

SIEMENS EDA

Calibre® OPCverify™ 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Unpublished work.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	
Calibre OPCVerify Quickstart.....	13
Product Overview	13
Calibre OPCVerify Quickstart Instructions	13
Syntax Conventions.....	14
Calibre OPCVerify Key Concepts	16
Brief Calibre OPCVerify Command Setup File Syntax	16
Chaining Setlayer Commands Together.....	17
Important Calibre OPCVerify Setlayer Operations.....	18
Brief SVRF Rule File Syntax	20
Requirements to Run Calibre OPCVerify	22
Calibre OPCVerify Workflow	22
Calibre FullScale Support in Calibre OPCVerify.....	23
Chapter 2	
Calibre OPCVerify Application Examples	25
Application 1a: Bridging Check	25
Application 1b: Alternative Bridging Check.....	26
Application 2a: Pinching Check	30
Application 2b: Alternative Pinching Check.....	32
Application 3: Gate CD Checking.....	35
Application 4: Generating a Process Variation Band	36
Application 5: Checking for Misalignment	38
Application 6: Viewing ldof Gauges in the Process Window Analysis Tool	39
Chapter 3	
Using Calibre OPCVerify	43
Usage Model For Calibre OPCVerify	43
Creating a Calibre OPCVerify Setup File	44
Defining Layers in the Design.....	46
Creating the SVRF Rule File.....	49
Concurrency Execution of Multiple Calibre OPCVerify Statements in Calibre	52
Topographical Models in Calibre OPCVerify	56
Topographical Model File Examples	57
Using Topographical Models in Calibre OPCVerify (No Litho Model).....	58
Using Topographical Models in Calibre OPCVerify (Litho Model).....	59
Best Practice: Using Tcl with Calibre OPCVerify	62
Frequently Asked Questions	64
Chapter 4	
Calibre OPCVerify Function Reference	67
Constraints	67

Error-Centric Section Blocks	69
Error-Centric Block Types and Rules	69
Property Block	70
Writing Properties to Outputs	71
Classification Block	73
Limit Block	85
Histogram Block	87
add_properties Block	91
Pinpoint Output Block	102
Using LITHO OPCVERIFY	104
LITHO OPCVERIFY	106
Litho Model Format	111
ZPlanes Model Format	118
EUV Through-Slit Litho Model Extension	121
RET INPUT	126
Lint Warnings	130
Calibre OPCVerify Setup File Configuration Commands	133
background	136
bbm_options	139
classify_options	141
clone_transformed_cells	143
collect_frame_stats	146
contour_options	150
critical_dimension	152
ddm_model_load	154
direct_input	155
direct_output	157
dynamic_output	159
etch_imagegrid	162
etch_model_load	163
euv_field_center	164
euv_slit_x_center	165
filter	166
final_upsample	167
flare_longrange	168
flare_model_load	170
gauge_set	171
image_options	174
image_set	177
imagegrid	185
layer	188
layer_properties	192
litho_model	195
log_options	196
mask_sample_grid	198
modelpath	199
optical_model_load	200
optical_transform_size	202
output_window	203

Table of Contents

processing_mode	208
progress_meter	209
promote_subframe_slivers	210
push	211
pw_annotate	212
pw_annotate_options	215
rasterizer_upsample_factor	216
resist_model_load	217
save_error_center_points	218
setlayer	224
shadow_bias_model_load	225
simulation_deangle	228
stairstep	229
stochastic_model_load	230
summary_report	231
svrf_var_import	249
tilemicrons	250
topo_model_load	251
zplanes_model_load	252
Using Setlayer Options as Operations	252
Image Operations	253
DRC-type Operations	253
Verification Control Operations	254
Setlayer Operations Reference	256
and	260
annotate	261
apa_check	264
area_compute	267
area_overlay	271
area_ratio	276
asraf_print_check	280
bandcheck	285
bridge	287
bridge_tolerance	296
build_tolerance	298
center_shift	300
circularity_compute	302
contour_diff	305
copy	309
cornerchop	310
curvature_check	313
dofcheck	316
empty_layer	321
enclosure	322
end_cap	325
external	330
extra_printing	333
filter_generate	336
gate_stats	341

gauges	352
holes	362
identify_corner	364
identify_edge	366
image	369
imax_check	380
imin_check	383
intensity_ilsccheck	386
intensity_meefcheck	390
intensity_nilscheck	393
interact	399
internal	402
maskgen	404
measure_cd	405
measure_cdv	415
measure_cross_section	419
measure_distance	423
measure_epe	432
meefcheck	439
nilscheck	443
not	447
not_printing	448
notchfill	450
or	452
pinch	453
pinch_tolerance	461
polygon_extent	463
pvband	464
pwcheck	465
shadow_bias	471
shift	473
show_annotation	474
size	475
sraf_apr	476
sraf_print_flux	478
tilegen	483
veb_simulate	485
width	486
window	487
xor	488
Appendix A	
Calibre OPCverify Best Practices and Troubleshooting	489
Essential Skillset	490
Best Practices	494
Use Original Target Layers for Pinch and Bridge Checks	495
Use extra_printing Instead of interact for Extra Printing Errors	495
Use not_printing Instead of interact for Not Printing Errors	495

Table of Contents

Choose Appropriate tilemicrons Settings	496
Choose Grid Settings Based On Your Configuration	497
Replace Older OPCVerify Bridge and Pinch Detection Methods With Bridge and Pinch	498
Use an Error-centric Flow	499
Use critical_dimension	500
Use Only the Innermost and Outermost Process Window Variation Contours	501
Avoid Using output_window to Output In-Spec Points	502
Add a Classification and Limiting Block to Process Window Contours Used With output_window	504
Set “reflections_rotations_match” for Optical Models Below Version 10	505
Use Setlayer Concurrency	505
Use Tcl for Dynamic String Construction	507
Include These Checks for Poly, Metal, or Contact Layers	507
Use DFM RDB on Limited and Classified Errors Only	508
Use Filter Layer Markers to Check Square Contacts	508
Use jog_filter With contour_diff	509
When to Use direct_input	509
Using ULTRAflex Versus TURBOflex	509
Common Coding Errors	510
Incorrectly Named Output Layers	511
ASCII Database Named the Same as Derived Layer	511
ASCII Database Files Do Not Take Multiple Input Types	512
Backslash Not at End of Line	512
Avoid Writing Out Full Chip Simulations	512
Do Not Use the size Command on Contour Layers	513
Do Not Use Large length Values for identify_edge	513
output_window is Not For Use With Classification keep duplicates	513
Post-OPC Layers are Not For Use in Classification	514
Classification max_size Set Too Small in Classification Blocks	514
Pinching Commands With max_tolerance Too Small	515
Multiple Property Layers Combined With OR	515
Usage of output_window on OR'd Output Layers	516
Improperly Used output_window filter_only Command	517
Avoid Using Large halo Settings	517
Soft Bridge and Soft Pinch Checks Have Strange Error Markers Near Line Ends	518
Use of DFM RDB NOPSEUDO Option	519
external, internal, and enclosure Inaccuracies When Used on Contours	520
Optical Models from 2005	520
Using the setlayer window Command to Return Classified Errors	520
Appendix B	
Scoring Information	523
Layout Scoring	523
Error Scoring	524
Scoring Functions	524

Appendix C	
Results Viewing in Calibre RVE	525
Invoking and Configuring Calibre RVE	525
Selecting a Property to View	527
Error Properties in Tabular Format	529
Plotting Properties as a Histogram	532
Appendix D	
Calibre OPCverify Source Example:	
Metal1 Layer Checks	535
About the Example File	535
Appendix E	
Calibre OPCverify Source Example:	
Process Windows and TVF Conversion	545
About the Example File	545
Converting This Module to TVF	553
Adding a TVF Header and VERBATIM Blocks	553
Converting LITHO OPCVERIFY Calls to a TVF Loop	555
Converting the setlayer image Operations to a Tcl Loop	558
Appendix F	
Creating DDE Runs in OPCverify	561
DDE and Litho Flat Overview	561
DDE and Litho Flat Workflow	561
Key Concepts	562
DDE and Litho Flat Product Requirements	562
List of Tasks	563
Finding Extents for Full Chip Runs	563
Creating a Litho Flat SVRF Rule File Header and OPCverify Calls	564
Setting up direct_input in the OPCverify Command File	566
Finding Extents for Small Designs	567
Lithoflat Sample Code	569

Index**Third-Party Information**

List of Figures

Figure 1-1. Calibre OPCverify Command Setup File Sample (opcverify.in)	16
Figure 1-2. Calibre OPCverify SVRF Rule File Sample.	21
Figure 1-3. Complete Post-Tapeout Flow on the Calibre Platform.	23
Figure 2-1. bridge Result (Small Value of output_expand).	26
Figure 2-2. Bridging Check Using an Outer Tolerance Zone	27
Figure 2-3. Plot Map of Bridging Areas	30
Figure 2-4. pinching Check (Small output_expand Value)	31
Figure 2-5. Inner Tolerance Zone for Pinching Areas.	33
Figure 2-6. Gate Check Inside the Active Area (Process Variation Band)	36
Figure 2-7. pvband Exposure Conditions	38
Figure 2-8. Misalignment Check	39
Figure 2-9. ldof Output Data in the Calibre WORKbench Process Window	41
Figure 3-1. Calibre OPCverify Setup File Example (opcverify.in)	45
Figure 3-2. Calibre OPCverify SVRF File Example	49
Figure 4-1. Anchoring	75
Figure 4-2. add_properties Example	100
Figure 4-3. meef Operation Measurement Points for add_properties	101
Figure 4-4. Multiple-Mask Exposure.	136
Figure 4-5. collect_frame_stats, Hierarchical Tiling.	148
Figure 4-6. collect_frame_stats, Flat Tiling.	149
Figure 4-7. Merge Action Layout	192
Figure 4-8. filter_only Mode in output_window	206
Figure 4-9. Two-layer Boolean AND Operation.	260
Figure 4-10. annotate Results for Orientation	263
Figure 4-11. area_compute operation	270
Figure 4-12. area_overlay Example	274
Figure 4-13. area_overlay With Shifted Contacts	275
Figure 4-14. area_ratio Operation	279
Figure 4-15. bandcheck Example	285
Figure 4-16. Bridge Tolerance Example	296
Figure 4-17. cornerchop Example	310
Figure 4-18. dofcheck Diagram	317
Figure 4-19. Basic Enclosure Rule Checks	322
Figure 4-20. enclosure Result	324
Figure 4-21. end_cap Measurement.	326
Figure 4-22. external Example.	332
Figure 4-23. filter_generate Example Output	340
Figure 4-24. Constructing Gates With gate_stats	342
Figure 4-25. Gate Equation	344
Figure 4-26. gate_stats exceptions_only Samples	346

Figure 4-27. gate_stats Example	351
Figure 4-28. identify_corner Example Output	365
Figure 4-29. Using measure_cd on 90 Degree Corners	406
Figure 4-30. Misaligned Corners Exception	406
Figure 4-31. max_search Diagram	408
Figure 4-32. measure_cd, output_expanded_edges Example	410
Figure 4-33. runlength Argument in measure_cd	411
Figure 4-34. measure_cd Example Output (Standard Mode)	412
Figure 4-35. measure_cd Example Output (output_expanded_edges Mode)	413
Figure 4-36. measure_cd Example Output (exceptions_only Mode)	414
Figure 4-37. measure_cross_section Diagram (OK versus Not Good)	420
Figure 4-38. measure_distance Clarification	425
Figure 4-39. measure_distance Check With Overlap	426
Figure 4-40. One-layer Separation Explained	428
Figure 4-41. Two-layer Separation Explained	428
Figure 4-42. measure_epe properties Example	438
Figure 4-43. notchfill Operation	450
Figure 4-44. Two-Layer Boolean or	452
Figure 4-45. Pinch Tolerance Example	461
Figure A-1. Layer Order in Calibre OPCverify	491
Figure A-2. Calibre WORKbench and Calibre RVE	492
Figure A-3. Inner Contours Versus Outer Contours	502
Figure A-4. Example Histogram File Output	503
Figure A-5. Filter Layer Suggestion for Square Contacts	508
Figure A-6. output_expand width_microns2 Argument	519
Figure C-1. Calibre RVE	526
Figure C-2. Calibre RVE Options	526
Figure C-3. Selecting Options from the RVE View Menu	527
Figure C-4. Viewing by Rule	528
Figure C-5. Viewing by Error	529
Figure C-6. Property Map	530
Figure C-7. Filtering by Value	531
Figure C-8. Results Table	531
Figure C-9. Invoking the Histogram Pane	532
Figure C-10. Histogram for a Property Displayed	533
Figure C-11. Invoking a Secondary Histogram	534
Figure C-12. Secondary Histogram (Example)	534

List of Tables

Table 1-1. Calibre OPCVerify Quickstart Table	14
Table 1-2. Syntax Conventions	14
Table 1-3. Required Configuration Command Tasks	16
Table 1-4. Contour and Tolerance Zone Generators	18
Table 1-5. Filter Generators	19
Table 1-6. Verification Checks	19
Table 1-7. Unsupported Calibre OPCVerify Commands in Calibre FullScale	23
Table 1-8. Conditional Support for FullScale in Calibre OPCVerify Commands	24
Table 1-9. FullScale Limited Feature Support in Calibre OPCVerify	24
Table 3-1. Litho Model, Including Topo Model Syntax	60
Table 4-1. Constraints	67
Table 4-2. Default Context Layers for Classification	76
Table 4-3. One-Sided add_properties Operation Value Used From a Two-Sided Gauge ..	93
Table 4-4. Conversion between Positive/Negative and Top/Bottom for SRAF	118
Table 4-5. Conversion between Mask/Resist Polarity and Printing Drawn Pattern Polarity	118
Table 4-6. OPCVerify Setup Configuration Commands	133
Table 4-7. critical_dimension New Settings	152
Table 4-8. layer Command transmission Argument Settings	189
Table 4-9. Summary Report Argument Scope	233
Table 4-10. Cross-Referencing Error and Warning Lists	236
Table 4-11. Format Specifiers Supported for snapshot_file_name	238
Table 4-12. SVG View Controls	246
Table 4-13. Calibre OPCVerify Setlayer Operations (DRC-type)	253
Table 4-14. Calibre OPCVerify Setlayer Operations (Verification Control)	254
Table 4-15. Calibre OPCVerify Setlayer Operations List	256
Table 4-16. gate_stats Property Keywords	349
Table 4-17. gauges properties_commands	353
Table 4-18. copy_props Syntax	399
Table 4-19. measure_cd Property Keywords	411
Table 4-20. measure_cross_section Properties	422
Table 4-21. measure_epe Property Keywords	436
Table 4-22. measure_epe properties Results	438
Table A-1. LITHO OPCVERIFY Variable Items	491
Table A-2. Best Practices for Calibre OPCVerify	494
Table A-3. Recommended tilemicrons Settings for Calibre OPCVerify	496
Table A-4. Recommended Failure Detection Methods	498
Table A-5. Using the Error-Centric Flow	499
Table A-6. Suggested Checks By Layer Type	507
Table A-7. Coding Error Checklist	510
Table C-1. Calibre RVE Results Viewing Process	525

Table F-1. Litho Flat in OPCverify Using direct_input Tasks	563
---	-----

Chapter 1

Calibre OPCverify Quickstart

Use this Quickstart chapter to learn the purpose of Calibre® OPCverify™ and to understand how to use it effectively.

Product Overview	13
Calibre OPCverify Quickstart Instructions	13
Syntax Conventions	14
Calibre OPCverify Key Concepts	16
Brief Calibre OPCverify Command Setup File Syntax	16
Chaining Setlayer Commands Together.....	17
Important Calibre OPCverify Setlayer Operations.....	18
Brief SVRF Rule File Syntax	20
Requirements to Run Calibre OPCverify	22
Calibre OPCverify Workflow	22
Calibre FullScale Support in Calibre OPCverify	23

Product Overview

Calibre OPCverify is a grid-based optical process simulator and OPC results verification tool that is designed to predict a manufacturing process window for wafer contours. It supports the industry-standard OPC usage models and seamlessly integrates with other Siemens EDA tools.

Audience

Users who intend to run Calibre OPCverify should be familiar with Calibre procedures for OPC and DRC. An understanding of programming Standard Verification Rules Format (SVRF) rule decks is highly useful.

Note All Calibre OPCverify functions are implemented using Tcl. The Tcl standard argument syntax and parameter conventions are required when coding for Calibre OPCverify.

Calibre OPCverify Quickstart Instructions

Use the following table to quickly understand the basics of Calibre OPCverify.

Table 1-1. Calibre OPCVerify Quickstart Table

You use Calibre OPCVerify as a programming language, building one or more Verification Rule checks to run versus the results of an OPC simulation.	
Calibre OPCVerify requires a previously-created optical model file and resist model file (if resist modeling is to be simulated) for use in the simulations.	
You Perform These Tasks	<ol style="list-style-type: none">1. Write an OPCVerify command setup file to set up your environment.2. Write an SVRF rule file.3. Run with Calibre® nmDRC™.4. Examine results.
Related Information — Learn How to Use Calibre OPCVerify	
Read First	<ul style="list-style-type: none">• “Calibre OPCVerify Application Examples” on page 25• “Calibre OPCVerify Best Practices and Troubleshooting” on page 489
Then Read	<ul style="list-style-type: none">• “Using Calibre OPCVerify” on page 43 <p>Contains detailed usage information for Calibre OPCVerify in Batch Mode, including in-depth coverage of sections covered in this chapter.</p> <ul style="list-style-type: none">• “Results Viewing in Calibre RVE” on page 525 <p>Shows how to use Calibre® RVE™ to view ASCII result file data.</p>
Command Dictionary	“Calibre OPCVerify Function Reference” on page 67

Syntax Conventions

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

Table 1-2. 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.

Table 1-2. Syntax Conventions (cont.)

Convention	Description
[]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.
‘ ’	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
or	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
...	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.

Example:

```
DEvice {element_name [('model_name')]}

device_layer {pin_layer [('pin_name')] ...}
[‘<auxiliary_layer> ...]
[‘('swap_list') ...]
[BY NET | BY SHAPE]
```

Calibre OPCVerify Key Concepts

The following sections describe key concepts to understanding Calibre OPCVerify.

Brief Calibre OPCVerify Command Setup File Syntax	16
Chaining Setlayer Commands Together	17
Important Calibre OPCVerify Setlayer Operations.....	18
Brief SVRF Rule File Syntax	20

Brief Calibre OPCVerify Command Setup File Syntax

Calibre OPCVerify command setup files are text files. They contain two types of statements: configuration commands and setlayer commands.

Figure 1-1. Calibre OPCVerify Command Setup File Sample (opcverify.in)

```
modelpath $env(DESIGN_DIR)/models
optical_model_load o1 DUV10tab
optical_model_load o2 DUV10tab_plus1
```

Optical Model Input

```
background clear
layer POLY visible dark
layer TEST hidden dark
```

Layer setup

```
imagegrid .04
stairstep 1
#setlayer commands below this line
```

```
setlayer ti0 = build_tolerance inner POLY max .16 stepsize .01
setlayer ti = cornerchop ti0 .03 .03
setlayer to0 = build_tolerance outer POLY max .16 stepsize .01
setlayer to = cornerchop to0 .03 .03

setlayer il1 = image optical o1 aerial_contour 0.3
setlayer il2 = image optical o2 aerial_contour 0.3
```

These setlayer commands create Calibre OPCVerify "context" layers for later use.

```
setlayer b1 = bridge_tolerance il1 ti
setlayer b2 = bridge_tolerance il2 to
```

- Configuration commands set up the working environment for Calibre OPCVerify.

Table 1-3. Required Configuration Command Tasks

Task	Command
Set Model Path	modelpath
Import Optical Model	optical_model_load

Table 1-3. Required Configuration Command Tasks (cont.)

Task	Command
Import Resist Model	resist_model_load
Set up Layers	background and layer
Set Grid Size	imagegrid

Tip

i The full list of Calibre OPCVerify setup file configuration commands can be found in the section “[Calibre OPCVerify Setup File Configuration Commands](#)” on page 133.

- Setlayer commands perform Calibre OPCVerify operations. Setlayer commands all have the form:

```
setlayer name = operation arguments
```

Calibre OPCVerify supports three types of setlayer commands:

- image generation
- DRC-type layer manipulation commands
- Verification commands, which include polygon modification and check commands

Tip

i The full list of Calibre OPCVerify setlayer commands can be found starting with the section “[Setlayer Operations Reference](#)” on page 256.

Chaining Setlayer Commands Together

Each setlayer command creates a context layer that can be used in subsequent setlayer commands. At least one context layer should be used as a derived output layer back to the SVRF file.

You will do most of your development programming in the Calibre OPCVerify command setup file, using multiple setlayer statements in sequence, as in the following example:

```
1 optical_model_load o1 mymodel.opt
2 layer ORIG visible dark
3
4 setlayer to1 = build_tolerance outer ORIG MAX 0.080 STEPSIZE .01
5 setlayer to = cornerchop to1 0.020 0.020
6 setlayer i1 = image optical o1 dose 0.96 aerial_contour 0.30
7 setlayer bridge1 = bridge_tolerance i1 to
```

This example, which is a bridging check, executes the following steps:

1. The [build_tolerance](#) verification command (line 4) modifies the input layer ORIG so that all of its polygons have an outer tolerance of .080 um, creating the Calibre OPCverify layer to1.
2. The [cornerchop](#) command (line 5) takes layer to1 and modifies it to remove the corners from the tolerance zones, creating the Calibre OPCverify layer to.
3. The image command (line 6) takes the loaded optical model o1, and creates an aerial contour simulation at a threshold of 0.30 and a dose value of 0.96, creating the Calibre OPCverify layer i1.
4. The [bridge_tolerance](#) command (line 7) takes the contour layer i1 and the tolerance zone layer **to**, and computes the results on the derived Calibre OPCverify layer bridge1. You will later use this layer name for the MAP argument for the SVRF LITHO OPCVERIFY, creating an SVRF output layer from the Calibre OPCverify layer.

For more information on setup files, see “[Creating a Calibre OPCverify Setup File](#)” on page 44.

Important Calibre OPCverify Setlayer Operations

Most users of Calibre OPCverify will probably utilize the following verification control operations.

Note

 This is a subset of the full operations list; the complete list and descriptions of the Calibre OPCverify operations can be found in the section “[Setlayer Operations Reference](#)” on page 256 for the batch commands.

Calibre OPCverify Generators

Generators create needed data (usually context layers) for subsequent Calibre OPCverify checks.

Contour and Tolerance Zone Generators

Use the following generators to build contours and tolerance zones around target layer shapes:

Table 1-4. Contour and Tolerance Zone Generators

Operation	Description
image	Simulates a contour. Many verification checks require a contour layer.
build_tolerance	Generates a tolerance zone around target layer shapes.
cornerchop	Chops out a triangular section off all corners.
veb_simulate	Applies an etch bias based on an etch model to a contour layer.

Filter Generators

Use filter layers to designate areas of interest, in order to exclude other areas from the computational checks.

Table 1-5. Filter Generators

Operation	Description
filter_generate	Generates a filter layer for the measure_cd command.
identify_corner	Identifies line fragments of edges near corners.
identify_edge	Identifies line ends or jogs.

Verification Checks

Use verification checks on constructed context layers to find results.

Table 1-6. Verification Checks

Operation	Description
area_compute	Checks for shapes that match a certain area constraint.
area_overlay	Performs a contact or via alignment check.
area_ratio	Checks for shapes that meet a certain area ratio constraint.
bandcheck	Checks CD accuracy for inner and outer tolerance band violations.
bridge	Checks for bridging problems.
bridge_tolerance	Checks for outer tolerance band violations.
contour_diff	Calculates the error factor between two contours, similar to meefcheck .
dofcheck	Calculates the depth of focus (DOF) for target shape edges under different defocus conditions.
end_cap	Checks endcap coverage.
extra_printing	Checks for extra printing features.
gate_stats	Computes statistics for a gate area.
holes	Returns holes inside layer shapes.
interact	Checks for shapes that interact with each other.
measure_cd	Checks width and space CD accuracy.
measure_cdv	Checks CD between two contours versus a target CD.
measure_epe	Checks if a layer's EPE fails a specified constraint.
measure_distance	Checks the distance between an edge and the nearest edge.

Table 1-6. Verification Checks (cont.)

Operation	Description
meefcheck	Checks if a polygon's MEEF values fail a specified constraint.
nilscheck	Checks the Normalized Image Log-Slope (NILS) for two or more contours for a specified range constraint.
not_printing	Checks for features that do not print.
notchfill	Fills in notches on polygons.
pinch	Checks for pinching problems.
shift	Shifts the polygons on a layer a distance in x and y.
polygon_extent	Checks the area of polygons in x and y dimensions.
pinch_tolerance	Checks for inner tolerance band violations.
pvband	Generates a process variation band from multiple contours.
window	Outputs context clips based on an error layer.

Brief SVRF Rule File Syntax

The SVRF rule file uses the same syntax as a Calibre nmDRC rule file. It must include a LITHO OPCVERIFY operation.

Figure 1-2. Calibre OPCVerify SVRF Rule File Sample

```

LAYOUT PATH ".tmp/opcverify_i.gds"
LAYOUT PRIMARY "PUB26"
LAYOUT SYSTEM GDSII
PRECISION 1000.0
UNIT LENGTH 1e-06
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE ".tmp/opcverify_o.gds" GDSII

LAYER IV2 2
LAYER IV0 0
Specify layers corresponding to the design; these are used for input to the Calibre OPCVerify command setup file.

i1 = LITHO OPCVERIFY FILE ".tmp/opcverify.in" IV2 IV0 MAP i1
i1 {COPY i1} DRC CHECK MAP i1 201
DRC CHECK MAP i1 ASCII "verify_output2.asc"

i2 = LITHO OPCVERIFY FILE ".tmp/opcverify.in" IV2 IV0 MAP i2
i2 {COPY i2} DRC CHECK MAP i2 202
DRC CHECK MAP i2 ASCII "verify_output2.asc"

ti = LITHO OPCVERIFY FILE ".tmp/opcverify.in" IV2 IV0 MAP ti
ti {COPY ti} DRC CHECK MAP t
DRC CHECK MAP ti ASCII "veri
These LITHO OPCVERIFY commands call the Calibre OPCVerify command setup file. They take SVRF input layers as input (the order matters) and return derived Calibre OPCVerify context layers.

to = LITHO OPCVERIFY FILE ".tmp/opcverify.in" IV2 IV0 MAP to
to {COPY to} DRC CHECK MAP to 102
DRC CHECK MAP to ASCII "verify_output2.asc"

b1 = LITHO OPCVERIFY FILE ".tmp/opcverify.in" IV2 IV0 MAP b1
b1 {COPY b1} DRC CHECK MAP b1 301
DRC CHECK MAP b1 ASCII "verify_output2.asc"

b2 = LITHO OPCVERIFY FILE ".tmp/opcverify.in" IV2 IV0 MAP b2
b2 {COPY b2} DRC CHECK MAP b2 302
DRC CHECK MAP b2 ASCII "verify_output2.asc"

bridge_score {
    DFM ANALYZE b1 b2 WINDOW 0.1
    [COUNT(b1)+COUNT(b2)] > 0
    RDB ONLY bridge_score.asc
}
This optional example scoring function command uses DFM ANALYZE, and operates on the derived SVRF output layers.

```

Call Calibre OPCVerify using a standard SVRF rule file, with the following requirements:

- LAYER statements that map to the design file layer numbers.

- One or more LITHO OPCVERIFY calls to the Calibre OPCVerify command setup file to generate derived layers.
 - The order you give the names of the input layers to the command determines the order of their assignment in the Calibre OPCVerify setup file.
 - The name of a context layer in the Calibre OPCVerify setup file must match the derived layer name.
- Rule checks to analyze the results.

For more information on Calibre OPCVerify SVRF files, see “[Creating the SVRF Rule File](#)” on page 49.

Requirements to Run Calibre OPCVerify

Running Calibre OPCVerify requires the following items:

- The Calibre OPCVerify license is required to run this tool. In addition, viewing the results of a Calibre OPCVerify run requires a Calibre WORKbench license. Refer to the [Calibre Administrator’s Guide](#) for more information on licensing these products.
- An SVRF file (see “[Brief SVRF Rule File Syntax](#)”) to call the Calibre OPCVerify command setup file
- A Calibre OPCVerify command setup file (see “[Brief Calibre OPCVerify Command Setup File Syntax](#)”) to define the Calibre OPCVerify operating parameters and run one or more user-programmed verification rule checks.

Once you have both of these items, you run Calibre OPCVerify by entering the following command in a console window:

```
calibre -drc -hier -turbo -turbo_litho svrf_filename
```

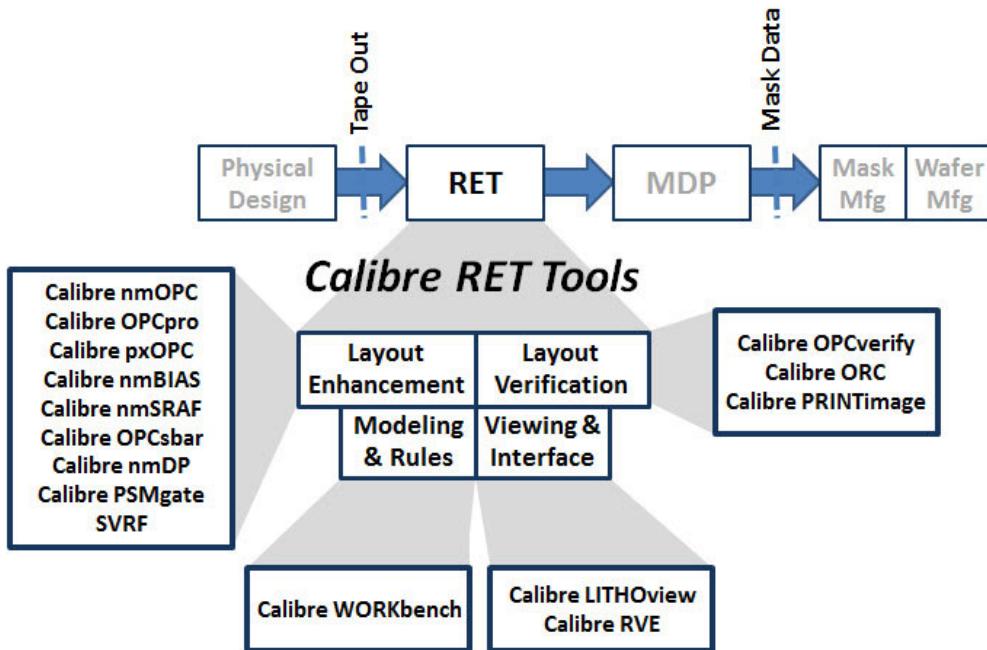
Calibre OPCVerify Workflow

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

[Figure 1-3](#) shows the complete design to mask flow based on the Calibre platform.

For more information on the tools in this workflow, see the [Calibre Post-Tapeout Flow User’s Manual](#).

Figure 1-3. Complete Post-Tapeout Flow on the Calibre Platform



Calibre FullScale Support in Calibre OPCverify

Calibre OPCverify is supported as a part of the Calibre® FullScale™ platform. While most setlayer commands support Calibre FullScale, there are some restrictions and limitations with the current release.

Note

For more information on Calibre FullScale, see the section “[Calibre FullScale Platform](#)” in the *Calibre Post-Tapeout Flow User’s Manual*.

Unsupported Calibre OPCverify Commands

The following commands are not presently supported in Calibre FullScale rule files. Depending on the command, the Calibre OPCverify parser will either produce a parse-time error, or silently have no effect.

Table 1-7. Unsupported Calibre OPCverify Commands in Calibre FullScale

bandcheck	dynamic_output	pw_annotate
save_error_center_points	window	RET INPUT FILE

Items Conditionally Supported by Calibre OPCverify

The following Calibre OPCverify commands are supported in Calibre FullScale, but with the following limitations:

Table 1-8. Conditional Support for FullScale in Calibre OPCverify Commands

Command	Notes
copy	Does not copy properties from Calibre OPCverify input layers.
dofcheck	Does not support the common_ldof and sgd_output options.
gate_stats	Does not support the no_split_marker option.
gauges	Does not support edge input layers, copied properties on input layers, the common_ldof operation, and the sgd_output feature of the ldof subcommand.
image	Does not support topo models or shadow bias models.
interact	Does not support the copy_props option.
log_options	Ignores report_cells, tile_warning_period, and tile_memory_warning_mb options.
pwcheck	Does not support the common_ldof and sgd_output options.

The following features have limited support for Calibre FullScale:

Table 1-9. FullScale Limited Feature Support in Calibre OPCverify

Feature	Notes
Classification	Does not support save_error_center_points.
DFM properties	Not supported for input layers.

Chapter 2

Calibre OPCverify Application Examples

Calibre OPCverify may be best understood by examining sample applications that highlight some of the best practices for OPC verification.

Application 1a: Bridging Check	25
Application 1b: Alternative Bridging Check.....	26
Application 2a: Pinching Check.....	30
Application 2b: Alternative Pinching Check.....	32
Application 3: Gate CD Checking	35
Application 4: Generating a Process Variation Band	36
Application 5: Checking for Misalignment	38
Application 6: Viewing Idof Gauges in the Process Window Analysis Tool.....	39

Application 1a: Bridging Check

Calibre OPCverify can detect both near-bridging conditions (soft bridging) and bridging conditions (where two contour edges touch) using the **bridge** command.

Tip The **bridge** command detailed in this section is significantly more accurate than the method described in “[Application 1b: Alternative Bridging Check](#)”. This version of the **bridge** command detects hard and soft bridging, as well as extra printing features. A separate [extra_printing](#) command exists to find only extra printing features.

A hard bridging check might look like the following code:

```
setlayer short = bridge m1 img_m1_ctr max_tolerance 0.05 \
    output_expand 0.005 property {min}
```

A soft bridging check might look like the following code:

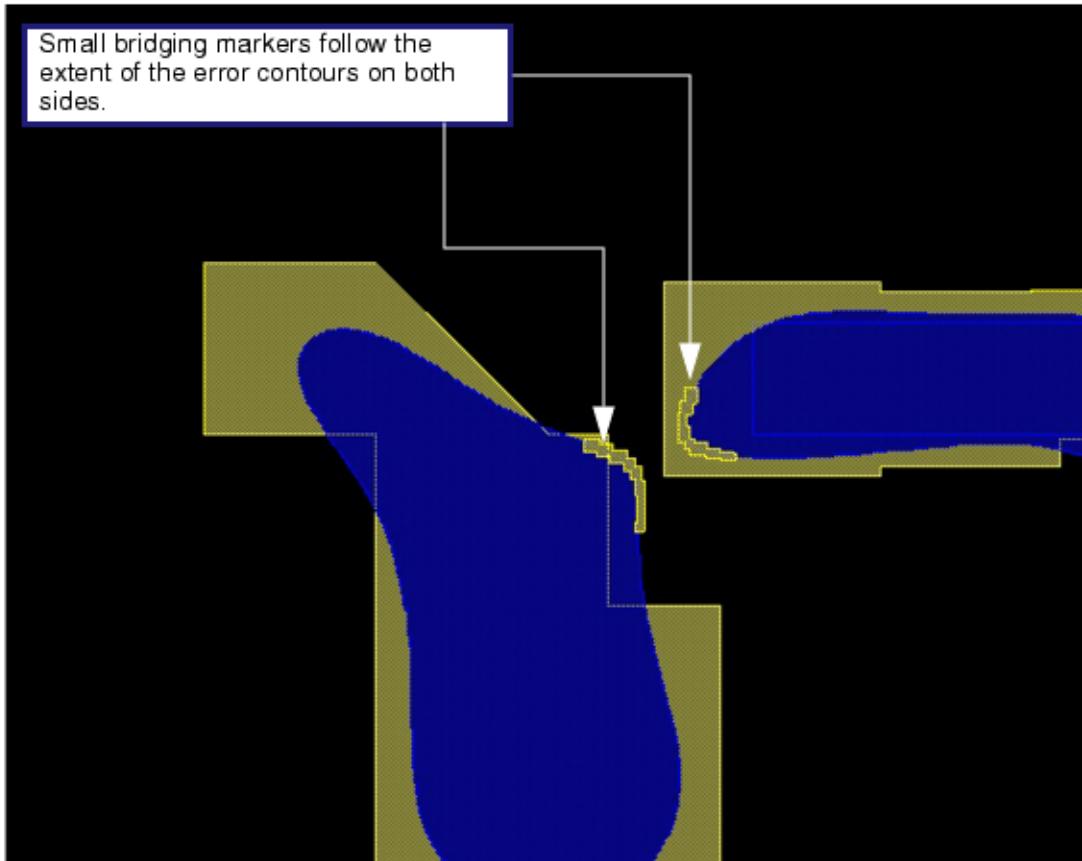
```
setlayer soft_bridge = bridge m1 img_m1_ctr < 0.060 separation 0.20 \
    max_tolerance 0.05 max_edge 0.050 output_expand 0.005 property {min}
```

The primary difference between the two checks is the inclusion of a distance constraint

(< 0.060 separation 0.200) in the soft bridging check; the hard bridging check uses a distance check of zero. The constraint determines the process failure boundary.

- The separation value sets the distance between bridging tests; it should be set to 3x the minimum target CD.
- The output_expand value determines the size of the output markers. Smaller markers will show as individual marks on the bridging contours. Larger values create a single block-like marker. Do not use large output_expand values (>250nm), as this may cause errors.

Figure 2-1. bridge Result (Small Value of output_expand)



- Specifying a property keyword attaches a value to the marker based on the measured minimum bridging distance found along the contour curve.

Application 1b: Alternative Bridging Check

Bridging checks can also be created using a tolerance method.

Tip

The method of bridge checks using the **bridge_tolerance** and **cornerchop** commands as described in this section is still supported by Calibre OPCVerify, but is not the recommended method of detecting bridging.

A simulated contour lying outside a generous tolerance band (defined as an area of space created by expanding polygon edges outward for bridging or inward for pinching) is either bridging or printing inaccurately enough that detection is warranted. Locations where the simulation contour exceeds the outer tolerance zone are flagged as errors, using the `bridge_tolerance` command.

