. Thus, if it takes two GDS layers to make Mask 1 (say, phase 1 and phase 2), enclosure rules are enforced between Mask 0 layers and both of the phase layers.

Turning on Enclosure Rule Checking

You control whether or not enclosure rules are checked using the following sse variables:

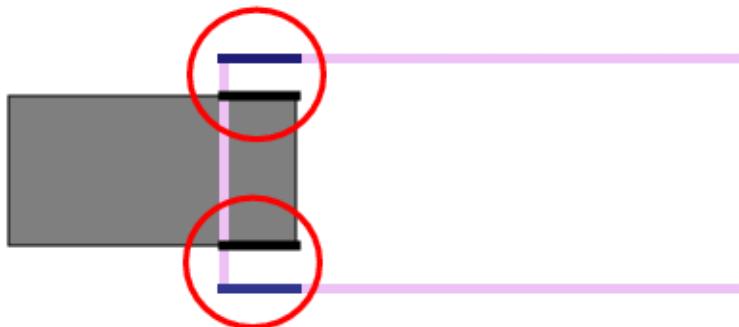
- **OPC_MIN_ENCLOSURE_10 val metric:** Turns on enclosure checking between Mask 1 and Mask 0 (that is, for fragments such that the Mask 1 fragments enclose the Mask 0 fragments).
- **OPC_MIN_ENCLOSURE_01 val metric:** Turns on enclosure checking between Mask 0 and Mask 1 (that is, for fragments such that the Mask 0 fragments enclose the Mask 1 fragments).

Turning Off Enclosure Rule Checking for Special Fragments

The following sse variables allow you to turn off enclosure checking between certain types of fragments:

- **OPC_ENCLOSURE_MIN_LENGTH *length***: Lets you turn off enclosure rule checking for jogs and other short fragments. This variable defines a minimum fragment length that both fragments must meet for the enclosure check to be enforced. If either fragment is shorter than this length, the enclosure rule does not apply. For example, this variable allows you to turn off enclosure checking for pairs of edges in which one edge is a user-created jog (it does not matter which mask they are on).
- **OPC_IGNORE_ENCLOSURE_AT_CORNER**: This variable allows you to turn off enclosure checking in corner fragments in situations where a corner on the block mask overlaps a corner on the phase-shifted mask.

Figure 7-29. Affected Edges With OPC_IGNORE_ENCLOSURE_AT_CORNER



Edges that are not checked when specifying
OPC_IGNORE_ENCLOSURE_AT_CORNER

Prioritizing Edge Movement When Enclosure Rules Are Violated

When edge movement is constrained through enclosure rules, there can be situations in which you want edges on one mask to receive the optimal correction and edges on the other mask move to compensate for those corrections. You can make this happen by assigning one mask priority over the other. The LITHO_MASK_PRIORITY_MOVEMENT variable allows you to do this. The argument you set for this variable defines the higher priority mask.

If you assign higher priority to one mask, you can override this prioritization scheme for selected fragments using the LITHO_MOVEMENT_PRIORITY_TAG variable. The argument to this variable is a tag name. Each fragment in this tag is given equal priority with the opposing fragment.

Example 4: Matrix OPC for Alternating Phase Shift Masks

This example of using Matrix OPC with Calibre OPCpro walks you through how to use alternating phase shift masks with Matrix OPC.

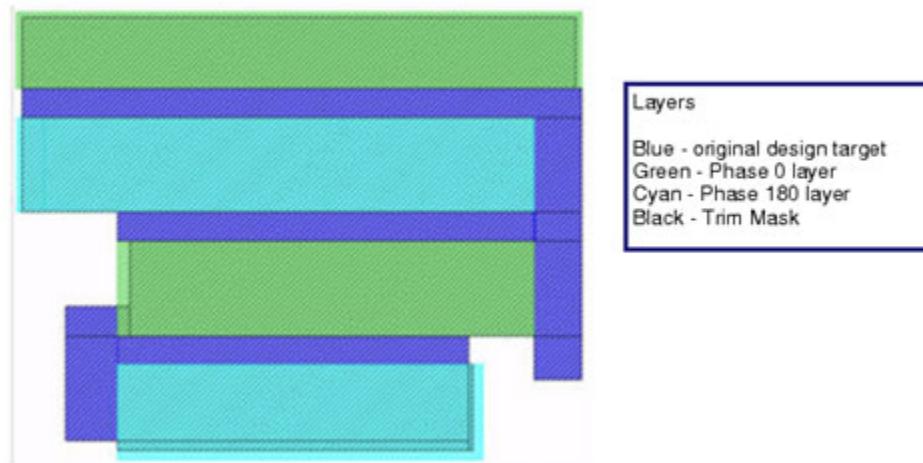
Prerequisites

- An input layout with design splits between phase 0 and phase 180.

Layout

The following layout is used for this example flow.

Figure 7-30. Test Layout With Phase 0 and 180 Separations



SVRF Rule File Analysis

The following rule file is used with Matrix OPC.

Example 7-10. SVRF Rule File for Alt PSM

```

LAYOUT SYSTEM GDSII           //This top portion of the SVRF file
LAYOUT PATH "GDS/altpsm_input.gds" //declares the input and output
LAYOUT PRIMARY "*"             //specifications of the Calibre run.
FLAG SKEW YES
LAYOUT ERROR ON INPUT YES
PRECISION 2000
RESOLUTION 1

//Output Section
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "altpsm_post_opc.gds" GDSII PSEUDO
DRC SUMMARY REPORT "output.report"
DRC MAXIMUM VERTEX 4096
DRC KEEP EMPTY YES
    
```

Example 4: Matrix OPC for Alternating Phase Shift Masks

```

//Layer Input Section //The layers specified here are fed into Calibre
LAYER TARGET 1 //for processing. As long as the numbers are unique
LAYER TRIM 20 //for each layer, Calibre does not error out. The
LAYER PH0 17 //layer numbers specified here for input must match
LAYER PH18 18 //the layer numbers from the input layout file.

//Layer Output Section
// The layers specified here output the original input polygons that are
// fed into Calibre. This is important to do for Pre and Post OPC
// comparisons. Layer numbers defined here appear in the results
// database. Layer names do not appear in the results database.
TARGET {COPY TARGET} DRC CHECK MAP TARGET 1
TRIM {COPY TRIM} DRC CHECK MAP TRIM 100
PH0 {COPY PH0} DRC CHECK MAP PH0 2
PH18 {COPY PH18} DRC CHECK MAP PH18 3

//LITHO OPC calls to execute OPCpro
// These LITHO OPC calls execute OPCpro on the input layers for processing
// and edge movements. Because the same OPC setup file is used, the
// correction of all three layers (PH0, PH180, and TRIM) is processed
// concurrently. MAPNUMBER is also required for Calibre to generate the
// output layers concurrently.
OPCTARGET = LITHO OPC MAPNUMBER 1 FILE "setup/altpsm.in"
    TRIM PH0 PH18 TARGET
OPCPH18 = LITHO OPC MAPNUMBER 2 FILE "setup/altpsm.in"
    TRIM PH0 PH18 TARGET
OPCPH0 = LITHO OPC MAPNUMBER 3 FILE "setup/altpsm.in"
    TRIM PH0 PH18 TARGET
OPCTRIM = LITHO OPC MAPNUMBER 100 FILE "setup/altpsm.in"
    TRIM PH0 PH18 TARGET

//Outputting Post-OPC Polygons
// These layer declarations output the layers generated from the LITHO OPC
// calls for viewing in WORKbench. Without the DRC CHECK MAP commands, you
// are not able to see the output of the LITHO OPC calls.
OPCTARGET {COPY OPCTARGET} DRC CHECK MAP OPCTARGET 201
OPCTRIM {COPY OPCTRIM} DRC CHECK MAP OPCTRIM 300
OPCPH0 {COPY OPCPH0} DRC CHECK MAP OPCPH0 202
OPCPH18 {COPY OPCPH18} DRC CHECK MAP OPCPH18 203

// This LITHO PRINTIMAGE call executes PRINTimage on the corrected layers.
// An output of the simulation is passed to the results database.
PI = LITHO PRINTIMAGE FILE "setup/altpsm_printimage.in"
    OPCTRIM OPCPH0 OPCPH18 TARGET
PI {COPY PI} DRC CHECK MAP PI 500

```

OPC Setup File Analysis

The setup file used with this flow is shown in [Example 7-11](#) and [Example 7-12](#).

Example 7-11. Matrix OPC Setup File

```

# ----- Simulation models -----
modelpath ./models/
opticalmodel trim
secondOpticalmodel psm
resistpolyfile vt5.mod
# ----- OPC algorithm -----
iterations 10
tilemicrons 100.000000
stepsize 0.0005
gridsize 0.0005
siteinfo RESIST -numx 15 -center 7
cornerSiteStyle SITES_ON_ARC
lineEndAdjDist 0.080000
convexAdjDist 0.080000
concaveAdjDist 0.080000
# ----- Fragmentation -----
minfeature 0.060000
minedgeLength 0.050000
maxedgeLength 50.000000
cornedge 0.05 0.05 0.05 0.05 0.08
concavecorn 0.05 0.05 0.05 0.05 0.08
interfeature -interdistance 0.460000
-ripplelen 0.070000 -num 0
seriftype 0
minjog 0.020000
lineEndLength 0.180000
# ----- Layer info -----
background clear dark
# Layers
layer 1 TARGET 17 0 opc dark 0 0.9
layer 2 PH0 17 0 correction clear 1 1
layer 3 PH18 17 0 correction phase180 1 1
layer 100 TRIM 17 0 correction dark 0 0.9
#---- Arbitrary Commands Can Follow This Line. Don't delete this line! ----
sse OPC_MIN_EXTERNAL 0.05
sse OPC_MIN_INTERNAL 0.05
sse OPC_FEEDBACK -.6 -.6 -.7 -.8 -.8 -.8 -.7 -.6 -.6 -.6
sse OPC_VISUALIZE_MATRIX 1

```

Layer Definitions:
The TARGET for EPE measurement is defined as the opc layer. It should always be placed on mask0.

The Phase layers are PH0 and PH18. They are placed on mask1.

The TRIM layer is placed on mask0. mask0 has a background of clear, and is dark-tone. mask1 has a background of dark, and is clear-tone or phase180-tone.

Alternating PSM processes often have low MEEF, so using a higher feedback results in faster convergence.

Example 7-12. Matrix OPC Setup File (continued)

```

fragmentLayer TARGET {
    maxedgelen 0.180000 -split 1
    cornedge 0.090000 0.090000 0.090000 0.120000 rem 0.09
    concavecorn 0.110000 0.090000 0.090000 0.090000 0.120000 rem 0.09
    interfeatLayers TARGET PH0 PH18
}
fragmentLayer PH0 {
    maxedgelen 0.180000 -split 1
    cornedge 0.090000 0.075000 0.070000 rem 0.09
    concavecorn 0.070000 0.070000 0.070000 rem 0.09
    coincidentLayer TARGET -overlay 1
    interfeatLayers TARGET PH0
}
fragmentLayer PH18 {
    maxedgelen 0.180000 -split 1
    cornedge 0.090000 0.075000 0.070000 rem 0.09
    concavecorn 0.070000 0.070000 0.070000 rem 0.09
    coincidentLayer TARGET -overlay 1
    interfeatLayers TARGET PH18
    visibleLayers
}
fragmentLayer TRIM {
    minedgelen 0.080000
    cornedge 0.080000 0.080000 0.080000 0.080000 0.080000 0.080000 rem 0.09
    concavecorn 0.080000 0.080000 0.080000 rem 0.080000 rem 0.09
    coincidentLayer TARGET -overlay 1
    interfeatLayers TARGET TRIM
}

MATRIX OPC {
    SEARCH 0.2
    OPTSLOPE NO
    MINSLOPE 0
    METHOD closest
    METRIC euclidean
}

```

Fragmentation details:

- 1.) Fragments are defined on the TARGET (opc) layer.
- 2.) Using "coincidentLayer -overlay 1", these opc layer fragments are copied to the correction layer.

Use Matrix OPC

- Correction layer fragments are mapped to an opc layer fragment, if the distance is <= 0.2um.
- Slope optimization is not performed.
- Maps the correction fragment to the "closest" opc fragment.
- Uses "euclidean" search metric for measuring the distance.

Example 7-13. PRINTImage Setup File

```
# ----- Layer info -----
background clear dark
# Layers
layer 1 TARGET 17 0 opc dark 0 0.9
layer 2 OPCPH0 17 0 external clear 1 1
layer 3 OPCPH18 17 0 external phase180 1 1
layer 100 OPCTRIM 17 0 external dark 0 0.9

----- Arbitrary Commands Can Follow This Line. Don't delete this line! --
----- The settings used after this line should be the same as the OPC
# setup file.
```

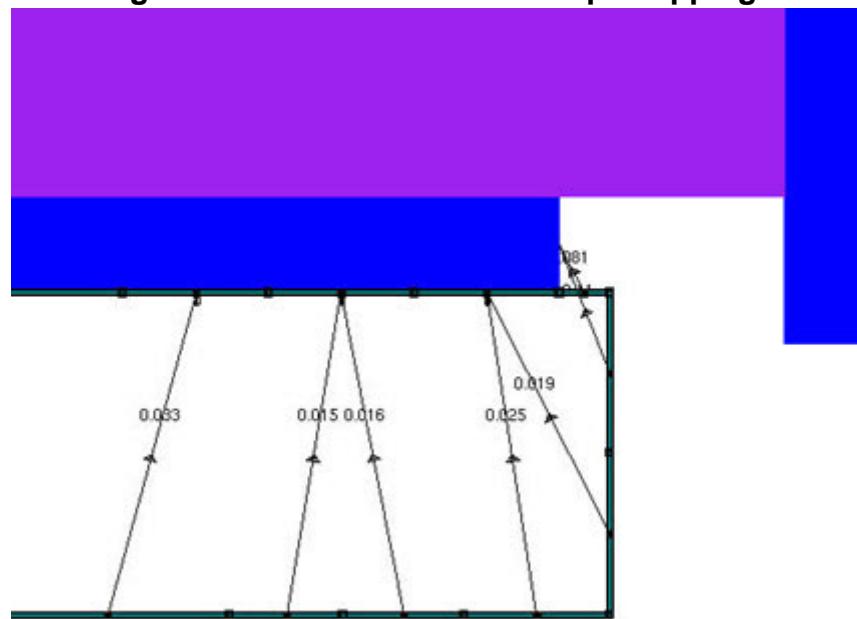
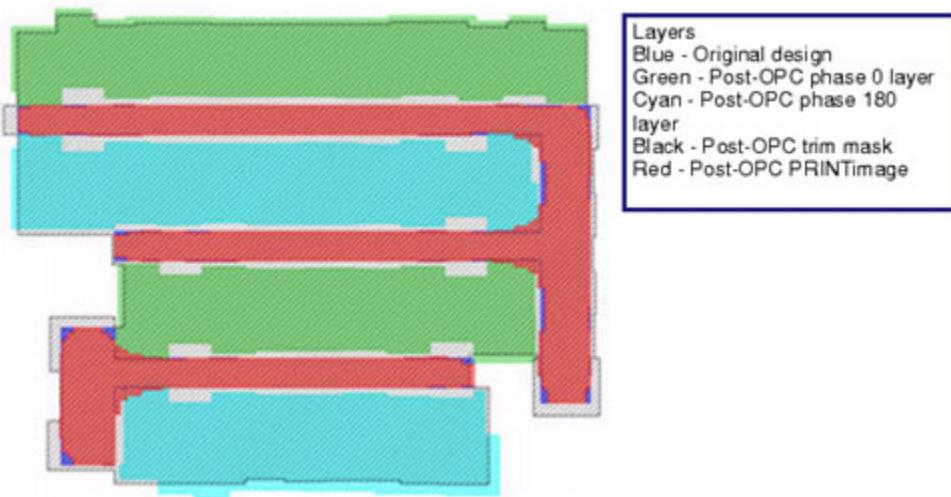
Layer definition of the PRINTIMAGE setup file is similar to the OPC setup, except:

We are inputting the post-OPC layers

Layers are defined as external instead of correction.

Results of Using Matrix OPC

Matrix OPC outputs a Tcl script that starts with “vis*N*” for each OPC iteration it performs. Mappings can be viewed for each iteration by choosing **Macros > Execute Tcl Script** in the menu options. The output of the Tcl script gives you an arrow which shows the correction layer to opc layer mapping. Alongside each fragment mapping is the MEEF value for that iteration. Each *vis<N>.tcl* file can be useful for debugging but the last one may be the most useful, because that states the final MEEF value for each mapping. [Figure 7-31](#) shows the mapping of the last iteration in this example. [Figure 7-32](#) shows the final OPC output from the Matrix OPC run.

Figure 7-31. Matrix OPC Tcl Script Mappings**Figure 7-32. OPC Results**

Chapter 8

Calibre ORC Procedures and Examples

Originally designed for performing Optical Rule Checking (ORC) to check the polygon edges in a layout to see whether or not they print within a user-specified tolerance, the Calibre ORC batch tool has evolved into a powerful tool that allows you to categorize polygon edges and fragments based on a wide range of properties and characteristics. You can use this tool many ways, chief among them being:

- Performing ORC to check the suitability for (before) and effectiveness of (after) an RET recipe.
- Categorizing edges and fragments prior to performing OPC, thereby providing additional control over all aspects of the correction process.
- Categorizing edges and fragments after performing OPC to support refinement and debugging of a setup file, model, or RET recipe.

The following sections are in this chapter:

| | |
|--|------------|
| Options for Running ORC | 596 |
| Tagging Basics | 597 |
| Properties Tagging | 597 |
| Relationship Tagging | 598 |
| Topological Tagging..... | 599 |
| Dimension Check Tagging..... | 600 |
| EPE Tagging..... | 601 |
| EPE Sensitivity Tagging..... | 602 |
| Image Tagging | 603 |
| Displacement Tagging | 605 |
| Boolean Tagging..... | 605 |
| Has_Site Tagging | 605 |
| MRC Tagging | 605 |
| Inspecting Tagged Data..... | 606 |
| Calibre Layer Output | 606 |
| Histogram Output | 611 |
| Control OPC With Tagging | 612 |
| Requirements for ORC in Standalone Mode | 614 |
| Requirements for ORC Within OPC Mode | 616 |
| Calibre ORC Example..... | 618 |

Options for Running ORC

You can run ORC in two modes: standalone or as part of a Calibre OPCpro run. Both modes are equally effective, and there is no appreciable run time difference between running OPC and ORC as separate processes versus running them together. The primary difference between running ORC standalone and running as a part of an OPC run is that in standalone mode, you can specify with much finer fragmentation than what is typically used for OPC.

The choice of mode is based on your workflow and individual need. [Table 8-1](#) highlights the differences between the two modes:

Table 8-1. Comparing ORC Standalone to OPCpro

| | Standalone | With OPCpro |
|--|---------------------------------------|--|
| Corrections performed | No | Yes |
| Ability to highlight tagged edges as boxes on highlight layers | Yes | Yes |
| Layer tagged | opc layer | correction layer if present, otherwise opc layer |
| EPE measurement | Between opc layer and simulated image | Between opc layer and simulated image |
| Tagging used to control OPC | No | Yes |
| Invocation from SVRF | LITHO ORC | LITHO OPC |
| Special tagging commands | | fastIter newTag -how DISPLACEMENT |

In both modes, using ORC can be thought of as a two step process:

1. Identifying edges of interest or concern (tagging).
2. Operating on those tagged edges, either inspecting them or controlling OPC.

Related Topics

[Inspecting Tagged Data](#)

[Control OPC With Tagging](#)

Tagging Basics

With the Calibre ORC batch tool, you use tagging commands to define conditions that are of special interest or concern. Like the Calibre OPCpro batch tool, this batch tool operates on edge fragments that result from fragmentation of polygon edges. The benefit of this is that you can identify exactly which portions of a polygon edge cause problems. A good fragmentation scheme breaks an edge into just enough fragments to ensure that portions of an edge with small errors or no errors are tagged differently than portions with larger errors.

Note



For more information on fragmentation, refer to “[Fragmentation Strategies](#)” on page 80.

ORC tagging commands let you tag fragments based on any combination of the following:

| | |
|---|---|
| Properties Tagging | Properties of individual fragments. |
| Relationship Tagging | Relationships with other fragments. |
| Topological Tagging | Relationships to marker layers. |
| Dimension Check Tagging | Width and spacing checks. |
| EPE Tagging | Differences between the drawn edge and the printed edge. |
| EPE Sensitivity Tagging | How responsive the etch or resist process results are to changes in image threshold. |
| Image Tagging | Aerial image properties such as intensity, contrast, IMAX, IMIN, aerial image factor, threshold, and slope. |
| Displacement Tagging | How far the fragment moved (was displaced) during correction (tagged after model-based OPC). |
| Boolean Tagging | Relationships between existing tags. |
| Has_Site Tagging | Whether or not they have a site. |
| MRC Tagging | Fragments that were not moved in the previous iteration of OPC. |

Properties Tagging

Property tagging refers to those -how methods or built-in tags that let you tag fragments based on length, angle, bias, and the type of fragment. The -how methods include:

- **edge** — Tags all fragments in an edge when the entire edge meets the constraints for proximity to a corner, corner angle, or length.
- **fragment** — Tags fragments based on their lengths, displacements, and the types of corners at either end.

The built-in tags let you manipulate various types of edges, without going through the trouble of isolating them using the -how methods described previously.

Table 8-2. Built-In Tags

| | |
|-------------------------|------------------------|
| all | convex_corner_adjacent |
| all_angle_line | horizontal_line |
| angle_45_line | line_end |
| concave_corner | line_end_adjacent |
| concave_corner_adjacent | opcable |
| convex_corner | vertical_line |

Relationship Tagging

Relationship tagging refers to seven -how methods that let you tag fragments based on their relationships to a nearby fragments. These are:

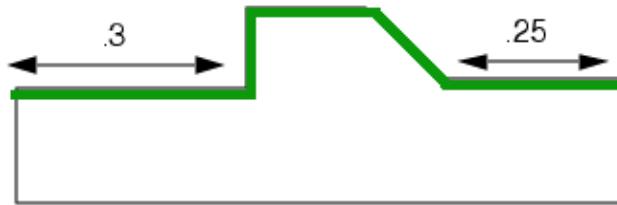
- **fragAfterEdge, fragAfterFrag, fragBeforeEdge, fragBeforeFrag, fragNextToEdge, fragNextToFrag** — Tag fragments based on their position relative to a specified edge or tag. All fragments are oriented in a direction so that the interior is always to the right of the fragment. To get to the fragment after a given fragment move along the fragment 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.
- **sequence** — Tags fragments based on whether or not they fit a specified length-angle-length pattern. This method identifies a sequence of *edges* (multiple fragments per edge) and places all fragments that make up the sequence into the output tag set. Using the optional argument, OUTPUT_ALL, it also lets you each fragment in the sequence into a second output tag indicating which edge of the sequence the fragment belongs to. The tag name for these additional output tags are derived from the *tagName* by appending *_n*, where *n* is an integer.

Sequence Tagging Example 1

Tag all fragments shown in the 5-edge sequence shown in green.

```
newTag t1 -how SEQUENCE all LENGTH == 0.3 ANGLE == 270 \
    LENGTH == 0.12 ANGLE == 90 \
    LENGTH < 0.2 ANGLE == 135 \
    LENGTH < 0.2 ANGLE == 225 LENGTH == 0.25
```

Figure 8-1. Sequence Tagging (Basic)

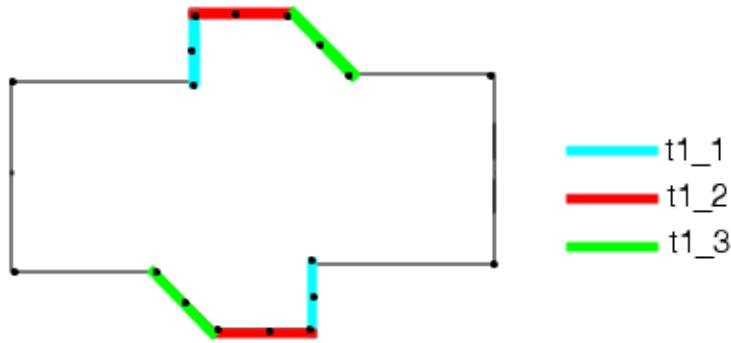


Sequence Tagging Example 2

Tag all fragments in the specified 3-edge sequence with t1, and also tags all the fragments making up the first edge with t1_1, all the fragments making up the second edge with t1_2, and all the fragments making up the third edge in the sequence with t1_3.

```
newTag t1 -how SEQUENCE all LENGTH == 0.12 ANGLE == 90 \
LENGTH < 0.2 ANGLE == 135 \
LENGTH < 0.2 OUTPUT_ALL
```

Figure 8-2. Sequence Tagging With OUTPUT_ALL



Topological Tagging

Topological tagging refers to sixteen -how methods that let you tag fragments based on their topological relationships to a marker layer. These methods are based on the SVRF operations with the same names. They select fragments only if the entire fragment meets the SVRF check constraints.

Note that these commands give different results depending on how island fragmentation is set for the marker layer. For all of the following examples, island fragmentation is on.

- **insideEdge** — Tags all *layer1* fragments that lie completely inside *layer2* polygons.

- **notInsideEdge** — Tags all *layer1* fragments not tagged by the corresponding insideEdge.
- **outsideEdge** — Tags all *layer1* fragments that lie completely outside *layer2* polygons.
- **notOutsideEdge** — Tags all *layer1* fragments not tagged by the corresponding outsideEdge.
- **coincidentEdge** — Tags *layer1* fragments coincident with *layer2*.
- **notCoincidentEdge** — Tags all *layer1* fragments not tagged by the corresponding coincidentEdge.
- **coincidentInsideEdge** — Tags *layer1* fragments inside-coincident with *layer2*.
- **notCoincidentInsideEdge** — Tags all *layer1* fragments not tagged by the corresponding coincidentInsideEdge.
- **coincidentOutsideEdge** — Tags *layer1* fragments outside-coincident with *layer2*.
- **notCoincidentOutsideEdge** — Tags all *layer1* fragments not tagged by the corresponding coincidentOutsideEdge.
- **touchEdge** — Tags all *layer1* fragments that are completely or partially coincidental with an edge on *layer2*.
- **notTouchEdge** — Tags all *layer1* fragments not tagged by the corresponding touchEdge.
- **touchInsideEdge** — Tags all *layer1* fragments that are completely or partially inside coincidental with an edge on *layer2*.
- **notTouchInsideEdge** — Tags all *layer1* fragments not tagged by the corresponding touchInsideEdge.
- **touchOutsideEdge** — Tags all *layer1* fragments that are completely or partially outside coincidental with an edge on *layer2*.
- **notTouchOutsideEdge** — Tags all *layer1* fragments not tagged by the corresponding notTouchOutsideEdge.

Dimension Check Tagging

Dimension check tagging lets you tag fragments based on width and spacing checks. The -how methods are:

- **External** — Tags fragments based on the DRC External dimension check.
- **Internal** — Tags fragments based on the DRC Internal dimension check.

By default, these tagging methods tag any fragment that would have been partially or completely selected by the DRC check in Calibre. That is, if any part of a fragment would be

returned by the DRC check, the entire fragment is tagged. Use the keyword FULL to tag only fragments that would be completely selected.

Depending on how you set up the newTag command, the dimension check can measure the distances between edges on one layer or on two, or belonging to one tag or two. In all cases, the results are a subset of *tag1*; that is, the results include ONLY fragments that were in the INPUT1 *tag1*.

- **1-input operation** — When only INPUT1 is specified, the check is a 1-layer DRC check between the fragments of the same layer.
- **2-input operation on same layers** — When INPUT1 and INPUT2 are both specified, and layer1 is the same as layer2, the check is a 2-layer check between two tags on the same layer.
- **2-input operation on different layers** — When INPUT1 and INPUT2 are specified, and layer1 is not the same as layer2, the check is a 2-layer check between the fragments of the first layer and the fragments of the second layer.

Note that both EXTERNAL and INTERNAL use the default orientation filters: PARALLEL ALSO, ACUTE ALSO, NOT PERPENDICULAR, and NOT OBTUSE.

EPE Tagging

EPE tagging tags fragments whose simulated edge lies within a specified distance from the drawn edge, which is on the opc layer. With this command, you can classify edges based on the type of edge, as well as the size of the placement error.

Usage is straightforward: you must specify the filter (the set of edges you are examining for errors) and the sizes of errors you are interested in (expressed as a range of distances between the drawn edge and the printed edge). Negative errors represent edges that print inside the polygon.

The following examples show this command in use.

Example 1: Tag any edge that deviates from the drawn edge by less than 0.02 microns in either direction.

```
newTag smallEPE -how EPE all -0.02 0.02
```

Example 2: Tag gates with an EPE larger than 0.03 microns. This example assumes you have already created a tag called “gates.”

```
newTag bad_gates -how EPE gates -0.03 0.03 not
```

Example 3: Tag line ends that print short, that is, inside the polygon. A very large negative lower bound and small negative upper bound ensures that the command tags all but the smallest negative errors.

```
newTag shortLE -how EPE line_end -0.3 -0.010
```

EPE Sensitivity Tagging

EPE sensitivity tagging lets you check how much the location of a printed edge varies as the process varies. This measure of process robustness is called the *EPE sensitivity*. It is expressed as a distance, defined as follows:

$$S = |EPE_1 - EPE_2|$$

where:

- EPE_1 is the distance between the drawn edge and the edge as it would print with the threshold at T_1 .
- EPE_2 is the distance between the drawn edge and the edge as it would print with the threshold at T_2 .

Negative EPEs represent edges that print inside the polygon.

The fragments are tagged using the `newTag ... -how EPESENSITIVITY` command. You can specify the change in threshold either of two ways:

- **VTR method** lets you specify the threshold as a function of the process model. With the VTR method, you define a model perturbation by specifying a fractional decrease in threshold (p_1) and a fractional increase in threshold (p_2). The Calibre ORC batch tool then computes two new thresholds using the current resist model.

$$T_1 = T_{\text{model}}(1 - p_1)$$

$$T_2 = T_{\text{model}}(1 + p_2)$$

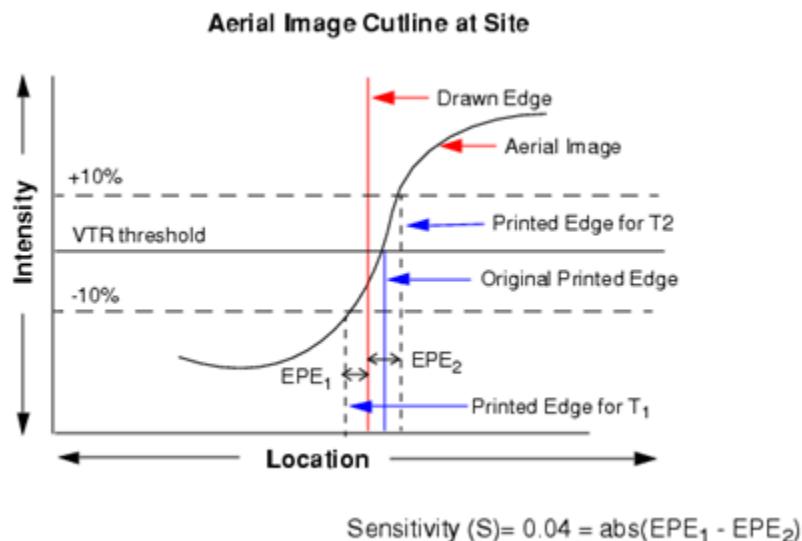
- **CTR method** lets you specify the change in threshold as a constant, explicitly defining upper and lower thresholds. With the CTR method, you specify the actual intensity values to use as the upper and lower thresholds.

$$T_1 = t_1$$

$$T_2 = t_2$$

[Figure 8-3](#) shows a simulation and the aerial image at that site. This figure shows how varying the threshold causes the edge to print differently. The VTR threshold pictured is varied by -10% and +10%. The sensitivity, S , is the difference between the two EPE values.

Figure 8-3. EPE Sensitivity



To use the `epeSensitivity` command, you must specify the filter (the set of edges you are examining for errors), the change in threshold, and the range of sensitivity values you are interested in. The following examples show how you can tag edges based on EPE sensitivity.

Example 1: Tag any gate edge whose EPE changes by more than 1 nm when the threshold varies by 5%.

```
newTag very -how epeSensitivity gates VTR 0.05 0.05 range .001 1
```

Example 2: Tag edges whose EPEs change more than 20, 30, and 40 nm, respectively, when the threshold varies from 0.25 to 0.28.

```
newTag twenty -how epeSensitivity all CTR 0.25 0.28 range 0.02 1
newTag thirty -how epeSensitivity all CTR 0.25 0.28 range 0.03 1
newTag forty -how epeSensitivity all CTR 0.25 0.28 range 0.04 1
```

Related Topics

[newTag ... -how EPESENSITIVITY](#)

Image Tagging

Image tagging tags fragments based on the properties of their aerial image. You can check for:

- **Intensity** — Specified as a range of aerial image intensity values plus distances along the control site. Distances inside the polygon are expressed as a range of negative values reflecting distances from the drawn edge; distances outside the polygon are expressed as a range of positive values reflecting distances from the drawn edge.

- **Slope** — Specified as an interval of Dintensity-per-micron values. In addition, you can either use the LOG keyword to use the LOG slope, or the YVAL keyword to evaluate the slope at the position where the image intensity matches the specified threshold.
- **Aerial Image Factor** — Specified as a range of values using the C_factor_constraint keyword.
- **IMAX** or **IMIN** — Specified as a single value.
- **Threshold** — Specified as a range of values.
- **Contrast** — Specified as

$$\text{contrast} = \frac{\max(I) - \min(I)}{\max(I) + \min(I)}$$

You must also include the extrema_distance, which defines a search window for the calculations.

For additional control over tagging based on aerial image properties, you can use the arguments NOT or EXTREMA_DISTANCE:

- **NOT** — This option complements all subsequent constraints, instructing the application to tag those fragments that do not meet the specified constraints.
- **EXTREMA_DISTANCE** — Used with IMAX, IMIN, or CONTRAST to define a search window for the calculations. This option requires a closed distance interval as an argument, such as “> -0.1 < 0.1”. The secondary keyword, QUAD_INTERP, instructs the operation to calculate these values using a three-point quadratic interpolation of the sample image values. By default, it does not use interpolation.

Note

 EXTREMA_DISTANCE is *required* when tagging is based on CONTRAST or tagging is based on intensity (IMIN or IMAX) using a resist model that is version 3 or later (VT5).

The following examples show the IMAGE tagging command in use.

Example 1: Identify line-end bridging by finding edges having an aerial image intensity greater than 0.3 for the entire range of 0 to 0.1 microns from the line end.

```
newTag t2 -how image line_end IMAGE > 0.3 DISTANCE >0 <=0.1
```

Example 2: Tag isolated lines that may not print because the aerial image is low for distances both inside and outside the drawn edge. The tag iso_line must be defined before this command is issued.

```
newTag t3 -how image iso_line IMAGE < 0.1 DISTANCE > -0.5 < 5.0
```

Example 3: Tag line ends with a contrast less than 0.7

```
newTag low_contrast_ends -how IMAGE line_end CONTRAST < 0.7
```

Displacement Tagging

Displacement tagging tags fragments *after model-based OPC* based on how far the fragment moved (was displaced) during correction. The -how method is DISPLACEMENT.

Displacement tagging only has meaning when used in ORC with OPCpro mode. To use it, you must place the command after the fastIter command. For more information on using the fastIter command, refer to “[Requirements for ORC Within OPC Mode](#)” on page 616.

Boolean Tagging

Boolean tagging tags fragments based on relationships between existing tags.