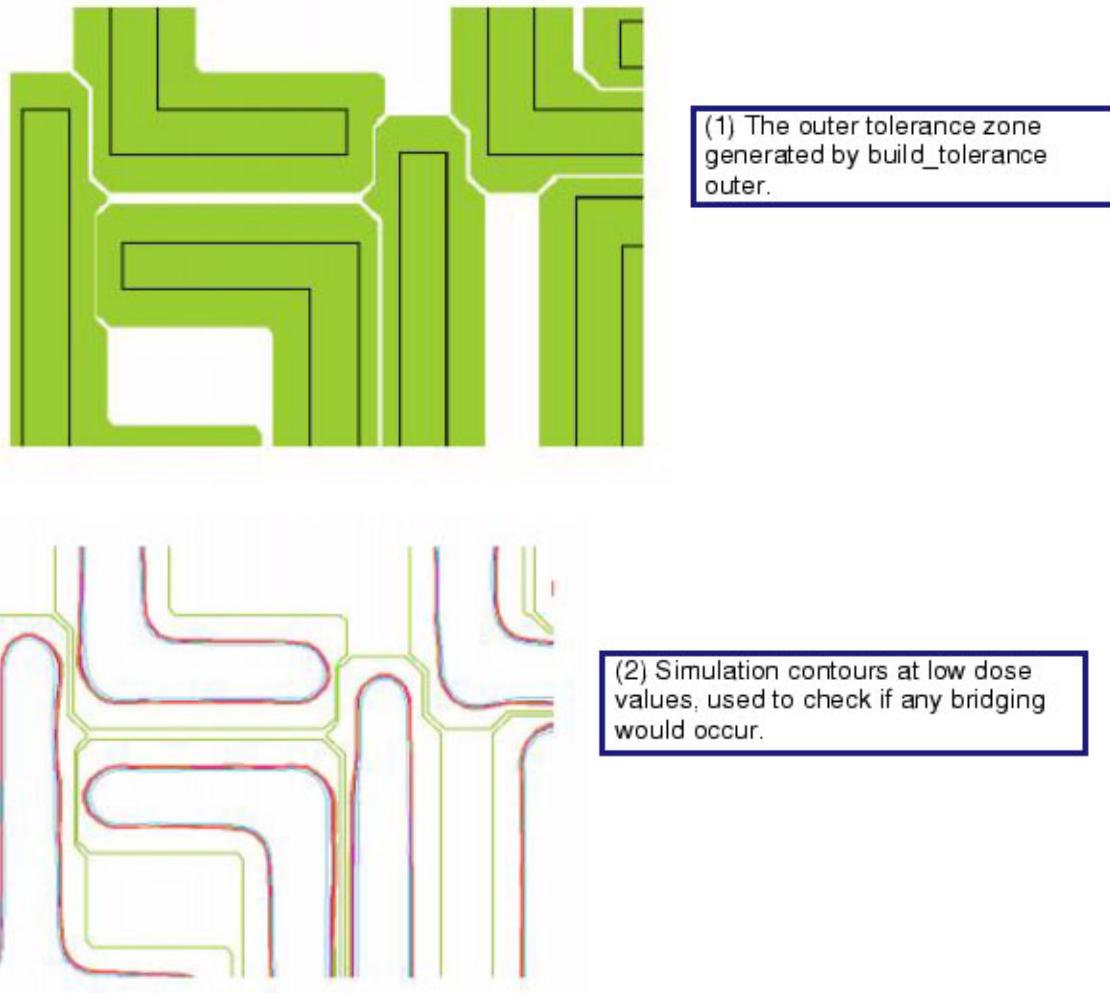
Process Flow

Step 1: Building the Outer Tolerance Zone

First, generate an outer tolerance zone using the following steps:

1. Build the initial tolerance zone using the `build_tolerance outer` command
2. Cornerchop the tolerance zone using the `cornerchop` command

Figure 2-2. Bridging Check Using an Outer Tolerance Zone



The outer tolerance zone generated from this method is shown in [Figure 2-2](#) above. It is clear that any cases of bridging will be detected (no missed errors). However it is possible for a simulation contour to cross a tolerance zone without actually bridging (a false error).

In this example, the design size is 130nm. The following OPCverify setup file generates an outer tolerance zone:

```
optical_model_load o1 mymodel.opt
background clear
layer SHAPES visible dark
layer ORIG hidden dark
imagegrid 0.040
stairstep 0
setlayer tol = build_tolerance outer ORIG max .08 stepsize .01
setlayer to = cornerchop tol .02 .02
```

Note

 If you are building tolerance bands on 45-degree angle layouts, you may run into some difficulties in accurately representing the geometry. Siemens EDA advises that you use multiple smaller, alternating increments of **build_tolerance** and **cornerchop** until you achieve a satisfactory result.

Step 2: Adding Checks for Bridging Violations

By adding the following lines to the OPCverify setup file, you can perform the tolerance zone check for bridging. The basic flow is:

1. Generate simulation contours for a set of given dose values:

```
setlayer i1 = image optical o1 dose 0.96 aerial_contour 0.3
setlayer i2 = image optical o1 dose 0.92 aerial_contour 0.3
setlayer i3 = image optical o1 dose 0.88 aerial_contour 0.3
setlayer i4 = image optical o1 dose 0.84 aerial_contour 0.3
setlayer i5 = image optical o1 dose 0.80 aerial_contour 0.3
```

2. Check for bridging at each dose:

```
setlayer bridge1 = bridge_tolerance i1 to
setlayer bridge2 = bridge_tolerance i2 to
setlayer bridge3 = bridge_tolerance i3 to
setlayer bridge4 = bridge_tolerance i4 to
setlayer bridge5 = bridge_tolerance i5 to
```

3. Add the following code to your SVRF file to output results:

```

bridge1 = LITHO OPCVERIFY POLY_OPC POLY_TARGET FILE \
    setup MAP bridge1
bridge2 = LITHO OPCVERIFY POLY_OPC POLY_TARGET FILE \
    setup MAP bridge2
bridge3 = LITHO OPCVERIFY POLY_OPC POLY_TARGET FILE \
    setup MAP bridge3
bridge4 = LITHO OPCVERIFY POLY_OPC POLY_TARGET FILE \
    setup MAP bridge4
bridge5 = LITHO OPCVERIFY POLY_OPC POLY_TARGET FILE \
    setup MAP bridge5

```

Optional Step: Scoring the Results Using the DRC Report Summary

The most likely bridging points in the layout are those that bridge closest to best dose (assuming best focus). You can create a distribution of number of failures as a function of dose by reading the DRC summary report file generated by Calibre (shown below).

```

RULECHECK bridge1 TOTAL Result Count = 1 (1) // DOSE = 0.96
RULECHECK bridge2 TOTAL Result Count = 1 (1) // DOSE = 0.92
RULECHECK bridge3 TOTAL Result Count = 1 (1) // DOSE = 0.88
RULECHECK bridge4 TOTAL Result Count = 8 (8) // DOSE = 0.84
RULECHECK bridge5 TOTAL Result Count = 32 (32) // DOSE = 0.80

```

The “Result count” shows two numbers, the first is the hierarchical count, the second is the flat count. The “worst” errors are those that show up in the “bridge1” check, because this check is closest to nominal dose. The number of errors grows for each check (starting with dose 0.84) as you decrease the dose further.

Optional Step: Scoring the Results Using DFM ANALYZE

The previous scoring technique, using the report summary file, represented a “global score” which gives a relative ranking of bridging count versus dose. Using DFM ANALYZE, a “local” score of bridging can be evaluated for each NxN micron window on the chip. The DFM analyze portion adds SVRF code as follows:

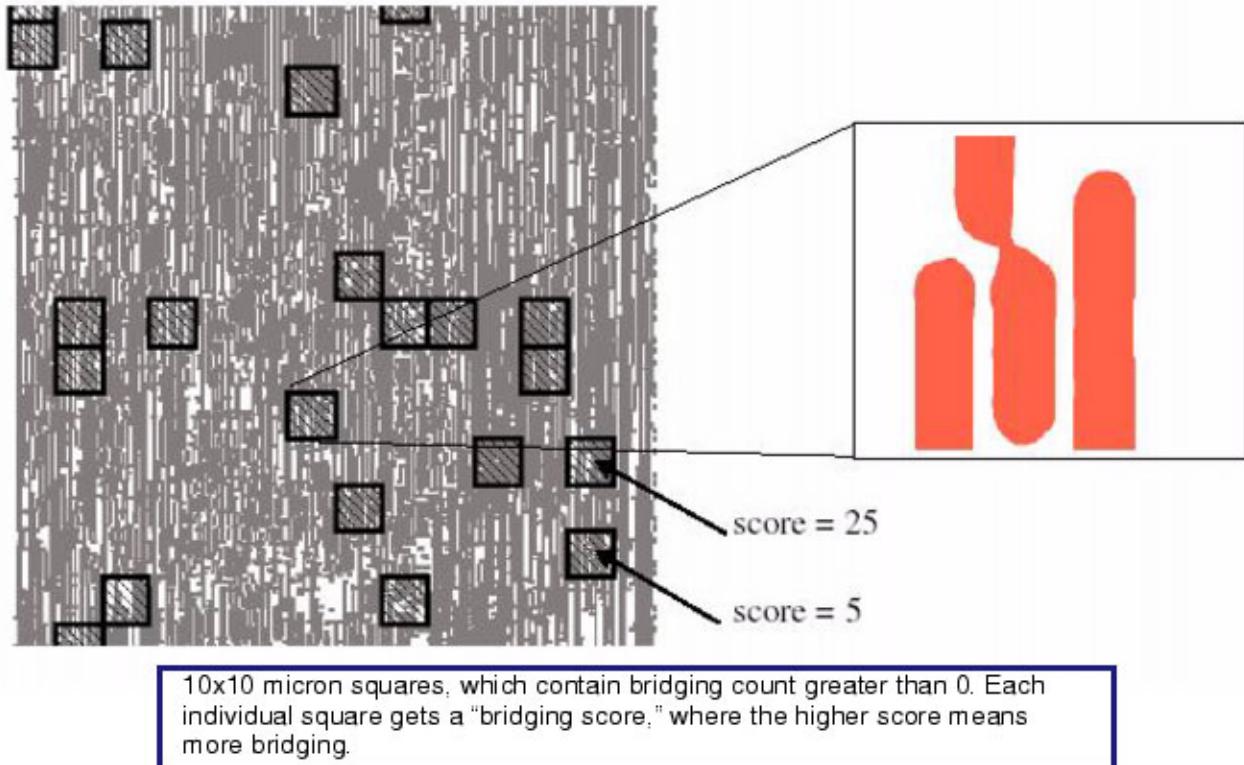
```

bridge_score {
    DFM ANALYZE bridge1 bridge2 bridge3 bridge4 bridge5 WINDOW 10
    [COUNT(bridge1)+COUNT(bridge2)+COUNT(bridge3)+COUNT(bridge4)+
    COUNT(bridge5) ] > 0
    RDB ONLY bridge.ascii
}

```

Here, DFM ANALYZE creates a score for each 10x10 micron window. If the score is greater than 0, it means at least one bridge occurred in the 10x10 micron window. A higher score indicates more bridging. Now, you can load the ASCII RDB file “bridge.ascii” into Calibre RVE and plot a map of the bridging areas, as shown in [Figure 2-3](#). By sorting the ASCII error results by the score, you can locate the worst bridging conditions.

Figure 2-3. Plot Map of Bridging Areas



Application 2a: Pinching Check

Pinching checks look for errors in printed contours where two contour edges have near-gaps (soft pinching) or gaps (hard pinching) where they should not. Use the **pinch** command to detect pinching problems.

Tip

The **pinch** command detailed in this section is significantly more accurate than the method described in “[Application 2b: Alternative Pinching Check](#)”. This version of the **pinch** command detects hard and soft pinching, as well as “not printing” features. A separate [**not_printing**](#) command exists to find only these features.

A hard pinching check:

```
setlayer short = pinch m1 img_m1_ctr output_expand 0.005 \
max_tolerance 0.05 property {min}
```

A soft pinching check:

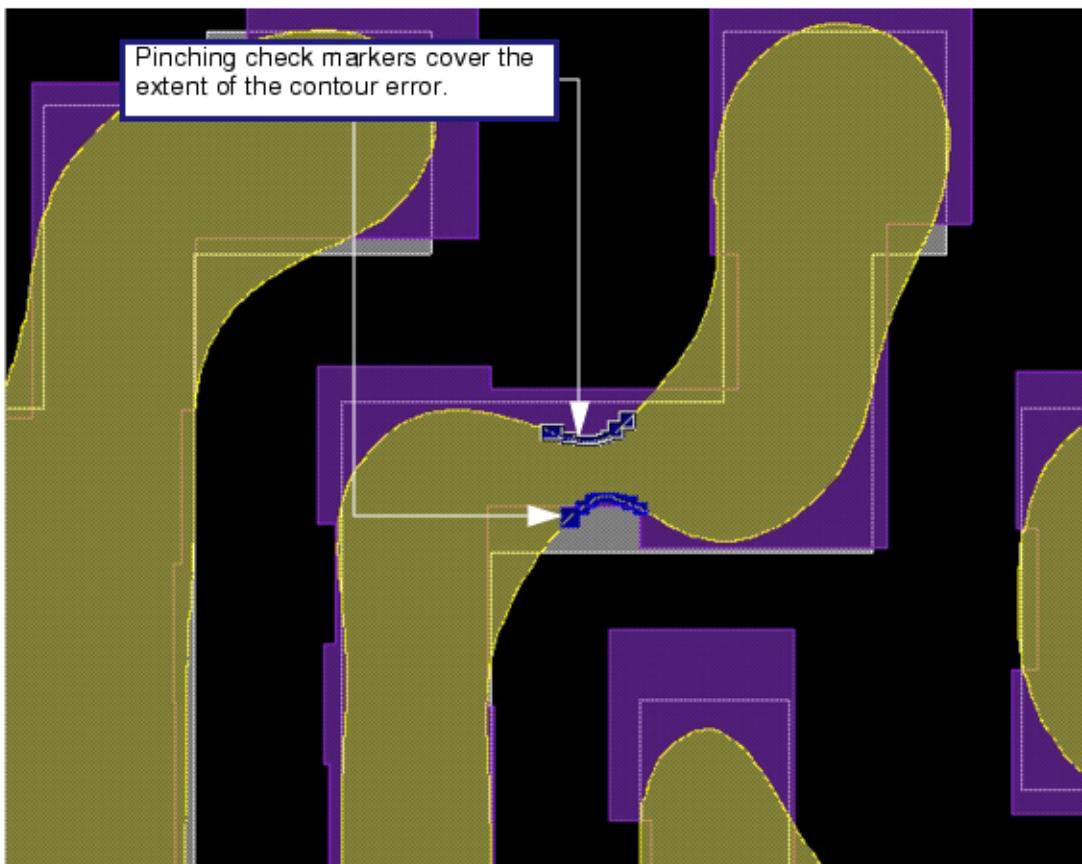
```
setlayer soft_pinch = pinch m1 img_m1_ctr < 0.050 separation 0.24 \
max_tolerance 0.05 max_edge 0.050 output_expand 0.005 property {min}
```

The primary difference between a soft pinching and hard pinching command is the inclusion of a distance constraint (< 0.050 separation 0.240, shown in bold above).

The constraint determines the process failure boundary.

- The separation parameter sets the distance along the contour between pinching tests; it should be set to 3x the minimum target CD. It is used to avoid detecting line ends as a possible pinch point.
- The output_expand value determines the size of the output markers. Smaller markers will show as individual marks on the pinching contours. Larger values create a single block-like marker. Do not use large output_expand values (>250nm), as this may cause errors.

Figure 2-4. pinching Check (Small output_expand Value)



- Specifying a property keyword attaches a value to the marker based on the measured minimum pinching distance found along the contour curve.

Application 2b: Alternative Pinching Check

Building a pinching check using the tolerance method is similar to creating a bridging check using a tolerance zone.

Note

 The method of bridge checks using the **bridge_tolerance** and **cornerchop** commands as described in this section is still supported by Calibre OPCVerify, but is not the recommended method of detecting bridging.

A simulated contour lying inside an inner tolerance band is either pinching or printing inaccurately enough that detection is warranted. Locations where the simulation contour exceeds the tolerance zone are flagged as errors, using the **pinch_tolerance** command.

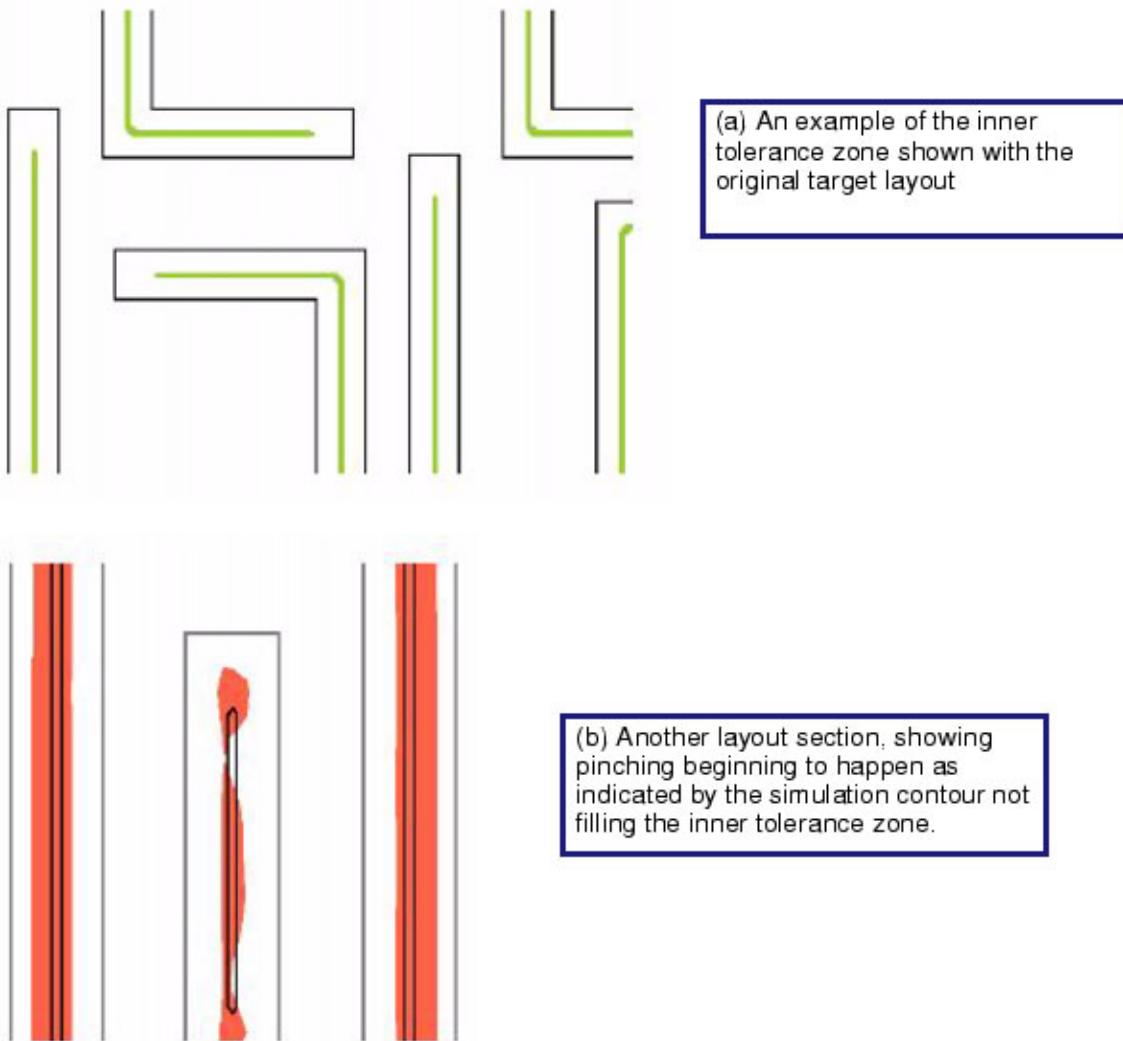
Process Flow

Step 1: Building the Inner Tolerance Zone

First you generate an inner tolerance zone using the following steps:

1. Build an initial tolerance zone using the **build_tolerance** inner command
2. Cornerchop the final tolerance zone using the **cornerchop** command

Figure 2-5. Inner Tolerance Zone for Pinching Areas



The inner tolerance zone generated from this method is shown in [Figure 2-5](#). A properly formed inner tolerance zone will detect all of cases of pinching that occur (no missed errors). However, again it would be possible for a simulation contour to cross a tolerance zone without actually pinching (false error).

In this example, the design size is 130nm. The following OPCverify setup file will generate an inner tolerance.

```
optical_model_load o1 mymodel.opt
background clear
layer SHAPES visible dark
layer ORIG hidden dark
imagegrid 0.040
stairstep 0
setlayer ti1 = build_tolerance inner ORIG max .06 stepsize .01
setlayer ti = cornerchop ti1 .02 .02
```

Step 2: Adding Checks for Pinching Violations

By adding the following lines to the OPCVerify setup file, we perform the tolerance zone check for pinching. The basic flow is:

1. Generate simulation contours for a set of given dose values:

```
setlayer i1 = image optical o1 dose 1.04 aerial_contour 0.3
setlayer i2 = image optical o1 dose 1.08 aerial_contour 0.3
setlayer i3 = image optical o1 dose 1.12 aerial_contour 0.3
setlayer i4 = image optical o1 dose 1.16 aerial_contour 0.3
setlayer i5 = image optical o1 dose 1.20 aerial_contour 0.3
```

2. Check for bridging at each dose:

```
setlayer pinch1 = pinch_tolerance i1 ti
setlayer pinch2 = pinch_tolerance i2 ti
setlayer pinch3 = pinch_tolerance i3 ti
setlayer pinch4 = pinch_tolerance i4 ti
setlayer pinch5 = pinch_tolerance i5 ti
```

3. Add the following code to your SVRF file to output the results:

```
pinch1 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP pinch1
pinch2 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP pinch2
pinch3 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP pinch3
pinch4 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP pinch4
pinch5 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP pinch5
```

Step 3a: Scoring Results Using a Report Summary

The most likely pinching points in the layout are those that pinch closest to best dose (assuming best focus). We can create a distribution of number of failures as a function of dose by reading the DRC summary report file generated by Calibre. The “Result count” shows two numbers, the first is the hierarchical count, the second is the flat count. The “worst” errors are those that show up in the “pinch1” check, because this check is closest to nominal dose. The number of errors monotonically grows for each check, as we increase dose further. Notice in the report file that the error count jumps to large numbers of errors quickly. This jump indicates that the dose range in the check should be reduced to a finer sampling of doses closer to the best dose. Modifying the check in that way can reduce the overwhelming number of errors to allow focus on the worst ones only.

```
RULECHECK pinch1 TOTAL Result Count = 1 (1) // DOSE = 1.04
RULECHECK pinch2 TOTAL Result Count = 184 (184) // DOSE = 1.08
RULECHECK pinch3 TOTAL Result Count = 4130 (4130) // DOSE = 1.12
RULECHECK pinch4 TOTAL Result Count = 7327 (7327) // DOSE = 1.16
RULECHECK pinch5 TOTAL Result Count = 8183 (8183) // DOSE = 1.20
```

Step 3b: Scoring Results Using DFM ANALYZE

Similar to the bridging check example, you can use [DFM ANALYZE](#) to perform “local” pinching scoring, checking pinching in every NxN micron window. The DFM ANALYZE portion of the SVRF is as follows:

```
pinch_score {
    DFM ANALYZE pinch1 pinch2 pinch3 pinch4 pinch5 WINDOW 10
    [COUNT(pinch1)+COUNT(pinch2)+COUNT(pinch3)+COUNT(pinch4)+
    COUNT(pinch5)] > 0
    RDB ONLY pinch.ascii
}
```

Here, DFM ANALYZE creates a score for each 10x10 micron window. If the score is greater than 0, it means at least one pinch occurred in the 10x10 micron window. A higher score indicates more pinching.

Application 3: Gate CD Checking

An important use of Calibre OPCverify is to perform CD checks for gates. You can create a CD gate check using the following flow:

1. Input the active region and mask shapes to OPCverify.
2. Simulate wafer contours at given dose and focus conditions.
3. Perform [gate_stats](#) checks within the active region to measure CD.

For this example, we verify the CD of 62 nm gates, where a double exposure alternating PSM is used to generate the image.

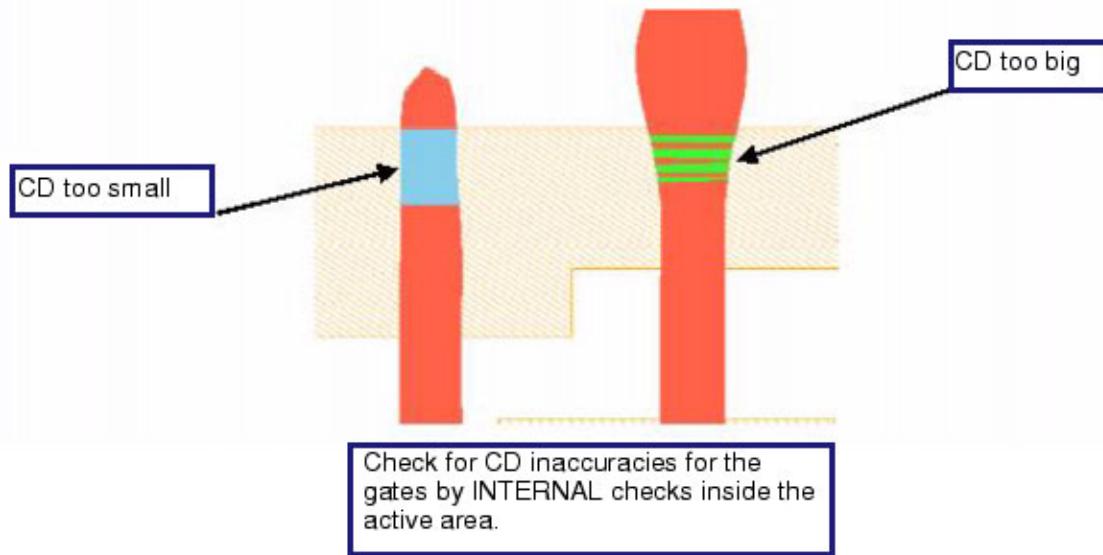
```
optical_model_load o1 trim
optical_model_load o2 psm
background clear dark
layer POLY hidden dark
layer ACTIVE hidden dark
layer TRIM visible dark 0 1.0
layer PH0 visible clear 1 1.0
layer PH180 visible phase180 1 1.0
imagegrid .02
stairstep 0
setlayer i1 = image optical o1 mask1 optical o2 aerial_contour 0.18
setlayer smallCD = gate_stats i1 POLY inside ACTIVE min < .056 cd_max .08 \
cd_min .05 max_extent 0.06
setlayer bigCD = gate_stats i1 POLY inside ACTIVE max < .09 > .068 cd_max \
.10 cd_min .05 max_extent 0.06
```

The “smallCD” check looks for gate regions which have a CD less than 56 nm. The “bigCD” check looks for gates which have a CD greater than 68 nm and less than 90 nm. The outputs of this CD check are shown in [Figure 2-6](#).

Note

 The internal check you specify for the “bigCD” check must be bounded in order for the REGION output to be generated. Non-bounded internal checks are not allowed by OPCverify.

Figure 2-6. Gate Check Inside the Active Area (Process Variation Band)



Application 4: Generating a Process Variation Band

Calibre OPCVerify can quickly simulate contours for variations of several different properties.

- Focus
- Dose
- Mask size (global)
- Relative mask shift (misalignment for double exposure)

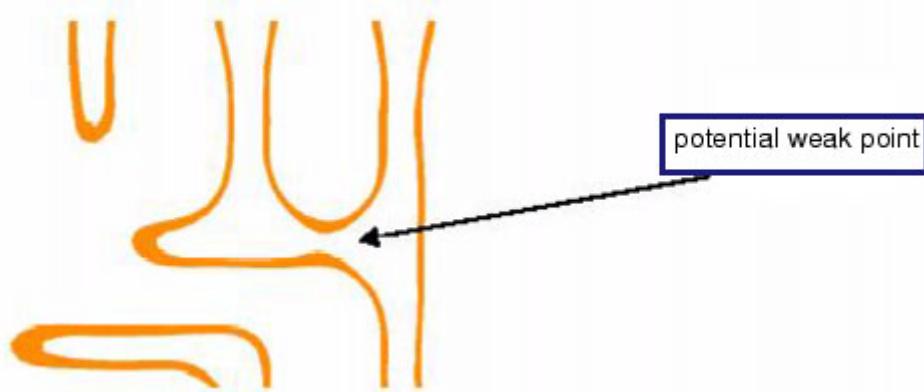
The study of the change of the simulation contours with respect to changing these parameters is what we call a variability check (also sometimes called a sensitivity check). When the sensitivity of the contour is studied with respect to mask sizing only, this is called MEEF calculations. The method of representing possible simulated contours for variations in any of the above parameters is known as the process variation band, or “PV-band.” OPCVerify has a built-in pvbnd calculator (see the [pvbnd](#) syntax page for more information).

In the following setup file, we vary the dose (via threshold) and focus to compute an overall pvband.

```
optical_model_load o1 trim
optical_model_load o2 psm
optical_model_load o3 trimd1
optical_model_load o4 psmd1
optical_model_load o5 trimd2
optical_model_load o6 psmd2
background clear dark
layer POLY visible dark
layer ACTIVE hidden dark
layer TRIM visible dark 0 1.0
layer PH0 visible clear 1 1.0
layer PH180 visible phase180 1 1.0
imagegrid .02
stairstep 0
setlayer i1 = image optical o1 mask1 optical o2 aerial_contour 0.15
setlayer i2 = image optical o1 mask1 optical o2 aerial_contour 0.155
setlayer i3 = image optical o1 mask1 optical o2 aerial_contour 0.16
setlayer i4 = image optical o1 mask1 optical o2 aerial_contour 0.165
setlayer i5 = image optical o1 mask1 optical o2 aerial_contour 0.170
setlayer i6 = image optical o1 mask1 optical o2 aerial_contour 0.180
setlayer i7 = image optical o1 mask1 optical o2 aerial_contour 0.185
setlayer i8 = image optical o3 mask1 optical o4 aerial_contour 0.15
setlayer i9 = image optical o3 mask1 optical o4 aerial_contour 0.155
setlayer i10 = image optical o3 mask1 optical o4 aerial_contour 0.16
setlayer i11 = image optical o3 mask1 optical o4 aerial_contour 0.165
setlayer i12 = image optical o3 mask1 optical o4 aerial_contour 0.170
setlayer i13 = image optical o3 mask1 optical o4 aerial_contour 0.180
setlayer i14 = image optical o3 mask1 optical o4 aerial_contour 0.185
setlayer i15 = image optical o5 mask1 optical o6 aerial_contour 0.15
setlayer i16 = image optical o5 mask1 optical o6 aerial_contour 0.155
setlayer i17 = image optical o5 mask1 optical o6 aerial_contour 0.16
setlayer i18 = image optical o5 mask1 optical o6 aerial_contour 0.165
setlayer i19 = image optical o5 mask1 optical o6 aerial_contour 0.170
setlayer i20 = image optical o5 mask1 optical o6 aerial_contour 0.180
setlayer i21 = image optical o5 mask1 optical o6 aerial_contour 0.185 \
setlayer pv = pvband i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 i11 i12 i13 i14 i15 \
i16 i17 i18 i19 i20 i21
```

In [Figure 2-7](#), the pvband is shown for these variations of exposure conditions. We observe a potential necking issue at some combination of focus and dose.

Figure 2-7. pvbnd Exposure Conditions



By inserting the following call to DFM ANALYZE into the SVRF:

```
DFM ANALYZE pv WINDOW 100 [AREA(pv)] > 0 RDB ONLY pvarea.ascii
```

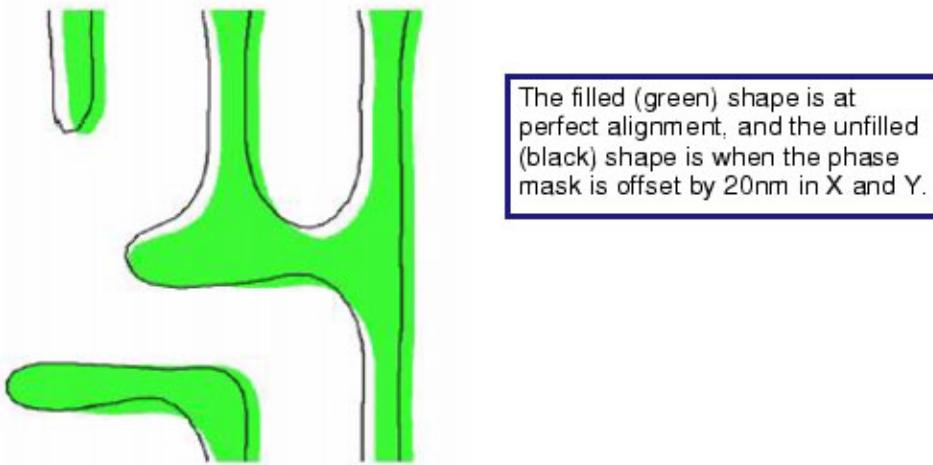
we generate a metric for the overall lithographic manufacturing quality through variations of conditions. In this case, we prefer the OPC settings which minimize the pvbnd area. Alternatively, you can select different OPC settings or PSM design rule settings in order to minimize the pvbnd area.

Application 5: Checking for Misalignment

We can quickly simulate contours for misaligned masks in a double exposure process. This is accomplished with a simple variation of the setup file from the previous example. For brevity, only the setlayer image commands are shown here.

```
setlayer i5 = image optical o1 mask1 optical o2 aerial_contour 0.170
setlayer i5shift = image optical o1 mask1 optical o2 \
xshift .02 yshift .02 aerial_contour 0.170
```

Figure 2-8. Misalignment Check



Application 6: Viewing Idof Gauges in the Process Window Analysis Tool

Calibre OPCVerify contains operations that create process windows. Calibre WORKbench contains a tool that displays process windows. This application shows an example of how to use them together.

- Calibre WORKbench requires a super gauge data file to create a process window display, and a design file loaded to locate the gauges on. In this example application, we create the super gauge data file by using the Calibre OPCVerify [gauges](#) command.
- The gauges are drawn polygons on a separate layer (or paths that are converted to polygons).
- The SVRF rule file must also read the gauges in the design file and pass the gauge layer into the Calibre OPCVerify command file.
- The SVRF rule file can also output the gauges as properties to an RDB file.
- Calibre OPCVerify requires the design file with gauges, and specific commands included in its command file:

- [image_set](#)

The image set should include a nominal (dose 1.0, focus 0.0) contour as part of its defined contour set list.

- [gauges](#)

The gauges command syntax must include the following:

- A specification of the gauge property that matches the gauges in the layout

- The “ldof” property type in the add_properties section
- An output super gauge data filename

The following image_set command creates multiple contours as a group (named “img”) using a step function to vary the dose at three values and the focus at five values.

```
image_set img = full img_pv dose (0.95 1.00 1.05) focus from -100nm to \
100nm step 50
```

It is used in the following gauges command:

```
setlayer gauges_out = gauges gauges_in max_size 0.02 \
add_properties { \
    cdl = ldof poly_opc image_set img auto_cd dose_latitude 0.05 \
    sgd_output output.sgd
}
```

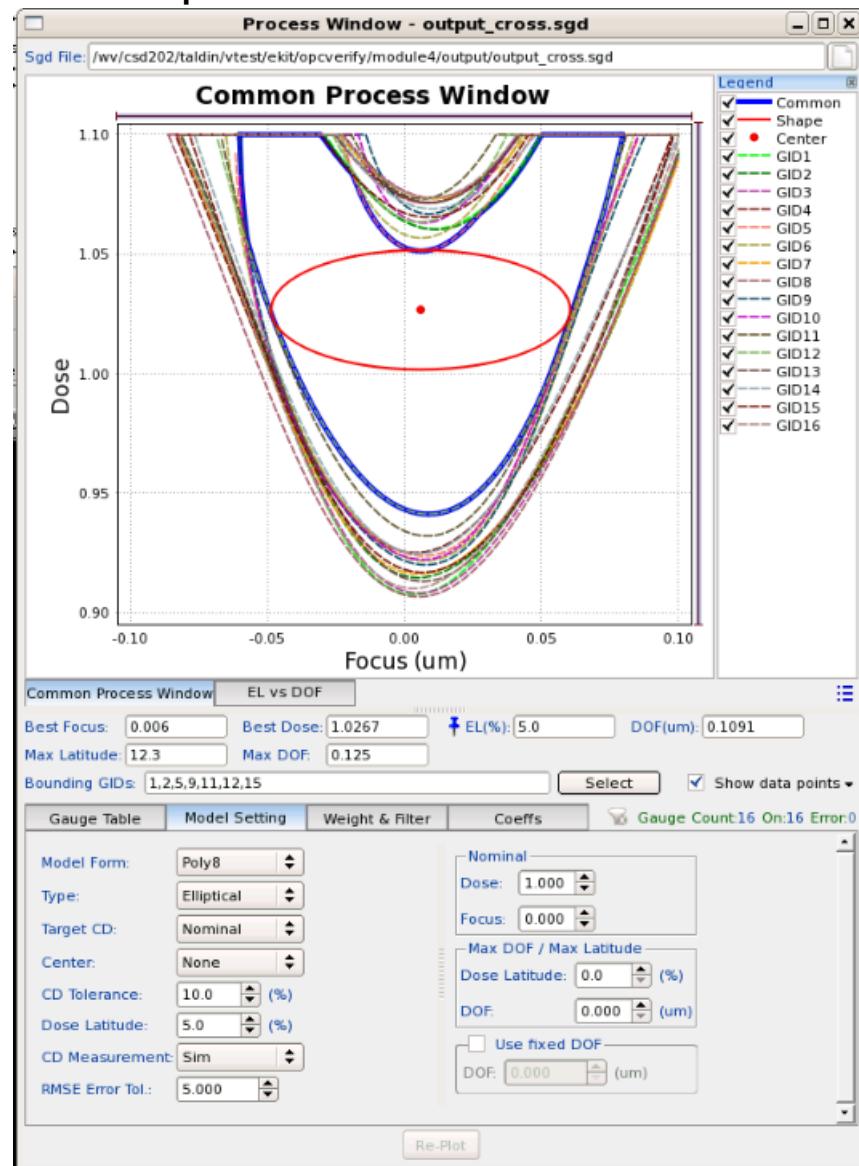
After running the rule file containing these commands, the resulting output files include the following:

- A design file with modified properties written to the gauges
- A super gauge data file containing the contour measurements at each dose and defocus value
- A Calibre RVE ASCII database

To load the gauge data into the Calibre WORKbench Process Window Analysis tool, follow these steps.