The options are:

- ANDTAGS
- ORTAGS
- SUBTRACTTAGS

Example 1: Identify all fragments that are not line ends.

```
newTag notLE -how subtractTags ALL LINE_END
```

Example 2: Tag vertical line ends with a different tag than horizontal line ends, for use with an asymmetrical illumination source

```
newTag vertLE -how ANDTAGS line_end vertical_line
newTag horzLE -how ANDTAGS line_end horizontal_line
```

Has_Site Tagging

The HAS_SITE -how method lets you tag fragments based on whether or not they have sites on them.

MRC Tagging

The MRC -how method lets you tag fragments that did not move in the previous iteration. The restrictions could have originated from an MRC rule or other movement limitation.

The following example creates a new tag, “blocked_fragments,” that applies to any movement-restricted fragment in the set “large_epe”:

```
newTag blocked_fragments -how MRC large_epe
```

Inspecting Tagged Data

The Calibre ORC batch tool identifies problem edges by processing the tagging commands you used to define the problems that interest you. To inspect the errors, you must define an output format explicitly. By default, the ORC operation does not return any results to the hierarchical engine.

The two options available to you are saving ORC results as a Calibre layer data or reporting ORC results in a histogram.

| | |
|-----------------------------------|------------|
| Calibre Layer Output | 606 |
| Histogram Output | 611 |

Calibre Layer Output

Writing ORC results as GDS layer data allows you to visually inspect the layout and distinguish between the various types of problems. To do this, you must generate new layers containing highlighting boxes drawn around each edge in the specified tag, then output the layer as a Calibre layer.

Generating Highlighting Boxes

The [tags2boxes](#) command does two thing: It defines a mapping between tag names and new (not defined in the setup file) polygon layers, and creates highlighting box drawn for every fragment in the associated tag, as shown in [Figure 8-4](#). The command can operate in either of two modes:

- TAG draws boxes around the fragments in the specified tags. (default)
- SITE draws boxes around sites on fragments in the specified tags.

Note

 The Calibre ORC batch tool executes the Tagging commands in the order in which they appear in the setup file. The commands that generate the tags must appear before the tags2boxes commands that reference the tags.

You define the mapping between the tags and the new layers through the -layers argument, followed by an ordered list of layer numbers. The order of the layer numbers matches the order of the tags. Thus, boxes for the first tag are drawn on the first layer, boxes for the second tag are drawn on the second layer, and so on. The following example shows this command in use:

```
tags2boxes -tags err1 err2 err3 err4 -layers 3 4 5 6
```

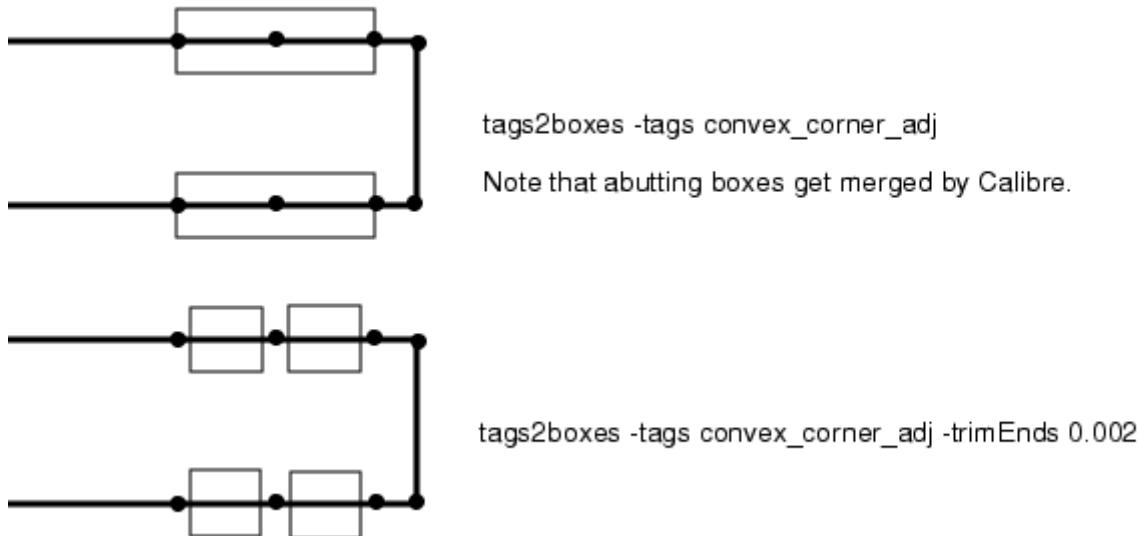
Additional arguments to the tags2boxes command, shown in [Table 8-3](#), control the placement, position, and size of the highlighting boxes.

Table 8-3. Arguments to tags2boxes, by Mode

| | TAG mode | SITE Mode |
|--------------------------------|---|--|
| Syntax | default mode | insert -sites between tags2boxes and -tags |
| Placement | drawn around the fragment | drawn around the site for the fragment |
| Box width, default | 0.02 | 0.02 |
| User control over box width | -halfWidths {hw1 hw2 ... hwn} defines $\frac{1}{2}$ width per tag. Width = 2* -halfWidths | — |
| Box length, default | length of fragment | length of site |
| User control over box length | -trimEnds trimD length = (frag length) - (2* -trimD) | -siteLength |
| Box position, default | centered over the fragment | centered over site |
| User control over box position | -offsets {d1 d2...dn} defines one offset per tag -offsets epe offset by the EPE distance | — |

Using -trimEnds is essential if you want to find the total number of tagged fragments by counting the number of boxes. The Calibre hierarchical engine merges all abutting geometries, so without -trimEnds you may end up with fewer boxes than tagged fragments.

Figure 8-4. Controlling the Size of Highlight Boxes



Returning the Boxes Layers as Output

While one tags2boxes command can map 4096 tags to as many layers, each LITHO operation in an SVRF file returns only one layer. By default, the operation merges all the tags2boxes layers into a single GDS layer. Using either the MAP or MAPNUMBER options to the LITHO operation and concurrent operations, you can write each tags2boxes layer to a different GDS layer.

MAP and MAPNUMBER specify which of the many tags2boxes layers the LITHO ORC operation returns. MAP lets you specify the layer by name, MAPNUMBER lets you specify it by number. When using MAP the layer name must be the same as the name of the tag being output on that layer. When using MAPNUMBER, the number must be the same as the layer number assigned to that tag in the tags2boxes command.

Concurrent operations let you issue the LITHO operation multiple times, yet process the data only once. Rather than create many different setup files, each identifying a single type of error and returning this information on a single tags2boxes output layer, you can create one setup file to identify all the errors and return each as a different layer. In the SVRF file, you include one LITHO operation for each output layer. Each LITHO operation references the same layers and the same setup file. The difference between them is that each must include the MAPNUMBER or MAP keyword identifying a different layer to output. For example, the following LITHO operations would be processed concurrently:

```
LITHO ORC File orc_poly.in POLY DIFF MAPNUMBER 3
LITHO ORC File orc_poly.in POLY DIFF MAPNUMBER 4
LITHO ORC File orc_poly.in POLY DIFF MAPNUMBER 5
LITHO ORC File orc_poly.in POLY DIFF MAPNUMBER 6
```

Because the LITHO statement is a layer constructor operation, you must save the results as a rule check statement in order for it to persist. The final step in saving the data as a GDS layer is to instruct the Calibre nmDRC hierarchical engine to write the specified rule check statement to the results database using the DRC CHECK MAP statement.

To store the data properly, the previous example can be written as follows:

```
err1 {LITHO ORC File orc_poly.in POLY DIFF MAPNUMBER 3}
err2 {LITHO ORC File orc_poly.in POLY DIFF MAPNUMBER 4}
err3 {LITHO ORC File orc_poly.in POLY DIFF MAPNUMBER 5}
err4 {LITHO ORC File orc_poly.in POLY DIFF MAPNUMBER 6}

DRC CHECK MAP err1 101
DRC CHECK MAP err2 102
DRC CHECK MAP err3 103
DRC CHECK MAP err4 104
```

Example 1: Equivalent Rule Files Using MAP and MAPNUMBER

SVRF Rule File Using MAP:

```
epc_0_10= LITHO OPC FILE "setup.in" poly island MAP epc_0_10
epc_0_n10 = LITHO OPC FILE "setup.in" poly island MAP epc_0_n10
convex_adj = LITHO OPC FILE "setup.in" poly island MAP convex_adj
line_end = LITHO OPC FILE "setup.in" poly island MAP line_end
```

SVRF Rule File Using MAPNUMBER:

```
epc_0_10 = LITHO OPC FILE "setup.in" poly island MAPNUMBER 100
epc_0_n10 = LITHO OPC FILE "setup.in" poly island MAPNUMBER 101
convex_adj = LITHO OPC FILE "setup.in" poly island MAPNUMBER 102
line_end = LITHO OPC FILE "setup.in" poly island MAPNUMBER 103
```

Setup file (for both cases):

```
tags2boxes -tags epc_0_10 epc_0_n10 convex_adj line_end \
-layers 100 101 102 103
```

Example 2: Passing Layers Between the Setup File and SVRF Rule File

Use the following methods to output tagging data with the tags2boxes command. Method 1 uses the tags2boxes option “-tags” and the SVRF LITHO option “MAP”. Method 2 focuses on using “-tags”, “-layers” and “MAPNUMBER”.

Method 1: Using Only “-tags” and “MAP”

- In the setup file:

```
tags2boxes -tags tag1 tag2 tag3
```

- In the SVRF rule file:

```
x = LITHO OPC FILE "setup.in" input_layer1 input_layer2 input_layer3
    MAP tag1
y = LITHO OPC FILE "setup.in" input_layer1 input_layer2 input_layer3
    MAP tag2
z = LITHO OPC FILE "setup.in" input_layer1 input_layer2 input_layer3
    MAP tag3
x {COPY x}
y {COPY y}
z {COPY z}
DRC CHECK MAP x 101
DRC CHECK MAP y 102
DRC CHECK MAP z 103
```

Method 2: Using “-tags”, “-layers” and “MAPNUMBER”

- In the setup file:

```
tags2boxes -tags tag1 tag2 tag3 -layers 101 102 103
```

- In the SVRF rule file:

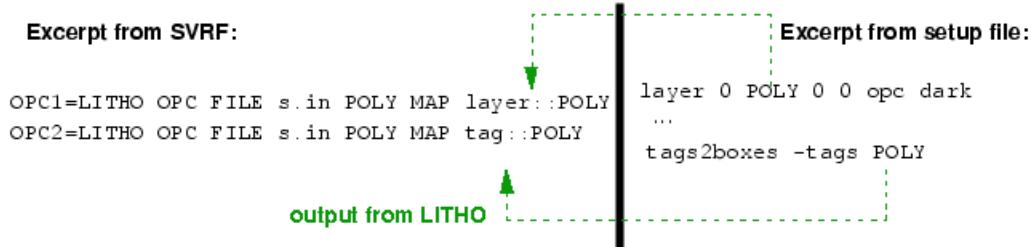
```
x = LITHO OPC FILE "setup.in" input_layer1 input_layer2 input_layer3  
    MAPNUMBER 101  
y = LITHO OPC FILE "setup.in" input_layer1 input_layer2 input_layer3  
    MAPNUMBER 102  
z = LITHO OPC FILE "setup.in" input_layer1 input_layer2 input_layer3  
    MAPNUMBER 103  
x {COPY x}  
y {COPY y}  
z {COPY z}  
DRC CHECK MAP x 101  
DRC CHECK MAP y 102  
DRC CHECK MAP z 103
```

Tag and Layer Name Qualifiers

When using MAP to return corrected opc or correction layers, use the layer name. When using MAP to return the output from a tags2boxes command, use the tag name. To prevent a conflict should your tag name match an existing layer name, prepend the name qualifier, “tag::” to output the layer tag in the setup file. To indicate the opc or correction layer results, you can use the name qualifier “layer::”, or it can be omitted since it is assumed by default.

The following example shows a case where the tags2boxes results and the OPC output results for POLY are mapped using MAP with names qualifiers.

Figure 8-5. MAP Using Name Qualifiers



Note

Do not name user tags that begin with “tag::” or “layer::” as they result in setup file errors.

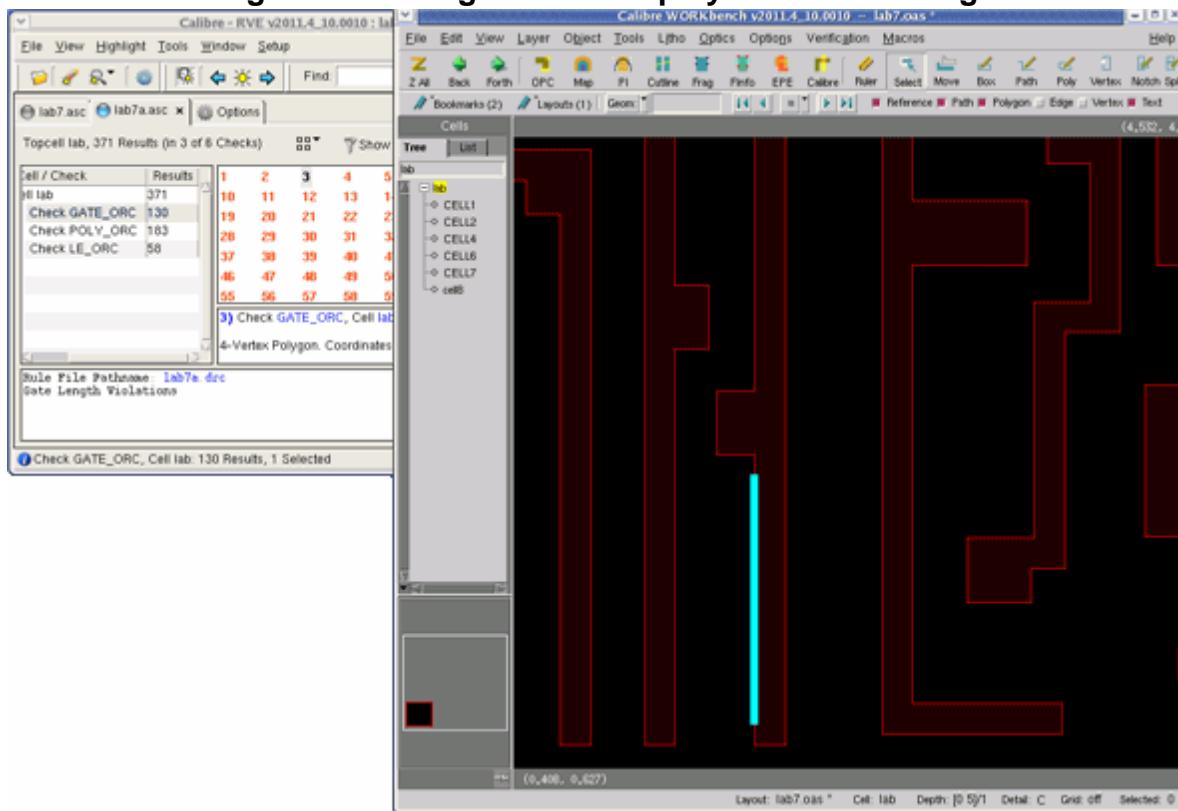
Viewing ORC Output Layers

Once the Calibre ORC batch process is complete, and the results (and input layers) are stored as a GDS file, you can view these results in either the Calibre WORKbench or Calibre LITHOview application. For larger files, you can use the Calibre WORKbench or Calibre LITHOview application concurrently with Calibre RVE to view and step through the errors.

Figure 8-6 shows the Calibre RVE and Calibre WORKbench applications open at the same time. On the left, the RVE user interface displays the error geometries, by number. On the right,

the actual highlight boxes drawn around problem fragments are highlighted in the Calibre WORKbench display area.

Figure 8-6. Using RVE to Simplify Results Viewing



Histogram Output

A histogram is a chart showing the number of occurrences of each type of error. You generate histogram data during ORC using the histogram tagging command, which instructs the Calibre ORC batch tool to generate a text file containing histogram data. To use this command, you should also specify the pathname for the file using the LITHO_HISTOGRAM_OUTPUT_FILE setup file variable. Otherwise, the histogram data is sent to the file. If no file is defined, histogram data is placed in the transcript.

To use this command, you must specify the name of the histogram, the tags you want to include in the histogram, and the label (or bin) to use for each tag count in the histogram. The following example shows how you can use histograms to output various types of design information.

Example 1: Classify edges with EPEs according to the size of the EPE for the edge, then generate a histogram showing how many edges there are for each size of error.

```
newTag badEPE0 -how EPE all -5 -1
newTag badEPE1 -how EPE all -1 -0.01
newTag badEPE3 -how EPE all 0.01 1
newTag badEPE4 -how EPE all 1 5
histogram MY_HIST -tags badEPE0 badEPE1 badEPE3 badEPE4 \
    -bins big_in small_in small_out big_out
```

The output file, when viewed in a text editor, looks like this:

```
oooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooo
>>> Histograms created by LITHO SVRF <<<
oooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooo
MY_HIST
Big_in 23
Small_in 103
Small_out 89
Big_out 17
oooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooo
```

Because the histogram is a text file, you can copy and paste the numbers from the histogram text file into a spreadsheet program for graphing purposes.

Note

 You should not use the histogram command to generate counts of errors before and after OPC.

Related Topics

[histogram](#)

[LITHO_HISTOGRAM_OUTPUT_FILE](#)

Control OPC With Tagging

One of the primary uses of ORC with OPCpro is gain more precise control over correction.