1. Invoke Calibre WORKbench.
2. Load the output design file (**File > Open Layout Files**) from the Calibre OPCverify run.
3. Open the PW Plot tool (**Litho> Pw Plot** menu).
4. In the file browser that appears, navigate to the super gauge data output file to open it.

The Process Window appears, showing the ldof values for the gauges.

Figure 2-9. Idof Output Data in the Calibre WORKbench Process Window

Chapter 3

Using Calibre OPCverify

In this chapter, you will learn how to set up and run Calibre OPCverify in batch command mode. Using this chapter requires knowledge of SVRF commands and DRC rule file creation.

Note

 The functions used in this chapter are listed in the “[Calibre OPCverify Function Reference](#)” chapter.

Usage Model For Calibre OPCverify.....	43
Creating a Calibre OPCverify Setup File	44
Defining Layers in the Design.....	46
Creating the SVRF Rule File	49
Concurrency Execution of Multiple Calibre OPCverify Statements in Calibre	52
Topographical Models in Calibre OPCverify.....	56
Topographical Model File Examples	57
Using Topographical Models in Calibre OPCverify (No Litho Model).....	58
Using Topographical Models in Calibre OPCverify (Litho Model).	59
Best Practice: Using Tcl with Calibre OPCverify.....	62
Frequently Asked Questions.....	64

Usage Model For Calibre OPCverify

The usage model for Calibre OPCverify requires you to follow a specific flow.

Usage Model Process Flow

Tip

 Setup file syntax is case-sensitive.

Step 1 - Create a Calibre OPCverify Setup File

You configure Calibre OPCverify from a setup file (or a setup file block inside an SVRF rule file) that you create, using the following steps:

1. Select one or more optical models.
2. (Optional) Select one or more resist models.

3. Define layers in the design for Calibre OPCverify:
 - a. Specify input layers using the layer command.
 - b. Specify Calibre OPCverify and SVRF output layers using the setlayer command.
4. Create rule checks inside the setup file using the setlayer command operations.

Step 2 - Create SVRF Rule File Calls

You run Calibre OPCverify from an SVRF file (also known as a “rule file”) that you create, using the following steps:

1. Define the layers in the design for SVRF for use as input layers.
2. Define one or more LITHO OPCVERIFY calls, deriving them as SVRF output layers from the input layers. The LITHO OPCVERIFY calls use the setup file you created in “Step 1 - Create a Calibre OPCverify Setup File” as their FILE argument.
3. Add one or more DRC CHECK MAP calls to output the derived layers to an ASCII or OASIS®¹ database file.
4. (Optional) Add additional rule checks in the SVRF file, such as using scoring checks or DFM ANALYZE (see “[Scoring Information](#)” on page 523 for more information).
5. Run Calibre on the SVRF rule file.

Creating a Calibre OPCverify Setup File

Calibre OPCverify uses a specific setup file (in ASCII text) that contains a block of configuration parameters and operations that you want Calibre OPCverify to perform. It is not the same as the setup file used by Calibre WORKbench; you will be calling this file exclusively from Calibre OPCverify.

Note

 Arguments for the Calibre OPCverify setup file can be found on the [LITHO OPCVERIFY syntax page](#).

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.

Figure 3-1. Calibre OPCVerify Setup File Example (opcverify.in)

```

modelpath      $env(DESIGN_DIR)/models
optical_model_load    o1 DUV10tab
optical_model_load    o2 DUV10tab_plus1
background clear
layer POLY visible dark
layer TEST hidden dark

imagegrid .04
stairstep 1
#inputs ordered

setlayer ti0 = build_tolerance inner POLY max .16 stepsize .01
setlayer ti = cornerchop ti0 .03 .03
setlayer to0 = build_tolerance outer POLY max .16 stepsize .01
setlayer to = cornerchop to0 .03 .03

setlayer i1 = image optical o1 aerial_contour 0.3
setlayer i2 = image optical o2 aerial_contour 0.3

setlayer b1 = bridge_tolerance i1 ti
setlayer b2 = bridge_tolerance i2 to

```

In this example, two optical models, o1 and o2, are identified for use with Calibre OPCVerify.

Two input layers are declared, POLY and TEST, which are mapped from design layer numbers specified in the SVRF file.

These setlayer commands create Calibre OPCVerify layers.

Notice how the bridge tolerance (b1 and b2) check layers are derived from operations on the generated contour layer (i1, i2) with the generated tolerance layer (ti, to).

Selecting Optical Models

You must supply at least one optical model (either individually or as part of a litho model) to Calibre OPCVerify. Specifying multiple optical models allows you to select them as part of your output layer process window.

Optical models simulate the behavior of the illumination source. Calibre uses this information to calculate the illumination effects on the design.

You can generate an optical model in Calibre WORKbench, using the Optical Model tool.

- Specify optical models with the **optical_model_load** setup file command:

```
optical_model_load model_label filename
```

- You may need to supply a pathname to the **modelpath** setup file command for Calibre to find your optical model director(ies).

An instance where you would use multiple optical models is if you want to vary your defocus settings; you would provide a different optical model for each defocus setting you want to evaluate.

Selecting Resist Models

You only need to supply a resist model to Calibre OPCverify if you want to account for resist and etch effects. Specifying resist models allows you to select them as part of your output layer process window.

Resist models simulate the behavior of true resist and etch effects. Calibre uses this information to calculate the post-illumination effects on the design.

You can generate a resist model in Calibre WORKbench.

- Specify resist models with the **resist_model_load** setup file command:

```
resist_model_load resist_model_label filename
```

- You may need to supply a pathname to the **modelpath** setup file command for Calibre to find your resist model director(ies).

Defining Layers in the Design

A design file consists of one or more named and numbered “layers,” which represent polygons in a topographical fashion. Layers can also be OPC correction layers, SRAF layers, or other non-printing layers (such as for text information).

- The Calibre OPCverify tool requires the identification of one or more of the layers in the design as input layers.
- Calibre OPCverify can create SVRF output on one or more output layers, when used in conjunction with the DRC CHECK MAP SVRF command.
- Calibre OPCverify also uses special layers for its own operations.

You can visually inspect layers (original/input and results/output) in Calibre WORKbench.

Layer Mapping Operations

Layers in the design must be mapped to identify them for processing by Calibre OPCverify.

Mapping Input Layers for Calibre OPCverify

Use the **layer** setup file command to identify the input layers present in your design, such as in the following example:

```
layer mask visible dark
```

- The order you define your layers in is important; each name will later be linked to the layer statement by the positional order that the SVRF rule file LITHO OPCVERIFY calls use.
- The defined layer inputs must match the layers defined in the design that you run Calibre OPCVerify on, or Calibre returns an error.

Note



The layer command you use inside Calibre OPCVerify setup files is a simplified version of the OPCpro layer command, and is not backwards compatible.

Once you define layers, you will be able to later use them to receive input layer arguments from the SVRF file call to Calibre OPCVerify:

```
derived_layer_name = LITHO OPCVERIFY input_layer1 ... input_layerN FILE  
setup MAP output_layer
```

Remember, Calibre OPCVerify uses the input layers in the order given (not their layer number or name in the design file) for its computations.

This mapping-by-order method used by Calibre OPCVerify is an enhancement over other Calibre LITHO products (such as Calibre® OPCpro™, Calibre® ORC™, and Calibre® PRINTimage™), which require the setup file layer names to match the SVRF layer names. The enhanced method allows you to write setup file layer statements using generic names that are not required to match the SVRF layer names passed in to Calibre OPCVerify, which means you can more easily re-use different existing SVRF files with different Calibre OPCVerify setup files.

Example (Mapping Input Layers)

In this example, the Calibre OPCVerify file defines layers named “mask” and “sraf.” However, the SVRF call to LITHO OPCVERIFY sends the input layers named POLY_OPC and SBARS_OPTIMAL to Calibre OPCVerify. Calibre OPCVerify automatically maps the layer POLY_OPC to “mask”, matching the first layer specified in the Calibre OPCVerify file to the first layer sent in the LITHO OPCVERIFY command. Similarly, Calibre maps SBARS_OPTIMAL (the second argument to the LITHO OPCVERIFY command) to “sraf” (the second layer defined in the Calibre OPCVerify file).

The following example code is placed in the Calibre OPCVerify file, *opcverify.in*:

```
optical_model_load m1 optical_model1  
background clear  
layer mask visible dark  
layer sraf visible dark  
setlayer image1 = image optical m1 aerial_contour 0.3
```

In the SVRF file, the following call code is placed:

```
L1 = LITHO OPCVERIFY POLY_OPC SBARS_OPTIMAL FILE opcverify.in MAP image1
```

Mapping Calibre OPCverify Layers and Output Layers

In this section, you will use the setlayer command to create Calibre OPCverify layers, and learn how to designate an output layer.

You must also define the Calibre OPCverify layer and output layers that will be used in your later SVRF call to Calibre OPCverify, which appears in the SVRF file as follows:

```
output_layer = LITHO OPCVERIFY input_layer1 ... input_layerN FILE setup MAP  
OPCverify_layer
```

Calibre OPCverify Layers

Many Calibre OPCverify operations rely on manipulating the input layers using the [setlayer](#) commands. These commands fall into one of three categories:

- The [image](#) command, which generates contour layers from an input layer
 - Use the aerial_contour command to create an aerial image (CTR) simulation. Use this command to perform critical feature detection.
 - Use the resist_model command to create a VT5 simulation. You should only use this command to perform CD checking applications.
- DRC-type commands, which perform geometrical boolean and measurement operations
- Verification commands, which are a number of utility commands to create and check design areas from input and contour layers

Some verification operations can generate error layers; others are designed to be used as intermediate generation steps.

All three types of commands use the same syntax:

```
setlayer OPCverify_layer_name = layer_operation
```

Example (Setlayer Command)

In this example, the [setlayer](#) commands produce Calibre OPCverify layers called “image1” and “image2.” Both of these layers are image type operations which produce an aerial contour layer using the optical model m1, but image2 uses a threshold value of 0.4 instead of 0.3, which will produce a different image simulation.

```
setlayer image1 = image optical m1 aerial_contour 0.3
setlayer image2 = image optical m1 aerial_contour 0.4
```

Output Layers

You can use any Calibre OPCVerify layer created with a **setlayer** command as an output layer in the SVRF rule file. You do not generate output layers directly from the Calibre OPCVerify setup file; instead, you designate output layers as part of the LITHO OPCVERIFY call in the SVRF rule file creation phase.

Creating the SVRF Rule File

You use the SVRF rule file as input to Calibre. These instructions assume that you understand the basic construction of a rule file; the instructions in this section only detail the additional information you will need to run Calibre OPCVerify as a Calibre LITHO command.

Figure 3-2. Calibre OPCVerify SVRF File Example

```
LAYOUT PATH ".tmp/opcverify_i.gds"
LAYOUT PRIMARY "PUB26"
LAYOUT SYSTEM GDSII
PRECISION 1000.0
UNIT LENGTH 1e-06
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE ".tmp/opcverify_o.gds"

LAYER IV2 2
LAYER IVO 0
```

Two design layers are declared, IV2 and IVO, which Calibre OPCVerify maps to its input layers POLY and TEST (declared in the setup file) due to the order sent

```
i1 = LITHO OPCVERIFY FILE ".tmp/opcverify.in"  IV2 IVO MAP i1
i1 {COPY i1} DRC CHECK MAP i1 201
DRC CHECK MAP i1 ASCII "verify_output2.asc"

i2 = LITHO OPCVERIFY FILE ".tmp/opcverify.in"  IV2 IVO MAP i2
i2 {COPY i2} DRC CHECK MAP i2 202
DRC CHECK MAP i2 ASCII "verify_output2.asc"

ti = LITHO OPCVERIFY FILE ".tmp/opcverify.in"  IVO MAP ti
ti {COPY ti} DRC CHECK MAP ti 201
DRC CHECK MAP ti ASCII "verify_output2.asc"

to = LITHO OPCVERIFY FILE ".tmp/opcverify.in"  IVO MAP to
to {COPY to} DRC CHECK MAP to 202
DRC CHECK MAP to ASCII "verify_output2.asc"

b1 = LITHO OPCVERIFY FILE ".tmp/opcverify.in"  IVO MAP b1
b1 {COPY b1} DRC CHECK MAP b1 302
DRC CHECK MAP b1 ASCII "verify_output2.asc"

b2 = LITHO OPCVERIFY FILE ".tmp/opcverify.in"  IVO MAP b2
b2 {COPY b2} DRC CHECK MAP b2 302
DRC CHECK MAP b2 ASCII "verify_output2.asc"
```

These LITHO OPCVERIFY commands create SVRF output layers from Calibre OPCVerify layers.

Each set of lines consists of:

- the derived SVRF layer creation, as a result of the call to Calibre OPCVerify (this line is also where the layer order is specified)
- a line to copy it to the GDS results file layer number
- a line to copy the results to an ASCII results file database

Defining OPCverify Operations in the SVRF

Mapping Design Layers

You define the relationship between the design file and Calibre OPCverify using the LAYER SVRF command:

```
LAYER label layer#
```

The layer label can be anything, as long as it is used consistently in the later calls to LITHO OPCVERIFY.

Note

 The layer numbers must correspond to layers that exist in the design file.

Defining LITHO OPCVERIFY Calls

To run LITHO OPCVERIFY, you construct one or more SVRF output layer statements, using the following syntax:

```
svrf_output_layer = LITHO OPCVERIFY input_layer_list FILE setupfile MAP  
OPCverify_layer_name
```

- *svrf_output_layer* is a unique name you will use if you are planning to create rule checks in SVRF.
- *input_layer_list* is the list of design layer names you are passing to Calibre OPCverify, in the order you want them to be used by Calibre OPCverify.

Remember that Calibre OPCverify maps the inputs (layer commands in the setup file) according to the order you pass them into Calibre OPCverify (LITHO OPCVERIFY commands in the SVRF file), which allows you to re-use existing SVRF rule files without renaming layers in either the rule file or Calibre OPCverify setup file.

Note

 A common error is to pass the layer names into your Calibre OPCverify file in the wrong order.

- *setupfile* is the fully qualified pathname and filename of the Calibre OPCverify file you created earlier.
- *OPCverify_layer_name* is the name of any Calibre OPCverify layer you created inside the setupfile (setlayer command). Each LITHO OPCVERIFY statement can have only one Calibre OPCverify layer mapped to it, as described in the following section.

Note

 After a call to LITHO OPCVERIFY, you can also further manipulate the derived SVRF layer as you would any other SVRF output layer (some Calibre OPCverify output layers may not be suitable for further modification, however).

Understanding Calibre OPCVerify to SVRF Output Layers

You instruct Calibre to map outputs from Calibre OPCVerify into SVRF layers using the MAP command inside each LITHO OPCVERIFY command, in the SVRF rule file.

You can only use the LITHO OPCVERIFY MAP (output) statement in the SVRF file on a Calibre OPCVerify layer you defined in the Calibre OPCVerify setup file, using the **setlayer** Calibre OPCVerify command:

```
setlayer OPCVerify_layer_name = layer_operation
```

Calibre automatically prunes any setlayer commands whose output is not in the dependency graph of outputs selected using MAP from the execution graph. The pruned layers are not run, and do not contribute to the runtime.

In this way, MAP is similar to DRC SELECT CHECK in a rule file.

Example (Output Layers)

In this example, the **setlayer** command produces a Calibre OPCVerify layer called “image1.” Calibre maps this layer to the output SVRF layer L1 using the MAP command on the LITHO OPCVERIFY command.

```
LITHO FILE OPCVerifysetup [
    optical_model_load m1 optical_model1
    background clear
    layer mask visible dark
    layer sraf visible dark
    setlayer image1 = image optical m1 aerial_contour 0.3
    setlayer image2 = image optical m1 aerial_contour 0.4
]
L1 = LITHO OPCVERIFY POLY_OPCT SBARS_OPTIMAL FILE OPCVerifysetup MAP image1
```

Note

 The image simulation for “image2” will be pruned from the graph and therefore not executed, because it is not selected by a MAP command.

If your Calibre OPCVerify layer has properties attached, you must also add a COPY statement to add the layer containing the properties to the layout:

```
L1 {COPY L1} DRC CHECK MAP L1 404
```

Attempting to use a layer with properties without using a COPY statement on it first will result in an error.

Add Database and Output Reporting

Some Calibre OPCVerify commands can output polygons with attached properties as a result of function calls.

- If your output does not contain any property reporting arguments, use the COPY and DRC CHECK MAP commands to output the results of a LITHO OPCVERIFY call to an ASCII (or OASIS) RDB database file.

```
L1 {COPY L1} DRC CHECK MAP L1 202 // copies the output to layer 202
DRC CHECK MAP L1 ASCII "verify_output2.asc" // sends output to a
file
```

- If you want to generate an output file that contains the properties assigned by Calibre OPCverify, you must use a **DFM RDB** statement inside of a rule check to capture the properties.

```
rdb_fetch_property { DFM RDB min_with_props "verify_output2.asc"
NOEMPTY}
```

This example code assumes that you have an Calibre OPCverify result layer “min_with_props” with attached properties inside your Calibre OPCverify setup file.

Concurrency Execution of Multiple Calibre OPCverify Statements in Calibre

Concurrency mechanisms allow Calibre to complete operations faster by taking advantage of operations that can be run independently.

Generating Multiple Output Layers

Calibre OPCverify can generate multiple output layers in a single run, using a concurrency mechanism. The concurrency mechanism is triggered automatically when multiple LITHO OPCVERIFY layer operations are contained in the SVRF call that have exactly the same arguments, but have different MAP outputs.

Example (Multiple Output Layers)

```
L1 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image1
L2 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image2
L3 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image3
```

In this example, the layers L1, L2, and L3 are run concurrently, because the LITHO OPCVERIFY commands match, except for the MAP statements, which point to different Calibre OPCverify layers.

The following Calibre transcript shows that concurrency was successfully recognized. You will notice that several LITHO OPCVERIFY statements are printed, followed by a single interpreter module message, followed by multiple cell and tile completion information lines.

```

OV1 = LITHO OPCVERIFY MAP i1 POLY TEST FILE opcverify.in
OV2 = LITHO OPCVERIFY MAP i2 POLY TEST FILE opcverify.in
OV3 = LITHO OPCVERIFY MAP i3 POLY TEST FILE opcverify.in
OV4 = LITHO OPCVERIFY MAP i4 POLY TEST FILE opcverify.in
-----
----- LITHO OPCVERIFY INTERPRETER MODULE -----
-----
[...]

```

Concurrency Execution of Multiple Setlayer Operations in Calibre OPCVerify

Concurrency mechanisms allow Calibre OPCVerify to complete certain setlayer operations in parallel to reduce runtime.

The following operations support concurrency (see the appropriate syntax page for information):

- [image](#)
- [external](#), [internal](#), and [enclosure](#)
- [measure_cd](#)
- [area_compute](#) and [area_ratio](#)
- [measure_distance](#), [measure_epe](#), and [end_cap](#)
- [gate_stats](#)
- [meefcheck](#)
- [bridge](#), [pinch](#), [extra_printing](#), and [not_printing](#)

Setlayer Operations

Within Calibre OPCVerify itself, concurrency is supported between certain setlayer commands. This means that in some cases, you can compute multiple “setlayer” commands at the same time. In contrast to Calibre’s layer operations, where concurrency of layer generation operations is automatically calculated, the “setlayer” concurrency within Calibre OPCVerify requires you to group together all lines in the setup file which are to be computed concurrently.

To observe which setlayer operations were able to be run concurrently, save the output transcript, and look for the Calibre OPCVerify section. A listing of “CONCURRENT GROUP” statements in the transcript indicates setlayer operations that were grouped for concurrency. This is shown in the following sample transcript.

Sample Transcript

```
----- LITHO OPCVERIFY INTERPRETER MODULE -----
-----
----- LOADING INPUT LAYER 'POLY'
----- LOADING INPUT LAYER 'ORIG'
----- CONCURRENT GROUP
setlayer ti0 = build_tolerance inner ORIG MAX .16 STEPSIZE .01
----- CONCURRENT GROUP
setlayer ti = cornerchop ti0 .03 .03
----- DELETE LAYER 'ti0'
----- CONCURRENT GROUP
setlayer too = build_tolerance outer ORIG MAX 0.16 STEPSIZE 0.01
----- CONCURRENT GROUP
setlayer to = cornerchop too 0.03 0.03
----- DELETE LAYER 'too'
----- CONCURRENT GROUP
setlayer il_1 = image optical o1 aerial_contour 0.16
setlayer il_2 = image optical o1 aerial_contour 0.15
setlayer il_3 = image optical o1 aerial_contour 0.14
. .
setlayer ih_3 = image optical o1 aerial_contour 0.46
setlayer ih_4 = image optical o1 aerial_contour 0.47
5
setlayer ih_5 = image optical o1 aerial_contour 0.48
----- CONCURRENT GROUP
setlayer b_1 = bridge_tolerance ih_1 to
----- CONCURRENT GROUP
. .
----- CONCURRENT GROUP
setlayer p_3 = pinch_tolerance il_3 ti
----- CONCURRENT GROUP
setlayer p_4 = pinch_tolerance il_4 ti
----- CONCURRENT GROUP
setlayer p_5 = pinch_tolerance il_5 ti
```

Concurrency Example - Multiple Dose Calculations

The following example code generates image outputs for multiple focus and dose combinations.

The following code fragment appears in the SVRF file:

```
L1 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image1
L2 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image2
L3 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image3
L4 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image4
L5 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image5
L6 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image6
L7 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image7
L8 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image8
L9 = LITHO OPCVERIFY POLY ASSIST FILE setup MAP image9
```

The following code fragment appears in the setup file *setup.in*:

```

modelpath ./
optical_model_load m1 focus1.opt
optical_model_load m2 focus2.opt
optical_model_load m3 focus_shift.opt
background 1
layer POLY visible
layer ASSIST visible
imagegrid 0.02
stairstep 0
setlayer image1 = image optical m1
setlayer image2 = image optical m1 dose .9
setlayer image3 = image optical m1 dose 1.1
setlayer image4 = image optical m2
setlayer image5 = image optical m2 dose 0.9
setlayer image6 = image optical m2 dose 1.1
setlayer image7 = image optical m3
setlayer image8 = image optical m3 dose 0.9
setlayer image9 = image optical m3 dose 1.1

```

Concurrency Examples - setlayer image Statements

Optical models o1 and o2 have same optical diameter while optical model o3 has different diameter in the following examples.

Example 1

When the rule file has the following setlayer commands:

```

setlayer o1_cm1 = image optical o1 resist_model cm1
setlayer o2_cm1 = image optical o2 resist_model cm1
setlayer o3_cm1 = image optical o3 resist_model cm1
setlayer o1_ac = image optical o1 aerial_contour 0.2
setlayer o2_ac = image optical o2 aerial_contour 0.2
setlayer o3_ac = image optical o3 aerial_contour 0.2
setlayer o1_vt5 = image optical o1 resist_model vt5
setlayer o2_vt5 = image optical o2 resist_model vt5
setlayer o3_vt5 = image optical o3 resist_model vt5

```

The concurrent groups will be (o1_cm1, o2_cm1), (o3_cm1, o3_ac), (o1_ac, o2_ac), (o1_vt5), (o2_vt5), and (o3_vt5).

Example 2

When aerial_contours are grouped with vt5 contours:

```

setlayer o1_ac = image optical o1 aerial_contour 0.2
setlayer o2_ac = image optical o2 aerial_contour 0.2
setlayer o1_vt5 = image optical o1 resist_model vt5
setlayer o2_vt5 = image optical o2 resist_model vt5

```

The concurrent groups will be (o1_ac, o1_vt5) and (o2_ac, o2_vt5).

Topographical Models in Calibre OPCverify

Topographical Models (also referred to as “topo models”) are extended optical models. They incorporate the effects of optical reflections from underlying layer topography. Most commonly, they are used in imaging of implant layers, where the reflective properties of underlying features (silicon, polysilicon) differ significantly from the properties of oxide on field area.

Tip

 These instructions assume that you already have calibrated models (optical, resist, topo, end optionally etch). Detailed information about creating and calibrating topo models is covered in the [Calibre WORKbench Topography Modeling User's and Reference Manual](#), available on Support Center.

This document uses the following layer names:

- “Implant” to mean the layers being imaged.
- “RX” or “Active” to mean the reflective monocrystal (bare) silicon underlayer.
- “PC” or “Poly” to mean the reflective polycrystal silicon underlayer.
- “Finfet” to mean a FinFET underlayer, that usually is a set of monocrystal (bare) silicon strips divided with oxide.

The topo model itself is a control file that contains the recipe for modeling the optical composite including effects near the boundaries between underlying silicon features and field oxide.

Associated with the optical composite are several optical model files, each of which applies to a specific subset of the problem. For example, there can be one optical model for Implant over Oxide, another for Implant over RX, and so on. These models follow the standard optical model format.

The type of configuration you have adds steps to the task of loading optical models into Calibre OPCverify. Additional density models may be included for both single or multiple density configurations. In most cases, single density is used for topo modeling, but there are cases where multiple density optical models are required.

- Pre-PC configurations are typically single density, and have only silicon and oxide stacks.
- Post-PC configurations might include a poly optical model.
 - If a Poly optical model is not included, then single density simulations have to be carried out with “interconnect” or “gate” topo signals.
 - If a Poly optical model is included, then multiple density simulations must be performed. The multiple density simulations are recommended for post-PC

configurations because they better take into account the optical properties of a poly layer stack.

- Finfet configurations include a separate film stack for the film composition in the optical model. The optical model for Finfet can be included into a stack-transitional optical model, or it can be separated in its own model.

Discovering the kind of topo model you have (single density versus multiple density) requires you to view the topo model in a text editor.

Look for the following lines in the topo model:

- **transit** — Indicates which optical models should be the Implant optical models and relevant layers.
- **from *umin* to *umax*** — Specifies the minimum and maximum limits for density exposure values that are used to combine transit exposure images into one composite aerial image.
- **density [exposure | underlying]** — Indicates exposures or underlying layers that are used for density calculations. Single-density exposure situations might omit the density keyword.

The type of density keyword in your topo model provides a reference for which layers and optical models should be configured as density layers.

Topographical Model File Examples	57
Using Topographical Models in Calibre OPCverify (No Litho Model)	58
Using Topographical Models in Calibre OPCverify (Litho Model)	59

Topographical Model File Examples

Use the correct syntax for topographical model files in order to get the best results.

Examples

Example 1 (Single Density)

If your topo model file contains these lines:

```
transit 0 1
```

Your Calibre OPCverify code requires a total of 2 masks.

- Implant layers must be assigned to masks 0 and 1.
- Mask 0 must have the oxide optical model assigned to it.
- Mask 1 must have the RX-silicon optical model assigned to it.

Example 2 (Multiple Density)

If your topo model file contains these lines:

```
transit 0 1 2
density exposure 3 -3 4
```

Your Calibre OPCverify code requires a total of 5 masks.

- Implant layers must be assigned to masks 0, 1, 2, 3, and 4.
- Mask 0 has the oxide optical model assigned to it.
- Mask 1 has the RX-silicon optical model assigned to it.
- The density exposure notation “3” corresponds to RX-silicon transit exposure #0.
- The density exposure notation “-3” corresponds to field transit exposure #1.
- The density exposure notation “4” corresponds to transit exposure #2.

An alternative density notation uses “density underlying” instead of “density exposure” and designates the layers used for underlying exposure calculations by name instead of by number.

```
transit 0 1 2
density underlying RX -RX POLY
```

Related Topics

[Using Topographical Models in Calibre OPCverify \(No Litho Model\)](#)

[Using Topographical Models in Calibre OPCverify \(Litho Model\)](#)

Using Topographical Models in Calibre OPCverify (No Litho Model)

Calibre OPCverify setup files must be modified to include awareness of topo models. The instructions assume that you have *not* created a litho model.

The primary changes for topo modeling are the topo_model_load command and a modified image command.

Prerequisites

- Calibrated optical, topo, and resist models. These should have been provided by your modeling team.
- A Calibre OPCverify setup file.

Procedure

1. Edit the Calibre OPCverify setup file, setting the modelpath and declaring layers.

```
modelpath ./models
layer Implant
layer RX
```

2. Load the topo model using the `topo_model_load` command and assign it an alias (in the following code, “topo_mdl” is the filename and “TOPO” is the alias used for that filename).

```
topo_model_load TOPO topo_mdl
```

A topo model can also be specified with an inline defined model inside curly braces:

```
topo_model_load alias {definition}
```

When you have two transit exposures (oxide and silicon) in the topo model, you must specify the transit exposure numbers using the “transit” keyword. In single density situations, the transit exposure for oxide optics must always have the number 0.

```
transit 0 1
# 0 - oxide exposure
# 1 - silicon exposure
```

3. Load all optical and resist models and assign them an alias. For single-density setups, you typically only need two optical models and a resist model.

```
/* single-density example */
optical_model_load OXIDE oxide_opt
optical_model_load SILICON silicon_opt
resist_model_load RESIST resist_model_file
```

4. Code `image` statements to use the topo model and to explicitly associate the optical models with the layers as mask layers, the underlying layer type, and the underlying layer name.

In this example, the underlying layer is layer RX. It is mapped using the “underlying *type* *layername*” keyword set, where *type* is “active”, “poly”, or “finfet”.

```
/* single-density example */
setlayer topo_single = image \
mask0 optical OXIDE background dark layer Implant 1.0 0.0 \
mask1 optical SILICON background dark layer Implant 1.0 0.0 \
underlying active RX \
topo_model TOPO \
resist_model resist_model_file
...
```

Using Topographical Models in Calibre OPCverify (Litho Model)

Modifying Calibre OPCverify code to include awareness of topo models contained in a litho model requires several steps to work with the pointers inside the litho model.

The primary tasks to configure Calibre OPCverify for topo modeling include the [topo_model_load](#) command and a modified image command.

Prerequisites

- Calibrated optical, topo, and resist models, or a litho model containing them. They must be located in a single directory or in a location that can be symbolically linked to.
- A Calibre OPCverify setup file.

Procedure

1. In a text editor, edit the *Lithomodel* file, or if you do not have a litho model, create a new text file named *Lithomodel* in the models directory, and enter the following lines:

Table 3-1. Litho Model, Including Topo Model Syntax

Item	Description	Suggested Value
version 1	Sets the version of the litho model. This must be the first line in the model, and currently only version 1 has been released.	1
resist <i>resist_model</i>	Sets the resist model.	The calibrated CM1 resist model name.
topo_model <i>topo_model_file</i>	Sets the topo model file. This topo model is associated with all masks in the litho model.	The calibrated topo model.

2. Add a definition block for mask 0, enclosed by braces, similar to this example:

```
mask 0 {  
    background dark  
    optical models/oxide_opt  
    mask_layer 0 TRANS dark CATEGORY oxide  
}
```

This mask block corresponds to the exposure for the oxide-only part of the film.

3. Add a definition block for mask 1, enclosed by braces, containing the following values:

```
mask 1 {  
    background dark  
    optical models/silicon_opt  
    mask_layer 0 TRANS dark CATEGORY silicon  
}
```

This mask block corresponds to the exposure for the silicon-only part of the film.

Note

 The background and mask_layer 0 parameters must be identical in every mask definition block, except for the CATEGORY keyword. Furthermore, the background for the RX layer must always be “dark” according to topo modeling conventions.

4. (Optional) Users with multiple density configurations need to add more masks to account for the additional models.
5. Save the *Lithomodel* file.
6. Edit the Calibre OPCverify setup file, setting the modelpath and layers.
7. Create an image_options block that has the following items:
 - A litho_model statement to indicate the directory containing the Lithomodel file.
 - Underlying layers using the “underlying” keyword to the layer declaration along with the type (active, poly, or finfet).
 - A mask number if you are using multiple underlying density layers. The mask number comes from the litho model. (See the Examples section.)

```
image_options A {  
    layer Implant visible mask 0 mask 1  
    layer Active underlying active  
    litho_model models_directory  
}
```

8. Code **image** statements to include the image options block:

```
setlayer cntr = image A
```

9. Code other Calibre OPCverify checks as needed.
10. Save the file and run it as normal.

Examples

The following example shows a Calibre OPCverify command file implementing a topo model using litho models.

```

modelpath .
layer implant
layer rx
layer pc
layer finfet
processing_mode flat
mask_sample_grid rsm
optical_transform_size 768
imagegrid aerial 3 8
image_options opts {
    litho_model lithoMod
    layer implant visible mask 0 mask 1 mask 2 mask 3 mask 4
    layer rx underlying active
    layer pc underlying poly
    layer finfet underlying finfet
}
setlayer contour = image opts

```

The corresponding litho model is defined as follows:

```

version 1
resist resist.mod
topo topo.mod
mask 0 {
    optical ox_opt
    background dark
    mask_layer 0 TRANS clear
}
mask 1 {
    optical si_opt
    background dark
    mask_layer 0 TRANS clear
}
mask 2 {
    optical pc_opt
    background dark
    mask_layer 0 TRANS clear
}
mask 3 {
    optical dense1_opt
    background dark
    mask_layer 0 TRANS clear
}
mask 4 {
    optical dense2_opt
    background dark
    mask_layer 0 TRANS clear
}

```

Best Practice: Using Tcl with Calibre OPCverify

The Calibre OPCverify setup file supports Tcl script constructs. You can use conditional and looping Tcl code to simplify the setup and SVRF code for Calibre OPCverify. All lines in the

setup file are evaluated at the beginning of the Calibre OPCverify job for Tcl code, and are converted to the raw commands.

Examples

Varying the Dose

For example, the following code in the setup file:

```
-----
OPCverify setup file code:
-----
set cnt 1
foreach dose {0.96 0.92 0.88 0.84 0.80} {
    setlayer i${cnt} = image optical o1 aerial_contour .3 dose $dose
    incr cnt
}
```

Will translate to:

```
setlayer i1 = image optical o1 aerial_contour 0.3 dose 0.96
setlayer i2 = image optical o1 aerial_contour 0.3 dose 0.92
setlayer i3 = image optical o1 aerial_contour 0.3 dose 0.88
setlayer i4 = image optical o1 aerial_contour 0.3 dose 0.84
setlayer i5 = image optical o1 aerial_contour 0.3 dose 0.80
```

When you use TVF (Tcl Verification Format) in the rule file instead of SVRF, you can insert concurrent calls to LITHO OPCVERIFY into a Tcl loop. This coordinated use of Tcl in the setup file and in the rule file (TVF) can greatly enhance the ease-of-use of verification scripts.

```
-----
TVF code:
-----
set output 101
foreach val { ti to i1 i2 i3 i4 i5 i6 smallCD bigCD} {
    SETLAYER $val "LITHO OPCVERIFY FILE opcverify.in POLY MASK MAP $val"
    RULECHECK $val "COPY $val"
    DRC CHECK MAP $val $output
    incr output}
```

Varying the Tolerance

Another effective example use of Tcl looping code is with the [measure_cd](#) command. Creating a variation using the tolerance values as the loop variable creates a concurrently executable command block that can be used in a binning test, as shown in this fragment:

```
setlayer contour = image optical f0 resist_model vt5
    set startCD 0.080
    set stepCD 0.002
    set numsteps 15
    for {set i 0} {$i <=$numsteps} {incr i} {
        set hibin [expr $startCD + ($i+1)*$stepCD]
        set lobin [expr $startCD + $i*$stepCD]
        setlayer cd$lobin$hibin = measure_cd contour target_layer internal \
            inside island_layer \
            cd_min 0.09 cd_max 0.110 cd_min_length 0.001 max_search 0.03 \
            tol >= $lobin < $hibin absolute cdeffective
    }
```

Frequently Asked Questions

The following questions and answers may be useful to new users of Calibre OPCverify.

Q: Why are my simulated results the opposite of what I expect, when I am using positive tone mask input data (also known as “fractured output” or “mask-ready data”)?

A: Calibre OPCverify assumes that the polygons in the layout represent the regions where you want the resist to appear. Because of this, the simulation needs to be run on the final mask shapes, instead of using the input to the mask. To use positive tone masks, you must calculate for the mask-input inversion either before or after you run Calibre OPCverify. Simply switching the layer and background illumination types is not sufficient.

Q: Why do I sometimes see discrepancies between dense printimage and Calibre OPCverify contours?

A: Simulation contours created by the [image](#) command can show a high degree of variability in the areas where bridging or pinching are about to take place under given simulation conditions. Depending on your grid size, the dense printimage selection grid origin point may not match the automatically selected Calibre OPCverify grid. This can cause apparent differences between dense printimage results and Calibre OPCverify results to be shown. However, both dense printimage and Calibre OPCverify use the same simulation engine, and the detected results status (pinching, bridging, and so on) will always be consistent.

Reducing the grid size (in the Calibre OPCverify imagegrid parameter) will generally reduce the size of differences between dense printimage and Calibre OPCverify simulation results, but will take longer to calculate.

Q: Can I use the contours generated by Calibre OPCverify outside of Calibre OPCverify for DRC operations?

A: Siemens EDA does not recommend output of the full chip simulations created using the **image** command directly to a GDS or OASIS file, as this can cause noticeable performance issues for large designs. You can output clips of the layout using the **window** and **output_window** commands.

Q: Why do the commands with “property” settings have a function argument? What are the property settings for?

A: You use the function argument to get a filtered subset of the input layer based on desired constraint criteria. You optionally assign properties and associated values by adding the property block to the command; Calibre OPCVerify calculates the requested properties only for the shapes collected by the initial function selector.

For example, `measure_epe` has function choices of average, min, max, and exception. The following code uses “function min” to filter out shapes that have a minimum EPE of greater than 0.02 nm (the constraint) from all the shapes on the input layer. Each of those shapes then have both their min and max EPE values recorded and added as properties.

```
measure_epe
...
function min
...
only >0.02
property {
min
max
}
```

Q: Properties must be specified ‘one per line’ according to the command descriptions. Does that mean I have to write a separate setlayer line for each property I want to apply?

A: No. Each property label is a keyword that must be given its own line within the property block of a single setlayer command. For example:

```
measure_epe contour1 targetA ...
property {
min
max
min_ratio
}
```

Remember that the braces ({}) must enclose the property command list. Each shape returned by the `measure_epe` command is given the specified properties and associated values calculated for the object. In the code above, any shapes that matched the specified `measure_epe` criteria have their min, max, and min_ratio values calculated and attached to the shape.

Chapter 4

Calibre OPCverify Function Reference

Calibre OPCverify is a command language that has an extensive set of setup commands and operations.

Constraints	67
Error-Centric Section Blocks	69
Using LITHO OPCVERIFY	104
LITHO OPCVERIFY	106
Litho Model Format	111
ZPlanes Model Format	118
EUV Through-Slit Litho Model Extension	121
RET INPUT	126
Lint Warnings	130
Calibre OPCverify Setup File Configuration Commands	133
Using Setlayer Options as Operations	252
Image Operations	253
DRC-type Operations	253
Verification Control Operations	254
Setlayer Operations Reference	256

Constraints

Certain layer operations are measurement-oriented and therefore carry constraints. Constraints are intervals of non-negative real numbers; input data that falls within the specified constraint of an operation is generally output data.

The syntax for constraints uses one of the six keywords (operators) shown in Table 4-1.

Table 4-1. Constraints

Operators	Constraint Notation	Alternate Constraint Notation	Mathematical Notation
<	< a	NA	x < a
>	> a	NA	x > a
<=	<= a	NA	x <= a

Table 4-1. Constraints (cont.)

Operators	Constraint Notation	Alternate Constraint Notation	Mathematical Notation
\geq	$\geq a$	NA	$x \geq a$
\equiv	$\equiv a$	NA	$x = a$
\neq	$\neq a$	NA	$x \neq a$
$>$ and $<$	$> a < b$	$< b > a$	$a < x < b$
\geq and $<$	$\geq a < b$	$< b \geq a$	$a < x \leq b$
$>$ and \leq	$> a \leq b$	$\leq b > a$	$a < x \leq b$
\geq and \leq	$\geq a \leq b$	$\leq b \geq a$	$a \leq x \leq b$

Notice that the syntax for the last four forms is simply a combination of that for the first four forms. In most cases, $a \geq 0$ and $a < b$. The constraint “ < 0 ” is not allowed because strictly negative constraint values are not possible in SVRF operations. The constraint “ ≤ 0 ” is permitted.

Not all operations accommodate all types of constraints or all values of the numbers a and b . For example, setting $a=0$ in the following constraint is not valid for use with the With Width operation, because a polygon with zero width is not possible for this operation.

```
WITH WIDTH layer1  $\geq a < b$  // not valid
```

Restrictions are discussed, when applicable, with the description of the operation.

As an example, the constraint < 4 denotes all non-negative numbers less than 4, and the constraint $\geq 5 < 7$ denotes all numbers greater than or equal to 5 and less than 7.

Some OPCverify operations have a ratio mode that can be specified; when constraints are specified with a ratio mode, the second value is used as a ratio of the first, using 1.0 as “equal”. For example, the constraint “ < 1.5 ” is read as “50 percent greater than.”

Error-Centric Section Blocks

Several Calibre OPCVerify commands can write out properties to an RDB database. Results of such OPCVerify checks that have this additional information are viewable in Calibre RVE. Command parameters that control the output of these properties are placed in the Error-Centric section after the other command parameters.

Error-Centric Block Types and Rules	69
Property Block	70
Writing Properties to Outputs	71
Classification Block	73
Limit Block	85
Histogram Block	87
add_properties Block	91
Pinpoint Output Block	102

Error-Centric Block Types and Rules

The error-centric section consists of one or more ‘blocks’, each of which is self-contained using braces ({}).

Error-centric section blocks come in the following types:

- [Property Block](#) — Calculates properties relevant to the command.
- [Classification Block](#) — Reduces the number of error markers returned using the geometry of the error shapes.
- [Limit Block](#) — Reduces the number of error markers returned using the property value.
- [Histogram Block](#) — Returns files with properties in tabular data form for charting purposes in Calibre RVE.
- [add_properties Block](#) — Computes additional property values based on the error location, with the capability of applying additional input parameters. Only available on a subset of commands that also support property blocks.
- [Pinpoint Output Block](#) — Changes the shape of an error marker. Only available on a subset of commands that also support property blocks.

Error-centric section blocks must obey the following rules:

- Error-centric sections must appear last, after all other command parameters have been specified.
- If an error-centric section is defined at all, it must include a [Property Block](#). All other error-centric blocks are optional.

- Each error-centric section block must be closed (with a closing brace) before starting another (they cannot be nested). However, when using two or more error-centric blocks, second and subsequent block declarations cannot start a new line (but they can be on the same line).

Correct:

```
property {
  ...
} classify {
  ...
}
```

Correct:

```
property {...} classify {...}
```

Incorrect:

```
property {...}
classify {...}
```

- Error-centric section blocks can be defined in any order within the error-centric section of the command.
- The limit, classification, and histogram blocks are mutually exclusive with each other.

Property Block

Rule File Construct

Calibre OPCVerify **properties** are floating point numbers representing additional information attached to error polygons.

Note

 Properties created using Calibre OPCVerify are not the same as GDS or OASIS properties that are stored in the design. Calibre OPCVerify property output must also be explicitly coded into your SVRF rule file. See the section “[Writing Properties to Outputs](#)” on page 71 for more information.

By default, error polygons simply mark a location. Use property blocks to define which properties are reported with the error markers. For example, suppose a **measure_cd** command returns 127 result polygons. If the **measure_cd** command was specified with the max, min, and average properties, each returned polygon would have three properties attached to it with the associated values.

Some commands have multiple property functions available. Only properties that are explicitly requested are attached.

The full syntax notation for a property block is as follows:

```
setlayer derived = function_name .... [property '{' propfunction1
[propfunction2]
[propfunction3]
... # more properties as needed
'}'] # closing brace
```

Properties require a specially formatted syntax block with the following rules:

1. The property keyword must be on the same line as the command that calls it and be followed by a left brace ({). For example:

```
setlayer derived = measure_cd .... property {
```

2. After the left brace, specify one or more property function calls. The first call can occupy the same line as the property keyword, but each additional optional property function must be on a new line. Use a right brace (}) to close the list.

Correct:

```
setlayer derived = measure_cd .... property { max
min
average
}
```

Incorrect:

```
setlayer derived = measure_cd .... property { max min
}
```

Writing Properties to Outputs

If a Calibre OPCverify output layer has a properties block, then a standard call to Calibre OPCverify returns a derived layer with properties. These properties can be manipulated with DFM PROPERTY commands, saved in an OASIS or a GDSII file with a DRC CHECK MAP command, or written to an RDB file with a DFM RDB command.

Examples

Example 1 (No Limits)

If your SVRF rule file contains the following Calibre OPCverify call:

```
cdr = LITHO OPCVERIFY FILE "setup/check.in" popc target MAP cdr
cdr { copy cdr } DRC CHECK MAP cdr 111 0
```

and your OPCverify command file contains the example code:

```
setlayer cdr = measure_cd contour target internal cd_min 0.11 \
cd_max 0.16 cd_min_length 0.01 max_search 0.06 tol not < 1.1 \
ratio property {max
min}
```

You can save both “cdr” geometries and their properties in an OASIS or GDSII file using the following command:

```
cdr { COPY cdr } DRC CHECK MAP cdr 42 PROPERTIES
```

Assuming that the derived layer “cdr” had properties returned, the following code creates the corresponding RDB database file with the properties information.

```
cdr_max { DFM RDB cdr "cdr_max.asc" MAXIMUM ALL ALL CELLS NOEMPTY }
```

The following code demonstrates the use of the DFM PROPERTY keyword to narrow the results to only results with a property “max” value less than a specified value:

```
cdr_n034 = DFM PROPERTY cdr [max_n034 = PROPERTY(cdr,max) ] <= -0.034
cdr_n034 = { COPY cdr_n034 } DRC CHECK MAP cdr_n034 43 PROPERTIES
```

Example 2 (Limits)

Using the same example SVRF rule file as Example 1, but adding limits to the OPCverify rule check as follows:

```
setlayer cdr = measure_cd contour target internal cd_min 0.11 cd_max 0.16
cd_min_length 0.01 max_search 0.06 tol not < 1.1 ratio property {max
min
} limit {
score property min smallest
worst 2
}
```

results in only two error bins being returned in the database file.

Classification Block

Rule File Construct

An OPCVerify classification block is an extension to some of the Calibre OPCVerify commands, allowing you to classify the filtered results of the command. It can be used in conjunction with a property function block, but is mutually exclusive with histogram and limit blocks.

Usage

setlayer *command*

```
...
[property '{' property_block
'}'] classify [classify_name]
'{'
context layer1 [layer2 ... layerN]
halo halo_microns [around {center | extent}]
[coarse_match cm_microns]
[unique_unclassified_ids {off | on}]
[maxsize ms_microns]
[maximum_error_number max_errors]
[suppress | keep | keep_no_context} duplicates]
[anchor alayer [anchor_max_snap ams_microns] [anchor_halo ah_microns]]
[reflections_rotations_match {decide | yes | no | reflections}]
[select_errors {lower_left | random}]
[[score [bin_size score_delta] [property propname] [smallest | largest]
[worst [unique | total] count [{duplicates | per_class} dup_count]]]
[pm_classify max_search value
    max_length value [match_tolerance value] [orig_runlength]]]
'{'
```

Description

Classification blocks can be specified for most setlayer checks (refer to the documentation for each setlayer check).

Note

 The command “setlayer x = copy y classify {..}” syntax allows you to classify any setlayer in the deck.

A centerpoint-style classification is used for errors; Calibre OPCVerify finds the center of the error and uses that as the center for the haloed context. Adding the “around extent” to the **halo** argument uses the halo area around the error marker as the extent.

When the classification process is complete, the result information is attached to output polygons as the properties CLASS_FLAT_COUNT and CLASS_ID.

- For unique errors, CLASS_FLAT_COUNT encodes the flat count of instances of this particular error including the duplicates and the unique itself.
- For duplicate errors, CLASS_FLAT_COUNT is set to 0.
- For unclassified errors, CLASS_FLAT_COUNT is set to the total placement count of the pseudo cell that unclassified errors are in.

If the keep or keep_no_context duplicates option was specified, CLASS_ID contains the IDs of errors.

- Duplicate errors have their CLASS_ID set to the ID of the corresponding unique error.
- Unclassified errors have their CLASS_ID set to 0.

MTFlex and TURBOflex May Return Different Classification Results

If you are working with both Calibre® MTflex™ and Calibre TURBOflex solutions, you may notice slightly different error counts and outputs when run on the same design. These are a result of slight differences between the way MTflex and TURBOflex handle tiling, and have the following known effects:

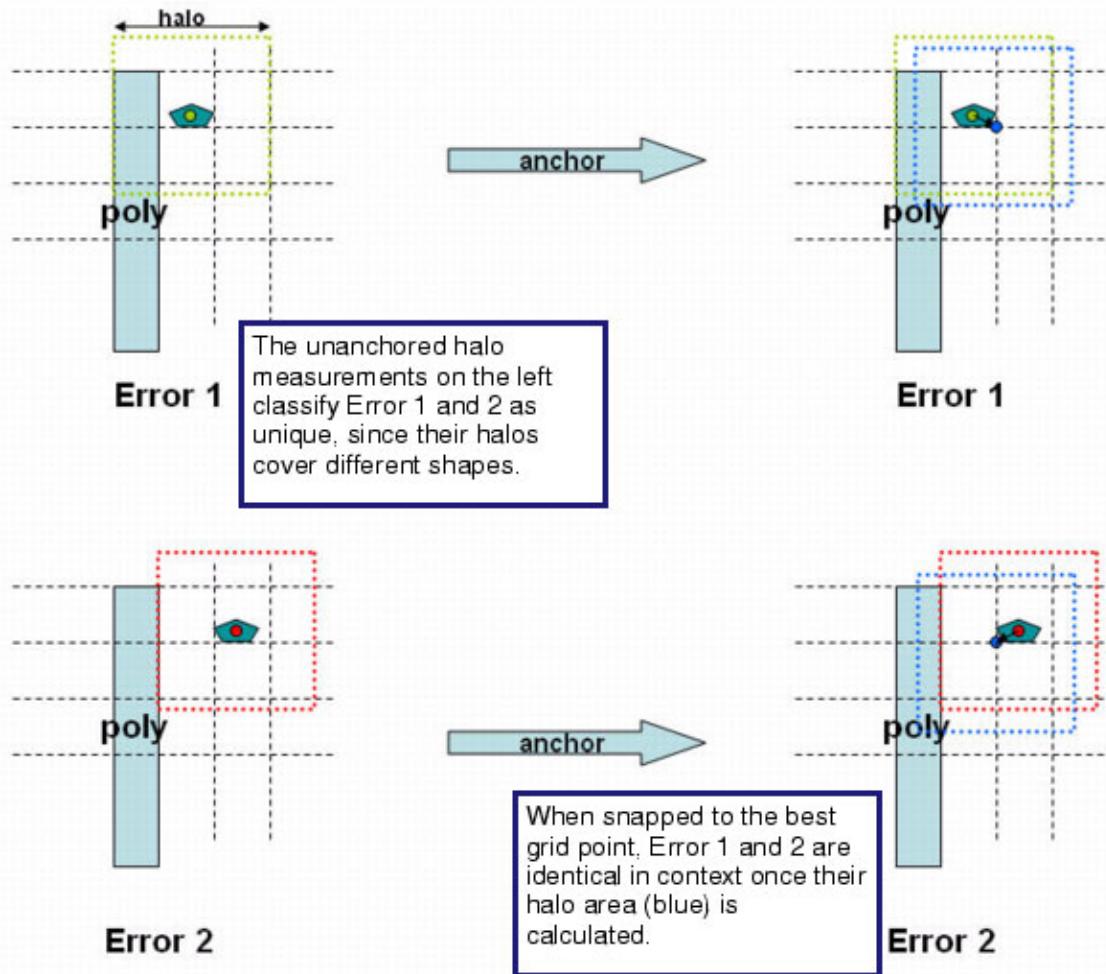
- **Limiting (when using the score keyword)** — The score threshold is updated as each tile is processed. The score threshold (and the screening done based on the threshold) depends which tile finishes first. The tile processing order can vary from run to run in MTFlex. The tile processing order will also change due to TURBOflex operations.
- **Different locations of unique classified returned results** — All unique classified results are found in order of the deepest cell in hierarchy in the lowest X,Y order. TURBOflex alters the cell hierarchy, which can cause duplicate classified shapes to be located in different cells. If the lowest (cell,X,Y) unique shape is moved into a cell higher in the cell hierarchy, a different unique-shape location from another cell will be returned instead. This means that the locations returned by a non-TURBOflex run may be different from the locations returned by a TURBOflex run.
- **Different flat counts of unique classified shapes** — Because TURBOflex alters the cell hierarchy, the cell placement count may be different. A change in the cell placement count changes the cell flat count.
- **Subtle contour differences due to different cell tiling and tile processing order** — OPC results for tiles are dependent on the processing order of the adjacent tiles. The OPC'd tile result is influenced by adjacent OPC'd tile results. Because TURBOflex can change the OPC tile processing order, formerly adjacent tiles can be either pushed up or down the cell hierarchy, thereby changing the contours. In addition, the TURBOflex restructured cells will have different tile boundaries.

Anchoring Mode

Anchoring is a process where each error's center is first snapped to a grid before the uniqueness comparisons are made. The grid is drawn based on the closest polygon vertex in context; the size of the grid is equal to $2 * \text{anchor_max_snap}$ option.

Anchoring mode makes classification less sensitive to the location of an error's center point and reduces the number of unique errors (see [Figure 4-1](#)). Use the drawn (target) layer as the anchor layer, and the `anchor_layer` keyword to activate this mode.

Figure 4-1. Anchoring



Usage Notes

- A `setlayer` operation with a limit or classify block can be used as input to another `setlayer` operation. However, it is pre-limited and pre-classified geometry that is fed into another `setlayer` operation.
- At most 4 billion errors can be handled by the classification code.

- Calibre OPCVerify classify produces slightly different results even when there are no unclassified errors in Calibre OPCVerify (due to the maxsize limit being reached) and anchoring is not used. The reason is a difference in handling of errors whose extents are odd in 1 or 2 dimensions.
- In a hierarchical Calibre run, a unique error is the one with the smallest ordered triplet of (Calibre-internal cell id (the 'bottom-most' cell), the error's smallest x coordinate, and the error's smallest y coordinate). However, due to hierarchy-injection, cell numbers might not be consistent from run to run. Use the following command:

```
setenv CALIBRE_DISABLE_MT_INJECTION on
```

to disable this source of unique error inconsistency (and get longer HDB constructor run time).

- If *classify_name* is specified, there must be a [classify_options](#) statement in the setup section defined with that name. A [classify_options](#) statement also allows you to skip required arguments if they are defined in the *classify_options* block. Arguments set in an individual *classify* block overwrite the *classify_options* block setting.

Arguments

- **classify** [*classify_name*]

A required argument declaring the start of a command's *classify* block. All arguments must be contained within a **classify** or **classify** *classify_name* block, enclosed by braces ({}).

- **context** *layer1* [*layer2* ... *layerN*]

An argument specifying at least one context layer to use to measure nearby polygons for contextual similarities. Layers must be Calibre OPCVerify input layers.

- The context argument is required for a *classify* block, unless the *classify* block is used with a command listed in [Table 4-2](#).
- For the commands listed in [Table 4-2](#), the context *layer* values used are taken from specific layer arguments to the command when no context argument is specified in the classification block.

For example, specifying a *classify* block without a context argument for the [measure_cd](#) command uses the input *ref_layer* argument as its **context** *layer1* argument.

Table 4-2. Default Context Layers for Classification

Command	Context Layer(s) Used
bridge , pinch , not_printing , extra_printing	<i>target_layer</i>
measure_cd	<i>ref_layer</i>
measure_cdv , measure_epe	<i>layer_target</i>
end_cap	<i>layer_active</i>

Table 4-2. Default Context Layers for Classification (cont.)

Command	Context Layer(s) Used
<code>area_ratio</code>	<code>ref_layer</code>
<code>gate_stats</code>	<code>layer_poly_target</code> and <code>layer_active</code>
<code>dofcheck, contour_diff, meefcheck, nilscheck, pwcheck</code>	<code>layer_target</code>
<code>copy</code>	<code>layer</code>

- **halo *halo_microns* [around {center | extent}]**

Note

 The halo keyword must appear on a new line in order for it not to be interpreted as a context layer argument.

A required argument that specifies a halo distance around an error used as the classification region for each error. A square of size $2 * \text{halo_microns}$ is clipped out around each error center by default. This option is ignored if pm_classify is also specified.

Specifying “around extent” sets the halo to instead be the shape of the error polygon, sized up by *halo_microns*. Error shapes must have their extents matching to be considered duplicates. The “around extent” argument is incompatible with the anchor argument.

Tip

 The `critical_dimension` setup command can be used in place of this argument. The value used for *halo_microns* is $2 * \text{critical_dimension_value}$.

- **coarse_match *cm_microns***

An optional argument specifying that if a pattern is larger than or equal to 3 dbu, patterns whose context layers differ by less than *cm_microns* in *layer2* may be classified as duplicates. This is an approximate matching method.

- **unique_unclassified_ids {off | on}**

An optional argument that, when set to “on”, specifies that unclassified errors be issued unique CLASS_IDs. When set to “off” (the default setting), this keyword specifies that unclassified errors are assigned as CLASS_ID = 0.

- **maxsize *ms_microns***

An optional argument specifying a maximum error size. Errors with an extent exceeding *ms_microns* in an X or Y direction will be considered unclassified. Default: 0.5 microns. This option is ignored if pm_classify is also specified.

- `maximum_error_number max_errors`

Note

 `maximum_error_number` is deprecated as of Calibre OPCVerify version 2021.1. It will be removed in the 2021.3 release.

An optional argument specifying a limit to the number of total errors recorded. If specified, it *turns off classification* after `max_errors` have been found, and marks errors in the remaining tiles as unclassified. The granularity of this option is per tile; it has no effect for flat runs or runs with a single tile.

Caution

 In a Calibre MTflex run with multiple remotes, the `maximum_error_number` keyword allows all remotes to finish classifying the current tile before turning off classification for later tiles. The exact subset of errors that are classified when this argument is in use is arbitrary.

- You should use `maximum_error_number ONLY` as a safety measure against extremely long classification times on unexpectedly large numbers of errors.
- Siemens EDA also recommends against the use of a `maximum_error_number` that is too similar in size to the “`worst count`” argument, as under certain conditions classification can be turned off too early to capture significant errors. The suggested value for `maximum_error_number` is at least ten times relative to `worst count`.

- `{suppress | keep | keep_no_context} duplicates`

An optional argument that specifies whether or not duplicates are suppressed. Default: Duplicates are suppressed. If `suppress duplicates` is specified (the default), a `CLASS_ID` property is not attached to the output, and the command will output only unique and unclassified errors.

- `keep duplicates` — Outputs context contours for both unique errors and duplicates.
- `keep_no_context` — Duplicates outputs context contours only for unique errors, not for duplicates.

Note

 `CLASS_ID` properties are arbitrary assignment numbers, and may differ between a `keep` versus `keep_no_context` duplicates run. This is expected behavior.

- `anchor alayer`

An optional argument that specifies an anchor layer (see “[Anchoring Mode](#)”). It must be a Calibre OPCVerify input layer.

Default: `no anchoring`. This option is incompatible with the `halo` around extent argument. This option is also ignored if `pm_classify` is also specified.

- `anchor_max_snap ams_microns`

An optional argument that specifies the largest distance by which the anchoring point may move away from the center of the error marker. Larger values yield fewer unique errors. Default: 0.2 microns. This option is ignored if `pm_classify` is also specified.

Tip

The [critical_dimension](#) setup command can be used in place of this argument. The value used for `ams_microns` is $2 * \text{critical_dimension_value}$.

- `anchor_halo ah_microns`

An optional argument used in anchored mode to widen the halo used to search for anchoring points.

The anchoring code searches for anchor vertices within a window of radius `ah_microns` around the error center. If an anchor point is found, it is snapped back towards the error center in multiples of `ams_microns` until it is within `ams_microns` of the error center.

Default is to use the value specified for the halo argument. This option is also ignored if `pm_classify` is specified.

- `reflections_rotations_match {decide | yes | no | reflections}`

An optional argument that specifies how rotated/mirrored matches are handled.

Note

 When using optical model versions less than 10, the default argument “decide” must be changed. See the section “[Set “reflections_rotations_match” for Optical Models Below Version 10](#)” on page 505 for more information.

- `decide` — Translates to no if the opcverify deck has asymmetric optical models. This is the default.
- `yes` — Forces classification to classify rotated or mirrored patterns as duplicates.
- `no` — Forces classification to classify rotated or mirrored patterns as non-duplicates.
- `reflections` — Forces classification to classify patterns that are reflected around the X axis, Y axis, or both as duplicates.

Note

 The “no” option will cause Calibre to clone all mirror and rotated cells during the construction of the database, which may result in slower Calibre runtimes.

- `select_errors {lower_left | random}`

An optional argument that specifies how to select errors. Typically, Calibre OPCVerify selects errors with the worst score. The requirements for scoring and what are considered the worst errors is specified through the “score” and “worst” arguments. When scoring is not specified or if errors have the same score, then one of two selection policies, namely,

lower_left or random are applied to break ties. These selection policies are applied when choosing a unique representative from a set of duplicates with identical scores, and, also during limiting of duplicates.

By default, the lower_left policy is used, and, it chooses errors by the smallest cell-ID, then smallest X-coordinate, and then by smallest Y-coordinate.

If select_errors is set to random then each of eligible duplicate have the same probability to be selected as the unique representative of the set. This procedure prevents all errors from concentrating in the lower-left corner of the layout. When errors before classification are spread uniformly throughout the layout using “select_errors random” leads to uniform distribution of the unique errors as well; when unclassified errors are concentrated in certain areas of layout the spatial distribution of the uniques will be the same as the spatial distribution of unclassified errors (that is there will be more unique errors in the areas where there were more unclassified errors). The random selection is done in such a manner that two identical Calibre OPCverify runs always produce the same result.

When Calibre OPCverify is run in hierarchical mode, error selection happens in individual cells which are then placed in the layout by the Calibre engine. This may not result in a uniform flattened distribution of errors across the chip as the wafer SEM inspection desires. In order to get a better distribution, it is recommended to run Calibre OPCverify in litho flat mode.

- score [bin_size *score_delta*] [property *propname*] [smallest | largest]

An optional argument that indicates the start of a score block. Calibre OPCverify considers any difference in property values between two shapes that is less than *score_delta* to be a duplicate if the geometric contexts match.

- If the bin_size *score_delta* argument is omitted, then classification uses only geometric contexts in classification. Classification selects a unique error with the worst score from among a set of duplicates.

Conceptually, when score is specified without bin_size, one large score bin is used.

- For performance reasons, bin_size is required when duplicates are kept and the “worst” argument is specified. Also note that classification with limiting uses less memory and time when a reasonable value of bin_size is specified, so bin_size specification is recommended.

Note

 The use of a score block requires a property block defined outside the classify block.

The following optional sub-keywords can be specified on the same line as the score argument:

- property *propname*

If specified, the named property (which must appear in the property list specified in the property block outside of the classify block) is used as the selection criteria. If not specified, the first property in the list is used.

- smallest | largest

Sorts the values for the requested property as smallest first or largest first so that the “worst” value returns the top of the list. The default is smallest.

Note

 “smallest” and “largest”, when used, must appear on the same code line as the “score” argument.

- worst [unique | total] *count* [{duplicates | per_class} *dup_count*]

An optional argument to restrict the number of errors. “worst” must be the start of a new line, and also requires that the “score” argument appear in the limit block.

Note

 Previous to the 2017.3 release, the “keep_bin” and “exact” keywords were used to handle errors in the final bin. These keywords are retained for backwards compatibility, but are otherwise ignored and have no effect on the output.

- By default, the error list is based on either the smallest or largest values in the bins with unique errors. “unique” also includes unique, unclassified errors totaling *count* errors.
- In “total” mode, all unique and duplicate errors are output, limited by *count*. This option is not permitted with *keep_no_context* duplicates. If this option is specified with suppress duplicates, it is ignored and unique-only limiting is performed instead.

Note

 The “worst total” setting is incompatible with “suppress duplicates”. Requesting a total limiting will result in the “suppress duplicates” setting being ignored and forces the behavior to “keep duplicates”.

- Specifying “worst” returns the exact number of errors requested, picking from the last bin in priority order: the worst *score*, then by smallest *cell_id*, then by smallest x-coordinate, then separately by smallest y-coordinate.

In the default “worst unique *count*” mode, duplicate errors are not counted towards the limit.

- The “duplicates” and “per_class” options can only be used when the “unique” limiting mode is explicitly specified. The options are not compatible with the “total” limiting mode and “suppress_duplicates” option.
 - duplicates — After unique limiting, the total number of duplicates is limited to *dup_count*. Duplicates are sorted in order, by unique (from worst), then by

property, then by cell ID, then by x and y coordinates. The duplicate keyword is not compatible with total limiting mode and suppress duplicates.

- **per_class** — after unique limiting, duplicates for each class of unique errors are limited to *dup_count*. Duplicates are sorted by property, then by cell ID, then by x and y coordinates.
- **pm_classify max_search *value* max_length *value* [match_tolerance *value*] [orig_runlength]**

An optional argument that instructs Calibre OPCVerify to perform enhanced classification, including splitting large errors, improved anchoring, and “fuzzy matching” of context geometry. It significantly reduces the final error counts. This argument invokes the Calibre OPCVerify Classify Plus feature, which requires a separate license.

Note



Enabling pm_classify mode disables (and ignores) multiple arguments in the main classify block as follows:

- **halo** — pm_classify uses max_search in its place.
 - **maxsize** — pm_classify uses max_length in its place, and cuts all errors to max_length.
 - **anchor** — pm_classify always does anchoring and uses all context layers as the anchor layers.
 - **anchor_max_snap** — pm_classify ignores this option because it is not relevant to the pm_classify anchoring algorithm.
 - **anchor_halo** — pm_classify uses max_search in its place.
-
- Prior to 2019.2, pm_classify was a post-processing step, reported in the transcript as a separate LITHO PM_CLASSIFY run. Starting with 2019.2, pm_classify results are transparently folded into the “OPCVERIFY OUTPUT GEOMETRY COUNTS” section.
 - The pm_classify command is supported only for the Calibre hierarchical engine (calibre -hier).
 - Due to licensing requirements, a check that uses pm_classify must have at least one input contour derived from a setlayer image command initiated in the same Calibre OPCVerify run.

All keywords for pm_classify must be on the same line:

- **max_search *value***

A keyword that specifies the halo size used for fuzzy matching. It replaces the halo and anchor_halo arguments, creating a search area of 2*value around the error center.

- **max_length *value***

A keyword that specifies the largest permitted size for an input error marker in microns. Markers larger than `max_length` are split before Classify Plus classification into sizes less than or equal to `max_length`. Classification is applied to the split pieces. The splitting of large errors is one important reason why Classify Plus is capable of large reductions in unique error counts.

`max_length` must be at least $1.0 * \text{max_search}$.

There is a trade-off in setting this keyword:

- Smaller values may split relatively small errors, and the cut pieces, if not classified away, may cause larger final error counts.
- Larger values usually produce smaller final error counts and less confusing final markers, but at the risk of discarding important information. The `pm_classify` algorithm matches geometry only within a “peephole” of diameter $2 * \text{max_search}$ around a point within `max_search` of the input marker center. Hence, a large `max_length` could result in errors with important differences outside this peephole being incorrectly considered identical.

Overall, setting `max_length` in the range 3 to $10 * \text{max_search}$ is recommended.

Tip  To learn about the results of error splitting by `max_length`, run classification twice, the second time with “keep duplicates”. Then use the with-duplicates result as the context when viewing the unique-only results in Calibre WORKbench.

- `match_tolerance value`

If this optional keyword is present, fuzzy matching is applied when comparing context geometry in `pm_classify`. The default is 0.0, specifying exact matching for all geometry comparisons. `value` indicates a tolerance in microns. Geometry comparisons are considered successful when the two patterns match within an envelope of the specified size around the pattern edges. Fuzzy matching comparisons ignore different placements of jogs in each pattern, as long as the jogs are fully contained within the envelope.

Note  Starting with the 2019.2 release, a list of what was detected by the fuzzy matching algorithm can be found in the “POSTPROCESSING” section of the transcript.

- `orig_runlength`

If this optional keyword is present, `pm_classify` generates an output property `ORIG_RUNLENGTH` for all errors. The property is 0.0 for all errors that were not cut up by `pm_classify`. For errors that were cut by `pm_classify`,

ORIG_RUNLENGTH specifies the run length, in microns, of the original error marker before cutting.

Examples

Example 1

The following examples add a classification block to a measure_cd command.

Without a property block (score bins cannot be used):

```
setlayer cderror = measure_cd contour target internal cd_min 0.11 cd_max
0.16 cd_min_length 0.01 max_search 0.06 tol not < 1.1 ratio classify {
context target
halo 0.1 }
```

With a property block:

```
setlayer cderror = measure_cd contour target internal cd_min 0.11 cd_max
0.16 cd_min_length 0.01 max_search 0.06 tol not < 1.1 ratio property {
min
max
} classify
{ context target
halo 0.1 }
```

Example 2 (Scoring)

With a property block and a scoring bin block as shown in the following code, the value of the first item in the property list (min) is used in addition to the geometric context to decide if two errors match. Each score bin in this example has a size of 0.01 microns and starts from 0. If two errors fall into different score bins, they are considered as non-matching errors even if they are less than 0.01 microns in difference (0.0195 and 0.0201, for example). Setting a very large value (10) will act if no score was specified (since the errors will land in the same bin); setting a very small value (0.001) will make classification highly sensitive to differences.

```
setlayer cderror = measure_cd contour target internal cd_min 0.11 cd_max
0.16 cd_min_length 0.01 max_search 0.06 tol not < 1.1 ratio property {
min
max
} classify { context target
halo 0.1
score bin_size 0.01}
```

Example 3 (Scoring and Worst)

With a property, scoring, and “worst count” block, the number of errors returned is limited to exactly the count worst uniques, plus all of their duplicates if “keep duplicates” is also specified.

```
setlayer cderror = measure_cd contour target internal cd_min 0.11 cd_max
0.16 cd_min_length 0.01 max_search 0.06 tol not < 1.1 ratio property {
min
max
} classify { context target
halo 0.1
score bin_size 0.01 property max
worst 3
}
```

Limit Block

Calibre OPCVerify commands can return large amounts of results, especially for bigger, more complex chip designs. In such cases, you can optionally add a limit restriction to return only the worst subset of error polygons to the SVRF rule file. You use limit blocks as a safety net to avoid excessive error generation times due to variations in the layout. A limit is based off of one property function, a number of errors to return, and a selection criteria (smallest or largest value for the property).

Note

 Error counts before limiting are available in the log file for reference; a text search on the status line LIMITING FOR LAYER will show how many errors were filtered out for each setlayer statement that includes a limit block.

Limit blocks are incompatible with classification and histogram blocks.

The full functional syntax notation for a property block with included limit block is as follows:

```
setlayer derived = function_name .... [property { propfunction1
[propfunction2]
[propfunction3]
} [limit {
[score [property propfunctionx] [smallest|largest]]
[worst value]
}]]
```

Limit block syntax uses the following rules:

1. Must be associated with a property keyword.
2. Must start the limit block on the same line as the end of the property keyword.
3. On a new line, limit the amount of error bins returned with the worst keyword and a closing brace (a bin contains errors with the same property value):

```
setlayer derived = measure_cd .... property { max
min
average
} limit {
worst 25
}
```

4. The optional keyword score can be specified with either “smallest” or ‘largest’ to indicate which end of the data set is considered to be the worst.

```
...
} limit {
score largest
worst 25
}
```

Tip

i The values “largest” and “smallest” must be selected to be relevant to your expected data and represent a non-absolute value selection criteria. For example, in order to find the worst line end pullback values (a negative EPE value), you would choose “smallest”. For line end extension values (a positive EPE value), you would choose “largest”. When measuring the magnitude of the EPE (absolute value of over/under the target), “largest” is the recommended selection.

5. If multiple types of property keyword have been specified, the score keyword can be modified with the property *propertyname* arguments between “score” and “largest/ smallest”:

```
...
} limit {
score property average largest
worst 25
}
```

- o Do not confuse the score property keyword inside the limit block with the initial property block keyword declaration.
- o The score property keyword must specify one of the property arguments previously given in the property block. For example, in the code above, score property can only use a min, max, or average argument.
- o If a score property is not specified, the first property block keyword is used (max in the example code above).

6. By default, the number of errors output is limited to the value specified in the worst argument, but also includes all the errors in the last score bin. This means that the number of errors returned may greatly exceed the value for “worst” if the last score bin contains a large number of errors.

Errors are sorted in priority order by the ‘worst’ value, then by smallest cell_id, then by smallest x coordinate, then separately by smallest y coordinate.

Histogram Block

Rule File Construct

A histogram block adds the ability to save tabular data from properties to a file. It can be added to any command that also has a properties block. Histogram blocks are mutually exclusive with classify and limit blocks.

Note

 The number of data points stored internally is limited to the number of hierarchical shapes processed (no zero-entries are kept internally). The main performance degradation will occur only if the zero_counts argument is set to keep and the number of bins computed from the **bin_sz** argument is very large.

Usage

```
setlayer x = command
...
property '{'
property_block
'}' histogram '{'
file output_file property prop_name bin_size bin_sz [zero_counts {suppress | keep}]
    [label {off | on | lbl_text}]...
'}
```

Description

Calibre OPCVerify supports histogram output to a file. A histogram output file has two tab separated columns:

- First column: centers of the bins
- Second column: integer number of flat occurrences of error polygons on the histogram-enabled Calibre OPCVerify layer with a property value in that bin.

Use of a histogram block requires a property block in the same setlayer command; only properties previously named in the property block can have histogram files created for them.

Note

 A histogram-enabled Calibre OPCVerify layer must also be an output layer in the SVRF rule file for the histogram to be generated.

Arguments

- histogram {

A required argument declaring the start of the histogram block. Arguments in this section must be inside a histogram block and enclosed by braces ({}).

- **file *output_file***

A required argument that specifies the filename to save the histogram data to. File name write permissions are checked at compile time and must be unique among all histograms; old histogram files are overwritten.

The remaining histogram arguments (property, bin_size, zero_counts, and label) must appear on a single line with the file keyword.

- **property *prop_name***

A required argument that specifies which of the properties in the previously-declared property block to write information for.

- **bin_size *bin_sz***

A required argument that specifies the bin size. Bin ranges are computed as follows:

$[-0.5 * \text{bin_sz}, 0.5 * \text{bin_sz}), [0.5 * \text{bin_sz}, 1.5 * \text{bin_sz}), [1.5 * \text{bin_sz}, 2.5 * \text{bin_sz})$

Note

 The notation $[a,b)$ is intended to be interpreted as $a \leq \text{value} < b$; for example, $[0.5 * \text{bin_sz}, 1.5 * \text{bin_sz})$ represents a range of 0.5 to 1.49999999.

bin_sz must be larger than 10^{-7} . The range of *bin_sz* is limited by the equation:
 $\text{abs}(\text{max}(\text{property-value})/\text{bin_sz}) \leq \text{INT_MAX}$.