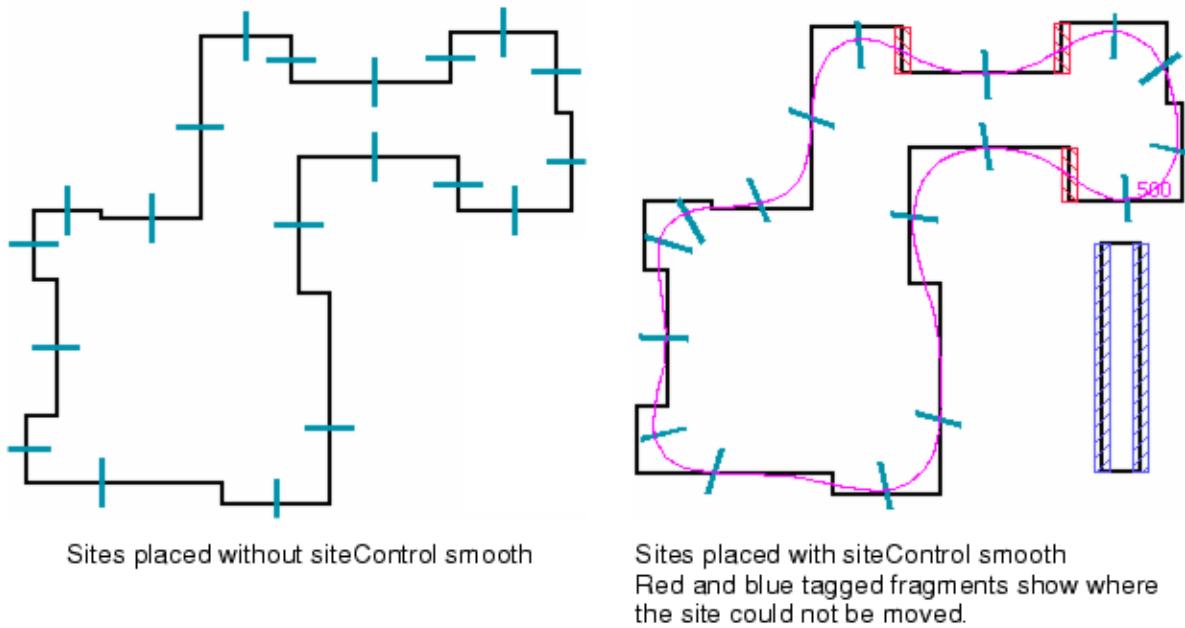
Using Tags To Control Sites

The siteControl tagging command lets you control site placement and orientation on a tag by tag basis. These are:

- siteControl DELETE — Deletes sites on all fragments in tag.

- siteControl OFFSET_FROM — Defines a placement and orientation for the sites on each of the fragments in the tag. If the fragment already had a site, this moves it. If the fragment did not have a site, it creates one.
- siteControl SMOOTH — Generates a smoothed contour of the polygon, then moves existing sites on each of the fragments in *tag onto* that contour, oriented perpendicular to it. This is one of the more complex tagging commands.

Figure 8-7. Moving Sites To Fit a Contour



Using Tags To Control Which Model To Use

With the bindModelToTag tagging command you can control which model is used for performing simulations on a tag by tag basis. Note that the model must be one that has been defined by the “processModel *modelname modelfile*” setting.

Using Tags To Define Separate Width and Space Rules

The MRC_RULE command lets you constrain OPC by defining tag-specific width and spacing rules. The Calibre OPCpro batch tool can only move a fragment if it does not create a violation of these rules.

You can define a prioritization scheme to resolve any conflicting rules by the order in which you list the rules, or by creating subsets of tags. Tag-specific spacing and width rules have higher priority than the default MRC_RULE.

Using Tags To Define Offsets

The opcTag tagging command lets you define offsets to apply to fragments either before or after they have been moved by the Calibre OPCpro batch tool. Once offset, you can instruct the Calibre OPCpro batch tool to perform additional iterations to arrive final corrections.

The opcTag command gives you five tagging options:

- **-freeze** — Prohibits further changes to the tagged fragments.
- **-hintOffset** — Calculates the fragment position by adding the offset to the original position.
- **-fixedOffset** — Calculates the fragment position by adding the offset to the original position, then freezes the fragment so no corrections can be applied.
- **-hintOffsetIncr** — Calculates the fragment position by adding the offset after movement by the Calibre OPCpro batch tool. After this operation, you typically use the fastIter command to instruct the Calibre OPCpro batch tool to “process” the hints into final bias offsets.
- **-fixedOffsetIncr** — Calculates the fragment position by adding the offset after movement by the Calibre OPCpro batch tool, then freezes the fragment so no further corrections can be applied.

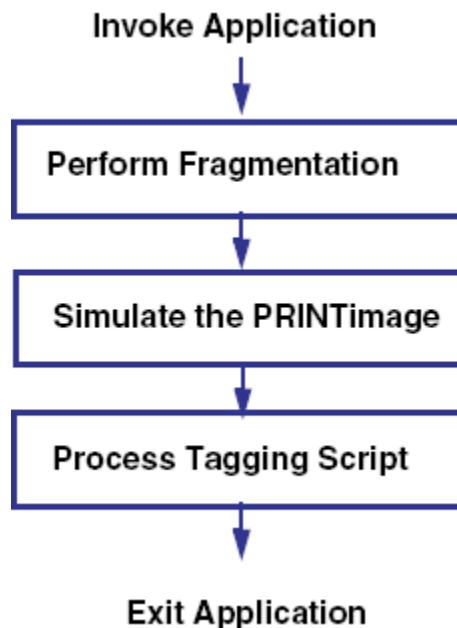
Related Topics

[MRC_RULE](#)

Requirements for ORC in Standalone Mode

Calibre ORC can be run as an independent operation, with no need to first run an OPC tool. Within the ORC setup file, there are keywords to perform fragmentation, simulate the print image, tag fragments and postprocess them.

Figure 8-8. Processing Order in Standalone Mode



Files Required to Run the Calibre ORC Batch Tool

The following files are required in order to run the Calibre ORC batch tool:

1. A layout database: the input OASIS or GDSII file.
2. An SVRF file containing the [LITHO ORC](#) operation.
3. A setup file containing the following:
 - o Layer definitions.
 - o In-line models or paths to optical and resist model files.
 - o Fragmentation parameters.
 - o Tagging commands identifying particular types of edges and tolerances.
 - o Output commands, generating either highlighting boxes or a histogram.

Layers Used by ORC

Table 8-4. Layer Usage With the ORC Batch Tool

| Layer | Allowed | Usage |
|-------|---------------|--|
| opc | yes, required | EPEs are measured between PRINTimage and this layer. Layer is tagged unless an external is supplied, then it is used only for calculating EPEs. |

Table 8-4. Layer Usage With the ORC Batch Tool (cont.)

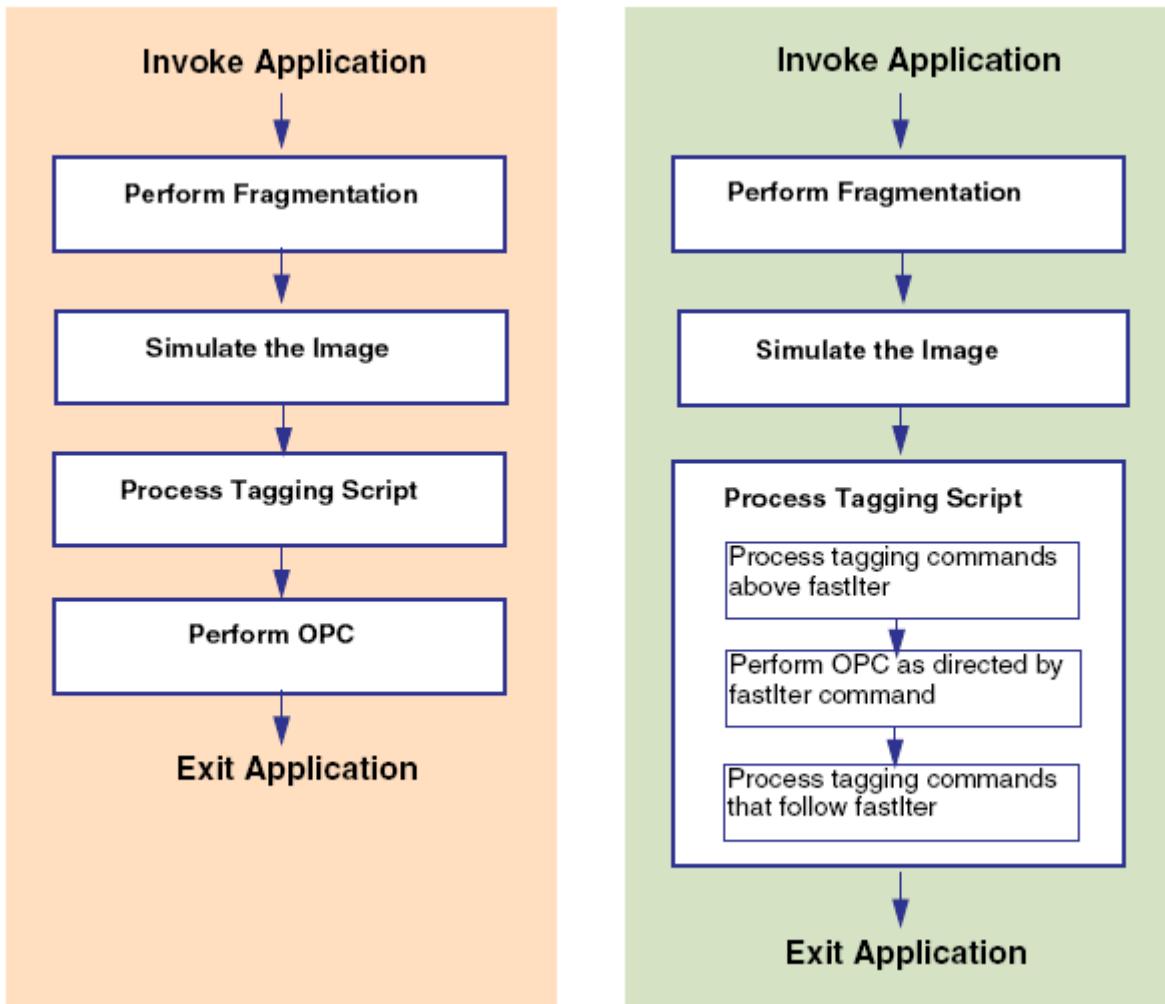
| Layer | Allowed | Usage |
|------------|---------|---|
| correction | no | No usage. |
| visible | yes | Factored into the aerial image, not tagged. |
| asraf | yes | Factored into the aerial image, not tagged. |
| external | yes | Tagged if present. |
| island | yes | island layer fragmentation. |

Requirements for ORC Within OPC Mode

When running ORC with Calibre OPCpro, you must choose between two possible flows. The difference lies in whether you use the fastIter command to trigger corrections or not.

The fastIter tagging command instructs Calibre OPCpro to perform the specified number of iterations of correction on the specified tags. Because the application evaluates tagging commands in the order it encounters them, inserting the fastIter command in the middle of the tagging scripts forces the application to evaluate some tagging commands with respect to uncorrected data and other tagging with respect to corrected data. Placing the fastIter command at the end of the tagging script and applying corrections to the built-in tag ALL is identical to running without fastIter.

Figure 8-9. Processing Order Using ORC With OPC



Things to keep in mind:

- When you use fastIter, Calibre OPCpro only calculates corrections for those fragments in the specified tag or tags.
- When you use fastIter, Calibre OPCpro ignores the iterations keyword. That is, it only performs the number of iterations specified by the fastIter command.
- Some tagging commands, such as DISPLACEMENT, are only meaningful when used after the fastIter command.
- Using fastIter does not affect the run time.

Layers Used by ORC When Run Within OPCpro

Table 8-5. Layer Usage With the ORC Batch Tool Usage

| Layer | Allowed | Usage |
|------------|---------------|--|
| opc | yes, required | EPEs are measured between PRINTimage and this layer. Layer is tagged unless an external is supplied, then it is used only for calculating EPEs. |
| correction | yes | Tagged if present. Corrected by OPC if present. |
| visible | yes | Factored into the aerial image, not tagged. |
| asraf | yes | Factored into the aerial image, not tagged. |
| external | no | Factored into the aerial image, not tagged. |
| island | yes | Used for island layer fragmentation. |

LITHO ORC Usage When Run Within OPCpro

When running ORC with Calibre OPCpro, you do not invoke the ORC application at all. Usage is identical to performing sparse OPC alone.

Calibre ORC Example

This section contains a Calibre ORC usage example. It uses the Calibre ORC batch tool to find line_ends and compare them to EPEs. You output both types of fragments (line_ends and errors) as GDS layers. This allows you to discover which portion of the errors are caused by line_ends.

Finding and Outputting Errors

After defining the input layers (in this case there is one input layer, type opc) you can reference them in the tagging script.

Note

 Tagging commands are always written with respect to the opc layer. If an external layer is defined, EPEs are measured as the distance between the external layer and the opc layer.

The following is an excerpt from the setup file used with this example:

```
# ----- Layer info -----
background clear
# Layers
layer 2 POLY 0 0 opc dark

----- Arbitrary Commands Can Follow This Line -----
clearTagScript
newTag dangerousEPE -how EPE all -10 -0.01
tags2boxes -tags dangerousEPE line_end -layers 101 102
```

Notice that you only created one new tag, dangerousEPE. The other tag, line_end is a built-in tag.

Performing the LITHO Operation

Issue the operation twice, using the MAPNUMBER option to the operation to signal concurrent operations and control which output layer each operation returns. The portion of the SVRF file that implements this functionality would look like this:

```
dangerous_errors = LITHO ORC FILE orc.in POLY MAPNUMBER 101
line_ends = LITHO ORC FILE orc.in POLY MAPNUMBER 102

dangerous_errors {copy dangerous_errors}
line_ends {copy line_ends}

DRC CHECK MAP dangerous_errors 101
DRC CHECK MAP line_ends 102
```

This set of SVRF statements and operations saves the ORC results as derived layers, creates rule check statements that are copies of the derived layers, then writes the rule check statements to the GDS results database. The layer GDS layer numbers happen to be the same as the Calibre layer numbers, but this is not a requirement.

Finding and Outputting Errors

The input GDS file contains both the original mask design and the OPC corrected design, with the original data on layer 2 (POLY) and the corrected data on layer 3 (CORR). In the setup file, set the RET layer type for the original mask design to opc layer (the target) and the layer type for the corrected data to type external (the data to be simulated.) Refer to “[Layer Usage](#)” on page 120 for layer information.

Using these layers as input, you can classify EPEs according to how far the simulated edge differs from the drawn layer. Create a different tag for each range of errors. The following is an excerpt from the setup file:

```
# ----- Layer info -----
background clear
# Layers
layer    0 CORR    0 0 external   dark
layer    1 POLY    0 0 opc        dark

----- Arbitrary Commands Can Follow This Line -----
clearTagScript
newTag badEPE0 -how EPE all -5 -1
newTag badEPE1 -how EPE all -1 -0.01
newTag badEPE3 -how EPE all 0.01 1
newTag badEPE4 -how EPE all 1 5
tags2boxes -tags badEPE0 badEPE1 badEPE3 badEPE4 -layers 100 101 102 103
```

Performing the LITHO Operation

Compare the following excerpts from two SVRF files, presented side by side.

Note

 Although the Calibre layer numbers in the previous example are the same as in the setup file, this is not required.

orc.drc Excerpts

```
LAYER POLY 2
LAYER CORR 3

// ORC Operation using LITHO
// The file orc.in is a LITHO setup file

MY_ORC {
    LITHO ORC FILE orc.in POLY CORR
}

POLY {COPY POLY}
CORR {COPY CORR}

DRC CHECK MAP POLY 2
DRC CHECK MAP CORR 3
DRC CHECK MAP MY_ORC 4
```

concur_orc.drc Excerpts

```
LAYER POLY 2
LAYER CORR 3
// ORC Operation using LITHO
// The file orc.in is a LITHO setup file
// MY_ORC {
// LITHO ORC FILE orc.in POLY CORR
// }

MY_ORC_L1 = LITHO ORC FILE orc.in POLY CORR MAPNUMBER 100
MY_ORC_L2 = LITHO ORC FILE orc.in POLY CORR MAPNUMBER 101
MY_ORC_L3 = LITHO ORC FILE orc.in POLY CORR MAPNUMBER 102
MY_ORC_L4 = LITHO ORC FILE orc.in POLY CORR MAPNUMBER 103
POLY {COPY POLY}
CORR {COPY CORR}

MY_ORC_L1 {COPY MY_ORC_L1}
MY_ORC_L2 {COPY MY_ORC_L2}
MY_ORC_L3 {COPY MY_ORC_L3}
MY_ORC_L4 {COPY MY_ORC_L4}
DRC CHECK MAP POLY 2
DRC CHECK MAP CORR 3
DRC CHECK MAP MY_ORC_L1 100
DRC CHECK MAP MY_ORC_L2 101
DRC CHECK MAP MY_ORC_L3 102
DRC CHECK MAP MY_ORC_L4 103
```

Chapter 9

Calibre PRINTimage Procedures and Examples

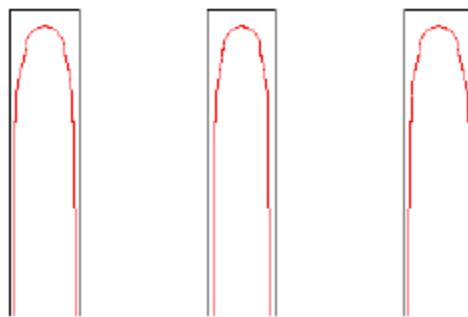
The Calibre PRINTimage batch tool performs optical and resist simulation to predict how mask features actually print on a wafer. You can use the Calibre PRINTimage batch tool in many ways, including the following:

- Analyze the simulated wafer to discover which mask areas or feature types require correction.
- Evaluate the effectiveness of OPC.
- Check for the existence of side-lobes or other residual features.
- Compare different RET approaches.
- Perform DRC operations on the simulated image.

The Calibre PRINTimage batch tool creates a derived layer containing the simulated data. Depending on how you configure the tool, this derived layer contains either a simulated scanning electron micrograph (SEM) image or a dense aerial image. The following sections describe both image types.

By viewing two or more layers superimposed on each other, you can see exactly how the exposure or printing process affects the individual mask features. You can compare the derived PRINTimage layer to the original drawn layer or compare the results from different correction methodologies or different processes. [Figure 9-1](#) shows the Calibre PRINTimage results for three uncorrected lines superimposed on the original drawn shapes. Notice the corner rounding and line-end shortening depicted by the red, simulated SEM layer.