Computed bin indices are integer values.

- **zero_counts {suppress | keep}**

An optional argument that specifies the behavior when the polygon count of a bin is zero. If set to keep, the output file includes bins with zero polygon counts; suppress (the default) skips the bin.

- **label {off | on | *lbl_text*}**

An optional argument that specifies whether the histogram is labeled in the output file. The histogram label is printed in square brackets on a separate line preceding the first line of the histogram. This feature is useful for identifying different histograms saved into the same file. The default setting is "off", meaning that the label is not printed. If "on" is specified, the histogram has the standard label, which consists of the layer name and histogram property name separated by a colon. The *lbl_text* option can be used to specify a user-defined text for the label. If *lbl_text* contains whitespace, it should be enclosed in quotes.

Examples

Given the following code:

```
setlayer e2 = area_overlay contact poly_contour not > 0.80 max_extent 0.6\
    shift .05 output_type worst property {
        min
    } histogram {
        file histogram_e2.txt property min bin_size 0.1
    }
setlayer e3 = area_compute contact < 1 max_extent 0.6    property {
    area
} histogram {
    file histogram_e3.txt property area bin_size 0.01
}
setlayer e4 = area_compute contact < 1 max_extent 0.6 property {
    area
}
```

The following information is printed to the transcript:

```
HISTOGRAM MEAN 0.236801 STANDARD DEVIATION 0.213629 FOR LAYER e2:
    HISTOGRAM COUNT 1, FLAT GEOMETRY COUNT 1388
TIME FOR LAYER aov_e2 HISTOGRAMMING: CPU TIME = 0  REAL TIME = 0
    TIMESTAMP 6
...
HISTOGRAM MEAN 0.050607 STANDARD DEVIATION 0.016282 FOR LAYER e3:
    HISTOGRAM COUNT 1, FLAT GEOMETRY COUNT 1917
TIME FOR LAYER aov_e3 HISTOGRAMMING: CPU TIME = 0  REAL TIME = 0
    TIMESTAMP 6
...
aov_e3 (HIER TYP=1 CFG=1 HGC=0 FGC=0 HEC=0 FEC=0 VHC=F VPC=F)
aov_e4 (HIER-FMF TYP=1 CFG=1 HGC=1354 FGC=1917 HEC=127681 FEC=178196 VHC=F
    VPC=F)
```

The mean and standard deviation is reported for all properties in the property block of the histogram layer. The algorithm for computing standard deviation is ‘biased’ in that the (sum-of-squares minus square-of-sum) term is divided by n, where n is the number of data samples. The calculation of the standard deviation and mean is performed incrementally (adding a single data point at a time) to reduce loss of precision errors that occur when computing the difference of large-almost-equal values.

Mean/standard deviation in the presence of the ‘exception also’ keyword may include exception property values (negative values for the area_ration property). Be careful when using exception also with histograms.

In this example, notice that the histogram FLAT GEOMETRY COUNT (FGC) for histogram-enabled layer e3 of 1917 agrees with the count in the SVRF transcript for an identical [setlayer](#) operation without the histogram block (aov_e4) of FGC=1917.

Note

 For the histogram SVRF layer aov_e3, the Flat Geometry Count (FGC) is 0.

The histogram output file *histogram_e3.txt* contains information similar to the following:

0.04	1060
0.05	2
0.06	462
0.07	261
0.09	132

Specifies the derived layer to extract from the OPCverify command setup file.

add_properties Block

Rule File Construct

An add_properties block computes additional properties for the errors generated by other checks (such as pinch and bridge). These properties are computed relative to the locations where the primary check found a problem.

Usage

```
setlayer name = command_supporting_add_properties cmd_arg1 cmd_arg2 ...
[...] property '{'
prop
...
[prop]
'}' add_properties {
prop_name = operation
...
[prop_name = operation]
'}
```

where **operation** is one of: cd, contour_diff, contrast, distance, dof, epe, ils, imax, imin, intensity_ils, intensity_meef, intensity_nils, meef, nils, or stochastic.

Description

An add_properties block computes additional properties for the errors generated by other checks. These properties are computed relative to the locations where the primary check found a problem.

Note

 Although the syntax and name of “add_properties” block is similar to that of “setlayer gauges”, this feature is not the same and supports a different set of properties. Instead of the user-provided gauges in “setlayer gauges”, this operation uses internal gauges generated by other Calibre OPCVerify checks. The following terms are used in this reference page:

- primary check: Refers to a setlayer command, which includes an add_properties block at the end.
- add_properties operation: Refers to a single operation within an add_properties block.
- internal gauge: A hidden gauge or marker produced by the primary check, used internally to determine the location of the worst error. This gauge is used as the measurement location for add_properties operations.
- customer gauge: A customer-specified gauge in the gauges command, used in place of the internal gauge for the add_properties operations in that command.

An add_properties block can be specified after the property block for most setlayer operations that support property blocks. The current list of supported commands are:

bridge, contour_diff, dofcheck, imax_check, imin_check, intensity_ilsccheck, intensity_meefcheck, intensity_nilscheck, measure_cd, measure_cdv, measure_distance, measure_epc, meefcheck, and pinch.

Note

 The [gauges](#) check also supports add_properties blocks, with some functional differences as described in that command's reference page.

The syntax uses *prop_name*, which can be any name that does not coincide with other property names in property or add_properties blocks.

add_properties Gauges

When setlayer commands are run, markers are placed on the output layer based on the command operation as its standard output.

When an add_properties block is included, the block can contain one or more operations to be run on the results, placing properties with computed values on the marker.

Note

 If a primary check produces an exception, all corresponding properties in its add_property blocks will also be exceptions.

Because a marker is typically a polygon covering an area, gauges are placed within the marker as a location to perform measurements at.

Add_properties blocks use the two types of property gauges created by the primary check:

- Standard gauges — A gauge with two endpoints. Most primary check operations create gauges with two ends.
- Point gauges — A single point placed anywhere along a contour. Only intensity_ilsccheck, intensity_meefcheck, and intensity_nilscheck primary checks produce point gauges.

Measurement Locations for add_properties

The exact location where an add_property measurement is taken depends on both the type of add_property gauge being used for the operation, and the type of add_properties operation being performed.

Note

 The marker or gauge placed by the primary check is not necessarily the location where measurements are taken and reported from. The gauge marker is used to mark the general location of the error.

The setlayer command checks that support add_properties are either two-sided (place a marker connecting two points in the layout) or one-sided (place a marker at a specific point).

- The bridge, contour_diff, dofcheck, measure_cd, imax_check, imin_check, measure_cdv, measure_distance, measure_epe, meef, and pinch commands are two-sided checks.
- The intensity_ils, intensity_meefcheck, and intensity_nils checks are one-sided checks.

All add_properties operations except distance accept both two-ended (standard) and point internal gauges, and can be used with any primary check that supports add_properties blocks. The add_properties distance operation accepts only standard internal gauges, meaning that it cannot be used with one-sided primary checks.

The combinations of primary check locations and add_properties operations follow these rules:

- If an add_properties operation accepts point gauges, and is provided with a point gauge, it calculates the property at (in the vicinity of) this point.
- If an add_properties operation accepts point gauges, but is provided with a two-sided gauge, it calculates properties at both ends of the gauge, and selects the worst one (“worst” is the maximum or minimum, depending on the nature of the add_properties check, according to the following table). If one of the two properties is an exception, the non-exception value is selected. If both properties are exceptions, the returned property is an exception.

Table 4-3. One-Sided add_properties Operation Value Used From a Two-Sided Gauge

add_properties	Selected Property Value
cd	min
contour_diff	max
contrast	min
dof	min
epe	same as function (min/max) specified in operation parameters
ils and nils intensity_ils and intensity_nils	min

Table 4-3. One-Sided add_properties Operation Value Used From a Two-Sided Gauge (cont.)

add_properties	Selected Property Value
imin	max
imax	min
meef, intensity_meef	max

- If an add_properties operation does not accept point gauges (distance is currently the only operation that does not accept point gauges), it cannot be added to a primary check producing such gauges (a parse error occurs).

Note

 It is important to understand that the actual measurement location can take measurements using contours and points that were not arguments in the original setlayer check. This is because the arguments to the add_properties operations are used instead of the original check, but the error marker from the original check is used as a starting context.

Arguments

- **cd** — An argument requesting a two-sided operation that runs a CD measurement check for the specified contour and target layers.

prop_name = cd contour target max_search MS {internal | external} [exception property_value] [with {{rel | ratio | target}...}]

The property applied uses the [measure_cd](#) behavior for which edges to check (internal or external).

If the with keyword is added, an additional calculated property ***prop_name.suffix*** is added to the output with the relevant value (rel adds the min relative CD, ratio adds the min relative ratio, and target adds the min CD within the target polygon).

- **contrast** — An argument requesting a one-sided operation that runs an image contrast check ($\text{imax}-\text{imin}/\text{imax}+\text{imin}$) for the specified image layer or image set. It returns the minimum contrast for the specified gauge set.

prop_name = contrast {image_layer | image_set_name} gauge_set {set_name | {imin set_name1 imax set_name2}} max_search MS [exception property_value] [with image]

The gauge set can either be a single gauge set or a pair of gauge sets, one for imin and one for imax parameters.

If an image set with more than one image contour is specified, the “with image” argument can be specified to add a property, ***prop_name.image***, that contains the name of the optical model inside the image set that had the minimum contrast.

- **contour_diff** — An argument requesting a one-sided operation that measures the distance between contours.

prop_name = contour_diff contour1 contour2 max_search MS

[exception *property_value*]

Where **contour1** and **contour2** are names of two different contours generated with the image command. A search region size is specified by **max_search**; however, if a **critical_dimension** statement is present in the deck, this argument is optional and defaults to one-half of the **critical_dimension** value. Optionally, a property value can be specified to be used as an artificial value when the distance cannot be calculated; a default value of 100 is used as the exception flag value.

- **distance** — An argument requesting a two-sided operation that measures the distance between opposing edges of a layer, similar to the “min” property as defined by “measure_distance”. This argument is used for pinch and bridge operations.

prop_name = distance layer separation S max_search MS [exception *property_value*]

A distance is measured on the provided layer, usually a contour generated with the image command. A minimum separation in microns can be specified to prevent the distance command from measuring a distance between two edges belonging to the same side of a contour. A search region size is specified by **max_search**; however, if a **critical_dimension** statement is present in the litho setup file, this argument is optional and defaults to one-half of the **critical_dimension** value. Optionally, a property value can be specified to use as an artificial value when the distance cannot be calculated; a default value of 100 is used as the exception flag value.

- **dof** — An argument requesting a two-sided operation that performs a depth of focus (DOF) check at the error location. It uses one of the following formats:

prop_name = dof target images img1 img2 img3 [...] defocus f1 f2 f3 [...] max_search MS
[width | space] [tolerance *tol*] [exception *property_value*]

or

prop_name = dof target image_set image_set_name [focus_units {nm | um}]

max_search MS [width | space] [tolerance *tol*] [exception *property_value*]

This command is equivalent to running the [dofcheck](#) operation.

- **epe** — An argument requesting a one-sided operation that computes the EPE value at the exact location of the main error. If the error is calculated for a pair of target edges (such as the two-sided marker created by bridge and pinch checks), it is treated like a gauge, and two points are considered, one on either end of the gauge, with the selected point dependent on the specified arguments. If the error is calculated for a single point edge (such as **contour_diff**), a single EPE value is calculated on the target closest to the main error.

prop_name = epe contour target max_search MS [max | min] [magnitude]

[exception *property_value*]

Where **contour** is a simulated contour layer and **target** is a target polygon or contour layer to compare the first contour with. A search region size is specified by **max_search**. In the case of gauge-type errors, use “max” and “min” to use either the maximum or minimum of the two values. Adding the “magnitude” argument calculates the EPE magnitude before max or min selection is performed. Optionally, a property value can be specified with the

“exception” argument to use as an artificial value when the distance cannot be calculated; a default value of 100 is used as the exception flag value.

- **ils** — An argument requesting a one-sided operation that calculates the non-normalized image log slope at the error location.

prop_name = ils target {images img1 img2 ... levels lv1 lv2 ... | image_set image_set_name} max_search MS [min_contour_separation mcs] [exception property_value]

At least two images or an image set containing two or more images must be specified. Only CTR images are expected for the image contours. If individual image layers are specified, corresponding aerial contour slice levels must also be specified, following the images keyword.

The optional min_contour_separation option sets the measurement distance between contours for ILS comparison purposes.

- **imax** — An argument requesting a two-sided operation that calculates the maximum intensity for the specified image layer or image set.

prop_name = imax {image_layer | image_set_name}... gauge_set gauge_set_name max_search MS [exception property_value] [with image]

This operation is the equivalent of the [imax_check](#) command applied to the gauges placed for found errors. If the with image option is specified with an image set, an additional property **prop_name.image** is added to the gauge with the name of the optical model that generated the largest imax value.

- **imin** — An argument requesting a two-sided operation that calculates the minimum intensity for the specified image layer or image set.

prop_name = imin {image_layer | image_set_name}... gauge_set gauge_set_name max_search MS [exception property_value] [with image]

This operation is the equivalent of the [imin_check](#) command applied to the gauges placed for found errors. If the with image option is specified with an image set, an additional property **prop_name.image** is added to the gauge with the name of the optical model that generated the smallest imin value.

- **intensity_ils** — An argument requesting a one-sided operation that calculates the ILS for a set of layers or an image set.

prop_name = intensity_ils options max_search2 MS [exception property_value]
[with '{'[image] [min] [max]'}']

where **options** are any options permitted by the [intensity_ilsccheck](#) command, except the constraint and markers options.

max_search2 is the search distance in microns that the add_properties operation searches from the initial error marker for a point on the add_properties input layer for calculation purposes.

The sub-properties min and max select the minimum or maximum of a two-ended gauge to apply as the result of the operation, respectively. If the primary check produces a single-ended gauge, min and max are the same. If the primary check produces an exception at one end, that minimum or maximum will have the exception value.

Choosing to use more than one of these options requires braces around the expression, similar to the following example:

```
int_ils = intensity_ils other_options with {image min}
```

- **intensity_meef** — An argument requesting a one-sided operation that computes intensity MEEF values at the error locations produced by the main check.

prop_name = **intensity_meef** <*options*> **max_search2 MS** [exception *property_value*] [with '{' [image] [min] [max] '}']

where *options* are any keywords allowed by the [intensity_meefcheck](#) command.

The **max_search2** option is the maximum distance to search for image from the end of the gauge produced by the main check. If **critical_dimension** setup command is specified, the default value is 0.5*CD, otherwise the parameter is mandatory.

The sub-properties min and max select the minimum or maximum of a two-ended gauge to apply as the result of the operation, respectively. If the primary check produces a single-ended gauge, min and max are the same. If the primary check produces an exception at one end, that minimum or maximum will have the exception value.

This operation executes **intensity_meefcheck** in the background and computes intensity MEEF values at the required locations. This is done by interpolating MEEF values from the nearest gauges produced by the underlying **intensity_meefcheck** operation. The intensity MEEF is always computed at the image layer (it cannot be computed on the target layer).

- **intensity_nils** — An argument requesting a one-sided operation that calculates the NILS value for a set of layers or an image set.

prop_name = **intensity_nils** *options* **max_search2 MS** [exception *property_value*] [with '{'[image] [min] [max]'}']

where *options* are any keywords permitted by the [intensity_nilscheck](#) command, except the constraint and markers options.

Starting with the 2020.3 release, the algorithm used for the NILS value is as follows for any add_properties block:

- a. The ILS value at the add_properties measurement point is interpolated from the ILS values at the nearest internal intensity gauges.
- b. The directions for CD measurement at the measurement point are interpolated from directions of CD measurements associated with the matching nearest intensity gauges, unless the primary check is **gauges**. For the **gauges** command, the CD is measured in the direction of the gauge.

- c. The local CD is measured at the add_properties measurement point in the interpolated CD direction, unless the primary check is gauges. For the gauges command, the CD is measured in the direction of the gauge.
- d. To obtain the NILS at the measurement point, the interpolated ILS value from (a) is normalized by the CD measured in (c).

The fix_small_cds, cd_angle_range_degs, and min_separation_fac options are not used, and will be ignored if specified. The max_local_cd option is still valid, however, and uses a value of 3*critical_dimension by default.

The **intensity_nils** operation also uses the **max_search2** option for calculation purposes as described for **intensity_ils**.

The sub-properties min and max select the minimum or maximum of a two-ended gauge to apply as the result of the operation, respectively. If the primary check produces a single-ended gauge, min and max are the same. If the primary check produces an exception at one end, that minimum or maximum will have the exception value.

Choosing to use more than one of these options requires braces around the expression, similar to the following example:

```
int_ilis = intensity_nils other_options with {image min}
```

- **meef** — An argument requesting a one-sided operation that computes MEEF at the exact location of the main error. It supports two different formats:
 - **prop_name = meef contour1 contour2 delta_mask DM max_search MS[exception property_value]**
 - **prop_name = meef image_set image_set_name [delta_mask DM] max_search MS [exception property_value]**

Where **contour1** and **contour2** are names of two different contours generated with the image command. The **image_set_name** parameter is the name of an **image_set** that contains exactly two images. Delta_mask is the sizing in microns on the mask of the second image, relative to the mask used on the first image. The **max_search** parameter specifies a search region size; however, if a **critical_dimension** statement is present in the litho setup file, this argument is optional and defaults to one-half of the **critical_dimension** value. The exception parameter is used to specify a optional property value to be used as an artificial value when the distance cannot be calculated. The default value is 100.

- **nils** — An argument requesting a one-sided operation that computes the normalized image log slope (NILS) for the location of the main error.
prop_name = nils target {images img1 img2 ... levels lv1 lv2 ... | image_set image_set_name} [auto_cd | width | space | both] max_search MS [min_contour_separation mcs] [exception property_value]

Similar to the **nilscheck** command, this operation calculates the NILS values for either a list of image layers and corresponding level slices, or an image set.

The NILS value can be calculated for the internal (width), external (space), or both gauge types. If auto_cd is specified, the operation attempts to automatically detect the type of gauge to use based on how the gauge interacts with the contours.

- **stochastic** — An argument that computes the stochastic failure probability between 0% to 100% of the error detected by the pinch or bridge operations.

prop_name = stochastic resist_image resist_layer stochastic_model

stochastic_modelName [exception property_value]

- **resist_image** — A required keyword specifying the name of a resist layer (usually a contour generated with the **image** command) to measure the **pinch** or **bridge** distance for. If the resist image has an etch model, this command may have longer run times due to the larger tile area to be analyzed.
- **stochastic_model** — A required keyword used to specify a specific stochastic model previously loaded using **stochastic_model_load**.
- **exception** — An optional keyword specifying an artificial property value to be used when the failure probability cannot be calculated. (Default is -1.)

Examples

Given the following code, which features a bridge command (a two-sided primary check) that contains two add_properties operations (distance and meef):

```
LITHO FILE OPCV_INPUT [
    critical_dimension 0.04
    layer trgt
    layer trgt_opc
    .....
    image_options img_options {
        .....
    }

    setlayer cntr_nom    = image img_options
    setlayer cntr_plus   = image img_options bias  0.001
    setlayer cntr_minus = image img_options bias -0.001

    setlayer bridg_pin = bridge poly cntr_nom < 0.05 separation 0.2
    pinpoint_output {gauges}

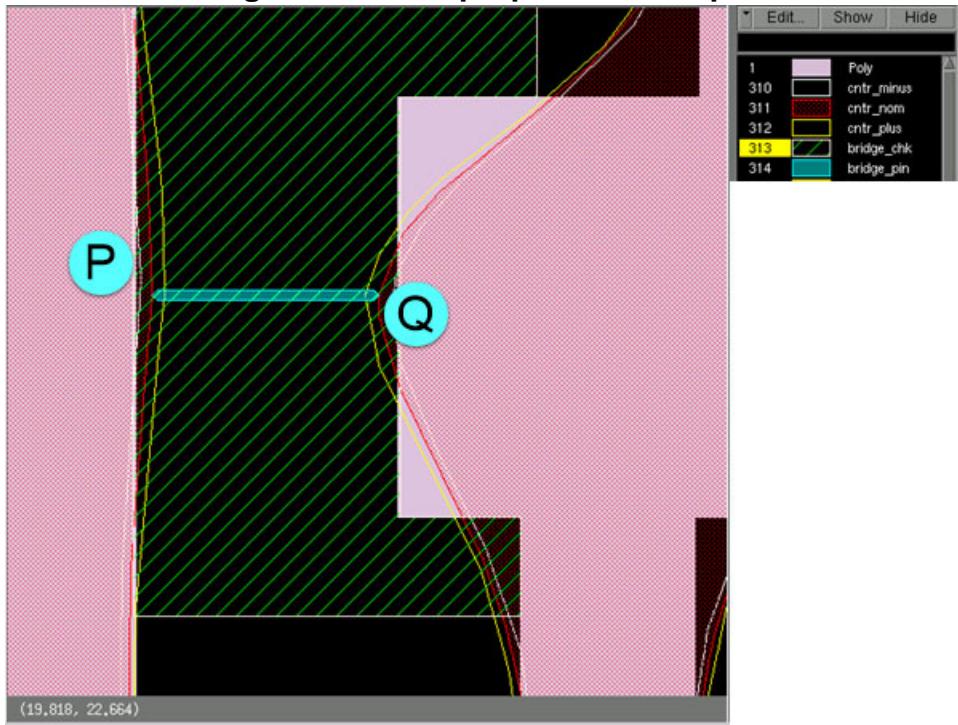
    setlayer bridg = bridge poly cntr_nom < 0.05 separation 0.2 property { min
} add_properties {

    dist = distance cntr_plus separation 0.2 max_search 0.01

    meef = meef cntr_minus cntr_plus delta_mask 0.002 max_search 0.01  }]
```

An error for this would look similar to the following figure:

Figure 4-2. add_properties Example



Examining the error properties for an error marker in Calibre RVE shows values similar to the following:

- min = 0.0490

Represents the length of the pinpoint error marker (PQ) for the bridge check, using the cntr_nom (nominal contour) layer as a measurement layer.

- dist = 0.04387

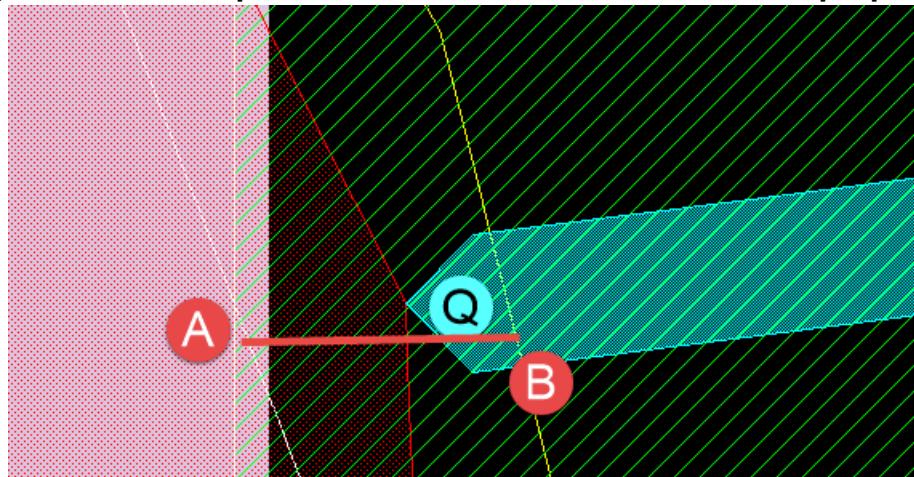
Because the distance operation uses a different contour layer (cntr_plus) as its contour source, its value is measured between two points on the contour nearest to the original error marker points (PQ). The distance measured is therefore smaller than min. A distance check with an argument of cntr_minus would give a dist value larger than 0.0490.

- meef = 2.889

Because the meef operation also uses a different pair of contours (cntr_plus and cntr_minus) than the contour source, the MEEF value is computed within the max_search vicinity of both P and Q and the larger of the two is returned.

The measurement points are selected as shown:

Figure 4-3. meef Operation Measurement Points for add_properties



- Point A is selected on the first contour argument (cntr_minus) closest to the starting location (Q).
- Point B is selected on the second contour argument (cntr_plus), closest to point A.
- The MEEF value is calculated as the distance |AB| divided by the delta_mask value.

Related Topics

[Pinpoint Output Block](#)

[bridge](#)

[contour_diff](#)

[dofcheck](#)

[imax_check](#)

[imin_check](#)

[intensity_ilsccheck](#)

[intensity_meefcheck](#)

[intensity_nilscheck](#)

[measure_cd](#)

[measure_epe](#)

[meefcheck](#)

[pinch](#)

Pinpoint Output Block

Pinpoint output markers can be placed at the ‘worst’ part of an error marker location. Pinpoint output blocks are an option for any command that supports add_property blocks.

Usage

```
pinpoint_output '{' options '}'
```

where *options* is one of:

- off
- markers [*size*]
- gauges [*size*] {length *length_um* | extra *extra_um*}
- stripes [*size*] {length *length_um* | extra *extra_um*}

Description

Pinpoint_output replaces the regular error markers produced by a check with pinpoint markers. Pinpoint output markers show where an error point has been detected with better precision than a standard error marker, which highlights the region of an error.

All properties associated with original error markers are attached to the corresponding pinpoint markers. Each error polygon is replaced with exactly one pinpoint marker placed at the location of the worst property value.

For many operations (such as pinch), there is an apparent determinant property, such that its worst value corresponds to the greatest constraint violation. Other operations may have multiple different properties, so it may not be as obvious which property defines the worst error (such as measure_cd with both “min” and “max” properties requested). When this is the case, the first property in the properties block is assumed to be the determinative property, and a pinpoint marker is output at the location of its “worst” value (for example, the minimum value for a “min” property, maximum value for a “max” property).

When a check produces an exception, there is no particular location for the error. In this event, the pinpoint marker is always a square marker, even if gauges or stripes have been requested. The exception pinpoint marker is placed in the geometrical center of the bounding box of the original “exception” error polygon produced by the check.

The pinpoint markers are generated in a post-processing step, after all setlayer operations have been completed. Therefore, within the current OPCVerify run, an output layer always contains original error markers, even if it has pinpoint_output specified. To use pinpoints as a geometrical layer for subsequent setlayer operations (e.g., a filter layer), the pinpoint layer must be passed to another OPCVerify run as an input layer.

The pinpoint_output block appears at the end of a setlayer command together with error-centric options (property, classify, add_properties, etc.). The braces around the options are required if pinpoint_output block contains more than one parameter.

```
setlayer bridge_add = bridge poly img_d100_f0 < 0.060 separation 0.2
max_edge 0.050 output_expand 0.005
pinpoint_output {markers 0.02} property {min}
```

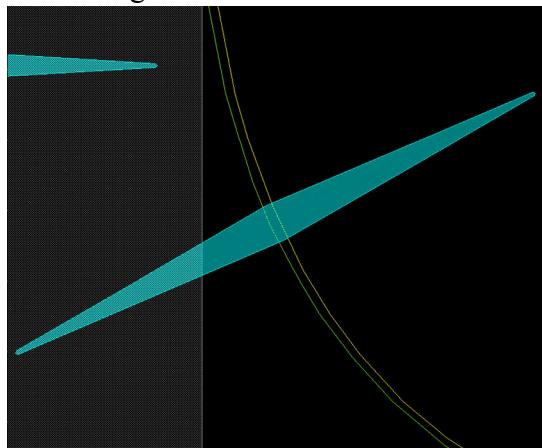
Arguments

- off

An argument that suppresses pinpoint marker output. The default marker output is used; this is the default behavior.
- markers [*size*]

An argument that creates markers of the specified *size* in microns at the midway point between the error points (minimum of 2dbu, the default).
- gauges [*size*] {length *length_um* | extra *extra_um*}

An argument that creates octagonal, rhombus-shaped markers indicating the points connecting error locations that are two-sided. (One-sided error locations, such as a hard pinch or hard bridge, have a single marker in the center of the error.)



The rhombus has a width specified by *size* in database units (minimum of 4dbu, the default).

The length and extra options override the endpoints of the marker touching the edges by either setting a specific length of the marker or an extra distance to extend the marker past the touching edges in microns, respectively.

- stripes [*size*] {length *length_um* | extra *extra_um*}

Creates rectangular markers between the connecting error locations, suitable for filtering purposes. Similar to gauges, one-sided markers place a single marker in the center of the error.

The stripe marker has a width specified by *size* in database units (minimum of 4dbu, the default).

The length and extra options override the endpoints of the marker touching the edges by either setting a specific length of the marker or an extra distance to extend the marker past the touching edges in microns, respectively.

Related Topics

[add_properties Block](#)

Using LITHO OPCVERIFY

Use these steps to create a Calibre OPCverify command file and one or more LITHO OPCVERIFY calls for use in an SVRF rule file.

Prerequisites

- An OASIS or GDS design file with layers to be used as input for Calibre OPCverify

Procedure

1. First, create an OPCverify command setup file, using the following sequence:
 - a. Setup commands that include OPCverify layer definitions
Setup commands are described in the section “[Calibre OPCverify Setup File Configuration Commands](#).”
 - b. setlayer commands that operate on OPCverify layers
Setlayer commands are described starting with the section “[Setlayer Operations Reference](#).”
2. In a separate SVRF rule file, define the mapping between the design file layer numbers and the OPCverify layers you defined inside the OPCverify command setup file.
3. Run OPCverify by defining one or more LITHO OPCVERIFY output layer definition calls from the SVRF rule file to the OPCverify command setup file, using the following syntax:

```
L1 = LITHO OPCVERIFY input_layer1 ... input_layerN FILE cmd_file \
    MAP OPCverify_layer1
L2 = LITHO OPCVERIFY input_layer1 ... input_layerN FILE cmd_file \
    MAP OPCverify_layer2
...
Ln = LITHO OPCVERIFY input_layer1 ... input_layerN FILE cmd_file \
    MAP OPCverify_layerN
```

4. (Optional) Output the results of OPCVERIFY to the design file using the following syntax:

```
Ln {COPY OPCverify_layerN} DRC CHECK MAP OPCverify_layerN\
    layer_number
```

5. Run Calibre nmDRC on the SVRF file:

```
calibre -drc -hier -turbo -turbo_litho filename.svrf
```

Results

The output of the SVRF rule file is written to the file designated in the SVRF rule file.

LITHO OPCVERIFY

SVRF Command

Runs the Litho OPCverify tool on the specified command file.

Usage

```
LITHO [EUV] OPCVERIFY input_layer1 [input_layerN] ... FILE cmd_file MAP  
OPCverify_layer [INSIDE OF LAYER region]
```

Arguments

- EUV
 - An optional argument that enables EUV libraries. Required when EUV models are used in the setup file referred to by the *cmd_file* argument. The EUV keyword requires an additional license for Calibre EUV to function. The EUV keyword must be specified if the wavelength is less than 50nm.
 - *input_layer1* ... *input_layerN*
 - A required argument that identifies one or more layers defined with LAYER commands in the SVRF rule file to be input to the Calibre OPCverify command file.
 - *cmd_file*
 - A required argument that specifies the name of the OPCverify command file to run.
 - *OPCverify_layer*
 - A required argument that specifies the name of the output layer from the Calibre OPCverify command file. A setlayer command inside the command file that results in a layer name matching *OPCverify_layer* is copied to the SVRF layer for this LITHO OPCVERIFY command.
 - INSIDE OF LAYER *region*
 - An optional argument that invokes the region iterator. The region iterator allows efficient execution of Calibre OPCverify inside a set of small user-defined regions, such as hotspots in a re-opc or repair flow. All regions that are smaller than the [tilemicrons](#) setting are processed as one tile, which can improve the quality of the results.
- Using INSIDE OF LAYER forces the use of these Calibre OPCverify features and commands even if they are not explicitly specified:
- [LAYOUT ULTRA FLEX YES](#)
 - [HDB construction mode](#)
 - [processing_mode flat](#)

The region iterator is equivalent to the following operations:

```
regs_plus = SIZE regions BY interaction_distance
clip_11 = 11 AND regs_plus
clip_12 = 12 AND regs_plus
out_plus = LITHO OPCVERIFY clip_11 clip_12 FILE opcv_file MAP out
out = out_plus AND regions
```

The *interaction_distance* is the Calibre OPCVerify distance sufficient to ensure that the results inside the region are the same as a full-chip run. This interaction distance is calculated automatically, and can be found in the Calibre OPCVerify run transcript as a line starting with “SE: INTERACTION DISTANCE.”

Limitations

- When using Calibre OPCVerify with the region iterator, there are some usage limitations.
 - Images computed with the region iterator are very close to but not exactly the same as the images computed in a full-chip run, because simulation frames are placed differently. For well-behaved models, the difference should be less than 0.1nm for 1D regions and slightly larger for 2D regions.
 - Even tiny changes in images can lead to some errors disappearing or new errors appearing. For example, if a check is “pinch target contour < 0.018” and the full-chip run finds a pinch with a “min” property of 0.01795, then the contour jitter of the region iterator of 0.1nm can lead to a new “min” value of 0.01805, causing the entire error to disappear.
 - If an error is only partially covered by the filter, its region iterator properties can differ significantly from full-chip run properties.

Description

Litho OPCVerify is a Calibre batch command. It performs a number of dense simulations on a simulation grid at various process conditions. It then performs a set of user-defined check operations on those dense simulation results.

The result of a successful Litho OPCVerify run is one or more derived layers added to the output design file. A derived layer that contains geometries, contours, or error markers is output as a layer. Empty layers are not output.

The LITHO OPCVERIFY Transcript

Litho OPCVerify adds the following transcript sections to the SVRF transcript under the heading “OPCVERIFY OUTPUT GEOMETRY COUNTS AND CLASSIFY/LIMITING”.

Tip

 The transcript section is useful because it contains a list of polygon counts for all outputs, and a concise summary of error classification.

The transcript tables follow the Calibre OPCverify run diagnostics (and the pm_classify information if you enabled it in any [Classification Block](#)). It groups the counts into the following two categories:

- Hierarchical Geometry Counts (HGC)

This is calculated as HGC = (unique + duplicates + unclassified + filtered) geometries for hierarchical layers.

- Flat Geometry Counts (FGC)

This is calculated as FGC = (unique + duplicates + unclassified + filtered) geometries for flattened layers.

The geometry count tables use the following column notation:

- “All” shows error counts before classification and limiting.
 - “All tot” is the total number of errors found.
 - “All [uncl]” is the number of unique-unclassified errors (those larger than maxsize) included in “All tot”.
- When pm_classify is specified, “All tot” is the total count after cutting up large errors. In this case, there are never any “All [uncl]” errors because they were all cut.
- The “Out unq/dup/tot” columns are the numbers of unique, duplicate, and total errors written to the OASIS or rdb file. If applicable, “Out [uncl]” is the number of unique-unclassified errors included in “Out unq”.
- If the classify block uses the “unique_unclassified_ids on” setting, “Out [uncl]” statistics are not reported.
- “All” and “Out” total geometry counts for layers with no classify or limit block are identical. For contours, the counts are after the output_window operation is applied.
- Fields marked “-” are not applicable for a layer (depends on classify options).

An example transcript might look similar to the following:

```

HGC:-----
HGC: | Hierarchical Geometry Counts for Output Layers
HGC:-----
HGC: Opcverify      =====All=====      ======Out=====
HGC: Layer Name    tot   [uncl]     (ung)  +dup  =tot)  [uncl]
HGC:-----
HGC: tile           1      -        -      -      1      -
HGC: cntr_out       910     -        -      -      910     -
HGC: brdge_cl       46      8        23     -      23      8
HGC: brdge_ncnp     46      -        -      -      46      -
HGC: pinch_nc       1290    -        -      -      1290     -
HGC: pinch_cl_11    1290    122      443    -      443      122
HGC: pinch_cl_rand  1290    122      443    -      443      122
HGC: pinch_cl_ext   1290    122      614     -      614      122
HGC: pinch_cl_unq  1290    122      614     676    1290     -
HGC: pinch_lm5000   1290    -        -      -      1290     -
HGC:-----
FGC:-----
FGC: | Flat Geometry Counts for Output Layers
FGC:-----
FGC: Opcverify      =====All=====      ======Out=====
FGC: Layer Name    tot   [uncl]     (ung)  +dup  =tot)  [uncl]
FGC:-----
FGC: tile           1      -        -      -      1      -
FGC: cntr_out       910     -        -      -      910     -
FGC: brdge_cl       46      8        23     -      23      8
FGC: brdge_ncnp     46      -        -      -      46      -
FGC: pinch_nc       1290    -        -      -      1290     -
FGC: pinch_cl_11    1290    122      443    -      443      122
FGC: pinch_cl_rand  1290    122      443    -      443      122
FGC: pinch_cl_ext   1290    122      614     -      614      122
FGC: pinch_cl_unq  1290    122      614     676    1290     -
FGC: pinch_lm5000   1290    -        -      -      1290     -
FGC:-----
=====          END OPCVERIFY GEOMETRY COUNTS          =====
=====
```

Examples

Example 1

A standard Calibre OPCverify run command that takes the layer POLY_OPCT as input and returns the result of the command “setlayer image1 = ...” as output, copied into the SVRF layer L1.

```
L1 = LITHO OPCVERIFY POLY_OPCT FILE opcverify.in MAP image1
```

Example 2

A Calibre OPCverify run with EUV-enabled library commands contained somewhere inside the command file.

```
outlayer = LITHO EUV layerlist FILE runfile MAP runfile_output
```

Example 3

This example uses the region iterator with a set of hotspots on the layer “regions”.

```
out = LITHO OPCVERIFY l1 l2 ... FILE opcv_file MAP out INSIDE OF LAYER
regions
```

Litho Model Format

Calibre OPCVerify File Format

Input for: [image_options](#)

The litho model consolidates all of the model and mask information into a single object in order to define a manufacturing technology configuration. It is intended to unify simulation conditions across Calibre tools to reduce the possibility of errors.

A litho model is loaded when its directory name is specified in an [image_options](#) block for the setlayer [image](#) command or in a post-tapeout product's [denseopc_options](#) block.

The current methods of specifying the information contained in the litho model are backwards compatible, but may be deprecated in a future release.

Litho models affect the following Calibre OPCVerify commands:

- Any [background](#), [resist_model_load](#), and [optical_model_load](#) commands in the setup command file are ignored. The values in the litho model are used instead.
- The [layer](#) command is only used to set the association-by-order of SVRF input layers to the names used in the setup file, and requires only the name alias of the layer. Other parameters for transmission and type are ignored.
- The [image](#) command uses a specified [image_options](#) command block to find its litho model and layer mapping information.

Note

 It is possible to mix the previous and litho model file versions of the [image](#) command, but it is not recommended.

Format

A litho model must conform to the following restrictions:

- The collection of files that comprise a litho model is a directory that must be in the existing [modelpath](#).
- Can be produced by Calibre WORKbench (in Calibre nmModelflow, CM1 Center, [modelflow_v2](#), and the Litho Model Tool) or created in a text editor.
- The directory must contain a text file named *Lithomodel* consisting of multiple configuration commands. The name of the directory containing the *Lithomodel* file location is the argument used by the [image_options](#) command to locate this file.

- A *Lithomodel* file uses the following format:

```
version 1
resist filename
[image_threshold value]
[etch filename]
[topo filename]
[flare filename
[black_border filename] ]
[shadow_bias filename]
[cdfa filename]
[stochastic filename]
[zplanes filename]
[{{n2r_model | n2e_model} filename]
mask n {
    background trans
    mask_layer m [TRANS transmission] [CX val] [CC val] [ALL_ANGLES]
        [CHOP_FULL_EDGE 0|1] [CHOP_NOTCHES diag | rect | all | none]
        [BIAS val] [XBIAS xval] [YBIAS yval]
        [XSHIFT xval] [YSHIFT yval]
        [DDM filename]
        [CATEGORY name]
    {optical filename[focus fnm] }+
    [dose val]
}
```

Parameters

- **version 1**
A required parameter that specifies the version of the litho model. This must be the first line in the file. Currently, the only available version is 1.
- **resist *filename***
A required parameter that specifies the resist model that is used for all exposures. Only one resist parameter is allowed in each *Lithomodel* file.
- **image_threshold *value***
An optional parameter that specifies the best constant threshold value to use, typically derived from the calibration flow. If specified, this parameter will be used in the [image](#) command when “aerial model” is specified.
- **etch *filename***
An optional parameter that specifies a VEB etch model that is used for all exposures.
- **topo *filename***
An optional parameter that specifies a topographical (topo) model that is used for all exposures.
- **flare *filename***
An optional parameter that specifies an EUV flare model that is used for all exposures.
Intended for use with LITHO EUV OPCVERIFY commands.

- **black_border *filename***
An optional parameter that specifies an EUV black border model, which is only used in conjunction with a flare model.
- **shadow_bias *filename***
An optional parameter that specifies an EUV shadow bias model file.
- **cdsa *filename***
An optional parameter that specifies a compact DSA model file.
- **stochastic *filename***
An optional parameter that specifies a stochastic compact model file.
- **zplanes *filename***
An optional parameter that specifies a filename containing additional information for special imaging conditions used for SRAF print avoidance or 3D images for toploss models.
- **{n2r_model | n2e_model} *directory_name***
An optional parameter that specifies a directory containing a machine learning model (n2r for resist, n2e for etch) to be applied during simulation. Only machine learning models supported by the simulation may be loaded through the litho model interface.

The machine learning directory should contain a *frozen_model.pb* file and a machine learning description file named *model.txt* using the following format:

```
version 1
framework tensorflow1
filename frozen_model.pb
mltype {N2R | N2E}
```

Note

 The “ml_model” or “nnam_model” parameters can also be used to specify a machine learning model, but it is preserved only for backwards compatibility.

- **mask *n* ‘{’**
A required parameter identifying the start of a mask definition block, which must be enclosed in braces ({}). At least one mask block is required for mask *n*=0. If multiple mask statements are used to define multiple exposures, further masks must use different values for *n*.
- **background *trans***
A required parameter inside the mask definition block that defines the background characteristics for the mask. It replaces the values specified for the **background** statement in the Calibre OPCVerify command file.

1. TensorFlow, the TensorFlow logo, and any related marks are trademarks of Google Inc.

- **mask_layer *m*** [TRANS *transmission*] [CX *val*] [CC *val*] [CHOP_FULL_EDGE 0 | 1] [CHOP_NOTCHES diag | rect | all | none] [BIAS *val*] [XBIAS *xval*] [YBIAS *yval*] [XSHIFT *xval*] [YSHIFT *yval*] [DDM *filename*] [CATEGORY *name*]

A required parameter that specifies the characteristics of a mask layer template inside the mask block. It replaces the values specified for the [layer](#) statement in the Calibre OPCverify command file.

At least one mask_layer statement is required (*m*=0). Additional mask_layer statements must use incrementally larger values for *m*, and are associated with layer statements inside an [image_options](#) block.

Each mask_layer statement consists of the following arguments:

- TRANS *transmission* — A keyword that specifies the transmission type of the mask_layer template. *transmission* must be one of the following types: dark, clear, atten *x*, phase90, phase270, phase180, or a *real imaginary* pair. This argument is required except when this layer will be designated as an asraf layer in the [image_options](#) block, in which case it is prohibited, because asraf layers are holes that use the background transmission value.
- CX, CC — A keyword set that specifies convex and concave cornerchop values respectively, in microns, to be applied to the mask layer before using it in simulation.

Note

 For litho model flows, corner chopping defaults to chopping right-angled corners only. This differs from non-litho model flows, where the [cornerchop](#) and the Calibre nmOPC [mask_chip_corner](#) commands chop all corners, regardless of angle, by default.

Note

 For negative SRAFs (layers of type asraf in the [image_options](#) block) the values of CX and CC must be swapped, because they model holes instead of polygon shapes. Use the original value of CC for CX and the value of CX for CC.

- CHOP_FULL_EDGE 0 | 1

A keyword that specifies the maximum size of a chopped corner. A value of 0 chops a corner that exceeds the half length of an edge on either side. A value of 1 chops a corner that exceeds the full length of an edge on either side.

- CHOP_NOTCHES diag | rect | all | none

A keyword that specifies how existing chopped corners and notches are handled before corner chop calculations.

- diag — Existing chopped corners smaller than the CC or CX values are removed before chopping.

- rect — Existing orthogonal notches at corners are removed if they have both edge lengths smaller than the corner chop length (CC or CX) and an area less than the area of the corner ($CC^2/2.0$ or $CX^2/2.0$) multiplied by the area limit scale factor set by the user environment variable LITHO_CHOP_NOTCHES_AREA_LIMIT_SCALE.

Note
 The default value of LITHO_CHOP_NOTCHES_AREA_LIMIT_SCALE is 0.88.

- all — Both existing chopped corners and orthogonal notches are removed.
- none — Removes nothing before performing corner chop calculations.
- BIAS, XBIAS, YBIAS, XSHIFT, YSHIFT — An optional keyword set that specifies geometry preprocessing characteristics.
 - When BIAS is specified in the image command as well as in this litho model, Calibre OPCVerify adds the values of BIAS together.
 - How BIAS affects holes and negative SRAFs depends on the layer type. For asraf layers, positive values for BIAS makes the shapes defining the negative SRAFs larger. This means that the holes created by the asraf cutouts will also be larger. However, if the negative SRAFs are included as holes in the main feature layer, increasing the BIAS value makes the holes smaller.
 - If BIAS is specified, XBIAS and YBIAS are not permitted, and vice versa.
 - If a mask_layer specifying XBIAS, YBIAS, XSHIFT or YSHIFT is used in an [image](#) command or in Calibre nmOPC, Calibre clones all rotated or reflected cell transforms.

Geometry preprocessing before simulation is applied in the following order:

- 1) BIAS
- 2) XSHIFT and YSHIFT
- 3) XBIAS and YBIAS
- 4) Corner chopping

Large values of XSHIFT and YSHIFT are not permitted for performance reasons. It is recommended that shifts be ≤ 1.0 microns. For larger values, use the SVRF SHIFT command.

- DDM *filename* — Specifies a DDM model to associate with the layer. The model must be located in the modelpath directory. This option cannot be used with an asraf layer.

- CATEGORY *name* — Specifies an identifying comment. This is not used by Calibre OPCverify.
- **optical** *filename* [focus *fnm*]
A required parameter specifying an optical model. At least one optical parameter is required inside a mask block to specify the nominal focus optical model. A mask block can have multiple optical statements, but each one must have a unique focus value. If no focus parameter is specified, the file is treated as the nominal optical model.
The optional focus parameter is a symbolic name for an off-focus optical model, where *f* is a positive or negative integer in nanometers. (Note the “nm” is a literal part of the argument, such as “-20nm”, and the decimal form of the integer is also accepted, such as “20.000nm”, but “20.334nm” is treated as 20 nanometers).
If focus is not specified, nominal focus (focus 0nm) is assumed.
- **dose** *value*
An optional parameter inside a mask block that defines the **relative** base dose for a mask. If not specified, 1.0 is assumed. This parameter is intended for multiple exposure setups, where the mask exposures may have different relative doses.

Caution

 When dose arguments are specified both in the litho model file and in the [image](#) command in Calibre OPCverify, or in the image or pw_condition commands in Calibre nmOPC, the actual dose used is derived by multiplying the litho model value by the dose argument in the Calibre OPCverify or Calibre nmOPC command. This may give unexpected results in regression testing.

Examples

Note

 For an example of the invocation of the litho model file, see the [image_options](#) command.

The following example code shows a basic *Lithomodel* and the corresponding OPC model directory structure.

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

The following example code shows a complex *Lithomodel* defining a double exposure (two mask statements), multiple focus values, and a different CTR threshold.

```
% cat duv10_2/Lithomodel
version 1
resist mymodel.mod
image_threshold 0.125
mask 0 {
    background clear
    mask_layer 0 TRANS atten 0.06 CATEGORY main
    optical    duv193_nom
    optical    duv193_m20x focus -20nm
    optical    duv193_p20x focus  20nm
}
mask 1 {
    background clear
    mask_layer 0 TRANS atten 0.06 CATEGORY main
    optical    duv193_nomy
    optical    duv193_m20y focus -20nm
    optical    duv193_p20y focus  20nm
    dose 1.1
}
```

Related Topics

[image](#)
[background](#)
[image_options](#)
[optical_model_load](#)
[layer](#)
[resist_model_load](#)
[ZPlanes Model Format](#)
[EUV Through-Slit Litho Model Extension](#)

ZPlanes Model Format

Calibre OPCverify File Format

The ZPlanes model describes resist 3D profiles for applications which require additional model parameters to match metrology or observed results.

A ZPlanes model is loaded when its filename is specified in a *LithoModel* file.

ZPlanes are used with other optical models. While ZPlanes use a top and bottom name convention, other files use a positive and negative naming convention. The conversion between these relies on the tone or polarity of the mask/resist process as shown below:

Table 4-4. Conversion between Positive/Negative and Top/Bottom for SRAF

	SRAF_POSITIVE	SRAF_NEGATIVE
Line Pattern	SRAF_BOTTOM	SRAF_TOP
Space Pattern	SRAF_TOP	SRAF_BOTTOM

Table 4-5. Conversion between Mask/Resist Polarity and Printing Drawn Pattern Polarity

	(Positive Tone Detection) PTD	(Negative Tone Detection) NTD
Clear field	Line pattern	Space pattern
Dark field	Space pattern	Line pattern

This requires the correct RESIST_POLARITY setting in the CM1 model file.

Format

A ZPlanes file uses the following format:

```
version 1
modelType ZPLANES
[sraf_top_plane z]
[sraf_top_dose dose]
[sraf_top_mincd cd]
[sraf_top_maxcd cd]
[sraf_bottom_plane z]
[sraf_bottom_dose dose]
[sraf_bottom_mincd cd]
[sraf_bottom_maxcd cd]
[toploss_plane z]
[toploss_slope s]
[toploss_mincd cd]
```

Parameters

- **version 1**
A required parameter that specifies the version of the ZPlanes model. This must be the first line in the file. Currently, the only available version is 1.
- **modelType ZPLANES**
A required parameter that specifies the model type. The only option is ZPLANES.
- **sraf_top_plane *z***
A conditionally required parameter that specifies the resist thickness in micrometers. This parameter is required if any sraf_top_* parameters are present. The value must be positive, as zero micrometers represents the resist bottom.
- **sraf_top_dose *dose***
An optional parameter that specifies the dose in the top resist. The default value is 1 and the value must be positive.
- **sraf_top_mincd *cd***
An optional parameter that specifies the minimum critical dimension of the top resist. The default value is -1, which means the value is disabled. The option is enabled when the value provided is positive.
- **sraf_top_maxcd *cd***
An optional parameter that specifies the maximum critical dimension of the top resist. The default value is -1, which means the value is disabled. The option is enabled when the value provided is positive.
- **sraf_bottom_plane *z***
A conditionally required parameter that specifies the resist thickness in micrometers. This parameter is required if any sraf_bottom_* parameters are present. The value must be positive, as zero micrometers represents the resist top.
- **sraf_bottom_dose *dose***
An optional parameter that specifies the dose in the bottom resist. The default value is 1 and the value must be positive.
- **sraf_bottom_mincd *cd***
An optional parameter that specifies the minimum critical dimension of the bottom resist. The default value is -1, which means the value is disabled. The option is enabled when the value provided is positive.
- **sraf_bottom_maxcd *cd***
An optional parameter that specifies the maximum critical dimension of the bottom resist. The default value is -1, which means the value is disabled. The option is enabled when the value provided is positive.

- **toploss_plane z**
Conditionally required parameter that specifies the resist thickness in micrometers. This parameter is required if any sraf_bottom_* parameters are present. The value must be positive, as zero micrometers represents the resist top.
- **toploss_slope s**
Optional parameter that specifies the slope for the sidewall angle for the height gauge.
- **toploss_mincd cd**
Optional parameter that specifies the minimum critical dimension of the top of the resist. The default value is -1, which means the value is disabled. The option is enabled when the value provided is positive.

EUV Through-Slit Litho Model Extension

Litho Model Format extension

An EUV through-slit litho model is an extended form of a litho model that supports multiple DDM and optical models (referred to as sub-litho models) at multiple x-coordinates in the EUV field.

Model Support

EUV through-slit models are still under active development and have limited cross-tool support.

- EUV through-slit litho models are currently supported only in Calibre OPCVerify (in the setlayer [image](#) command), Calibre nmOPC (in the [image](#), [pw_condition](#), and [DENSE_SIMULATE](#) commands), and Calibre nmModelflow.

In addition, these models may be specified as inputs to the RET FLARE_CONVOLVE and [shadow_bias](#) commands. This means that they can be used uniformly through a full Calibre nmOPC or Calibre OPCVerify flow.

- Through-slit litho models are only supported when running in litho flat mode (see “[processing_mode](#)” on page 208), or in the flat Calibre engine (using “calibre -drc” without the -hier switch). With the flat Calibre engine, the entire design is processed using the sub-model where the slit position is closest to the center of the design. This means that the results are only valid for relatively small clips.

All standard features of a litho model are supported, except for the following models.

- topography (topo) model
- VT5 resist models

All litho models referenced by an EUV through-slit litho model must be identical to each other in all respects except for optionally specifying different DDM and optical models. Starting with the 2019.2 release, signal libraries no longer have to be identical; there is a 0.05 radian tolerance for the theta and phi settings to allow for slightly different signal locations.

Creating and Using Through-Slit Litho Models

Through-slit litho models must be built by hand with a text editor. The sub-litho models are in the standard litho model file format and may be created or calibrated in any supported tool.

To use a through-slit litho model, simply specify its name (a directory name in the modelpath, or a full path to the directory) in place of a standard litho model in any supported command.

For imaging and OPC, the x-coordinate of the slit center must be specified using the Calibre OPCVerify setup command [euv_slit_x_center](#). This value must be specified in the OASIS coordinate system of the design being simulated. It is used to convert the relative x-coordinates specified in the litho models to absolute coordinates in the design.

The granularity of through-slit simulation is per-tile. This is sufficient because a typical EUV tile (~25um) is much smaller than any distance over which the variation of through-slit effects is noticeable. For each tile, the x-coordinate of the tile center is compared with each of the through-slit x-coordinates in the litho model list, in order to determine which sub-litho model is closest. Simulation for the tile then uses the models from that sub-litho model.

There is no model interpolation and no smoothing at tile boundaries. Because through-slit effects are typically small (<~ 2nm across the 13mm half-field), it is expected that supplying a reasonable number of models across the slit, for example 25 for the whole field, will limit any jogs at tile boundaries to less than the computational noise of simulation. If unacceptable jogs are seen, increase the number of through-slit models.

Internal Format of Through-slit Litho Models

Like standard litho models, the through-slit litho models are a directory that has a “key” file called “*Lithomodel*”. The through-slit litho model directory must contain subdirectories for each of the sub-litho models. It is also often convenient to place all of the model files being used by the sub-litho models in the through-slit directory (or softlinks to them). Then each of the sub-litho models can be built using softlinks to the models in the through-slit directory.

Compatibility Requirements for Sub-Litho Models

- All sub-litho models must be identical in all respects except that they may have different DDM or optical models.
- For a given mask and focus, all per-slit optical models must have the same optical diameter (hoodpix setting).

Performance

Using EUV through-slit litho models may increase memory usage on remotes.

EUV Information in the Transcript

When you use EUV models, the transcript includes some additional information that can help you diagnose setup problems.

When you use global litho models, the parser checks slit positions. Currently the maximum spread across all slits of a single source is 26 mm.

Messages like these indicate that a global litho model is not set up correctly:

```
...
//-- WARN: Slit position 79362.4 um is outside expected range of +/- 14000 um
//-- WARN: Slit position 48028.7 um is outside expected range of +/- 14000 um
//-- WARN: Slit position -58346.2 um is outside expected range of +/- 14000 um
...
//-- WARN: Outermost EUV slits, centered at x = -120.315968 mm and 109.965310 mm,
are 230.281278 mm apart. This exceeds the fixed maximum expected distance of
26.000000 mm. Check global litho model x values for accidental overscaling.
```

The transcript also includes information about the positions of the outermost slits when reporting the coordinates to be used for interpolating DDM libraries, as in this example:

```
//-- INFO: Global Lithomodel ct98c_thruslit_correct will use interpolated
DDM libraries across slit
//-- INFO: Outermost EUV slits, centered at x = -12.031597 mm and 10.996531 mm,
are 23.028128 mm apart.
```

Format

The EUV through-slit model format is a list of sub-litho models, each with an associated x-coordinate that is relative to the slit center. An EUV through-slit model must conform to the following formatting and syntax rules:

- The first parameter must be “version.” Subsequent statements can be in any order.
- Comment lines beginning with # are allowed at any point in the file.

```
version 1
lithotype EUV_RETICLE
[dynamic_model_generation {all | none} [optics] [ddm]
litho_model sub_dir_name_1 x xcoord_um_1
litho_model sub_dir_name_2 x xcoord_um_2
[litho_model sub_dir_name_i x xcoord_um_i]
... ]
```

Parameters

- **version 1**

A required parameter specifying the version number of the file format, which must be the first line in the file. The only available option at this time is 1.

- **lithotype EUV_RETICLE**

A required parameter declaring that this *Lithomodel* key file contains an EUV reticle definition. It must be typed exactly as shown.

- **dynamic_model_generation {all | none | ddm | optics}**

An optional parameter that controls which models are dynamically generated between slit positions. If dynamic_model_generation is not specified, the default is “ddm”. At least one of the following arguments must be specified when dynamic_model_generation is used:

- “all” means both ddm and optical models are dynamically generated.
- “none” means that no models are dynamically generated. The model for the slit position closest to each tile center is used instead.
- “ddm” means that DDM model interpolation is performed by dynamically generated based on nearby DDM libraries. This is the default behavior.
- “optics” means that optical model dynamic generation is performed using a weighted combination of all the optical kernels for the two adjacent slit positions.

When a tile is not located between two slit positions, the optical model for the nearest slit position is used.

If “ddm” or “optics” are specified alone, then the model that was not specified will not be dynamically generated.

- **litho_model sub_dir_name x xcoord_um**

A required parameter identifying a subdirectory containing a discrete litho model to be used at a slit position. There must be at least two **litho_model** lines, and a maximum of 128 **litho_model** lines are supported. *xcoord_um* is the x-coordinate in microns relative to the slit center to which this sub-model applies.

All x-coordinates must be unique. They are not required to be at fixed intervals.

Examples

Model EUV_LITHOMODEL has three slit positions.

All models are stored in the main directory for convenience (not required).

```
ls EUV_LITHOMODEL:  
-rw-r--r-- 4557 May 15 04:33 flare.fmf  
-rw-r--r-- 121 Oct 22 17:23 Lithomodel  
drwxr-xr-x 4096 Oct 22 17:39 LM_slit01  
drwxr-xr-x 4096 Oct 22 17:40 LM_slit02  
drwxr-xr-x 4096 Oct 22 17:41 LM_slit03  
drwxr-xr-x 4096 Oct 9 17:07 optical_slit01  
drwxr-xr-x 4096 Oct 9 17:10 optical_slit02  
drwxr-xr-x 4096 Oct 9 17:12 optical_slit03  
-rw-r--r-- 217 May 15 04:32 resist.mod  
-rw-r--r-- 6007432 May 22 07:05 slit01.ddm  
-rw-r--r-- 6007445 May 20 14:05 slit02.ddm  
-rw-r--r-- 6007351 May 20 12:43 slit03.ddm  
  
cat EUV_LITHOMODEL/Lithomodel:  
version 1  
lithotype EUV_RETICLE  
litho_model LM_slit01 x -9000.0  
litho_model LM_slit02 x 0.0  
litho_model LM_slit03 x 9000.0  
  
ls EUV_LITHOMODEL/LM_slit01:  
lrwxrwxrwx 12 Oct 22 17:27 flare.fmf -> ../flare.fmf  
-rw-r--r-- 136 Oct 22 17:28 Lithomodel  
lrwxrwxrwx 13 Oct 22 17:39 optical_slit01 -> ../optical_slit01  
lrwxrwxrwx 13 Oct 22 17:26 resist.mod -> ../resist.mod  
lrwxrwxrwx 13 Oct 22 17:39 slit01.ddm -> ../slit01.ddm
```

```
cat EUV_LITHOMODEL/LM_slit01/Lithomodel:  
version 1  
mask 0 {  
    background dark  
    mask_layer 0 TRANS clear DDM slit01.ddm  
    optical optical_slit01  
}  
resist resist.mod  
flare flare.fmf  
  
ls EUV_LITHOMODEL/LM_slit02:  
lrwxrwxrwx 12 Oct 22 17:27 flare.fmf -> ../flare.fmf  
-rw-r--r-- 136 Oct 22 17:28 Lithomodel  
lrwxrwxrwx 13 Oct 22 17:39 optical_slit02 -> ../optical_slit02  
lrwxrwxrwx 13 Oct 22 17:26 resist.mod -> ../resist.mod  
lrwxrwxrwx 13 Oct 22 17:39 slit02.ddm -> ../slit02.ddm  
  
cat EUV_LITHOMODEL/LM_slit02/Lithomodel:  
version 1  
mask 0 {  
    background dark  
    mask_layer 0 TRANS clear DDM slit02.ddm  
    optical optical_slit02  
}  
resist resist.mod  
flare flare.fmf  
  
ls EUV_LITHOMODEL/LM_slit03:  
lrwxrwxrwx 12 Oct 22 17:27 flare.fmf -> ../flare.fmf  
-rw-r--r-- 136 Oct 22 17:28 Lithomodel  
lrwxrwxrwx 13 Oct 22 17:39 optical_slit03 -> ../optical_slit03  
lrwxrwxrwx 13 Oct 22 17:26 resist.mod -> ../resist.mod  
lrwxrwxrwx 13 Oct 22 17:39 slit03.ddm -> ../slit03.ddm  
  
cat EUV_LITHOMODEL/LM_slit03/Lithomodel:  
version 1  
mask 0 {  
    background dark  
    mask_layer 0 TRANS clear DDM slit03.ddm  
    optical optical_slit03  
}  
resist resist.mod  
flare flare.fmf
```

Related Topics

[Litho Model Format](#)

[euv_slit_x_center](#)

RET INPUT

SVRF Command

Loads a non-SVRF file into an SVRF rule file. Use this command to load a gauge file into SVRF for use in Calibre OPCverify.

Usage

RET INPUT FILE *cmd_file* [EDGE] **MAP** *layer_name*

Arguments

- ***cmd_file***

A required argument that specifies the location and characteristics of a gauge file, using one of the following variants:

- The literal description of the input file in brackets ([])
- The name of a LITHO FILE statement with a description of the input file's contents (inline)

However, note that using an inline file descriptor for this command means that the command cannot be run concurrently.

- The name of an external file with a description of the input file

The file loading keywords are as follows:

- **read {sgd | gg | rdb } *filename***

A required argument specifying an input file. *filename* is a required file path to either a super gauge data file (sgd), gauge data file (gg), or RDB file (rdb). The path must include the filename and file extension.

Environment variables are allowed inside *filename*, for example:

```
LITHO FILE gauge_in [
    READ sgd "$::env(MY_DIR) /gauge.sgd"
    ...
]
```

The read argument can only occur once in the command file. It must also be the first line in the command file.

All properties of the input layer (if any) are copied to the output.

If rdb format is specified, Calibre OPCverify reads the specified RDB file and maps layers from the file to the MAP *layer_name*. A layer statement is not permitted when reading from RDB file format. The layer name is instead taken from the check string in the RDB file, but without the prefixes to the name (before first occurrence of ":"). If the layer in the RDB file is an edge layer, the EDGE argument must be specified before MAP. There is no way to convert an edge layer from the RDB file to a

polygon layer. Similarly, a polygon layer cannot be mapped to a DRC EDGE layer. Precision and cell structures must be the same in the SVRF rule file and in the RDB file.

- layer *layer_name* [edge] [{[vertical] [horizontal] [acute] [obtuse]} | all] [exact]

An argument that specifies a layer containing gauges. Additional keywords limit the types of gauges to load. If the sgd or gg filetype is input, layer is a required argument. Cannot be specified with rdb input. Multiple layer statements are allowed.

- *layer_name*

A keyword that specifies the name of an input layer to be generated from the sgd file. This layer name should also appear in a MAP argument, unless there is only one layer operator, in which case the MAP argument may be omitted from the RET INPUT statement.

- edge

A keyword that specifies that the output is an edge layer. If the keyword is omitted, a polygon layer is created. If an edge layer is specified, the MAP EDGE keyword must also be specified.

- vertical

A keyword that specifies output of the vertical gauges.

- horizontal

A keyword that specifies output of the horizontal gauges.

- acute

A keyword that specifies output of the 45° gauges.

- obtuse

A keyword that specifies output of the 135° gauges.

- all

A keyword specifying output of all gauges. This is the default behavior. If “all” is specified without “exact,” all the gauges from the SGD file are output.

The keywords vertical, horizontal, acute and obtuse may be specified in any combination, but they are not compatible with the “all” keyword.

- exact

Specifies that only edges within ±1 DBU tolerance from any specified directions are output. Otherwise, edges are allowed within +/- pi/8 radians angle tolerance. If “all” is specified, only gauges along primary directions (0°, 45°, 90° and 135° axes) are output with ±1 DBU tolerance.

If a layer statement is not included as part of the file description, the LITHO FILE statement returns an implicit layer. Implicit layer names must be lowercase quoted strings named as one or more of the predefined gauge layer types “vertical,” “horizontal”, “acute”, or “obtuse”, and must be specified in the MAP argument. Implicit layers are specified either singly or together as quoted strings separated by spaces (for example, “vertical horizontal”).

- **EDGE**

An optional argument that causes the output to be an edge layer. If this keyword is not specified, gauges are converted to rectangular polygons with a 1 DBU width.

- **MAP *layer_name***

A recommended argument that maps the output layer from the file read operation to the specified *layer_name*. If a MAP argument is not included, RET INPUT behaves as though you had entered the implicit layer specification MAP “all”.

Description

The RET INPUT command reads geometries from non-standard sources that are not directly supported by Calibre DRC operations and converts them into Calibre layers. The command conveys additional geometry information found in the input files to Calibre via DFM properties associated with objects in the created layers.

The RET INPUT statement is designed for loading super gauge data (sgd) files, gauge data files (gg), or RDB files (rdb).

Gauges are converted to edges when RET INPUT is used with the EDGE modifier, or to rectangular polygons with a 1dbu width when RET INPUT is used without EDGE.

For super gauge data input, every gauge (whether a polygon or an edge) has a set of associated properties added to it:

- **struct** — A string property taken from the Struct field from the super gauge data file.
- **gid** — An integer property, identifying an individual gauge, taken from the Row field.
- **gauge_id** — A string property, which is a copy of the Struct field.

Note



For more information on super gauge data files, see the *Calibre WORKbench User’s and Reference Manual*.

Examples

Example 1

This example creates a polygon layer L1, importing all gauges from the super gauge data file “*my_file.sgd*”.

```
L1 = RET INPUT FILE [
    read sgd my_file.sgd
]
```

Example 2

This example creates three layers. Layer LV contains only gauges in a vertical 45° sector. Layers LH and L2 are identical because they both use the MAP from the layer “lh” inside the LITHO FILE block. Layers LH and L2 contain horizontal gauges within ±1 DBU tolerance. The sgd file is in the parent directory of current work directory.

```
LV = RET INPUT FILE gauge_in MAP lv
LH = RET INPUT FILE gauge_in MAP lh
L2 = RET INPUT FILE gauge_in MAP lh
LITHO FILE gauge_in [
    read sgd "../my_file.sgd"
    layer lv vertical
    layer lh horizontal exact
]
```

Example 3

The following example creates an edge layer Le by importing all gauges from the sgd file with the name “*my_file.sgd*”.

```
Le = RET INPUT FILE [
    read sgd my_file.sgd
] EDGE
```

Example 4

The following example maps implicit layers from the file. Notice the lack of declared layers (no layer statements). Instead, the example uses one of the automatically defined layer types, specifying MAP “vertical” to return vertical gauges.

```
Vert = RET INPUT FILE [
    read sgd my.sgd
] MAP vertical
```

Example 5

The following example maps implicit layers in sets; multiple layer definition keywords are returned via the MAP statements.

```
LITHO FILE my_sgd [
    read sgd my.sgd
]
VertE      = RET INPUT FILE my_sgd MAP "vertical exact"
HorizVert = RET INPUT FILE my_sgd MAP "vertical horizontal"
AcuteP     = RET INPUT FILE my_sgd MAP "acute"
```

Example 6

Given an RDB file produced by the following statement:

```
my_layer_rdb { DFM RDB my_layer my_file.rdb }
```

Read the file into Calibre OPCVerify using the following statement:

```
my_layer = RET INPUT FILE [
    read rdb my_file.rdb
] MAP my_layer_rdb
```

If my_layer is an edge layer, an EDGE statement must be included:

```
my_layer = RET INPUT FILE [
    read rdb my_file.rdb
] EDGE MAP my_layer_rdb
```

Related Topics

[gauges](#)

[pwcheck](#)

Lint Warnings

The Calibre OPCVerify tool has a set of built-in Lint checks that are run on the setup command file. The Lint checker reports settings that are not necessarily errors, but might be a misuse of options or might give suboptimal quality results. Lint warnings are reported in the Calibre OPCVerify transcript.

Activating Lint Warnings

Lint warnings are active by default. If the Lint checker finds any problems, it adds the following header to the transcript:

```
// =====
// === Lint Check for OPCVERIFY Setup File ... (LITHO RUN .. ===
// =====
```

To deactivate Lint warnings, set the [log_options parsing_messages](#) parameter to “off”.

Common Lint Warnings

- The SVRF PRECISION (precision) is too high, set it ≤ 40000 to get correct results.
- Tilemicrons value of ... is not optimal for OPCVerify: values $> 100\mu m$ can increase memory usage and runtime.
- Tilemicrons value of ... is not optimal for OPCVerify: values $< 30\mu m$ can increase runtime.
- Both “simulation_consistency” and “optical_transform_size” specified: size used is from last command seen.
- Option “mask_sample_grid” should be set to “rsm.”

- DDM model(s) loaded. Background transmissivity from DDM models overrides those in setup file.
- “contour_options max_edge_merge_error” is set to ..., recommended value: 0.0002 to 0.0005.
- Non-default setting of “preprocessing_version” might result in sub-optimal performance.
- A classify_block has option maximum_error_number set to Errors occurring after this count has been exceeded are not classified. It is strongly recommended not to use maximum_error_number.
- Output_window halo > 0.5um is not recommended as it may significantly impact runtime.
- Output contour “...” has no output_window, full contour will be output (output of full contours can generate excessive data and impact performance).
- Optical model symmetry checking has been disabled by setup command: “clone_transformed_cells {decide | rotations | yes | no} no_optical_symmetry_check.”
- Setlayer deangle is deprecated because its output is inherently wrong on tile boundaries. It may be obsoleted in a future Calibre version. Use SVRF DEANGLE instead.
- Setlayer meefcheck: delta_mask is not an exact integer number of dbu. Calibre has rounded the value to an integer number of dbu.
- Setlayer meefcheck: delta_mask is < 3 dbu. Calibre has rounded the value to an integer number of dbu.
- Setlayer image: mask xshift/yshift/bias is not an exact integer number of dbu. Calibre has rounded the value to an integer number of dbu.

Lint Warnings for Calibre OPCVerify, but not EUV OPCVerify

- optical_transform_size set to ..., should be 768 for optimal performance.
- Simulation grid sampling size is set to ..., for OPCVerify should be set to 3/8/1 via “imagegrid aerial” and “final_upsample.”
- imagegrid aerial set to ..., should be set to 3 8.
- final_upsample set to ..., should be set to 1.
- Non-default setting of “rasterizer_upsample_factor” might result in sub-optimal performance.

Lint Warnings for EUV OPCVerify Only

- optical_transform_size set to ..., should be 512 for optimal performance.

- Simulation grid sampling size is set to ..., for EUV OPCVerify should be set to 2/3/1 via “imagegrid aerial” and “final_upsample.”
- imagegrid aerial set to ..., should be set to 2 3.
- final_upsample set to ..., should be set to 1.
- rasterizer_upsample_factor set to ... should be 2 1.
- Global Lithomodel ... will use interpolated DDM libraries across slit.
- Global Lithomodel ... will use the nearest DDM library across slit.

Calibre OPCverify Setup File Configuration Commands

The *cmdfile* argument to LITHO OPCVERIFY specifies a file (or an inline file) containing a *case-sensitive* command initialization block consisting of commands, specified one per line in roughly the order shown.

Tip

 Setlayer commands are described starting with the section “[Setlayer Operations Reference](#).”

Table 4-6. OPCverify Setup Configuration Commands

Command	Description
background	Defines the background transmission values of the mask for each exposure for a Calibre OPCverify command file.
bbm_options	Defines an options set that overrides one or more parameters of an EUV black border model inside a litho model for an image command.
classify_options	Sets default values for classification blocks as an ease of use utility.
clone_transformed_cells	Sets the type of transformed call placements.
collect_frame_stats	Computes frame processing statistics for Calibre OPCverify.
contour_options	Specifies the contouring style used by Calibre OPCverify for any image and Calibre nmOPC denseopc commands in this setup command file.
critical_dimension	Sets new default critical dimension values for multiple commands.
ddm_model_load	Loads a DDM model into Calibre OPCverify.
direct_input	Enables direct data entry (DDE) in Calibre OPCverify.
direct_output	Enables direct data output of OASIS files. Used with litho-flat configurations only.
dynamic_output	Instructs Calibre OPCverify to output interim layers during a simulation.
etch_imagegrid	Describes the grid size for etch model simulation purposes.
etch_model_load	Loads the specified etch model in Calibre OPCverify.
euv_field_center	Specifies the center of a EUV field.
euv_slit_x_center	Sets a placement x-coordinate for EUV through-slit models.
filter	Designates a user-defined <i>input_layer</i> as a filter layer.

Table 4-6. OPCVerify Setup Configuration Commands (cont.)

Command	Description
<code>final_upsample</code>	Sets the upsampling from the resist grid to the final grid.
<code>flare_longrange</code>	Associates a precomputed long range flare convolution file with a flare model file contained inside a litho model.
<code>flare_model_load</code>	Loads a flare model into Calibre OPCVerify. Used when the flare model is not in a litho model file.
<code>gauge_set</code>	Creates a definition block that describes how Calibre OPCVerify adds gauges to a target layer.
<code>image_options</code>	Creates a named option block containing settings for use with the image command in litho model mode.
<code>image_set</code>	Generates a group of image contours. Image sets and their auxiliary layers can be used to simplify setup file coding, and also enable some features.
<code>imagegrid</code>	Defines the contour image grid maximum size.
<code>layer</code>	Declares a layer input to a Calibre OPCVerify command file.
<code>layer_properties</code>	Specifies property names and property merge actions of an input layer.
<code>litho_model</code>	Defines an inline litho model. Used for encrypted setup files only.
<code>log_options</code>	Controls the output of various diagnostic messages in the output log.
<code>mask_sample_grid</code>	Optionally specifies the mask sampling grid. In most cases, the default setting (rsm) is sufficient.
<code>modelpath</code>	Specifies the directories to search for optical and resist models.
<code>optical_model_load</code>	Loads an optical model.
<code>optical_transform_size</code>	Overrides the default optical transform size setting.
<code>output_window</code>	Outputs a context-sensitive clip around specified errors.
<code>processing_mode</code>	Sets the processing mode for Calibre OPCVerify.
<code>progress_meter</code>	Toggles the reporting of cell processing information in the output log.
<code>promote_subframe_slivers</code>	Promotes small slivers geometries out of frames.
<code>push</code>	Performs a light push on generated output layers.
<code>pw_annotation</code>	Modifies image set error markers with extra properties for errors in each component image (pw_image) of the image set.

Table 4-6. OPCverify Setup Configuration Commands (cont.)