Figure 9-1. Corner Rounding



| | |
|--|------------|
| SEM, Dense Aerial, and Dense Print Images | 622 |
| SEM Image Controls | 624 |

| | |
|---|-----|
| Calibre PRINTimage Results | 626 |
| To Run the Calibre PRINTimage Batch Tool. | 627 |
| Calibre PRINTimage Example. | 628 |

SEM, Dense Aerial, and Dense Print Images

The Calibre PRINTimage batch tool generates three types of images: SEM images, dense aerial, and dense print images. Differences in how and where this application simulates the images lead to differences in the types of information they provide, which in turn lead to different uses. The following sections describe these differences.

PRINTimage Sampling Points

To create an image, the Calibre PRINTimage batch tool calculates simulation data at sampling points throughout the design. The three image types (SEM, dense print, and dense aerial) differ in terms of the locations of the sampling points.

For SEM images, the application samples data at control points. Because the RET batch tools only create control sites on fragments, the SEM image simulator only predicts the image of the wafer for those portions of the mask that contain fragmented polygon data.

For a complete discussion of fragmentation and control sites, refer to “[Key Concepts](#)” on page 71.

You can also generate the equivalent of a SEM image for the full chip by using the Calibre PRINTimage batch tool to generate a dense print image. Dense print images are essentially simulated SEM images that can simulate the entire chip, sampling at grid points rather than just those areas of the chip that contain fragmented polygons. Dense print images provide image information for portions of the layout that do not contain structures on the opc layer.

For dense print images using VT5 resist models, which allow you to separate out the optical, resist, and etch components of a model, the simulation generates three outputs.

- **Dense PI layer** — The PRINTimage generated using the entire model: the threshold polynomial and bias polynomial, which simulate resist plus etch effects.
- **Threshold layer** — The PRINTimage generated using the threshold polynomial alone. This simulates the resist effects.
- **Reference layer** — A contour of the aerial image at the (constant) reference threshold. This represents the optical model results.

These show you the intermediate results as well as the final PRINTimage results.

For dense aerial images, the Calibre PRINTimage batch tool samples data at grid points. Grid points are evenly spaced throughout the entire design. The application calculates simulation

data for dense aerial images at every grid point on the design. This also results in a full-chip simulation, not limited to the locations of fragmented data.

PRINTImage Simulation Models

The image types (SEM, dense print, and dense aerial) differ in terms of the models used to simulate the image:

- The Calibre PRINTImage batch tool uses both an optical model and a resist model to simulate a *SEM image*. Graphically, the flow looks like this:
`mask -> optical model -> VT5 (resist) model -> wafer image`
- *Dense print images* use the same flow as SEM images, using both an optical model and resist model:
`mask -> optical model -> VT5 (resist) model -> dense printimage`
- The Calibre PRINTImage batch tool uses the optical model alone to simulate a *dense aerial image*. Graphically, the flow looks like this:
`mask -> optical model -> dense aerial image`

For a complete description of the optical and resist models, refer to the *Calibre WORKbench User's and Reference Manual*.

Table 9-1 summarizes the differences in how and where the Calibre PRINTImage batch tool calculates the three image types.

Table 9-1. Comparison of Image Types

| | Models Used | Sampling Points | Area |
|----------------------------|------------------|-----------------|--|
| SEM | Optical & Resist | Control Sites | Areas containing fragmented polygons |
| Dense Aerial | Optical | Grid Points | Full Chip or areas contained in the filter layer |
| Dense (SEM) Print Image | Optical & Resist | Grid Points | Full Chip |

Using Simulated Images

SEM images and dense print images predict image distortions introduced by the exposure, resist, and etch processes. Use SEM and dense print images when you want to:

- Discover which areas or features require correction.
- Check how well OPC actually corrected the mask.
- Compare different RET approaches.

- Use dense print images if you want to analyze features for the full-chip rather than a specific portion of the layout.

Dense aerial images predict the light intensity on the entire resist surface. Because the light intensity is separate from the resist behavior, you can use this information to compare the effectiveness of different exposure settings. You can also use dense aerial images to check for:

- Side lobes resulting from attenuated phase shifting masks.
- Residual features produced by Sub-Resolution Assist Features (SRAFs) such as scattering bars.

SEM Image Controls

You control the process the Calibre PRINTimage batch tool uses to generate and draw printed (SEM) images by using variables and keywords in the setup file. The following sections describe the control you have over PRINTimage behavior.

PRINTimage Fragmentation

Because the Calibre PRINTimage batch tool calculates SEM images by sampling data at control points, the degree of fragmentation has a major impact on how coarse or fine the image is. By default, the application ignores any fragmentation parameters in the setup file, and fragments edges based on built-in heuristics designed with the intention of producing a fine, visually attractive image.

The [SEM_USE_SUF_FRAG](#) setup file variable lets you override this behavior and force the batch tool to use the fragmentation parameters in the setup file. To use this option, set `SEM_USE_SUF_FRAG` to true. You can also use the [SEM_USE_SUF_SITES](#) setup file variable to force use of user-defined sites.

SEM Drawing Mode

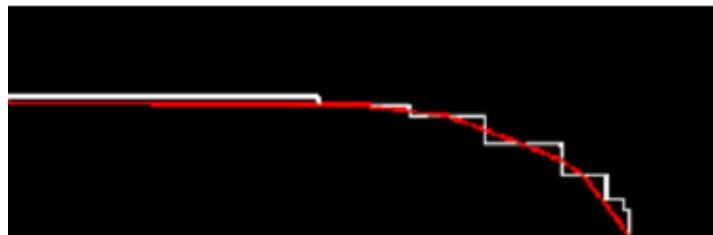
The Calibre PRINTimage batch tool supports two modes for drawing contours for SEM images. These are rectilinear and piecewise linear. The application uses the same method for calculating the image data for both modes. That is, it simulates the print location for the center of the fragment by calculating simulation data at each control point along a control site. The difference between the two drawing modes lies in how the tool connects up the many print locations:

- In rectilinear mode, the Calibre PRINTimage batch tool draws contours by moving each fragment to line up with the print location, then connecting the vertices of adjacent fragments. The resulting image appears somewhat jagged.
- In piecewise linear mode, the Calibre PRINTimage batch tool draws contours by drawing straight lines connecting the print locations on adjacent control sites.

Figure 9-2 shows the rectilinear image in white superimposed on the piecewise linear image in magenta. By default the application uses rectilinear mode.

Set the [SEM_USE_PWL](#) setup file variable to true to change the mode to piecewise linear.

Figure 9-2. Piecewise Linear Versus Rectilinear SEM



Jagged Edges

The Calibre PRINTimage batch tool allows you to clean up jagged edges near concave and convex corners on simulated SEM images by performing morphological over- or under-sizing.

Set the [SEM_DO_MORPH](#) setup file variable to off if you want to turn off over- or undersizing. Use the [SEM_MORPH_SIZE](#) setup file variable to control how much over or undersizing occurs. This variable specifies the maximum jog or jag to be corrected, and is expressed as a floating point number.

Use the [SEM_PWL_JOT_SIZE](#) setup file variable to filter edges excluded from piecewise linear conversion. Any edge in the stairstepped PRINTimage shorter than the *floating_point_number* length is excluded from the piecewise linear “connect the dots” operation. SEM_PWL_JOT_SIZE allows you to define a different value for jogs to be cleaned up by PRINTimage than for edge fragments to be skipped when generating a PWL image.

Control Site Placement

The results of an OPC run depend largely on how the control sites are placed on a fragment’s edge. You can make adjustments to the sites using the [SEM_USE_SUF_SITES](#) environment variable, which allows you to control the position and orientation of the sites when generating a SEM image.

Use the [SEM_CONSIDER_SITE_OFFSET](#) variable to adjust for an offset based on site direction when calculating EPE for PRINTimage operations. EPE is typically calculated along the site direction, which may or may not be normal to the edge, particularly around corners. This can cause false errors during PRINTimage generation. Set the SEM_CONSIDER_SITE_OFFSET variable to 1, and the EPE is calculated along the direction perpendicular to the edge that gives a correct EPE result.

Caution

 Calibre does not process the tag script in the setup file if PRINTImage is used. This means that fragmentTag and siteControl operations are ignored if they are in the setup file. Instead, use the OPC call to process the tag script and to output a SEM image by setting the iterations keyword to 1 and setting OPC_FEEDBACK to 1. This forces OPC to behave similarly to PRINTImage.

SEM Options Summary

Table 9-2 summarizes the SEM options you control through variables.

Table 9-2. Calibre PRINTImage Environment Variables

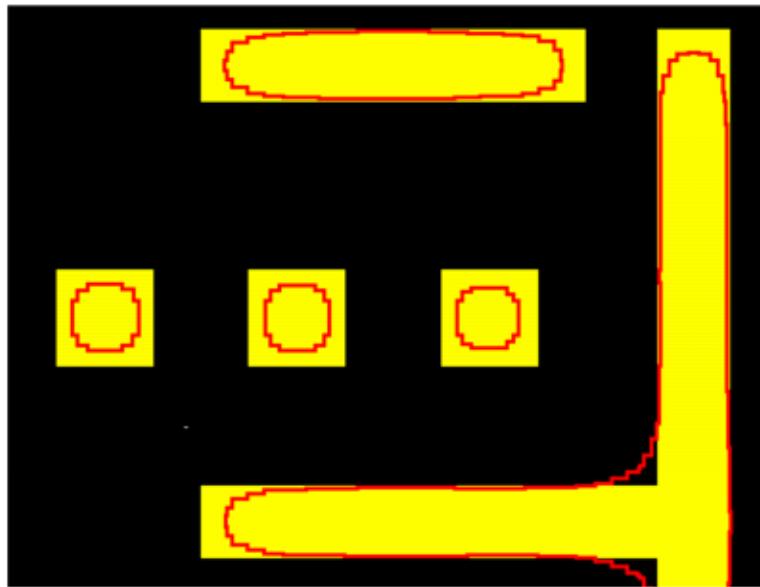
| Variable | Description |
|--------------------------|--|
| SEM_CONSIDER_SITE_OFFSET | Calculates a site offset along the direction perpendicular to the edge that gives a correct EPE result. Automatically set to 1 if SEM_USE_SUF_SITES is set to 0. |
| SEM_DO_MORPH | Turns morphological over- or under-sizing on (true) or off (false). The default is true. |
| SEM_MORPH_SIZE | Defines the amount of over- or under-sizing operation. Used when SEM_DO_MORPH = true. The default is 0.01. |
| SEM_PWL_JOT_SIZE | Filters edges excluded from piecewise linear conversion. |
| SEM_USE_PWL | Turns piecewise linear mode on or off. Default is rectilinear mode. |
| SEM_USE_SUF_FRAG | Instructs the Calibre PRINTImage batch tool to use the fragmentation parameters in the setup file. |
| SEM_USE_SUF_SITES | Control the position and orientation of sites when generating a SEM image with Calibre PRINTImage. |

Calibre PRINTImage Results

The Calibre nmDRC hierarchical engine saves PRINTImage results as a derived layer. You can view this layer using any GDS viewer or layout editor, such as the Calibre WORKbench or Calibre LITHOview application, or the Pyxis Layout Editor application.

In most situations, you view the PRINTImage derived layer along with some other layer, to visually inspect and compare images. Figure 9-3 shows a typical SEM image displayed in red with the original drawn geometries shown in yellow.

Figure 9-3. Calibre PRINTImage Results



In other cases, such as using a dense aerial image for side lobe detection, you may want to use the Calibre RVE application to help locate critical output geometries.

To Run the Calibre PRINTImage Batch Tool

You run the Calibre PRINTImage batch tool from within the Calibre nmDRC hierarchical engine, which you invoke using the `calibre` command from a Linux command line.

Files Required to Run the PRINTImage Batch Tool

The following files *are required* to run the Calibre PRINTImage batch tool:

- A layout database: the input OASIS or GDSII file.
- An SVRF file containing the **LITHO PRINTIMAGE** operation.
- A setup file containing the following:
 - Layer definitions
 - In-line models or references to optical and resist model(s)
 - Variables that control PRINTImage simulation and (optional) fragmentation parameters

Related Topics

[calibre](#)

Calibre PRINTImage Example

This simple example uses the Calibre PRINTImage batch tool to generate an SEM image for a simple binary mask.

Setting up the PRINTImage Batch Tool to Produce an SEM Image

Because generating SEM images is the default behavior, this example requires very little setup. This example contains a single mask layer with layer type OPC. Simulation uses the default fragmentation and generates the image using piecewise linear drawing mode. The following excerpts show the key portions of the SVRF and setup files.

SVRF file

```
LAYER POLY 2

// The file printimage.in is a litho setup file

MY_SEM {
    LITHO PRINTIMAGE FILE printimage.in POLY
}

POLY {COPY POLY}

DRC CHECK MAP POLY 2
DRC CHECK MAP MY_SEM 3
```

Setup file

```
modelpath ../models
# ----- simulation models -----
opticalmodel DUV10tab
resistpolyfile aerialmodel.30.mod

background clear
layer 2 POLY 0 0 opc dark

----- Arbitrary Commands Can Follow This Line -----

sse SEM_USE_PWL true
```

Viewing the SEM Results

The resulting SEM image is on layer 3 of the output GDS file. You can compare with the original as shown in [Figure 9-4](#) and [Figure 9-5](#).

Figure 9-4. PRINTImage Result and Original Layer

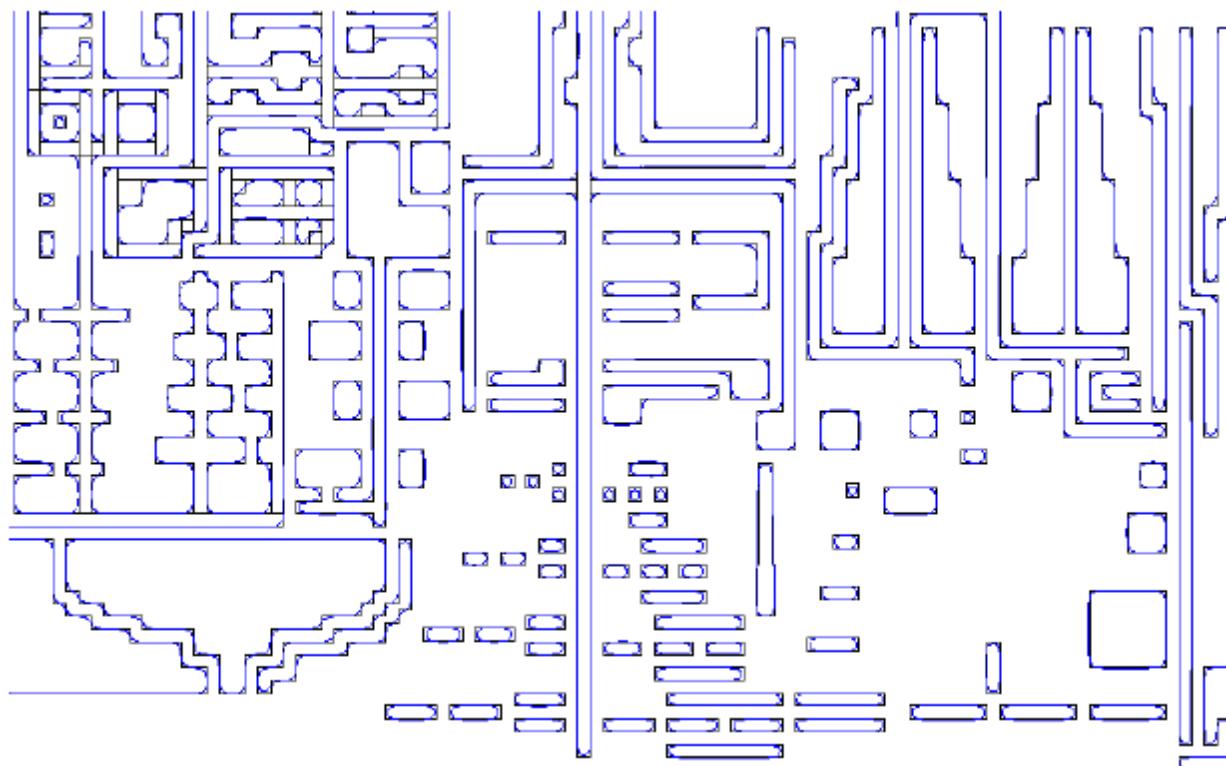
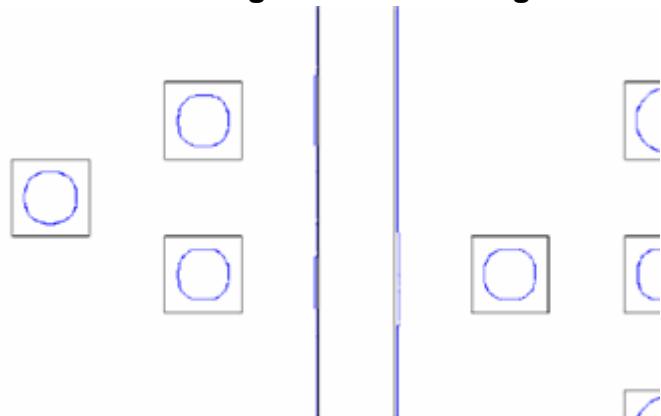


Figure 9-5. PRINTImage Result and Original Enlarged



Chapter 10

Calibre OPCpro Best Practices

This chapter outlines several best practices when using Calibre OPCpro to correct distortions on a mask input layer introduced by optical and resist processes. This chapter contains the following sections:

| | |
|--|------------|
| Recommended Settings for Best Results | 631 |
| Recommended Settings for Best Run Time..... | 635 |
| Fragmentation Best Practices..... | 636 |
| Practices to Improve OPC Output Consistency | 638 |
| Fix Simulation Context..... | 638 |
| Fix Recipe Non-Convergence..... | 639 |
| Improve MRC Rules..... | 640 |
| Practices to Reduce the Impact on Quality of Jogs | 642 |
| Layer-Based Best Practices..... | 643 |

Recommended Settings for Best Results

The following table summarizes recommended settings for OPCpro setup file commands for better performance and results.