Command	Description
pw_annotate_options	Sets run options for the pw_annotate command.
rasterizer_upsample_factor	Sets the rasterizer upsample factor relative to the optical (Nyquist) grid.
resist_model_load	Loads a resist model into Calibre OPCverify.
save_error_center_points	Creates an output file containing the center points for error shapes.
setlayer	Creates and derives layers, depending on the options used.
shadow_bias_model_load	Loads a shadow bias model. Used when the shadow bias model is not contained in a litho model file.
simulation_deangle	Runs the deangler on all image contours.
stairstep	Controls the aerial contour stairstep behavior.
stochastic_model_load	Loads a stochastic model from the specified location.
summary_report	Creates an HTML page containing summary information and snapshots of selected errors. Includes configuration output options for snapshots and histograms.
svrf_var_import	Imports a variable from the SVRF rule file.
tilemicrons	Specifies the size of an Calibre OPCverify tile.
topo_model_load	Loads a topographical (“topo”) model into Calibre OPCverify.
zplanes_model_load	Loads a zplanes model into Calibre OPCverify.

background

Setup command

Defines the background transmission values of the mask for each exposure for a Calibre OPCverify command file.

Usage

```
background {dark | clear | {attenfactor} | {real imag}}...
```

Description

Note

 If a litho model is used, the **background** command is not required. This is because the litho model definition includes a background statement, and any explicit background command is ignored.

The background command is intended for use with non-litho model Calibre OPCverify command files.

This configuration command describes the imaging background. This is used only in [image](#) commands that do not explicitly specify a background. In other words, the background command must appear before the first image command that does not have a background specified.

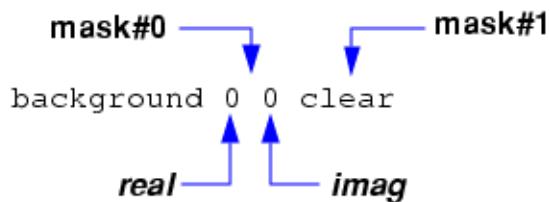
You must specify one background type per mask exposure. When multiple masks are used, the arguments must be supplied in order, with the first argument referring to Mask 0, the second referring to Mask 1, and so on.

[Figure 4-4](#) shows how to specify multiple-mask exposures using different arguments.

Figure 4-4. Multiple-Mask Exposure



Or



The first background applies to the first mask exposure, “mask#0”; the second background applies to the second exposure, “mask#1”.

Note

 The layer transmission value is tied to the background value. For example, a clear field mask does not support any layers that are defined as clear.

Arguments

- **dark**
A literal argument designating dark field imaging.
- **clear**
A literal argument designating clear field imaging.
- **atten*factor***
An argument specifying an attenuated phase-shifting background with an attenuation *factor*. This *factor* is specified as the normalized intensity that should transfer to the wafer for a clear exposure.

factor — a *required* numerical argument to the attenuated option. It is the attenuation factor for attenuated backgrounds. The accepted values are a floating point value representing a percentage. For example, for an 8% attenuated phase shift mask, the factor is specified as 0.08. The maximum allowed value is 0.36 (36%).

When Calibre LITHO tools encounter atten*factor*, they translate the factor into a real imaginary pair such that the attenuated background has a transmission value

$$\text{RE(bg)} = -\sqrt{\text{factor}}$$

$$\text{IM(bg)} = 0$$

Do not use an attenuated background with a phase180 layer, as this will cause 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:

$$\text{background} (\text{real } \text{i}\text{mag}) + \text{layer} (\text{real } \text{i}\text{mag}) = \text{transmission_value}$$

For example:

```
background 1 0
name      opt      type
chrome   visible 0 0
```

The background is specified with a (1, 0) pair, while the chrome layer is defined with a transmission value of (0,0). With clear background (1,0) and a layer specified with (0,0), the resulting transmission layer is (1,0) + (0,0) = clear (1,0).

```
background 1 0
name      opt      type
p180    visible -1 0
```

In this case, the layer p180 is defined as (-1 0). With a clear background (1,0) and a layer specified with (0,0), the resulting transmission layer is (1,0) + (-1,0) = dark (0,0).

Examples

This example shows the combination of layer and background settings needed to define an attenuated phase shift mask. The background is attenuated and the layer type is clear because light is not obstructed when passing through the polygons in the VIA layer.

```
background atten 0.08
layer VIA opc clear
```

bbm_options

Configuration command

Defines an options set that overrides one or more parameters of an EUV black border model inside a litho model for an image command.

Usage

```
bbm_options name {  
    [dose val | dose_delta val %]  
    [dose_ring_right_left_width val_um]  
    [dose_ring_top_bottom_width val_um]  
    [blade_shadow_right_left_width val_um]  
    [blade_shadow_top_bottom_width val_um]  
}
```

Description

This command defines a set of modifications (overrides) to the parameters of an EUV black border model. The overrides are used by specifying the “bbm_options *name*” parameter in a litho model-type call to the setlayer [image](#) command that uses a litho model containing a black border model. This allows per-image variation of the black border effect simulation.

All parameters except *name* are optional, and if not specified the existing value in the model is used.

Arguments

- ***name***
A required parameter specifying the unique name of the bbm_options block.
- **dose *val***
An optional parameter that specifies a dose value (in units of mJ/cm₂) that overrides the dose value in the black border model. This parameter cannot be specified with the dose_delta parameter.
- **dose_delta *val* %**
An optional parameter that specifies a percentage value by which the dose value is modified. The “%” is required. This parameter cannot be specified with the dose parameter.
- **dose_ring_right_left_width *val_um***
An optional parameter that specifies a positive value in microns that overrides the width of the dose ring at the right and left of the design.
- **dose_ring_top_bottom_width *val_um***
An optional parameter that specifies a positive value in microns that overrides the width of the dose ring at the top and bottom of the design.

- `blade_shadow_right_left_width val_um`

An optional parameter that specifies a positive value in microns that overrides the width of the blade shadow falloff region at the right and left of the design.

- `blade_shadow_top_bottom_width val_um`

An optional parameter that specifies a positive value in microns that overrides the width of the blade shadow falloff region at the top and bottom of the design.

classify_options

Setup command (see “[Classification Block](#)” on page 73 for more information).

Sets default values for classification blocks as an ease of use utility.

Usage

```
classify_options [name] '{  
    option1  
    [option2]  
    ..  
}'
```

Description

Specifying the classify_options setup command sets one or more option default values for [Classification Block](#). The following rules apply when a classify_options block is created:

- If an option specified in the classify_options block is not explicitly set in a specific instance of a classification block, the options specified in the classify_options block are used.
- Any options specified in a specific instance of the classification block overwrite the defaults, with the sole exception of the score sub-argument; score sub-argument values are merged.
- A specific instance of a classification block can be declared with an empty brace ({} {}) after defining a classify_options block. All of the classify_options block settings are used, since there are no specific instance values to override the classify_options block settings.
- A named classify_options block can now be specified after the classify keyword. If a matching classify_options *name* declaration exists, its settings are used instead of the default.

Arguments

- *name*

An optional name for a default settings block; using this option allows you to maintain multiple sets of default classification block settings.

If a *name* is specified for the classify_options block, its default settings will not be used unless explicitly requested by a classify block.

If *name* is not specified, the options specified in this block are considered the default classification options.

- *option1*

A required argument specifying any classification block option and related argument. At least one option must be defined.

- *option2*

An optional argument specifying additional classification block options and related arguments.

Examples

Given the *name*-style version:

```
classify_options one_thousand_no_dup{
    context poly_target via
    halo 0.1
    maxsize 0.25
    suppress duplicates
    reflections_rotations_match yes
    worst 1000
}
```

and the anonymous version:

```
classify_options {
    halo 0.2
    maxsize 0.40
    keep duplicates
}
```

In this example, all the following classify block options used are the ones specified in the named *classify_options* block.

```
setlayer layer1 = ..... classify one_thousand_no_dup {}
```

In this example, some of the classify block options are set from the anonymous *classify_options* block, but one option (keep duplicates) is overwritten.

```
setlayer layer2 = ..... classify {
    context poly_target via
    suppress duplicates
}
```

Related Topics

[Classification Block](#)

clone_transformed_cells

Setup command

Sets the type of transformed call placements.

Usage

```
clone_transformed_cells {decide | yes | no | rotations | x_axis_reflection | y_axis_reflection}
```

Description

This command causes the SVRF compiler to use the Calibre OPCVerify parser to set the type of transformed cell placements cloned when the HDB is built.

The SVRF parse-time optical source symmetry is reported after the “--- RULE FILE = ...” line in one of following formats:

```
// OPTICAL MODEL 'model_name...' SOURCE SYMMETRY: ALL  
    (Annular or Quadrupole Type)  
// OPTICAL MODEL 'model_name...' SOURCE SYMMETRY: XY_ONLY (Dipole Type)  
// OPTICAL MODEL 'model_name...' SOURCE SYMMETRY: ASYMMETRIC
```

If there are multiple models, the worst case model symmetry is also reported:

```
// WORST OPTICAL MODEL SOURCE SYMMETRY: ...
```

This option only affects designs that have an asymmetrical source. Because cell orientation can have a significant impact on results, this command sets how Calibre OPCVerify handles cell orientation for the aerial image in a hierarchical design.

Note

 Calibre OPCVerify determines symmetry from a reconstructed TCC matrix formed from the input optical kernels. Therefore, source maps need not be present to correctly compute model symmetries; only the kernels are needed. However, this means that any optical model without kernels present (such as inlined optical models) are determined to have no symmetry, because its kernels are not accessible at SVRF parse time. This can potentially cause excessive cloning and an increase in runtime.

The runtime source symmetry report appears after the “ADJUSTED TILEMICRONS = ...” line in the transcript. The HDB may not show the requested cloning state due to:

- Other LITHO or SVRF command(s) requesting cloning
- LAYOUT command overrides such as LAYOUT CLONE ROTATED PLACEMENTS NO or LAYOUT CLONE TRANSFORMED PLACEMENTS NO

Tip

 LAYOUT CLONE ROTATED PLACEMENTS NO will not block a “clone_transformed_cells yes” request. “LAYOUT CLONE TRANSFORMED PLACEMENTS NO” must be specified to block all cloning.

The HDB clone state is reported as:

HDB CELL TRANSFORM CLONING: {NO | ROTATED | YES}

If the HDB clone state is insufficient to protect against incorrectly rotated or mirrored placements of asymmetrically processed cells, the following warning message is issued:

```
//-- WARN: HDB Cell Transform Cloning: ??? is incompatible  
with Optical Model Symmetry: ???
```

Note

 The classify {reflections_rotations_match decide} option depends on the runtime optical model source symmetry, not the SVRF parse-time source symmetry.

Arguments

- **decide**

A required mode argument that if specified clones all transformed cells if at least one of the following is true:

- A non-symmetric optical model is being used
- The classify {reflections_rotations_match no} block command (see “[Classification Block](#)” on page 73 for more information) has been specified
- The [shift](#), [image shift](#), or [image](#) bias commands have been specified

Clones only rotated cells if at least one of the following is true:

- An x/y symmetric optical model (such as cross dipole) has been specified
- The classify {reflections_rotations_match reflections} block command has been specified
- dense opc with orientation-dependent processing (such as horizontal or vertical tagging) has been specified

If none of the above conditions are true, no asymmetric cells are cloned.

- **yes**

A required mode argument that if specified clones all transformed (rotated or mirrored) cells as if they were asymmetric. Can increase cell count by up to 8 times (equivalent to the SVRF [LAYOUT CLONE TRANSFORMED PLACEMENTS YES](#) command).

- **no**

A required mode argument that if specified skips cloning asymmetric cells.

- **rotations**

A required mode argument that if specified clones only cells that were rotated. Using this mode can increase cell count up to 2 times (equivalent to the SVRF **LAYOUT CLONE ROTATED PLACEMENTS YES** command).

- **x_axis_reflection**

A required mode argument that if specified clones rotated cell placements and reflections around the X-axis.

- **y_axis_reflection**

A required mode argument that if specified clones rotated cell placements and reflections around the Y-axis.

collect_frame_stats

Setup command

Computes frame processing statistics for Calibre OPCVerify.

Usage

collect_frame_stats {on | off}

Description

This command computes tile and frame processing statistics for any command that performs frame simulation, such as [image](#).

(Frames are much smaller than tiles and are always square; tiles follow cell boundaries and can assume quite exotic shapes. When the simulation engine computes an image for a tile, it covers the tile with frames.)

Initial and final reports with these statistics are generated.

The initial processing report is prefixed in the processing report by “IF:” It contains the following sections:

- ESTIMATED PROCESSED AREA — This is the estimated layer output area (total sum of “filter” areas).
- ESTIMATED PROCESSED AREA INCLUDING OVERHEAD — This is the effectively processed area, which includes tile overhead (total sum of “size filter by extra1” areas).

Note



These two items above are appended with an additional report string: “(Aaval%/
Hhval%/Eeval%/Ffval%)”. This breaks the values into the percentage contributions from the various kinds of litho processing. “A” is array-mode processing, “H” is hierarchical-mode processing, “E” is EUV-mode processing, and “F” is litho-flat mode processing.

- ESTIMATED TILE EFFICIENCY FACTOR — This is equal to the processed area/processed area including overhead.
- ESTIMATED FLAT AREA — This is an estimation of the amount of data that would be processed in a litho-flat run. It is calculated by just using the rectangular database extent. It is the area of the smallest rectangle enclosing all shapes in the input database. This may therefore be an overestimation.
- ESTIMATED FLAT AREA INCLUDING OVERHEAD — This is an estimation of the effectively processed area in a litho-flat run. It is derived by dividing the rectangular database extent into tiles, sizing them up by extra1, then summing the resulting areas.

- ESTIMATED FLAT TILE EFFICIENCY FACTOR — This is equal to the estimated flat area divided by the estimated flat area including overhead. This number should increase with increasing tile size, and decrease with increasing the interaction distance.
- ESTIMATED PROCESSED AREA BENEFIT FACTOR — This is equal to estimated flat area including overhead divided by the estimated processed area including overhead. It quantifies the estimated benefit derived from using hierarchical processing.

The final processing report is labeled “FF:” in the log. It contains the following information:

- ACTUAL PROCESSED AREA — This is the layer output area (total sum of “filter” areas).
- ACTUAL PROCESSED AREA INCLUDING OVERHEAD — This is the effectively processed area, which includes tile overhead (total sum of “size filter by extra1” areas). The actual hierarchical areas may differ from the estimated hierarchical areas.

Note

 These two items above are appended with an additional report string: “(Aaval%/
Hhval%/Eeval%/Ffval%)”. This breaks the values into the percentage contributions
from the various kinds of litho processing. “A” is array-mode processing, “H” is
hierarchical-mode processing, “E” is EUV-mode processing, and “F” is litho-flat mode
processing.

- TILE EFFICIENCY FACTOR — This is equal to actual processed area/actual processed area including overhead.
- HIER BENEFIT FAC (FLAT/ACTUAL PROC AREA) — This is equal to estimated flat area including overhead divided by the actual processed area including overhead. It quantifies the estimated achieved hierarchical efficiency.

Each concurrent group of simulation commands reports the following items:

- OUTPUT LAYERS — These identify the concurrent group.
- PROCESSED AREA — This value may be different from the ACTUAL PROCESSED AREA INCLUDING OVERHEAD section, because each concurrent simulation typically needs less input than the whole deck. For example, a setlayer command whose output is not used inside the same setlayer deck may need to process data inside (filter+command_extra) instead of (filter+global_extra).
- FRAME COUNT — For image commands, this is the total number of frames simulated for this concurrent group. For denseopc commands, this is the total number of frames processed for the “nominal” pw condition in the first iteration.
- FRAME KEEP — This is the size of the effective frame area, excluding frame overhead.

- **HIER FRAME EFFICIENCY FACTOR** — This is the (ideal number of frames assuming no tile or cell overhead) / (actual number of frames).

Arguments

- **on | off**

An optional argument that toggles collection of frame processing statistics. The default is on.

Examples

The following figure shows a sample design layout.

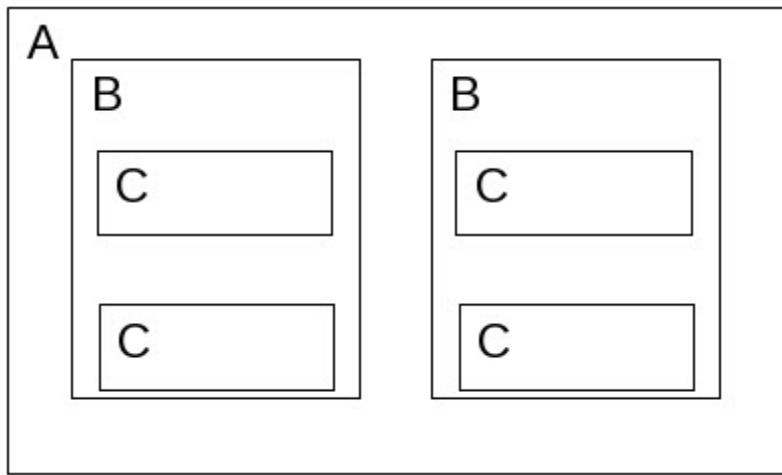


Figure 4-5 shows how hierarchical tiling is calculated.

Figure 4-5. collect_frame_stats, Hierarchical Tiling

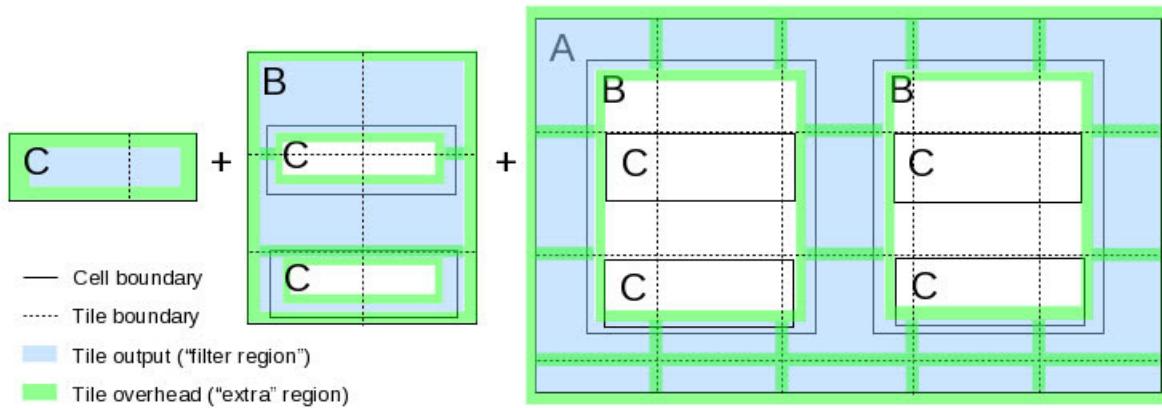
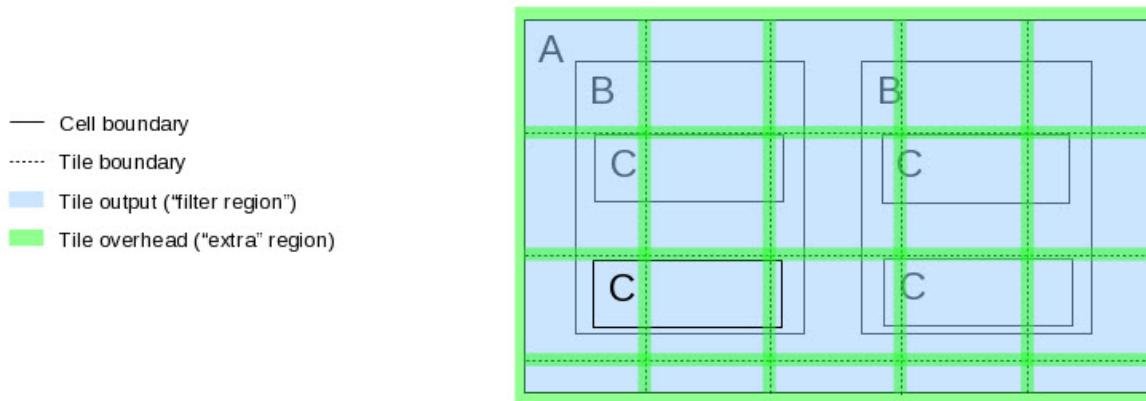


Figure 4-6 shows how flat tiling is calculated.

Figure 4-6. collect_frame_stats, Flat Tiling



An example of the initial report is as follows:

```

IF: ESTIMATED PROCESSED AREA = 70.63 square microns (A0%/H100%/E0%/F0%)
IF: ESTIMATED PROCESSED AREA INCLUDING OVERHEAD = 114.31 square microns
(A0%/H100%/E0%/F0%)
IF: ESTIMATED TILE EFFICIENCY FACTOR = 61.79 %
IF: ESTIMATED FLAT AREA INCLUDING OVERHEAD = 33.61 square microns
IF: ESTIMATED FLAT AREA = 12.32 square micron
IF: ESTIMATED FLAT TILE EFFICIENCY FACTOR = 36.66 %
IF: ESTIMATED PROCESSED AREA BENEFIT FACTOR = 0.29 X
  
```

An example of the final report is as follows:

```

FF: ACTUAL PROCESSED AREA = 44.10 square microns (A0%/H100%/E0%/F0%)
FF: ACTUAL PROCESSED AREA INCLUDING OVERHEAD = 82.67 square microns (A0%
H100%/E0%/F0%)
FF: TILE EFFICIENCY FACTOR = 53.35 %
FF: HIER BENEFIT FAC (FLAT/ACTUAL PROC AREA) = 0.41 X
FF: OUTPUT LAYERS: contour1 contour2 contour3
FF: PROCESSED AREA = 47.06 square microns
FF: FRAME COUNT = 1
FF: FRAME KEEP = 6.873400 microns
FF: HIER FRAME EFFICIENCY FACTOR = 93.35%
  
```

contour_options

Setup command

Specifies the contouring style used by Calibre OPCverify for any image and Calibre nmOPC denseopc commands in this setup command file.

Usage

contour_options

```
[interp_algo {cubic | lagrange}]  
[interp_degree integer]  
[max_edge_merge_error err_microns] [edge_snap_factor factor]  
[max_segment_length length_microns]
```

Arguments

- `interp_algo {cubic | lagrange}`

An optional argument that specifies the mode of subpixel interpolation used for contour generation. The cubic mode (the default) is faster but less accurate, and lagrange is slower but more accurate (when `interp_degree` is also specified with a value greater than or equal to 5).

- `interp_degree integer`

An optional argument that specifies the interpolation degree. The higher the value, the slower the interpolation function is. This option is only used by the ‘lagrange’ `interp_algo` option. The default value is 5.

- `max_edge_merge_error err_microns`

An optional argument that specifies almost collinear edges in contours will be combined into a single edge as long as this does not require moving any contour vertex more than `err_microns`. This setting reduces the number of edges in the contour, speeding up simulation commands and all further operations using the results. The default value is 1 dbu, expressed in microns. (Previous to the 2016.2 release, it was 0.0001 (0.1nm)).

When the dbu setting is $\geq 2 * \text{err_microns}$, almost collinear edges are not combined.

Tip

 When `max_edge_merge_error` is not set to 0, the final contour may contain segments longer than `max_segment_length`. In order to achieve the best possible contour accuracy, set `max_edge_merge_error` to 0. However, Siemens recommends against this for full chip runs, because a setting of `max_edge_merge_error` 0 may significantly slow Calibre OPCverify and increase memory consumption.

- `edge_snap_factor factor`

An optional argument that modifies the behavior of the `max_edge_merge_error` algorithm around almost collinear edges.

This argument instructs Calibre OPCVerify to snap X-coordinates of the contour vertices from a 1 dbu grid to a coarser grid with a step of k dbu, where k is the largest integer number that is smaller than the max_edge_merge_error value in dbu times two times *factor*.

For example, with a PRECISION setting of 5000, a *factor* value of 1.0, and a max_edge_merge_error setting of 0.0003, the step of this grid will be 3dbu = 0.6nm. For a max_edge_merge_error setting of 0.00025 it will be 2dbu = 0.4nm.

The default is 0 (off); allowed values are between 0.0 and 1.0 (maximum snapping).

The snapping never produces errors larger than the limit specified by the max_edge_merge_error option.

For Calibre OPCVerify operations with contours, snapped contours are faster to process in downstream operations, saving time for Calibre OPCVerify checks like pinch, bridge, measure_distance, measure_epe, area_compute, and so on.

Note

 Snapping may occasionally degrade contour consistency inside almost vertical 1D contour regions; instead of keeping at mostly the same X coordinate, the contour may vary slightly between two close X values.

- max_segment_length *length_microns*

An optional argument that adjusts the maximum length of an individual contour segment in order to ensure that the contour point density can resolve sub-nanometer contour features. Typical segment lengths will vary from *length_microns*/2 up to *length_microns*, so a setting of 0.001 (1 nm) results in segments of 0.5nm to 1.0nm. The default value is infinity.

Although max_segment_length has a maximum value of infinity (DBL_MAX or $1e^{308}$ um on a typical machine), setting max_segment_length to a value larger than $\sqrt{2} \times (\text{final pixel size})$ disables the feature, because line segments are left at their initial size.

If the value specified is less than max(0.1nm or 3 dbu), it is adjusted to that value.

critical_dimension

Setup command

Sets new default critical dimension values for multiple commands.

Usage

critical_dimension cd_micron

Description

The **critical_dimension** setup command is designed to globally specify new default values for multiple Calibre OPCVerify commands when the defaults may not be appropriate for your technology node. It also sets defaults for certain commands that did not originally have them.

Caution

 Using this command alters multiple command defaults and may cause unexpected effects on existing rule file outputs.

Table 4-7. critical_dimension New Settings

Operation(s)	Argument	Default Without critical_dimension	CD-Based Default
area_compute	max_extent	—	3*CD
area_overlay	max_extent	—	3*CD
area_ratio	max_extent	—	3*CD
bridge, pinch	max_tolerance	0.15 um	1*CD
	separation	—	3*CD
	max_edge output_expand small_extra_printing small_holes	max_tolerance	1*CD
	cd_search_dist	2*max_tolerance	4*CD
center_shift	max_extent	—	3*CD
contour_diff	max_search	—	0.5*CD
	contour_diff_spacing	0.01 um	0.2*CD
dofcheck	max_search	—	0.5*CD
	dof_spacing	0.01 um	0.2*CD
end_cap	max_search	0.05 um	1*CD
	max_edge	0.2 um	3*CD

Table 4-7. critical_dimension New Settings (cont.)

Operation(s)	Argument	Default Without critical_dimension	CD-Based Default
extra_printing, not_printing	max_extent	0.15 um	1*CD
gate_stats	cd_min	—	0.5*CD
	cd_max	—	1.5*CD
gauges	max_size	—	1.5*CD
measure_cd, measure_cdv	cd_max	—	1.5*CD
	max_search	—	0.5*CD
measure_cross_section	max_extent	—	3*CD
measure_distance	separation	—	3*CD
	max_edge	0.1 um	1*CD
measure_epe	min_featsize	—	1*CD
meefcheck	max_search	—	0.5*CD
	meef_spacing	0.01 um	0.2*CD
nilscheck	max_search	—	0.5*CD
	nils_spacing	0.01 um	0.2*CD
Classification Block	halo	—	2*CD
	anchor_max_snap	0.2 um	2*CD

Arguments

- **cd_micron**

A required argument that defines the critical dimension (CD) unit for the rule file. CD is defined as the minimum width of a feature or a minimum space between features found in the layer(s) in the rule file.

ddm_model_load

Setup command

Loads a DDM model into Calibre OPCverify.

Usage

```
ddm_model_load ddm_model_name {filename |  
‘{’ inline_model ‘}’}
```

Description

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

Note

 For more information on DDM models, see the *Calibre WORKbench User’s and Reference Manual*.

This command is ignored if a litho model is present; the DDM file is set inside the Lithomodel file instead (see “[Litho Model Format](#)” on page 111 for more details).

Arguments

- ***ddm_model_name***

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

- ***filename***

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

- **‘{’ *inline_model* ‘}’**

A required argument defining a DDM model. The entire specification must be enclosed in braces ({}).

direct_input

Setup command

Enables direct data entry (DDE) in Calibre OPCVerify.

Usage

```
direct_input '{  
    vboasis_path filename  
    {[input input_number] layer number [datatype] [layer number datatype]}  
    [vboasis_precision_multiplier {n/d | AUTO}]  
    ...}...  
'}
```

Description

This command enables direct data entry (DDE) in Calibre OPCVerify. It works in conjunction with litho-flat mode and automatically activates it when a direct_input block is found.

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

There is a one-to-one mapping between the layers specified in this command and the layers specified using the **layer** command. This mapping is based on the layer order.

Arguments

- **vboasis_path *filename***

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

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

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

input input_number — An optional keyword set that specifies which input layer from the setup file to map. The index starts at 1 for the first layer listed in the setup file. The specified layer is mapped to the VB:OASIS layer **number**, allowing you to use the input layers out of order. The value you specify for *input_number* cannot exceed the number of setup layers. If this keyword set is not specified, layers are assumed to be in the same order as the setup file.

Either all layers must use input *input_number*, or none of them.

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

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

- **vboasis_precision_multiplier {n/d | AUTO}**

An optional argument that modifies the PRECISION of *filename*. It may be specified once per vboasis_path. Either AUTO or a ratio must be specified.

n/d — Multiplies the PRECISION by *n/d*, and then scales the data from the file by the same amount to retain the same absolute scale as the original PRECISION setting.

AUTO — Attempt to infer the correct ratio. AUTO exits from the parser with an error if the ratio is not a rational number or would cause arithmetic overflow.

Examples

```
LAYOUT PATH "dummy.oas"

LAYER layout_window 1
layout_window { COPY layout_window } DRC CHECK MAP layout_window 1000
layer dummy1 998
layer dummy2 999

M1 = LITHO OPCVERIFY FILE "opcv" layout_window dummy1 dummy2 MAP M1_copy
M1 { COPY M1 } DRC CHECK MAP M1 43

M1_OP = LITHO OPCVERIFY FILE "opcv" layout_window dummy1 dummy2 MAP
M1_OP
M1_OP { COPY M1_OP } DRC CHECK MAP M1_OP 56

LITHO FILE opcv [
    direct_input {
        vboasis_path test.oas
        input 1 layer 43
        input 3 layer 46
        vboasis_path "test2.oas"
        input 2 layer 33 100
    }
    background dark
    layer M1 visible clear # this will be layer 43 in the file "test.oas"
    layer sraf visible clear # this will be layer 33, datatype 100, in the
                            # file "test2.oas"
    layer CONT hidden clear # this will be layer 46 in the file "test.oas"
    setlayer M1_copy = copy M1
    setlayer M1_OP = denseopc M1]
```

direct_output

Enables direct data output of OASIS files. Used with litho-flat configurations only.

Usage

```
direct_output '{'
    {vboasis_path vboasis_filename [append]
        {output name layer layer_number [layer_datatype]}}{...}} [...]
        [{vboasis_output_primary source}]{...}
    '}'
```

Description

This command enables direct data output from Calibre OPCVerify. It must be run in litho-flat mode.

Enabling direct_output re-routes the specified layers mapped out of Calibre OPCVerify to a direct output file. In the output design, these layers are generated as empty layers. The layers mapped out to a direct output file must be mapped out using setlayer statements, and they cannot be optimized out by the Calibre DRC compiler (they have to be output to a dummy result file).

One or more vboasis_path blocks may be present.

All remotes must be able to access the path specified in *vboasis_filename*, and that path has to point to the same physical directory. For example, /net/machine1/export/local/out.oas should work, but /tmp/out.oas might not. Siemens EDA suggests using absolute paths where possible, since on some machines a path may be different from on another machine.

Limitations

- Classification blocks, properties output, and the output_window command are not supported in Calibre OPCVerify scripts that use direct_output.
- A local host directory must be defined in the remote file in order for direct_output to work in Calibre MTflex mode.
- This statement can appear only once per litho setup file. Use multiple vboasis_path arguments in a single statement to output to multiple files.

Arguments

- **vboasis_path** *vboasis_filename* [append]

A required argument specifying a VBOASIS filename for output. If the optional “append” argument is specified, the file is written to the end of the specified filename. If the file does not exist, it is created.

- **output name layer layer_number [layer_datatype]**

A required argument specifying one or more output layers. This argument is used for both VBOASIS and online modes. It designates an output layer that maps to a specified VBOASIS or online layer number. Adding the layer datatype is optional.

- **vboasis_output_primary source**

An optional argument that can be specified once per direct_output command. It specifies the source of the top cell name in *filename*. Use one of the following:

HDB — The topcell name from the HDB.

DIRECTIN — The top cell hname from the first vboasis_path input.

FILE — Same as DIRECTIN.

explicit *name* — Explicitly names the top cell.

Examples

```
LAYOUT PATH "clip.oas"
LAYER M1 100
M1_copy = LITHO DENSEOPC FILE "opencv" M1 MAP M1_copy
M1_copy { COPY M1_copy } DRC CHECK MAP M1_copy 43
M1_img = LITHO DENSEOPC FILE "opencv" M1 MAP M1_img
M1_img { COPY M1_img } DRC CHECK MAP M1_img 45 LITHO FILE opcv [
    direct_output {
        vboasis_path "test.oas"
        output M1_copy layer 43
        output M1_img layer 45
    }
    background dark
    layer M1 visible clear
    setlayer M1_copy = copy M1
    setlayer M1_img = image M1
]
```

dynamic_output

Setup command

Instructs Calibre OPCVerify to output interim layers during a simulation.

Usage

```
dynamic_output {'  
    [path dyn_out_path]  
    [format {oasis | rdb}]  
    [max_frequency mfreq]  
    [max_file_queue mqueue]  
    layers layer_name1 [layer_name2 ...] '{' layer_keywords '}'  
    [layers layer_name1 [layer_name2 ...] '{' layer_keywords '}']  
    [...]  
}'
```

Description

Tip

 Dynamic output blocks produce no output if Calibre is not specified with the -turbo switch “calibre -drc -hier **-turbo** myruledeck.svrf”.

When specified, this setup command causes Calibre OPCVerify to save interim design files during a run. It is designed for developers who want to examine error results prior to the final result files.

- For limited layers, dynamic output files will contain more information than the final layer due to the fact that less important errors generated in the beginning of the run are filtered out from final output layer after the tile/cell loop.
- For classified layers, only errors that do not interact with tile/cell/hierarchical boundaries in each tile are saved to the dynamic output files.
- For classified, unique-only layers, a unique error may appear multiple times in the dynamic output file sequence if errors generated in the later part of processing have a smaller (score, cell_id, x, y) four-tuple than the ones that came before it.

The dynamic_output command has two sections: a configuration section followed by one or more **layers** blocks. Only layers specified in the *layer_name* list will be dynamically output. If no **layers** blocks are specified, the dynamic_output command has no effect.

Notes

- An error is generated if a specified layer name is not a Calibre OPCVerify layer.
- A warning is generated if a specified layer name is not one of Calibre OPCVerify’s output layers.

- Each layer name can only be present in a single **layers** block only.
- All layers from the layers list will be stored together in one file sequence.
- In an OASIS file, layers will be numbered in order of appearance in the **layer_name** list starting with 1.

Arguments

- path *dyn_out_path*

An optional argument that specifies a data dump directory relative to where the Calibre tools were invoked, or an absolute directory if the argument starts with a forward slash (/). If the last subdirectory in the specified string does not exist, it will be created. Dynamic output files will be created in *dyn_out_path* using the following naming convention:

YYYYMMDD.HHMMSS.rdb

where:

- *YYYYMMDD* - year, month and day when file was created
- *HHMMSS* - hour, minute and second when file was created

Default value: _opcverify_dynamic_output in the current working directory

- format {oasis | rdb}

An optional argument that specifies the format of the dynamic output files. ‘oasis’ results (the default) occupy less disk space, but are not human-readable. OASIS files do not include hierarchy and store the same information as results database (RDB) files.

- max_frequency *mfreq*

Specifies the minimum time between dumps; dumps will be generated no more than once every *mfreq* seconds. Default: 600.

- max_file_queue *mqueue*

Specifies the default number of most recent non-empty files to keep in the dynamic output directory. This value may be overridden in a **layers** block. If *mqueue* is 0, all files will be kept in the directory. Default: 0.

- layers *layer_name1* [*layer_name2* ...] {*layer_keywords*}

A required argument that specifies one or more layers to be output using the conditions defined in the **layers** block. The layers block is enclosed by braces ({}) and is a set of *layer_keywords*:

- file_prefix *string*

An optional keyword that specifies a string to prefix each layer in this layers block with. If there is only one **layers** block, this parameter is optional. If there are two or more **layers** blocks, each additional block must have a unique file_prefix to avoid overwriting output files.

- o max_file_queue *block_mqueue*

An optional keyword specifying that only the *block_mqueue* most recent files will be kept in output directory. If this keyword is not specified, the default global value (0) will be used.

- o max_dump_layer_edges *max_dump_layers*

An optional keyword that specifies only the first *max_dump_layers* edges will be stored for any single layer in the **layers** block. Granularity is per tile. If the keyword is not specified, the default global value (50000) will be used.

Examples

```
setlayer i1 = image optical o1 aerial 0.3
setlayer err_i = measure_distance i1 internal < 0.04 separation 0.1
setlayer wrn_i = measure_distance i1 internal > 0.04 < 0.06 separation 0.1
dynamic_output {
    max_frequency 600
    layers err_i {
        file_prefix err_
    }
    layers wrn_i {
        file_prefix wrn_
        max_dump_layer_edges 10000
    }
}
```

etch_imagegrid

Setup command

Describes the grid size for etch model simulation purposes.

Usage

etch_imagegrid *etch_grid_microns*

Description

Defines the grid size to use for etch simulations only.

- If the **imagegrid** option is explicitly specified, **etch_imagegrid** defaults to the value specified for **imagegrid**.
- If **etch_imagegrid** is specified, however, it overrides the supplied value for **imagegrid** for etch simulation (see **vob_simulate**) only.
- If neither **etch_imagegrid** nor **imagegrid** are specified, the grid value used for etch simulation is derived from the etch model's parameters.