Table 10-1. Recommended Settings for Best OPC Results

| Command | Default Setting | Recommended Setting | Notes |
|-------------------|----------------------------|---|------------------------------------|
| iterations | | | |
| | 4 (Calibre WORKbench only) | 130 nm: 6 to 8 90 nm: 6 to 8 65 nm: 8 to 10 45 nm: 8 to 10 | Balance between quality and speed. |
| gridsize | | | |
| | 0.001 microns (um) | 1 dbu | Required input. |

Table 10-1. Recommended Settings for Best OPC Results (cont.)

| Command | | |
|-------------------------------------|--|---|
| Default Setting | Recommended Setting | Notes |
| stepsize | | |
| 0.010 um 10 nanometers (nm) | >= 130 nm: 1 nm 90 nm: 0.5 nm 65 nm: 0.25 nm 45 nm: 0.25 nm | |
| tilemicrons | | |
| 100um | 130 nm: 100 um 90 nm: 100 um 65 nm: 80 to 100 um 45 nm: 50 to 80 um | Required input. |
| maxedgelength | | |
| tilemicrons/2 | Use default. | Balance between quality and speed. |
| minedgelength | | |
| 0.100 um | Use default. | No fragments with length < minedgelength should be created. |
| minjog | | |
| minfeature/4 | ~Nyquist/2 | Required input. Controls the definition of a jog. |
| movelimit | | |
| The larger of 80 nm or minfeature/2 | Use default. | Some special cases may require a lower movelimit to prevent wild edge movement. |
| LITHO_ASYM_SOURCE_CELL_CLONE | | |
| ENABLE | ENABLE | Calibre automatically checks the optical model and only clones when necessary. This gives the best performance. |
| LITHO_AUTO_PUSH | | |
| 1 | 1 | Better performance for hierarchical processing. |

Table 10-1. Recommended Settings for Best OPC Results (cont.)

| Command | | |
|---|---------------------------------------|--|
| Default Setting | Recommended Setting | Notes |
| LITHO_CONTEXT_RELAX_MRC | | |
| $N - 2$, where N is the number of iterations | Use default. | Use with LITHO_SIMULATE_CONTEXT Improves consistency between hierarchical and flat OPC. |
| LITHO_DEANGLE | | |
| 0 (off) | Use the SVRF command DEANGLE instead. | LITHO_DEANGLE uses the same code as the DEANGLE SVRF command. For consistency across OPC tools, the SVRF command is recommended. Avoid values < 0.0005; a value between 0.001 and 0.01 (1 to 10 nm) depending on the technology works well in most cases. |
| LITHO_PRIORITY_BASED_MOVEMENT | | |
| 1 | 1 | Improves OPC consistency during OPC edge movement when there is competing movement priority between line_end and line_end_adjacent fragments because of MRC restrictions. |
| LITHO_PROMOTION_TILING | | |
| 0 | 1 | Required to be set to 1 for MRC_RULE . Recommended to set it to 1 for new recipe development. Provides better scalability compared to the old tiling method, but might cause changes when adding to an existing recipe. |

Table 10-1. Recommended Settings for Best OPC Results (cont.)

| Command | Default Setting | Recommended Setting | Notes |
|---|---|---------------------|---|
| LITHO_SIMULATE_CONTEXT | | | |
| 1 (on) | 1 | | Better consistency between hierarchical and flat OPC. Does cause longer run time. |
|  Note: If you explicitly set LITHO_SIMULATE_CONTEXT, even to its default value, it changes the value of LITHO_SIMULATE_CONTEXT_DISTANCE to unlimited. You must manually set it to use a constrained distance. | | | |
| LITHO_SIMULATE_CONTEXT_DISTANCE | | | |
| 0.25 microns | Start at 0.25 um and adjust based on simulations. | | Decreases the longer run time caused by LITHO_SIMULATE_CONTEXT Smaller distances provide faster run time; larger distances provide better consistency. |
| OPC_AUTO_STEPBY_THRESHOLD | | | |
| 0.4 nm | For contact layers, 0 (off). For other layers, only use if stepsize < 0.25 nm. | | Improves OPC performance when using very small values. May affect quality when stepsize is above 0.25 nm. |
| OPC_EPE_TOLERANCE_FRAC | | | |
| 0.6 | 1 | | Controls the tolerance for OPC correction. |
| OPC_FEEDBACK | | | |
| -0.4 | See notes. | | Required input. Usually lower for initial and final iterations. |
| OPC_MAX_ITER_MOVEMENT | | | |
| (Lambda/NA) / 8 | See notes. | | Use lower values for contact layers or where large edge movement is expected. |

Table 10-1. Recommended Settings for Best OPC Results (cont.)

| Command | Default Setting | Recommended Setting | Notes |
|------------------|-----------------|-------------------------|---|
| newTag | | | |
| No length limits | | <= interaction distance | During length-based tagging, set an upper bound on length checks to keep the tags out of invalid regions. Allowing tags to apply to fragments longer than the interaction distance can cause false tags, which gives bad OPC results. |

Recommended Settings for Best Run Time

The recommendations described in the following table are important for creating the fastest possible OPCpro runs.

Table 10-2. Recommended Settings for Best OPCpro Run Time

| Setting | Recommendation and Explanation |
|---|---|
| Interaction distance | |
| | Interaction distance is determined by the model's hoodpix and site_length. |
| INTERACTION_DISTANCE | Recommend to <i>not</i> use INTERACTION_DISTANCE to avoid unnecessarily large interaction distance, which could cause layout promotion and hierarchy degradation. |
| hoodpix (optical model parameter described in <i>Calibre WORKbench User's and Reference Manual</i>) | Hoodpix defines the kernel diameter of the optical model. Minimizing it decreases OPC run time. |
| approxorder (optical model parameter) | By decreasing the number of optical kernels you decrease OPC run time. |
| kerngrid (optical model parameter) | Increasing the size of the optical kernel grid decreases OPC run time. |

Table 10-2. Recommended Settings for Best OPCpro Run Time (cont.)

| Setting | Recommendation and Explanation | | | |
|--|--|--|--|--|
| <code>hoodpix</code> (VT5 resist model parameter described in <i>Calibre WORKbench User's and Reference Manual</i>) | Set the VT5 hoodpix to less than or equal to the optical model hoodpix. | | | |
| density kernels | Using density kernels in VT5 models increases the OPC run time. | | | |
| General modeling recommendation | Follow the best practices for accuracy, but take run time also into consideration. See “ Global Recommendations for Optical Models ” and “ Best Practices for VT5 Modeling ” in the <i>Calibre WORKbench User's and Reference Manual</i> . | | | |
| Fragmentation optimization | | | | |
| Try to avoid excessive fragmentation for straight edges in the same context. Exact recommendations depend on the technology node and CD control requirements. | | | | |
| <code>maxedgelength</code> | 0.5 * <code>tilemicrons</code> is a good starting point for most layers. If <code>maxedgelength</code> is larger than that, it is reset to equal it. Critical layers on 45nm or smaller nodes require more fragmentation to ensure quality. | | | |
| OPC iterations | | | | |
| The number of OPC iterations has a significant impact on the OPC run time. Improving the recipe convergence could potentially reduce the number of iterations without compromising accuracy. See “ Fix Recipe Non-Convergence ” on page 639 for suggestions. | | | | |
| Site placement | | | | |
| <code>siteinfo</code> | Unnecessarily long site placement can cause longer run time without improving OPC accuracy. Define a reasonable site length with the RESIST option of <code>siteinfo</code> . | | | |
| <code>cornerSiteStyle</code> | Avoid using the orthog option. Site orthogonalization increases run time by up to 30% in each iteration. | | | |
| Cell cloning | | | | |
| If you are using a symmetrical optical source, set the <code>LITHO_ASYM_SOURCE_CELL_CLONE</code> to DISABLE. | | | | |

Fragmentation Best Practices

Most best practices for fragmentation depend on the layer type and the interaction of settings. Fragmentation is controlled by the “frag bit” in the `layer` keyword. For example:

```
layer 100 M1_in 1 0 opc clear
```

The line identifies layer 100 as M1_in, and sets the frag bit to 1. (This turns off intrafeature fragmentation caused by jogs for the layer.) The following frag bit settings are recommended for the various layer types:

Table 10-3. Layer Fragmentation Recommendations

| Layer Type | Frag Bit | Comments |
|-------------------|--|---|
| opc or correction | <ul style="list-style-type: none"> • 0 (all fragmentation on) • 1 (turn off intrafeature fragmentation from jogs) • 17 (turn off intrafeature and interfeature fragmentation from jogs) | 0 is better for general layouts. 1 or 17 is better for layouts with many jogs. |
| island | <ul style="list-style-type: none"> • 0 (keep island fragmentation on) • 32 (turn off island fragmentation) | 0 is used for real island layers. 32 is used for layers that were imported for tagging only. |
| asraf | <ul style="list-style-type: none"> • 0 (no interfeature fragmentation caused by anti-SRAF) • 128 (turn on interfeature fragmentation caused by anti-SRAF) | Use 128. |
| visible | <ul style="list-style-type: none"> • 0 (no interfeature fragmentation caused by visible layer) • 128 (turn on interfeature fragmentation caused by visible layer) | Use 128. |

Note

 The frag bit can be overridden by the [fragmentLayer](#) or [flaglayer](#) keywords.

Another best practice is that all fragment settings, in particular [minedgelen](#), should be greater than the MRC constraints. Failure to do so overconstrains the correction.

Practices to Improve OPC Output Consistency

There are three types of inconsistency generally found in OPCpro runs:

- Differences in hierarchical and flat runs.
- Identical polygon input generates different OPC results in different locations. (For example, in an SRAM array a bit cell may have different OPC output between neighbors.)
- Differences in output of 1 DBU.

Some of these differences are caused by theoretical limits. For example, output differences of a DBU are most likely due to floating-point calculations. These cannot be “fixed” because of limits in hardware.

Other sources of inconsistency are better able to be reduced. These sources are

- Different simulation context near hierarchical and tile boundaries.
- Recipe non-convergence.
- The particular sequence of hierarchical processing and MRC rule restrictions.

In general, be cautious when defining MRC rules. Using overly constrained MRC rules could cause fragments not to move enough (under correction), and using non-Euclidean metrics could cause inconsistent OPC outputs.

| | |
|---|------------|
| Fix Simulation Context | 638 |
| Fix Recipe Non-Convergence | 639 |
| Improve MRC Rules | 640 |

Fix Simulation Context

The variable LITHO_SIMULATE_CONTEXT resolves the majority of inconsistencies found near hierarchical and tile boundaries. It does this by resimulating the context region at the current hierarchy level (promoting) for valid fragments.

Add the following lines to your setup file to use LITHO_SIMULATE_CONTEXT:

```
sse LITHO_SIMULATE_CONTEXT 1
sse LITHO_SIMULATE_CONTEXT_DISTANCE 0.1
sse LITHO_CONTEXT_RELAX_MRC val
```

where *val* should be two less than the number of iterations.

Here is what the lines do:

- sse [LITHO_SIMULATE_CONTEXT](#) 1
This instructs the OPC engine to re-simulate the context data when performing hierarchical data processing.
- sse [LITHO_SIMULATE_CONTEXT_DISTANCE](#) 0.1
This controls how much context is simulated. Context fragments that are within this distance of a “valid” fragment are re-simulated. Setting the distance to a smaller value decreases the effect on run time.
- sse [LITHO_CONTEXT_RELAX_MRC](#) *val*
This causes MRC checks to only occur against current locations in the first *val* iterations. For iterations after *val*, MRC is enforced against final positions as is typical.

Fix Recipe Non-Convergence

Recipe convergence is also referred to as stability. If a recipe is unstable, improve it by optimizing feedback and making smaller adjustments.

Control Feedback

For high MEEF processes and dense layers, you want to use very small adjustments. (Contact and via layers are usually very high MEEF.) Use lower feedback for initial iterations. The tag-based controls [OPC_FEEDBACK](#) or [opcTag](#)-feedback let you set different values based on fragment types, or for initial and final iterations.

For low MEEF processes, higher feedback values cause recipes to converge more quickly. To reduce oscillations, use [opcTag](#)-hintOffset on fragments with large desired displacement.

Another technique for avoiding overly large edge movements is to limit the maximum edge movement per iteration. Set [OPC_MAX_ITER_MOVEMENT](#) to a layer-dependent value. For high MEEF processes, 0.004 microns is reasonable.

Restrict Small Fragments

After the first iteration, edges are broken into fragments which are likely not collinear. The displacement between fragments is considered a small step or s-shape curve if it is larger than minedgelength. Small steps are by default adjusted as though they were intentional fragments.

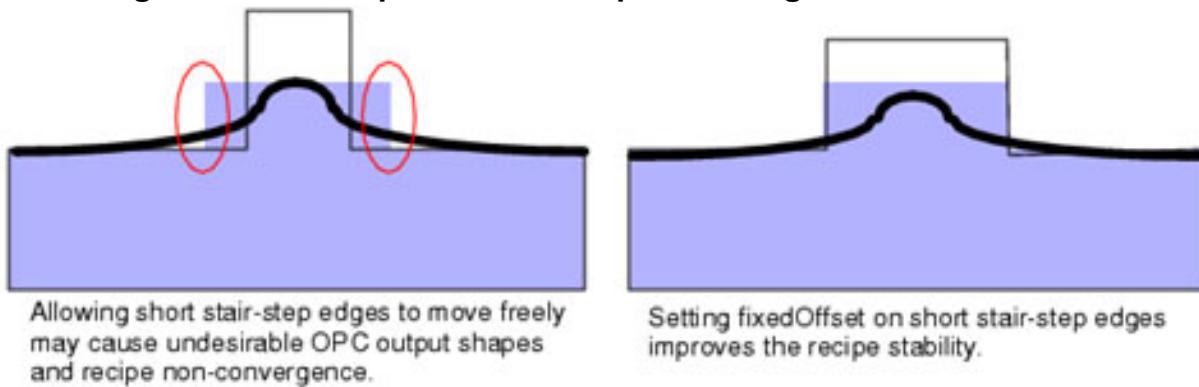
To improve recipe stability and decrease computation time, restrict the edge movement of jogs or small steps between a convex and a concave corner. Some methods of doing that are

- Set [minjog](#) to disable OPC correction for s-shaped fragments less than a certain size. A good initial value is 0.5*Nyquist.

- For steps whose target is not achievable, apply a unique tag to the fragment and use `opcTag tag -fixedOffset` or `-freeze` to selectively limit the movement. (See [Figure 10-1](#).)

You may also want to change the EPE tolerance of non-critical fragments (not just s-shaped curves). This can be done with `epiToleranceTag`. It allows different tolerances for inward and outward measurements.

Figure 10-1. Example of Small Step Interfering in OPC Movements



Improve MRC Rules

The MRC_RULE setup keyword allows you to specify internal or external constraints between layers based on the distance between fragments as well as fragment length. The following practices improve your OPC results when using MRC_RULE:

- MRC constraints should always be set to less than the minimum fragment length, `minedgelength`. Failure to do so overconstraints the correction.
- The input layer should not contain polygons that are already in violation of MRC rules.
- Use tags to define different MRC rules for different situations. For example:

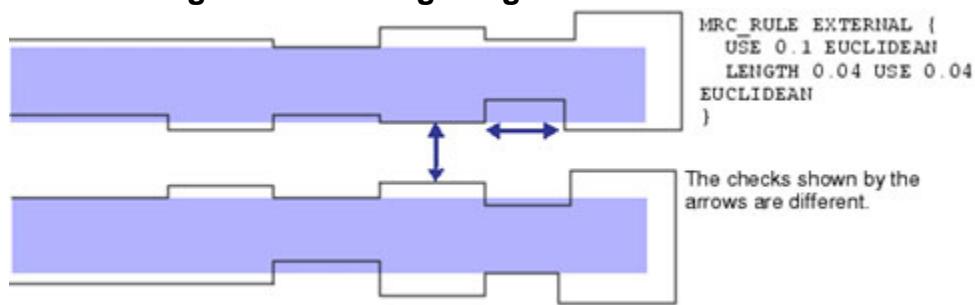
```
MRC_RULE EXTERNAL line_end line_end {USE 0.03}
```

```
MRC_RULE INTERNAL pc pc {USE 0.03}
```

If multiple rules apply to the same fragment, the first applicable rule in the setup file is used.

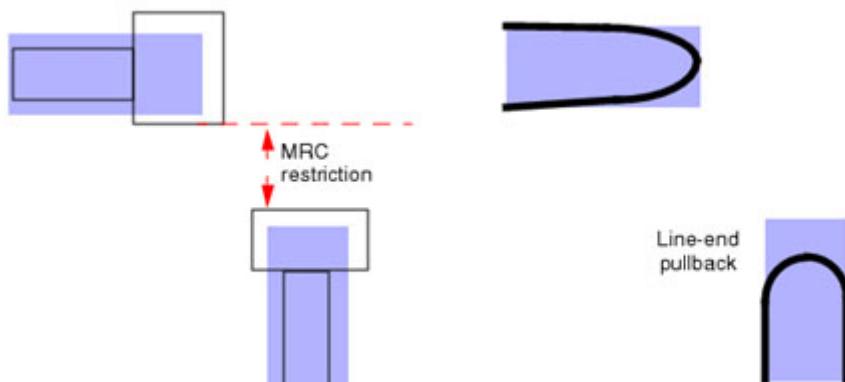
- Use length-based MRC rule specifications to prevent notches and corners from limiting edge movement, as shown in [Figure 10-2](#).

Figure 10-2. Using Length-Based MRC Rules



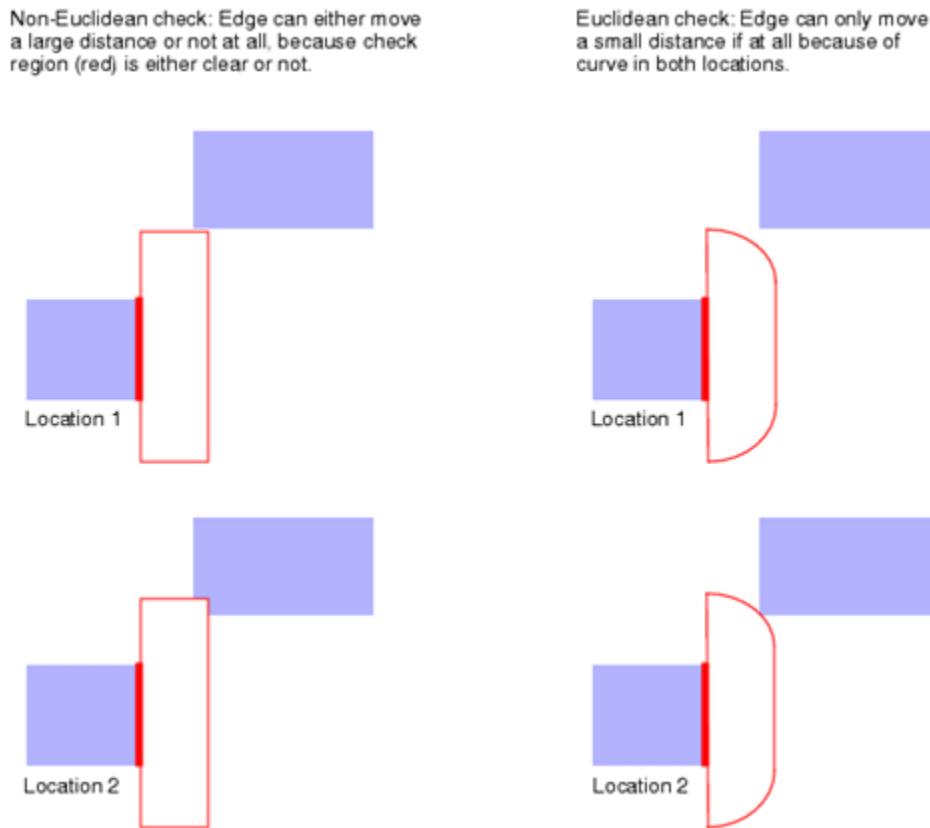
- Minimize MRC restrictions. Unnecessarily rigid MRC restrictions can increase problems such as pullback, as shown in [Figure 10-3](#).

Figure 10-3. Pullback Due to Excess MRC Restriction



- When specifying constraints for nubs or jog edges, be sure the MRC_RULE LENGTH matches the length definition of the nubs or jogs. For example, if minjog is 5 nm, then LENGTH should also be 5 nm so that the subcheck can be applied properly at the nubs or jog edges.
- Avoid non-Euclidean MRC metrics. The non-Euclidean metrics are rectangular (SQUARE or OPPOSITE). The Euclidean metric uses a curve. As shown in [Figure 10-4](#), the use of non-Euclidean metrics can result in inconsistencies.

Figure 10-4. MRC Metrics Can Cause Inconsistency



MRC_RULE is the preferred method for mask rule checking and enforcement, both INTERNAL and EXTERNAL. To enable MRC_RULE checks, you must add the following line to your setup file:

```
sse LITHO_PROMOTION_TILING 1
```

Related Topics

[MRC_RULE](#)

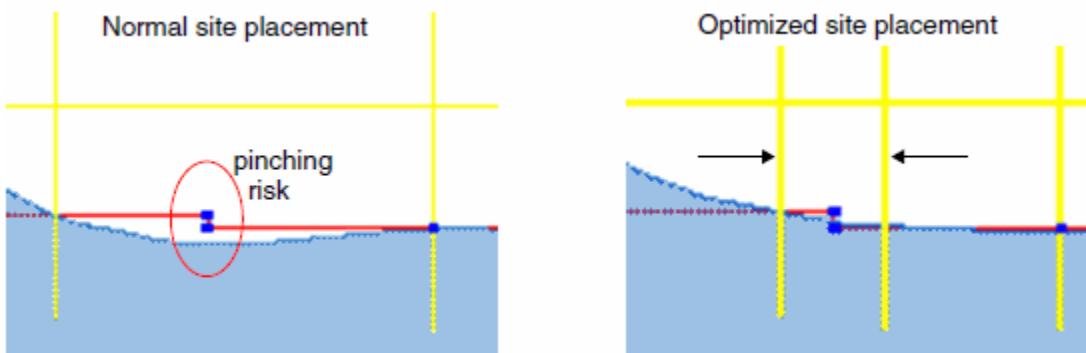
Practices to Reduce the Impact on Quality of Jogs

Jogs can be simulated and moved just like any other fragment. However, their small size typically means that controlling the EPE on the jog edge is unimportant. Furthermore, moving the jog edges can cause more critical fragments to be blocked by MRC constraints while not substantially improving the printed image.

The following are some general techniques to reduce the impact of jogs:

- Define “jog” as the edge size below which you are not concerned. Jogs do not receive a control site and so are not adjusted directly. Jogs are defined by the [minjog](#) keyword.
- Allow adjustment but constrain it. As shown in [Figure 10-1](#) of “Fix Recipe Non-Convergence” on page 639, without constraints the OPC engine might use large edge displacements to try reaching a target. S-shaped curves are relatively insensitive. Use [opcTag -fixedOffset](#) to achieve reasonable output without penalizing neighbors.
- For especially tight situations, or after initial adjustments, freeze the fragment to prevent it being moved at all. Use [opcTag -freeze](#).
- Optimize the placement of control sites on the fragments next to the jog. The existence of the jog can interfere with the important fragment’s EPE calculation, as in the example in [Figure 10-5](#) where placing control sites near the jog keep the contour from pulling in so much.

Figure 10-5. Optimized Site Placement Around a Jog



To control site placement, use the `siteControl` keywords. For example:

```
siteControl tag1 OFFSET_FROM CURRENT 0.005
siteControl tag2 SHIFT_TOWARD CONVEX 0.01
```

Related Topics

[siteControl OFFSET_FROM](#)

[siteControl SHIFT_TOWARD](#)

Layer-Based Best Practices

Different layer types can be optimized to produce better results.

Poly or Active Layers

When building a poly or active layer in an OPC recipe, follow these guidelines:

- Poly and active layers 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).
- Fragmentation and site placement need to be carefully tuned near gate regions. For better control of gate CD, use the gate area as an island layer to inject fragmentation points at the intersection of the poly and active layers. (Set the island layer's frag bit to 0 in the [layer](#) command to turn on island fragmentation.)
- Do not allow very long fragments, especially for the 65nm and smaller nodes. Control fragment length with [maxedgelength](#) (global) or [fragmentTag ... maxedgelength](#) (tag specific).

Metal Layers

When building a metal layer in an OPC recipe, follow these guidelines:

- The primary objective for this layer type is to avoid bridging and pinching. You also want to maximize coverage of contacts. Unlike poly layers, ripples and CD uniformity are not as critical.
- A layout could have many jogs due to complicated pre-OPC bias rules and complex 2-dimensional shapes. Be prepared to use complicated fragmentation rules (the [fragmentTag](#) family) to reduce the impact of jogs. Also adjust site placements on jog-adjacent fragments. See “[Practices to Reduce the Impact on Quality of Jogs](#)”.
- For older technology nodes, a longer [maxedgelength](#) value is useful. However, when you use it at the default value of $0.5 * \text{tilemicrons}$, you should also keep [Implicit Fragmentation](#) on to avoid crossing the tile boundary with long fragments.

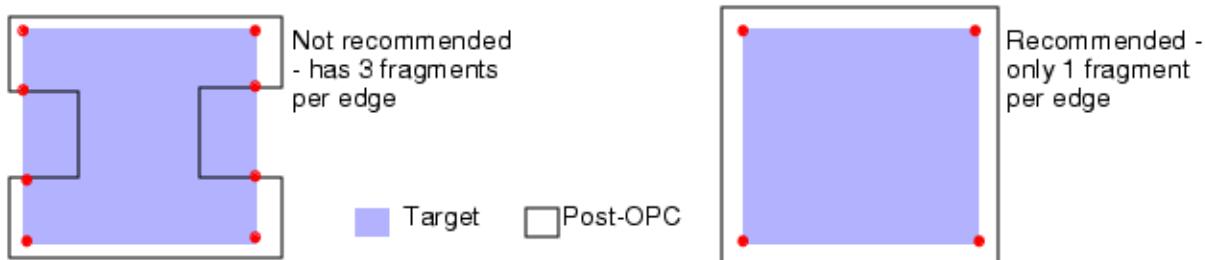
Contact Layers

When building a contact layer in an OPC recipe, follow these guidelines:

- The primary objective is to maintain a minimum coverage area between contact and metal or device layers (poly and active).
- Lower feedback is often required than for other layers.
- For 45nm and smaller nodes, also follow these additional guidelines.
 - Keep [stepsize](#) less than or equal to 0.25nm.

- Use a smaller value than the default $((\text{Lambda}/\text{NA})/8)$ for **OPC_MAX_ITER_MOVEMENT** to avoid recipe oscillation.
- Turn off the stepby heuristic by setting **OPC_AUTO_STEPBY_THRESHOLD** to 0.
- Have only one fragment per contact edge to avoid I-shaped OPC output (see Figure 10-6).

Figure 10-6. I-Shaped OPC Output



Calibre Litho Glossary

CD

critical dimension

concave corner

A corner whose interior angle is greater than 180 degrees.

concurrent operations

A set of two or more Calibre operations that are processed at the same time.

context

The geometry within a set distance of a fragment being simulated, measured from the center point of the fragment's control site without regard to hierarchy. The set distance is referred to as the promotion radius, and by default is based on the optical model and control site spacing.

context fragment

A fragment that was optimized in a child cell that gets promoted to the parent cell in order to participate in simulating a fragment in the parent. Context fragments may be shifted during the parent simulation depending on the litho setup file variables, but their own final positions are determined in the child cell.

control point

The points on a control site at which image intensity is calculated during simulation. Control points are numbered from lightest area to darkest, beginning with point 0.

control site

The locations at which Calibre applications sample simulation data to evaluate relative to the desired results. Control sites occur on layers of type opc. All EPEs are generated with respect to control sites.

convex corner

A corner whose interior angle is less than 180 degrees.

correction layer

A layer whose edges are moved during OPC to compensate for optical and processing errors.

deck

A set of SVRF commands used for a single Calibre run and saved in an SVRF rules file. Decks are also referred to as "job decks" or "rule decks."

derived layer

A new layer that is created in a Calibre application by modifying another layer or layers in some way.

DRC

design rule checking

DUV

deep ultraviolet

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”, which is more common in DRC, is synonymous with “fragment”, used in RET.

edge placement error

The difference between the drawn edge of a polygon and either the simulated silicon edge or actual silicon edge when printed.

EPE

see edge placement error

external layer

A layer that contains a corrected mask design. Its presence signals LITHO that the opc layer's geometries represent the target design rather than the mask design itself.

fragment

Either a layout polygon edge or a part of an edge.

fragmentation

The process of breaking up a layout polygon's edges into smaller segments called fragments.

fragmentation parameters

Parameters in a litho setup file that affect the way fragmentation vertices are created on a layer.

fragmentation vertex

A vertex added to an edge or fragment to divide that edge or fragment into smaller fragments. Fragmentation vertices are generated by the system during the run.

gauge

A pair of control sites for simulating and measuring critical distance. The sites are placed across a feature or space.

hidden layer

A layer whose data do not affect the outcome of a litho tool run. However, ORC 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.

implicit fragmentation

A type of fragmentation that inserts vertices where tile boundaries and hierarchy boundaries intersect polygon edges.

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 independent of other polygons and adds vertices at set distances from the figure's corners.

island layer

A layer that defines areas on the mask 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 fragment

A fragment that has a concave corner attached to one endpoint and a convex corner attached to the other endpoint and has a length less than the minjog.

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

A specific type of fragment that has a length less than or equal to lineEndLength, has convex corners at both end points, and has adjacent edges with lengths greater than or equal to the lineEndLength.

maxedgelength fragmentation

A type of fragmentation triggered by fragments whose length exceeds $2 * \text{maxedgelength} + \text{minedgelength}$.

MRC

mask rule constraints

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.

NA

numerical aperture

OPC

optical proximity correction

opc layer

The target layer for most LITHO operations. This layer contains the actual geometry that you intend to transfer onto silicon.

ORC

optical rule checking

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.

PSM

phase shifting mask

PWL

piecewise linear

resist model

A set of parameters that predicts the effects of a specific resist or etch process based on experimental or simulated measurements.

ripple fragments

Symmetrical fragments created by interfeature fragmentation on either side of an initial vertex.

RVE

results viewing environment

SEM

scanning electron microscope

site

see control site

space-end fragment

A specific type of fragment that has a length less than or equal to minedgelength, has concave corners at both end points, and has adjacent edges with lengths greater than or equal to the minedgelength.

SRAF

sub-resolution assist feature

SVRF

standard verification rule format

tagging

The process of defining named subsets of edge fragments so you can control how the different types of edge fragments are treated.

tagging commands

Keyword-based commands in the setup file, used for creating named subsets of fragments.

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.

visible layer

A layer that defines shapes that are visible on the manufactured mask but are not to receive OPC correction. Typically these shapes represent SRAFs and anti-SRAFs. In some processes, anti-SRAFs are placed on their own layer type, asraf.

visible-layer fragmentation

A type of fragmentation that inserts vertices where edges of opc or correction layers intersect edges on visible or asraf layers.

VT model

A family of resist models. The types are VTR (variable threshold resist), VTR-E (variable threshold resist extended), and VT5. The only currently supported version is VT5, version 3.

Index

— Symbols —

[]²¹
{}²¹
//¹³⁸
\¹³⁷
#¹³⁸
|²²

— A —

Aerial images
interaction distance, [282](#)
model required, [241](#), [243](#)
tagging, [469](#)
Aerial sites, [114](#), [257](#)
After
with respect to edges, [455](#)
And operations (SVRF), [53](#)
Angled edges
controlling OPC, [276](#), [322](#)
snapping to grid, [280](#)
Appropriate edges
definition, [41](#)
illustration, [44](#)
measuring angles for, [44](#)
orientation of, [41](#)
Asraf layers, [124](#)
Attenuated masks transmission values, [211](#)

— B —

background command, [155](#)
Bins for histograms, [407](#)
Block comments, [48](#)
Bold words, [21](#)
Built-in tags, [104](#)

— C —

Calibre data, [33](#)
Calibre Litho
calling, [65](#)
controlling, [67](#)

layer usage, [120](#)
Calibre nmDRC hierarchical engine
input, [29](#)
invoking, [30](#)
results, [31](#)
rule files, [29](#)
Calibre OPCpro, *see* OPCpro
Calibre ORC, *see* ORC
Calibre PRINTimage, *see* PRINTimage
Calibre transcript, [31](#)
Calibre WORKbench
viewing fragmentation, [103](#)
CALIBRE_HOME environment variable, [23](#)
Case sensitivity
in rule files, [48](#)
setup keywords, [139](#)
Change parameters for a specific layer, [98](#)
Clean-up of PRINTimage results, [360](#), [362](#)
Clear
background, [155](#)
features, [210](#)
CLOSEPOINT metric, [414](#)
CLOSEPOINT weighting, [420](#)
Coincident
relative to edges, [34](#)
relative to polygons, [34](#)
Coincident Edge operation (SVRF), [53](#)
COINCIDENTEDGE group commands
COINCIDENTEDGE, [430](#)
COINCIDENTINSIDEEDGE, [432](#)
COINCIDENTOUTSIDEEDGE, [434](#)
NOTCOINCIDENTEDGE, [484](#)
NOTCOINCIDENTINSIDEEDGE, [486](#)
NOTCOINCIDENTOUTSIDEEDGE, [488](#)
COINCIDENTEDGE tag command, [430](#)
COINCIDENTINSIDEEDGE tag command, [432](#)
coincidentLayer command, [158](#)
Commands
Litho setup file keywords, [152](#)

- set variables, 259
- sse, 259
- SVRF, 136
- tagging, 64, 368
- Comments
 - in rule files, 48
 - Litho setup file, 138
 - logging, 138
- Compare equally close fragments, 421
- Concave corners, 82, 161, 647
- concaveAdjDist setup file keyword, 160
- concavecorn setup file keyword, 161
- Concurrent operations, 647
- Conservative sites, 169
- Control OPC
 - fixed OPC feedback factor, 349
 - for angled edges, 276, 322
 - for line ends, 87
 - for multiple masks, 331
 - layer-specific fragmentation, 174
 - line ends, 252
 - line-end adjacent fragments, 215
 - maximum fragment length, 217
 - maximum movement, 177, 226
 - minimum feature size, 223
 - minimum fragment length, 222
 - minimum movement, 261
 - tolerance factor, 337
- Control points
 - definition, 647
 - on control sites, 112
 - used with PRINTimage, 622
- Control sites
 - at corners, 173
 - defining, 258
 - definition, 112, 647, 650
 - keywords, 113
 - length, 114
 - location, 115, 315
 - types of, 114
- Convex corners
 - definition, 647
 - fragmentation of, 82
 - priority, 85
- Convex Edge operation, 53
- Copy operation (SVRF), 53
- cornedge setup file keyword, 164
- Corner fragmentation
 - concave, 158, 160, 162
 - convex, 163, 165, 199
 - line-end adjacent, 165
 - non-90, 83, 161
 - non90 corners, 165
 - prioritizing, 164
 - sites at concave corners, 118, 172
 - space-end adjacent, 84, 162
- Corners
 - concave, 647
 - convex, 647
- cornerSiteStyle
 - radius used with, 167
- Correction layers, 209
- Courier font, 21

— D —

- Dark
 - background, 155
 - features, 210
- Data
 - edge clusters, 33
 - edges, 33
 - intermediate, 37
 - layer of origin, 35
 - merging, 34
 - passed to RET batch tools, 38
 - polygon, 33
 - types in GDS files, 40
 - used with Calibre, 33
 - writing to results database, 37
- Database constructor,*see*Calibre nmDRC
 - hierarchical engine
- Databases
 - design, 65
 - results, 29, 31, 69
 - used by Calibre nmDRC hierarchical engine, 29
- DDL controls, 177
- Deangle, 292
- Decks, 647
- Define MEEF matrix, 412
- Delete tags, 375

DELTA Matrix OPC keyword, 417
Dense aerial images
 grid points, 622
 models used to create, 623
 uses, 624
Dense print images, 623
Derived layers
 converting to rule check statements, 40
 creating, 38
 definition, 37
 global, 38
 glossary, 647
 local, 38
 names, 38
 saving with rule check statements, 40
 types, 33
Design layers, 33
Diagonal character, 138
Dimension checks
 and layer of origin, 36
 data returned from, 46
 definition, 41
 Enclosure, 41
 example, 46
 External, 41
 identifying edge pairings, 41
 illustration, 41
 Internal, 41
Distance mapping, 579
Dose, 212
Double pipes, 22
Drawn layers, 33
DRC Check Map statement, 40, 55, 66
DRC Maximum Results (SVRF), 51
DRC Results Database statements, 51
DRC Results Summary, 31
DRC Summary Report statement, 51

— E —

Edge pairings
 measured by Enclosure operation, 43
 measured by External operation, 42
 measured by Internal operation, 42
Edge Placement Errors (EPEs)
 and fragmentation, 73
 and process variation, 602

and simulation images, 73
calculating, 113, 122
definition of, 648
EPE sensitivity, 602
external layers, 123
newTag command, 446, 450
tagging, 601
tagging example, 108
tolerance region, 379

Edges
 appropriate, 41, 44
 as data, 33
 clusters, 33
 definition, 72, 648
 dividing space, 34
 false edges, 125, 207
 fragments, 648
 illustration of dividing space, 34, 35
 moving with Matrix OPC, 421

EMPTY, 126, 612
ENCLOSURE, 43
Enclosure, 41
Enclosure checks, 585
Enclosure operation (SVRF), 53
Environment variables
 CALIBRE_HOME, 23
 LITHO_WRITE_DEBUG_SVRF, 323

EPE, *see* Edge Placement Errors (EPEs)

Errors
 logging, 367
 minedgelength violation, 198
 projecting, 452, 475
 promotion tiling, 306

Euclidean keyword
 Matrix OPC, 414, 420

Euclidean metric
 used with dimension checks, 45

Event logs, 31

Examples
 OPCpro, 572
 ORC, 618
 PRINTImage, 628

Expand Edge operation, 54

External, 41

External layers

and ORC, 123
definition, 122, 648
illustration, 123
layer parameter, 209
External operation (SVRF), 53

— F —

False edges, 125, 207

Files

histogram, 294
Litho setup, 138
model, 224

Flag Acute statement, 51

Flag Offgrid statement, 51

Flag Skew statement, 51

fragAfterEdge tagging keyword, 598

fragAfterFrag tagging keyword, 598

fragBeforeEdge tagging keyword, 598

fragBeforeFrag tagging keyword, 598

Fragment mapping

many-to-one, 577
one-to-many, 578
one-to-one, 577

Fragmentation

and OPC, 58

blocks, 98

concave corner, 162, 207

concave corner adjacent, 158, 160

controlling by layer, 207

convex corner adjacent, 163, 207

definition, 80, 648

implicit, 649

interfeature, 86, 193, 649

intrafeature, 81, 199, 649

island-layer, 90, 92, 207, 649

jogs, 207, 223

line-ends, 207

maxedgeLength setup file keyword, 89

maximum fragment length, 217

minimum fragment length, 222

parameters, 648

vertices, 648

viewing, 103

fragmentLayer command blocks, 98

Fragments

adjacent space ends, 162, 197

adjacent to line-ends, 165
adjacent to non-90 concave corners, 161
adjacent to non-90 convex corners, 165
concave adjacent, 161
convex corner adjacent, 163
definition, 72, 74, 648
edge, 72, 648
fragmentation process, 80
line-end, 649
ripple, 88, 193
fragmentTag command, 406
fragNextToEdge tagging keyword, 598
fragNextToFrag tagging keyword, 598
Full-Serifs, 87, 252

— G —

GDS files

as input to Calibre, 29
as input to Calibre RET batch tools, 65
used as Results Database, 31

GDS layers

identifying, 126
output from ORC, 606
viewing, 103

Global derived layers, 38

Grid units

Calibre, 50

LITHO, 179

gridsize command, 179

— H —

Hammerheads, 87, 252

Heavy font, 21

Hidden layers, 124, 209

Hierarchical gaps, 59, 289

Hierarchy boundaries, 648

Histogram files, 294

Histograms, 407

-how argument, 107, 426

— I —

ignore_jogs setup option, 170

ignore_radius setup option, 169

Illumination types, 210

Image tag, 603

Implicit fragmentation, 295, 306, 649

In-line comments
 in rule files, 48
inline_optical2 setup keyword, 182
Inside
 relative to edges, 34
 relative to polygons, 34
Inside operation (SVRF), 53
INSIDEEDGE group
 NOTINSIDEEDGE, 490
 NOTOUTSIDEEDGE, 492
 OUTSIDEEDGE, 501
Interaction distance, 282
Interfeature fragmentation
 definition, 86, 649
 illustration, 89
 interfeature setup file keyword, 86
 minedgelength setup file keyword, 88
Intermediate data, 37
 within a rule check statement, 38
INTERNAL, 41, 42
Internal operation (SVRF), 53
Intersection of tags, creating a new tag from, 429, 500
Intrafeature fragmentation
 concavecorn, 161
 cornedge, 164
 definition, 81, 649
 illustration, 83
 layer setup file keyword, 85
 minedgelength setup file keyword, 86
 prioritizing, 85
Intrafeature setup file keyword
 split, 198
Island layers, 209
 and OPC, 125
 definition, 124
Island-Layer fragmentation
 definition, 90, 649
Italic font, 21
Iterations
 movement limits, 349
 OPCpro example, 574

— J —

Jogs
 definition, 223, 649

— K —

Keywords
 options versus arguments, 139

— L —

Layer and tag fragment mapping, 579

Layer constructors, 53

Layer definition statements, 55

Layer keyword

 arguments, 120

 properties, 210

 types, 208

Layer Map statement (SVRF), 52

Layer of origin, 35

Layer operations

 distinct from statements, 47

 in rule check statements, 38

 in SVRF rule files, 52

Layer selectors, 53

Layer statements (SVRF), 51

Layer types

 correction, 647

 external, 648

 hidden, 648

 opc, 650

 used with Calibre nmDRC, 33

Layer usage

 RET batch tools, 120

Layers

 assisting fragmentation, 124

 Calibre, 126

 derived, 33, 37, 126, 647

 edge, 33

 GDS, 126

 names, 207

 numbers, 126, 207

 original, 33

 polygon, 33

 used to tag fragments, 478, 479

Layout Path (SVRF), 50

Layout Primary (SVRF), 50

Layout System (SVRF), 50

lea, option to cornedge keyword, 165

Length operation (SVRF), 53

Limits

-
- on feature size, 223
 - on OPC movement, 177, 226
 - Line continuation (\), 137
 - Line end, 649
 - Line end adjacent fragments
 - controlling fragmentation, 165
 - defining, 215
 - specifying site locations, 172
 - Line-end adjacent (lea), 83
 - lineEndLength
 - definition of line end fragment, 75
 - Litho operations
 - as layer constructor, 65
 - OPC, 146
 - ORC, 148
 - PRINTimage, 150
 - Litho Printimage operation, 150
 - LITHO_MASK_PRIORITY_MOVEMENT, 587
 - LITHO_PROMOTION_TILING, 306
 - LITHO_SUPPRESS_INTERACTION_ERROR, 317
 - LITHO_SUPPRESS_MAPNUMBER_ERROR, 318
 - Local derived layers, 38
 - Loops
 - definition, 480
- M —**
- MAP_DISTANCE_MULTIPLIER Matrix
 - OPC keyword, 416
 - MAP_PRIORITY Matrix OPC keyword, 416
 - MAPNUMBER
 - used with ORC, 608
 - Mapping
 - OPC and multiple masks, 331
 - Masks
 - phase shifting, 331
 - Matrix OPC
 - flow, 578
 - fragment mapping, 577
 - overview, 576
 - syntax, 412
 - visualizing, 580
 - MATRIX_OPCT, 412
 - MaxedgeLength fragmentation, 89
- N —**
- Measurement regions
 - used with dimension checks, 44
 - used with RET batch tools, 61
 - MEEF_MODE Matrix OPC keyword, 416
 - Merged data, 34
 - Method for newTag, 426
 - METHOD Matrix OPC keyword, 412
 - Method, for newTag, 107
 - METRIC Matrix OPC keyword, 413
 - Metrics
 - default for dimension checks, 46
 - Matrix OPC, 414
 - used with dimension checks, 45
 - MGC_HOME, *see* CALIBRE_HOME
 - Midpoint parameter
 - Matrix OPC, 414
 - Minimum keyword, 21
 - Models
 - optical, 182, 241, 243, 650
 - path to, 224
 - resist, 250, 650
 - used with multiple masks, 251, 263
 - Modes, drawing, 624
 - Morphological over/under sizing, 360, 362
 - Multiple masks, 331
 - defining mask number and dose, 212
 - models used for, 251, 263
 - Multithreaded processing, 30, 649

NOTCOINCIDENTINSIDEEDGE tag
command, 486

— O —

Offgrid, 292
OPC, 551
 cleaning up, 59, 289
 controlling with tagging commands, 514
OPC layers
 and ORC, 123
 definition, 121, 650
 illustration, 123

Opc layers, 209
OPC_ENCLOSURE_MIN_LENGTH, 587
OPC_MIN_ENCLOSURE_01, 587
OPC_MIN_ENCLOSURE_10, 586
OPC_MIN_EXTERNAL, 585
OPC_MIN_INTERNAL, 585
OPC_USE_LE_DEF, 357
OPC_VISUALIZE_MATRIX, 580

OPCpro
 and tagging, 64
 controlling, 57
 product description, 57
 SVRF and the tagging commands, 572

Operations
 Litho OPC, 146
 Litho ORC, 148
 Litho Printimage, 150
 SVRF, 47, 52

Opposite Extended metric
 used with dimension checks, 45

Opposite metric
 used with dimension checks, 45

Opposite Symmetric metric, 46

Optical models, 69, 650

Optical Process Correction, *see* OPC

Optical Rule Checking, *see* ORC

Optically visible data, 123, 650

Or operations (SVRF), 54

ORC

 histogram files, 294
 identifying problem edges, 108, 597
 LITHO command syntax, 618
 methodology, 595
 outputting data, 606

product description, 57
setup file requirements, 615
viewing output from, 610

Orientation mapping, 579

Original layers, 33

Outside

 relative to edges, 34
 relative to polygons, 34
 SVRF operation, 53

Over-sizing, 360, 362

— P —

PARALLEL_ONLY Matrix OPC keyword,
 416

PARALLEL_PROJ_MULTIPLIER keyword,
 415

Parentheses, 21

Path, for models, 224

Performance, 635

Phase shifting masks, 331

Phase shifting transmissions, 211

Piecewise linear mode, 361, 364, 625

Pipes, 22

Polygon reference data, 34

Polygons
 dividing space, 34
 false edges, 315

Post processing, 59, 289

Precision statement, 50

PRINTimage
 clean-up after, 360, 362
 drawing mode, 624
 fragmentation with, 365
 mode used with SEM, 361
 product description, 57
 rectilinear mode, 626
 requirements, 627
 viewing results of, 627

Priority in Matrix OPC, 420

Product descriptions

 Calibre OPCpro, 57
 Calibre ORC, 57
 Calibre PRINTimage, 57

PROJECTING metric, 414

Projection mapping, 579

Promotion distance, 282, 301

— Q —

Quotation marks, 22

— R —

Radius

 used for correcting corner rounding, 167

Rectilinear mode, 361, 364, 624

Reference data

 dependence on layer of origin, 35

 dimension checks and layer of origin, 37

 for edges, 34

 illustration, 36

Required keywords, 140

Reserved words

 setup parameters, 139

Residual features

 detecting, 621

Resist models, 69, 250, 650

RESIST sites, 114, 257

Resolution (SVRF), 50

Results

 from ORC, 606

 from PRINTimage, 626

 from rule check statements, 38

 from SVRF statements, 37

 viewing, 611

Results database

 overview, 31

 specifying layer numbers in, 40

 writing data to, 37

RET batch tools

 data passed to, 38

 descriptions of tools, 57

 results from, 69

RET layer types

 asraf, 124

 external, 122

 hidden, 124

 island, 124

 opc, 121

 visible, 123

Ripple fragments

 controlling, 193

 from interfeature fragmentation, 193

interfeature, 88

Rule checks

 comments in, 48

 converting derived layers to, 40

 definition, 37

 names, 39

 results from, 38

 statement, 55

 syntax, 38

Rule files

 abbreviations in, 48

 case sensitivity in, 48

 Comments in, 48

 creating, 47

 definition and contents, 47

 DRC statements, 51

 flag statements, 50

 front matter, 49

 functions of, 29

 layer creation statements, 54

 layer operations statements, 52

 layer specification statements, 51

 layout statements, 49

 order in, 47

 reserved words in, 48

 unit statements, 50

Runtime improvements, 635

Runtime summary, 31

Runtime warnings, 31

RVE

 and ORC, 610

— S —

SAMPLE sites, 114, 257

Scanning electron micrograph,*see*SEM

sea, concavecorn option, 162, 197

SEARCH Matrix OPC keyword, 413

SEM Images

 drawing modes, 364

SEM images

 and fragmentation, 365

 clean up of, 625

 control points, 622

 controlling, 624

 drawing modes, 624

 models used to create, 623

 rectilinear vs. piecewise linear, 361

- uses, 623
- SEM_DO_MORPH, 625
- SEM_MORPH_SIZE, 625
- SEM_USE_SUF_FRAG, 624
- Sensitivity of EPEs, 450
- Setup file keywords
 - background, 155
 - concaveAdjDist, 160
 - concavecorn, 161
 - cornedge, 164
- Setup file variables
 - list, 267
 - LITHO_SUPPRESS_INTERACTION_E_RROR, 317
 - LITHO_SUPPRESS_MAPNUMBER_ER_ROR, 318
 - OPC_USE_LE_DEF, 357
- Setup files
 - case sensitivity, 139
 - syntax, 138
- Setup keywords
 - inline_optical2, 182
- Shielding, 193
- Short edges, 357
- Side-lobes
 - detecting, 621
- Simulation data, 123, 124
- Site orthogonalization, 172
- SITES_ON_ARC control sites, 169
- SITES_ON_EDGE control sites, 169
- Sites, *see* Control sites
- Size operations, 54
- Skewed input, 292
- Slanted words, 21
- Slash, 138
- Sliver features, 59, 289
- Slivers
 - definition, 236
- slopeToleranceTag Matrix OPC keyword, 418
- Snap, 292
- Source layer, 35
- Space end adjacent corner fragmentation, 83, 161
- Sparse OPC, 146
- Specification statements, SVRF, 49
- Speed, 635
- Split
 - intrafeature, 198
- Split fragments, 406
- Square metric
 - used with dimension checks, 45
- Square parentheses, 21
- sse command, 267
- Statements
 - SVRF, 47
- Status messages
 - logging, 367
- Stepsize
 - and tolerance factor, 337
 - definition, 261
- SVRF operations
 - And, 53
 - Coincident Edge, 53
 - Convex Edge, 53
 - Copy, 53
 - Enclosure, 53
 - Expand Edge, 54
 - External, 53
 - Inside, 53
 - Internal, 53
 - layer constructors, 53, 54
 - layer definition statement, 55
 - layer operation statements, 52
 - layer selectors, 53
 - Length, 53
 - Or, 54
 - Outside, 53
 - Size, 54
- SVRF rule files,*see* Rule files
- SVRF statements
 - DRC Check Map, 40, 55, 66
 - DRC Maximum Results, 51
 - DRC Results Database, 51
 - DRC statements, 51
 - DRC Summary Report, 51
 - Flag Acute, 51
 - Flag Offgrid, 51
 - Flag Skew, 51
 - flag statements, 50
 - Layer, 51

Layer Map, 52
layer specification statements, 51
Layout Base Layer, 50
Layout Path, 50
Layout Primary, 50
layout statements, 49
Layout System, 50
Layout Top Layer, 50
Precision, 50
Resolution, 50
results from, 37
rule check statement, 37, 55
specification statements, 49
Unit Length, 50
unit statements, 50

— T —

Tag
based on aerial image, 469
definition, 104
edge placement errors, 446
edges by angle, 442
edges by length, 441, 442
EPEs, 601
epoSensitivity, 602
fragments after edges, 455
image errors, 603
intersection of layers, 478
intersection of two or more tags, 429, 500
not intersecting with layer, 479

Tagging, 64, 650

Tags
built-in, 104, 541
clearing, 375
creating, 426
re-fragmenting, 391, 395, 399, 403, 406
statistics for, 407
viewing, 109

Target layer
with OPCpro, 121
with ORC, 620

Tcl, 142

tilemicrons keyword, 265

Tiles
boundaries, 126
definition, 651

size, 265
Tolerance factor, 337
TOUCHEDGE group
NOTTOUCHEDGE, 494
NOTTOUCHINSIDEEDGE, 496
NOTTOUCHOUTSIDEEDGE, 498
TOUCHEDGE, 507
TOUCHINSIDEEDGE, 509
TOUCHOUTSIDEEDGE, 511

Transcripts

Calibre, 31
printing comments, 138

Transmission value

related to background, 155
related to layer, 210

Triple exposure, 182

-turbo, 30
-turbo_all, 30
-turbo_litho, 30

TYPE parameters
edge tagging argument, 442

— U —

Underlined words, 21
Under-sizing, 360, 362
Unit Length statement, 50

— V —

Variables
list, 267
returning values, 180

Version information, 249

Visible layers, 210
definition, 123

— W —

Warnings
logging, 367
Weighted_Euclidean Matrix OPC keyword,
414
WEIGHTED_PROJECTING Matrix OPC
keyword, 415
Weighting
Matrix OPC, 420
Wrap commands, 137

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.