Caution

 The optimal value for **etch_imagegrid** is 0.008 microns and specifying a value above 0.008 microns is not recommended (especially for technology below 14nm).

Arguments

- ***etch_grid_microns***

A required argument that specifies the etch grid size.

etch_model_load

Setup command

Loads the specified etch model in Calibre OPCVerify.

Usage

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

Description

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

Arguments

- ***etch_model_name***

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

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

- ***filepath***

A required argument that specifies a path to the etch model.

- '{' *inline* '}'

A required argument that specifies the etch model specifications inline; the braces ({}) are required and must surround the inline model definition. For the definition of VEB etch models, see the “[Variable Etch Bias \(VEB\) Model File Format](#)”reference page in the *Calibre WORKbench User’s and Reference Manual*.

euv_field_center

Setup command

Specifies the center of a EUV field.

Usage

euv_field_center *x_um* *y_um*

Description

This optional command specifies the center of an EUV field for purposes of centering additional black border models containing the “field_size” keyword. The EUV field is defined as the region surrounded by the REMA blades. The EUV field size is not required to correspond to the slit center. In this case, the field center is located on the right side of the reticle, while the slit center is in the center of the reticle, with an appropriate “field_size” in the black border model.

Arguments

- ***x_um***

A required argument specifying the x-coordinate center of the EUV field in microns. It must use the coordinate system for the input design file for the current Calibre OPCverify run.

- ***y_um***

A required argument specifying the y-coordinate center of the EUV field in microns. It must use the coordinate system for the input design file for the current Calibre OPCverify run.

euv_slit_x_center

Setup command

Sets a placement x-coordinate for EUV through-slit models.

Usage

euv_slit_x_center [-field] *x_um*

Description

This command is required for usage of EUV through-slit Litho Models and shadow bias models. The coordinate is used only for offset of through-slit and shadow bias models. Other EUV field offsets (for example, for flare maps) must be specified separately.

Note

 Specifying `euv_slit_x_center` is required for all usages of shadow bias models, even if this keyword is set to 0

Arguments

- **-field**

An optional argument that specifies that the `euv_slit_x_center` is relative to the specified field center rather than relative to the global coordinates of the layout. This means the location of the `euv_slit_x_center` would be centered at the field center, and any `x_um` offset is calculated relative to the field center.

- ***x_um***

A required argument specifying the x-coordinate center of the EUV field in microns. It must use the coordinate system for the input design file for the current Calibre OPCVerify run.

Related Topics

[EUV Through-Slit Litho Model Extension](#)

filter

Setup command

Designates a user-defined *input_layer* as a filter layer.

Usage

filter *input_layer*

Description

Filter layers are used by other operations to limit the scope of the command in relation to the filter layer. All filter operations are performed using context from outside the filter (out to the promotion distance), so that inside the filter all simulations are performed exactly as though the filter were not present (except for slight rasterization differences).

Arguments

- *input_layer*

A required argument specifying the layer to use as a filter. This layer must be one of the input layers to OPCverify.

Examples

After declaring the layer (TARG), the new layer is defined as a filter layer:

```
layer TARG visible dark
filter TARG
```

final_upsample

Setup command

Sets the upsampling from the resist grid to the final grid.

Usage

final_upsample *value*

Description

The **final_upsample** command sets the upsampling factor from the resist grid to the final contour grid for contours and intensity interpolation. The contour grid pixel size is $1/\textit{value}$ of the resist grid pixel size.

Note

 For VT5 resist models, resist and contour grids are identical, and the **final_upsample** setting is ignored.

Arguments

- ***value***

A required argument that sets the final upsampling value. Accepted values are 1 or 2. The default value is 1 if the [imagegrid](#) command was specified, or 2 if it was not specified.

Related Topics

[imagegrid](#)

[optical_transform_size](#)

flare_longrange

Setup command

Associates a precomputed long range flare convolution file with a flare model file contained inside a litho model.

Usage

```
flare_longrange litho_model_name
{none | constant value | {png pngfile [{constant {average | minimum | maximum}} | {delta value [%]}] [flare_map_shift [-field] x_um y_um]}} [name name_string]
```

Description

The flare_longrange command specifies how intra- and inter-field flare effects are handled.

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

Note

 Flare effects are considered separate from the EUV black border effect, and thus a black border model is simulated by default if it is specified in the Litho Model, irrespective of what is specified by flare_longrange. The black border model can be disabled separately by specifying the “no_black_border” flag in setlayer image or the denseopc image option.

Arguments

- *litho_model_name*

A required argument that specifies the name of a directory in [Litho Model Format](#) that contains a flare model with long range flare. The model name must either be found in the modelpath or be a fully qualified path.

- **none | constant *value* | {png *pngfile* [{constant {average | minimum | maximum}} | {delta *value* [%]}] [flare_map_shift *x_um* *y_um*]}}**

A required argument set that specifies where the result of a pre-computed convolution of a fractal kernel (long range part) present in the loaded model and the currently processed layout is stored.

- **none** — No pre-computed long range convolution will be used when doing simulations using this model.
- **constant *value*** — The long range flare is a constant percentage of clear field transmission given by *value*, where a value of 1 is equal to 100%. The intensity is calculated as: $I_{\text{constant_flare}} = k_f * I_{\text{clear}} * \text{value}$, where k_f is the coefficient from the flare model, and I_{clear} is the clear field value taking into account both optics and ddm mask transmission.

- **png *pngfile*** [{constant {average | minimum | maximum} } | {delta *value [%]*}]
[flare_map_shift *x_um* *y_um*] — The file Calibre OPCVerify uses is the PNG file located at *pngfile*. This is a file that is generated using [RET FLARE_CONVOLVE](#). *pngfile* may be an explicit path, in the current directory, or the file found by searching in the modelpath.

If the “constant” parameter is specified, the long range flare is represented as a constant value that is a property of the long range flare map. Currently supported properties are “average” which uses the average value in the flare map, “minimum” which uses the smallest value anywhere in the flare map, and “maximum” which uses the largest value anywhere in the flare map. When the “constant” parameter is used, this constant flare value is used for the entire simulated region, independent of the original flare map extent.

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

If the flare_map_shift parameter is specified, the flare map is shifted by the amounts specified for *x_um* and *y_um* in microns before being used in the run. Set this to the negative of any coordinate shifts applied to the layout before creating the flare map. The default value for flare_map_shift is 0 0.

If flare_map_shift -field is specified, the shifts are considered relative to the field center instead of the global coordinates stored in the field map.

Note

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

If the “delta” parameter is specified, the values in the flare map are adjusted. If “delta value %” is specified, a percentage adjustment is made, up (positive) or down (negative). If only “delta value” is specified, a constant offset is added, up or down. In this case, the value is interpreted as a fraction of clear field transmission, as specified above under constant value. If “delta value” causes the flare value to fall locally below zero, it is truncated to zero at that point.

- **name *name_string***

An optional argument to specify a name for the flare_longrange command. This argument enables the use of multiple flare_longrange specifications for the same Lithomodel. The *name_string* must be unique for each flare_longrange command specified in the setup file. Named flare specifications may be used in the litho model mode of the setlayer image command (with the “flare name” argument), but are not supported anywhere else.

Related Topics

[flare_model_load](#)

flare_model_load

Setup command

Loads a flare model into Calibre OPCverify. Used when the flare model is not in a litho model file.

Usage

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

Description

Note

 This command requires you to include the EUV keyword in the [LITHO OPCVERIFY](#) statement (so that it appears as LITHO EUV OPCVERIFY). It also requires an additional license (caleuv) to function.

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

Arguments

- ***name***

A required argument specifying the name to use for the flare model inside the OPCverify command file.

- ***filename* | '{' *inline_model_definition* '}'**

A required argument defining the flare model file. If *filename* is specified, it is loaded from the specified location. If an *inline_model_definition* is specified, it must be enclosed in braces.

- **longrange {none | png *pngfile*}**

A required argument set that specifies a location that contains the result of a pre-computed convolution of a fractal kernel (long range part) used by the loaded flare model and the currently processed layout.

- **none** — Uses simulations without pre-computed long range convolution.
- **png *pngfile*** — Uses the specified png file representing the long range flare convolution results, generated beforehand with [RET FLARE_CONVOLVE](#).

Examples

```
flare_model_load FM fmodel longrange png png1.png
```

gauge_set

Configuration command

Creates a definition block that describes how Calibre OPCVerify adds gauges to a target layer.

Usage

```
gauge_set name {  
    target_layer targ_layer [[not] {inside | outside} filter_layer]  
    [critical_dimension cd]  
    [algorithm {projecting | corner | all}]  
    [min_space_fac minfac]  
    [max_space_fac maxfac]  
    [min_projection_fac projfac]  
    [gauge_spacing dist]  
    [gauge_extend_fac extfac]  
    [min_separation_fac sepfac]  
}  
'}
```

Description

This command defines how a set of gauges are placed on a target layer for intensity-based checks such as imin_check or imax_check.

A gauge_set does not specify whether the gauges are internal (across the lines) or external (across the spaces). For each check that uses a gauge_set, internal and external gauges are determined automatically by Calibre OPCVerify using the name of the check and the polarity (clear or dark field) of the image being checked.

Arguments

- ***name***
A required argument that specifies a user name for the gauge set.
- ***target_layer targ_layer* [[not] {inside | outside} *filter_layer*]**
A required argument specifying the drawn target layer where the gauges will be placed. If the optional *filter_layer* argument is also specified, gauges are only generated for the regions that meet the conditions set for the filter.
targ_layer and *filter_layer* must be defined in the setup file before the gauge_set command.
- ***critical_dimension cd***
An optional argument specifying the smallest line width or line spacing in the design (depending on which is being checked) in user units. This argument is required if the **critical_dimension** command has not been specified. Otherwise, gauge_set uses the value specified in the critical_dimension command.

Note

 Setting critical_dimension correctly is important. If the gauge_set will be used for imin_check or imax_check, the command automatically decides to measure across lines or across spaces based on *both* whether the check is imin or imax *and* on whether the image is clear or dark.

You should set critical_dimension to the smallest line or space width in the design, depending on which will be measured. For many designs, the smallest line width and smallest space width are substantially different from each other.

- algorithm {projecting | corner | all}

An optional argument that controls how gauge_set places gauges on the target.

- project — Gauges are placed on the target between pairs of parallel facing edges only, at intervals defined in the gauge_spacing argument. This is a geometric, rule-based algorithm and does not use the intensity grid for comparisons.
- corner — Gauges are placed extending outwards from manhattan corners to nearby geometry. This is a model-based algorithm that uses the intensity grid for analysis to find the optimal gauge placements.

For the corner option, the Imin and Imax values for all outward angles within a wide arc centered on the corner bisector are directly investigated. Gauges are then placed in directions with the worst properties (in this case, local extrema with respect to the angle from the corner). If there are no worst properties in any angular direction tested, then no corner gauges are needed, because the adjacent projecting gauges are guaranteed to catch any errors.

- all — Combines the behavior of projecting and corner options. This is the default behavior.

- min_space_fac *minfac*

An optional argument that defines the smallest spacing between parallel edges that is eligible for analysis. The *minfac* should be specified as a multiple of the *cd*. Use this argument to avoid measuring around small notches.

The default value is 0.5, and *minfac* must be less than 1.0.

- max_space_fac *maxfac*

An optional argument that defines the largest spacing between parallel edges that is eligible for analysis. The *maxfac* should be specified as a multiple of the *cd*.

The default value is 1.5, and *maxfac* must be greater than 1.0.

- min_projection_fac *projfac*

An optional argument that defines a minimum length such that gauges are not generated between edge pairs whose mutual projection length (parallel overlap) is shorter than the specified length. The *projfac* argument must be a multiple of the *cd*.

Use this argument to prevent long, shallow kinks from generating gauges. The projection lengths of edge pairs are checked before any filter is applied. Thus, if a pair of edges has a sufficient projection length, it will not be rejected if the filter leaves only tiny projecting segments of the edges.

The default value is equal to the setting for the min_space_fac argument. The value specified for projfac must be less than 1.0.

- gauge_spacing *dist*

An optional argument that defines the spacing interval between the measurement gauges that are placed between pairs of parallel edges, in user units.

The default value is 0.015.

- gauge_extend_fac *extfac*

An argument that defines the distance that gauges will extend at each end beyond the region defined by the edge pair, as a multiple of the CD.

The default value is 0.3. The value specified for extfac must be between 0.2 and 0.8, inclusive.

Note

 Values larger than the default may be necessary to achieve valid results when analyzing images that are substantially shifted relative to the target. If an intensity check produces any invalid gauge results (flagged by special “invalid” property values), try increasing gauge_extend_fac.

- min_separation_fac *sepfac*

An argument that sets the minimum separation value permitted between the ends of a generated gauge. The CD value is multiplied by the specified sepfac value to set the separation value; the default is 0.0, which is effectively a zero separation check. Using fractional values (0.5) results in separation values less than the CD (for example, 0.5 is half the CD).

This argument is intended to prevent the creation of corner-cutting gauges where the two ends of a gauge are on the same polygon corner and very close together. Using values that are too small still generates corner-cutting gauges. Using values that are too large can result in suppressed gauges. A reasonable value for this argument is between 2.0 and 3.0.

Related Topics

[imax_check](#)

[imin_check](#)

image_options

Configuration command

Creates a named option block containing settings for use with the [image](#) command in litho model mode.

Usage

```
image_options name {'  
    litho_model litho_model_name  
    layer name visible [mask n]... [mask_layer m]  
        [layer name filter]  
        [layer name underlying {active | poly | finfet}]  
        [layer name etch_underlying n]  
        [layer name asraf [mask n]... [mask_layer m]]  
    ...  
}'
```

Arguments

- ***name***

A required argument that specifies a user-defined name that is used to refer to this block from the [image](#) command. The option block must be enclosed in braces ({}).

- ***litho_model* *litho_model_name***

A required argument that specifies a directory name. This directory must contain all optical, resist, etch, flare, and topo models to be used for the image options block, and a *Lithomodel* file in [Litho Model Format](#).

- **layer *name* visible {[mask *n*]...} [mask_layer *m*]**

A required argument that specifies a visible layer. At least one visible layer is required.

- mask *n* — Optionally designates one or more exposure masks to associate with the layer.

If no mask parameter is specified but mask_layer *m* is specified, mask 0 is assumed as the default.

- mask_layer *m* — Maps this layer to a corresponding mask_layer *m* statement inside the Lithomodel file. The Lithomodel file defines the optical and resist models, the mask transmission, geometry preprocessing, and optional ddm model for the layer.

The mask_layer parameter also defaults to 0 if not specified. For a simple litho model with only one mask_layer defined inside the Lithomodel file, you may ignore this parameter.

Only one mask_layer parameter can be specified per layername line, but multiple mask *n* arguments can be associated with that mask_layer *m* parameter. This is intended for use with topo models.

Note

 If a layer is mapped to a mask_layer that specifies XBIAS, YBIAS, XSHIFT or YSHIFT, this will cause Calibre OPCVerify to clone all rotated or reflected cell transforms.

- layer *name* filter

An optional argument that specifies a filter layer. Only one filter layer may be specified for this command. When specified, simulation is performed to generate contours only inside the filter layer.

- layer *name* underlying {active | poly | finfet} [mask_layer *n*]

An optional argument that specifies a topo model underlying layer and its type. Multiple-density configuration users may need to also specify the mask_layer for an underlying layer (see “[Topographical Models in Calibre OPCVerify](#)” on page 56 for more information).

- layer *name* etch_underlying *n*

An optional argument that specifies a VEB underlying etch layer with an id of *n*, pointing to an SKERNEL in the VEB etch model that has an underlying=*n* parameter. For the current release, only *n*=1 is supported, and the VEB model can therefore only have SKERNELS specified with an underlying=1 parameter.

- layer *name* asraf {[mask *m*]...} [mask_layer *n*]

An optional argument that specifies a negative SRAF definition layer. Negative SRAF definition layers contain positive features specifying the positions of negative SRAFs, which are holes in the main mask features. These polygons are cut out from the main features before simulation. If an asraf layer is specified, there must also be at least one positive feature layer defined inside the litho model as well.

Examples

Example 1

The following example defines an image_options block “opt1” that designates two layers and a filter layer to be used from the litho_model directory 32nmPOLY, and how the image_options block can be used with image statements.

```
image_options opt1 {
    layer poly.main visible mask_layer 0
    layer poly.sraf visible mask_layer 1
    layer filter_lay filter
    litho_model 32nmPOLY # This is the litho model
}
setlayer a = image opt1
setlayer b = image opt1 dose 1.1 focus -20nm
setlayer c = image opt1 bias 0.002
```

Example 2

The following example defines a shift for one mask in a double exposure:

```
image_options A {
    layer mainfeature visible mask 0 mask_layer 0
    layer mainfeature2 visible mask 1 mask_layer 0
    litho_model 32nmPOLY
}
setlayer x = image A
setlayer a = image A mask0 xshift .002
```

Example 3

The following example defines an image_options block for a topo model. While the topo model itself is only referenced inside the litho model the following items indicate that this is a topo model:

- A layer declaration with multiple mask declarations
- A layer declaration with an underlying argument.

```
image_options A {
    layer Implant visible mask 0 mask 1
    layer Active underlying active mask 2
    litho_model models_directory
}
```

image_set

Configuration command

Generates a group of image contours. Image sets and their auxiliary layers can be used to simplify setup file coding, and also enable some features.

Usage

`image_set image_set_name = generator_spec`

Description

This command defines a set of image contours using different exposure conditions. The sets are created using generators, which are either range-based or list-based.

The set can be constructed from previously generated images, or the images can be generated automatically. After an image set is defined, some auxiliary layers (.inner, .outer, and .nominal) are also generated, and can be used later in the setup file.

An `image_set` may be used subsequently in `setlayer` commands that support `image_sets`, or in `output_window`. An `image_set` is used in these commands by coding `image_set_name` as a parameter in place of a list of layer names (and where applicable, exposure settings) in the existing syntaxes.

Note

 The “.inner” and “.outer” images cannot be used as input to intensity-based checks because the associated aerial image intensity is not defined.

When used, `image_set` is equivalent to generating those images explicitly and passing them explicitly to the command. For `output_window`, `image_sets` may be used in `context_layer` lists but not in `filter_layer` lists.

- Every component image in a set has an individual layer name created for it. That name may be used freely in the Calibre OPCVerify command setup file after the `image_set` is defined, and may be mapped back to SVRF by coding “MAP `name_of_one_image_from_set`”. The image layer names are generated using rules discussed below.
- There is no way to map an entire `image_set` to SVRF with one MAP statement. Each component must be mapped individually by name.
- The maximum number of images allowed in a set is 500; attempting to specify more than 500 returns a parser error. However, be aware that image contours are the largest component of Calibre OPCVerify memory usage. Remote CPUs typically have limited memory, and if you define a large number of images, you may need to reduce the tile size.

Image Layer Names Generated for Range-Based Sets

On execution of the [image_set](#) command, generators create new image contours with layer names created as concatenated strings using each element that you specified.

Tip

 When a component of a range-based image_set is to be mapped back to SVRF, the MAP statement should specify its layer name as defined for this command. If you are unsure of the exact name, try mapping back the layer using the closest approximation you can make. If the name is wrong, the Calibre OPCverify parser error includes a list of all valid layer names and you can choose the correct one.

If the generated layer name contains a minus sign, its name in the SVRF code must be enclosed in quotes (“myset_b-0.005”, not myset_b-0.005). SVRF generates a syntax error if a minus sign is used in a layer name without quotes.

For the “full” and “value” generators, the image layer names are of the form:

“*image-set-name_range-type-codevalue[_range-type-codevalue]...*”

(for example: “myset_b-0.005”).

where

- *range_type_code* is the first letter of the range_type:
 - focus — f
 - dose — d
 - bias — b
 - aerial — a
 - thr_delta — t
- *value* is the value used for this layer instance for the range type.

Additionally, the following rules are applied:

- focus values do not include the nm suffix.
- thr_delta values that were specified with a percent (%) keep the % suffix.
- No plus (+) signs are generated for positive values (for example: “myset_f40_t2.5%”) but minus (-) signs are generated for negative values (for example: “myset_f-40nm_t2.5%”).

- For floating-point range types (“dose”, “bias”, “aerial”, and “thr_delta”), the number of decimal places is the largest precision that the user specified. For example, for “range dose from 1.000 to 1.06 step 0.02”, the names would be similar to “myset_d1.060”.
- If values with no decimal places are specified, like “5%” or “5.%”, no decimal point is produced in the names.

The order of range types in the output strings is set using a priority list, so that output names are always in the same order for consistency. The following ordering is always followed no matter how the range types were originally specified:

“f”, “d”, “b”, “a”, “t”

For example, coding “image_set myset = values A (dose focus) (0.94 -40) ...” creates image names be similar to “myset_f-40_d0.94”, which reverses the order of focus and dose from the way range’s order.

Arguments

- ***image_set_name***

A required argument that specifies the name for the image set. Names may be up to 256 characters and must be ASCII characters. *image_set_names* may not be the same as any other layer names in the setup file.

Once the image set is created, the following three derived auxiliary contour layers can be referenced from it:

- ***image_set_name.inner***

image_set_name.outer

These are respectively the AND and OR of all the image contours defined by the set. These layers may be passed to any subsequent command which accepts image contours as input, or may be mapped back to SVRF. The .inner and .outer layers are automatically generated when their names are used.

- ***image_set_name.nominal***

If the *image_set* contains a nominal image, for example (focus=0, dose=1.00, bias=0.0) in a focus/dose/bias set, then you can refer to that image contour by the special name “*image_set_name.nominal*”. Image_sets that do not contain a nominal image do not have a .nominal contour.

- ***generator_spec***

A required argument that specifies the contents of the *image_set*. There are three kinds of generator, each with its own syntax. Generators that create images (“full” and “values”) are only supported when using litho models, and require an [image_options](#) block (as the *image_options_name* argument) in the same format used for [image](#).

The three generator types are:

- **full *image_options_name range_spec ...* [‘*other_argument...*’]**

At least one **range_spec** combination is required. If multiple ranges are given, images are generated for *every* combination of values in the **range_spec**. For example, specifying three **range_spec** lists consisting of M (3 focus), N (5 dose), and P (4 bias) values creates MxNxP (3x5x4=60) images.

- **values *image_options_name* ‘(*range_type ...*)’ ‘(*value [value...]*)’ [‘*other_argument...*’]**

Creates a list of arbitrary sets of values for one or more range types. It requires a list of item sets that *must be enclosed in parentheses*. The first set specifies the range types being varied, for example “(focus dose)”. The other sets contain corresponding values for the specified types. One image is generated for each set of parameter values. For example, “(-40nm 1.08) (-35nm 1.06)” applies a focus of -40nm and a dose of 1.08 to the first image contour, and a focus of -35nm and a dose of 1.06 to the second set.

- **list *list_item...***

Creates an image set using only the specified list items. **list_items** are previously-generated contour layers or image sets. Arguments to a “list” generator may contain duplicate items.

“list” generators do not create new image layers. Their components retain the names with which they were originally generated.

Calibre OPCVerify recognizes list-based sets as being equivalent to a **values** generator if both of the following are true:

- The set contains only images defined either by single-mask [Litho Model Format image](#) commands or by other range-based image_sets.
- The set varies one or more setlayer image parameters that are valid **range_type** arguments, with other parameters kept constant across the set.

If a **list**-based set can be so classified, it is considered valid input to any setlayer command that requires range-based sets of that type. If not, the set is classified as free-form. Free-form sets can be used only for *image_set_name.inner*, *image_set_name.outer* and in [output_window](#) commands. Free-form sets produce a parser error in setlayer commands that require a range-based set.

list_item is a required argument used in the list generator. Each **list_item** is separated by spaces, and must be one of the following:

- the name of an image layer, created with setlayer [image](#)
- the name of a previously declared **image_set**

- **range_spec**

A required argument that is used in the **full** generators to produce a range of values for a specified **range_type**. The available range specifiers are:

- **range_type from val1 to val2 step step**

Generates values starting from **val1**, incremented by **step**, until the next value would be greater than **val2**.

- **range_type from val step step count N**

Generates values by iteratively adding **step** starting from **val** and stopping when **N** values have been created.

Note

 **val1** must be less than **val2**, and **step** and **N** must be positive numbers.

- **range_type ('val ...')**

Generates the range explicitly with the values contained in the parentheses. Values do not have to be in ascending numerical order, but no duplicates are permitted.

- **range_type**

A required argument used in the **full** and **values** generators to specify which **image** parameter is being varied. Supported parameters are:

- focus — Note that some commands may use the “focus” argument specified for **image_set** for that command’s “defocus” argument.
- dose
- bias
- aerial
- thr_delta — “thr_delta” arguments require that all or none of the values specified for it have the “%” suffix. Also, if **thr_delta** is varied, the keywords “aerial model” are automatically inserted into the “other_arguments” if not explicitly specified.

For generators that vary multiple parameters the choice of range types to vary must result in a valid **image** syntax. For example, specifying the pair “aerial” and “thr_delta” is invalid, because both specify an aerial image threshold (as mentioned above, “thr_delta” automatically adds “aerial model”).

- **val**

A required argument used in various generators. It is defined as one of the following types of values:

- **integer_number[nm]** — Used for focus. This value is taken from the litho model when one is supplied.

- **floating_point_number** — Used for dose, bias, and aerial.
- **floating_point_number[%]** — Used for thr_delta. If a % symbol is specified for one argument, it must be specified for all of them.

Tip
 Any *val* arguments specified must be valid for the corresponding *range_type* in setlayer [image](#).

- ‘(*other_argument* ‘)’

An optional list of one or more arguments that can be any arbitrary setlayer [image](#) argument from the types (focus, dose, aerial, dose, bias, or thr_delta), excluding ones that are already specified inside a range generator. The list must be enclosed within parentheses ().

- If the focus argument is specified here, it must include the ‘nm’ suffix for any values.
- If the thr_delta argument is specified as one of the *range_type* arguments, it is optional to code “aerial model” in the *other_argument* list. If not found, it will be added.

Examples

Example 1 — A “list” generator using non-Litho Model syntax images

First, a set of image contours is generated using standard methods.

```
setlayer n1_ctrl = image optical opt-40 dose 0.94 resist_model r1
setlayer n2_ctrl = image optical opt-40 dose 1.00 resist_model r1
setlayer n3_ctrl = image optical opt-40 dose 1.06 resist_model r1
setlayer z1_ctrl = image optical optnom dose 0.94 resist_model r1
setlayer z3_ctrl = image optical optnom dose 1.06 resist_model r1
setlayer p1_ctrl = image optical opt+40 dose 0.94 resist_model r1
setlayer p2_ctrl = image optical opt+40 dose 1.00 resist_model r1
setlayer p3_ctrl = image optical opt+40 dose 1.06 resist_model r1
```

Next, we use `image_set` commands with the “list” type to create two sets; “`def_imgs`”, which contains the defocus image contours (`n1_ctrl`, `n2_ctrl`, `n3_ctrl`, `p1_ctrl`, `p2_ctrl`, and `p3_ctrl`) and “`all_imgs`”, which contains “`def_imgs`” plus `z1_ctrl` and `z3_ctrl`.

```
image_set def_imgs = list n1_ctrl n2_ctrl n3_ctrl p1_ctrl p2_ctrl p3_ctrl
image_set all_imgs = list def_imgs z1_ctrl z3_ctrl
```

Next, we code verification checks using the built-in functions of `all_imgs`. The `min_pinch` check is used with the contour defined as `all_imgs.inner` (which is the derived AND of the contours), and the `max_bridge` check is used with the contour defined as `all_imgs.outer` (which is the derived OR of the contours).

```
setlayer min_pinch = pinch target all_imgs.inner < 0.055 ...
setlayer max_bridge = bridge target all_imgs.outer < 0.055 ...
```

Finally, an output window is defined using the all_imgs set of contours around the results of the min_pinch and max_bridge commands.

```
output_window all_imgs halo 0.5 around min_pinch max_bridge
```

Note the following:

- all_imgs is used in [pinch](#), [bridge](#) and [output_window](#), but may not be used in [pwcheck](#), because it is an explicit list that contains non-litho model image commands.
- all_imgs contains eight image contours; six from def_imgs and two more added during the definition of the all_imgs image set.

Example 2 — A “list” generator using Litho Model syntax images

Note

 In these examples, “A” is a predefined [image_options](#) block.

Similar to the first example, a set of contours is generated using standard methods. This example features the use of the image_options block “A”, which uses the optical model from the litho model defined inside A.

```
setlayer n1_ctr = image A focus -40nm dose 0.94
setlayer n2_ctr = image A focus -40nm dose 1.00
setlayer n3_ctr = image A focus -40nm dose 1.06
setlayer z1_ctr = image A           dose 0.94
setlayer z2_ctr = image A           dose 1.00
setlayer z3_ctr = image A           dose 1.06
setlayer p1_ctr = image A focus +40nm dose 0.94
setlayer p2_ctr = image A focus +40nm dose 1.00
setlayer p3_ctr = image A focus +40nm dose 1.06
```

An image_set block is defined using the image contours as a list:

```
image_set all_imgs = list n1_ctr n2_ctr n3_ctr z1_ctr z2_ctr \
z3_ctr p1_ctr p2_ctr p3_ctr
```

The list is used as part of a [pwcheck](#) command as shown:

```
setlayer pw = pwcheck target dose_latitude 0.04 dof 0.03 \
images all_imgs max_search 0.05 ...
```

Notes:

- The set is recognized as a (focus dose) set and may be used in setlayer pwcheck.
- setlayer pwcheck automatically reads the focus and dose values from the image_set.

Example 3 — A single-range set for bias

In this example, a “full” generator is created, but only for a single range, “bias”.

```
image_set bias_set = full A bias (-0.002 0.0 0.002) \
(focus -40nm dose 1.06)
```

Notes:

- The resulting set bias_set has 3 images. The image layer names are “bias_set_b-0.002”, “bias_set_b0.000”, and “bias_set_b0.002.”
- All three images are generated with the “other_arguments” constants focus -40nm and dose 1.06.

Example 4 — A “full” generator for focus and dose used in setlayer pwcheck, bridge, pinch, and output_window

In this example, a “full” generator is used to create a set of nine images; each of the three elements in the explicit dose range (0.94, 1.00, and 1.06) is paired with the three elements in the from X to Y step Z range for focus (-40, 0, 40) to create an image contour.

```
image_set fd = full A dose (0.94 1.00 1.06)
          focus from -40nm to 40 step 40
```

The image_set.inner contours are used in a [pinch](#) check, the image_set.outer contours are used in a [bridge](#) check, and the entire set is used for the pwcheck command.

```
setlayer min_pinch = pinch target fd.inner < 0.055 ...
setlayer max_bridge = bridge target fd.outer < 0.055 ...
setlayer pw_err      = pwcheck target dose_latitude 0.04 dof 0.03 \
images fd max_search 0.05 ...
```

Finally, the results of all three checks are output using the [output_window](#) command.

```
output_window fd fd.inner fd.outer halo 0.5 \
              around min_pinch max_bridge pw_err
```

Notes:

- The image layer names generated are similar to “fd_f0_d1.00”.

Example 5 — A “values” generator for focus, dose, and aerial.

In this example, the “values” generator is used to create image contours varying the focus, dose, and aerial arguments with the specified value lists. Six sets of arguments (enclosed in parentheses) are supplied and produce six image contours when the command is run.

```
image_set set3 = values A (focus dose aerial)
          (-40 1.00 0.171) (0 1.02 0.18) (40 0.92 0.189)
          (-40 1.04 0.184) (0 1.03 0.19) (40 1.07 0.189)
```

Notes:

- The image layer names are similar to “set3_f-40_d1.00_a0.171”.

imagegrid

Setup command

Defines the contour image grid maximum size.

Usage

imagegrid *max_grid_microns*

imagegrid aerial *numerator denominator* [*coarse numerator denominator*]

Description

The imagegrid command typically sets the upsampling from the optical grid to the grid on which resist images are computed. This command operates in conjunction with the [NO TITLE](#) and [optical_transform_size](#) commands. It obeys the following rules:

- When VT5 models are being used, the resist grid and the contour generation grid are identical.
- When VT5 models are not used, the contour grid is defined relative to the resist grid by final_upsample.

In most cases, the default value for imagegrid is aerial 1 2. However, note that there is a special case where the rules are different. If both of the following are true:

- VT5 models are NOT used
- imagegrid IS specified but final_upsample is NOT specified

then imagegrid is treated as the total upsample from the optical grid to the *contour* grid. The upsampling to the resist grid is then computed by working backwards, using the default value of final_upsample. The default value for final_upsample is 1, unless imagegrid specifies an upsample value greater than 3, in which case a value of 2 is used for final_upsample. So the effect of this rule is only visible in a case such as the one shown in [Example 2](#).

The setting of imagegrid has a large effect on simulation runtime, so this command is for expert users only. Its effect permeates all aspects of the simulation time except for the optics portion.

Arguments

- ***max_grid_microns***

A required argument that defines the upper bound of the contour generation grid, in microns.

- If this option is not specified, CM1 and CM1-concurrent aerial contours are produced on a grid roughly equal to $\lambda/(16*NA(1+\sigma))$.
- If this option is not specified, VT5 and VT5-concurrent aerial contours are produced on a grid roughly equal to $\lambda/(8*NA(1+\sigma))$.

The lambda, NA and sigma values are determined by the optical model used in the calculation of a particular contour.

Note

 The **max_grid_microns** format above is typically used in older setup files. For new files, **imagegrid aerial** is recommended.

- **aerial numerator denominator** [coarse numerator denominator]

A required argument for aerial mode. Specify the numerator and denominator arguments to define the resist computation grid. The relation between the arguments sets the upsampling factor relative to the Nyquist grid. The default is imagegrid aerial 1 2.

If the optional coarse parameter is specified, it sets a coarser upsampling factor from the optical to the resist grid. The specified numerator and denominator can be equal to or coarser than the base setting.

The coarse parameter is used by and required for multiple-grid OPC configurations.

Examples

Example 1

The following code sets the imagegrid value to 1/2 of the Nyquist grid pixel:

```
imagegrid aerial 1 2
```

Example 2

A case where the imagegrid is treated as the total upsample from the optical grid to the contour grid.

```
(CM1 resist model)

optical_transform_size 256

imagegrid aerial 1 4

(no final_upsample)
```

The grids used for the example above are: optical 256, resist 1024, contour 1024. Given the following command settings:

- `optical_transform_size o`
- `imagerid aerial num den`
- `final_upsample f`

The following conversions are applied:

- $\text{optical grid} = o$
- $\text{resist grid} = o * (\text{den}/\text{num})$

- contour grid = $o * (den/num) * fs$

Related Topics

[final_upsample](#)

[optical_transform_size](#)

layer

Setup command

Declares a layer input to a Calibre OPCverify command file.

Note

 The layer command is order dependent, and must appear before any command that uses the declared layer.

Usage

layer *layer_name* [{hidden [*transmission*] | visible *transmission*} [*mask_num dose*]]

Description

Names and describes optical properties of a Calibre OPCverify input layer. Every layer to be used in Calibre OPCverify commands (see “[Setlayer Operations Reference](#)”) must be declared in the setup parameters using this command.

This command is *not* the same command as the Calibre OPCpro [layer](#) command.

Note

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

Note the following:

- The layer command must appear before the first setlayer command.
- The *transmission* and *mask_num* values are only used for the implicit version of the image command and are only needed for **visible** layers. The explicit version of the image command includes these values as part of the syntax, and a litho model file (see “[Litho Model Format](#)” on page 111) similarly contains the values as part of its mask_layer syntax requirement. If these parameters are supplied when not needed, a Lint warning is generated and the parameters are ignored. Lint warnings indicate setup issues that, once corrected, can better optimize the performance of the run.
- The *transmission* and *mask_num* values are never required for the **hidden** layer type. They can still be supplied, but Calibre OPCverify ignores them and generates a Lint warning.
- When using the implicit form of the [image](#) command, Calibre OPCverify does not simulate layers designated as the **hidden** layer type.
- **hidden** layers can otherwise be used in all Calibre OPCverify commands.

Arguments

- *layer_name*

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

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

Note

 As stated above, all further arguments to this command are only used when you are using an [image](#) command with an implicit format. If you are using only explicit image commands or a litho model (see [Litho Model Format](#)), only *layer_name* is required, and all other arguments are ignored with a warning.

- *hidden [transmission]*

An optional argument specifying a hidden layer, which can be used in any operation except image-based. The optional *transmission* argument, which uses the same syntax as the visible layer transmission argument, is ignored unless the named layer is used for a Calibre OPCVerify command that uses the information.

- *visible transmission*

An optional argument specifying a visible layer, which are used in optical simulations. The *transmission* argument defines the mask layer's optical transmission type, representing the electric field transmission of a plane wave coming in to the mask.

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](#) command in this chapter for more information about physical layer transmission in relation to background. The following table lists the valid *transmission* choices.

Table 4-8. layer Command transmission Argument Settings

Transmission	Description
clear	Defines the layer's optical transmission to be 100%. (clear features)
dark	Defines the layer's optical transmission to be 0%. (dark features)

Table 4-8. layer Command transmission Argument Settings (cont.)

Transmission	Description
<i>real imag</i>	<p>Defines the layer's optical transmission to be as specified by the real and imaginary value pair. Note that the attenuated syntax for the transmission is recommended over the REAL,IMAGINARY pair syntax.</p> <p>If you specify background with the <i>real imag</i> pair for a specified layer transmission value, the resulting transmission value for the output layer with a REAL,IMAGINARY pair is resolved relative to the background pair. The layer transmission value is resolved according to the following equation:</p> $\text{background (real imag)} + \text{layer (real imag)} = \text{transmission_value}$ <p>For example:</p> <pre>background 1 0 name opt type chrome visible 0 0</pre> <p>The background is specified with a (1, 0) pair, while the chrome layer is defined with a transmission value of (0,0). With a clear background (1,0) and a chrome layer specified with (0,0), the resulting transmission layer is (1,0) + (0,0) = clear (1,0).</p> <pre>background 1 0 name opt type p180 visible -1 0</pre> <p>In this case, the layer p180 is defined as (-1,0). With a clear background (1,0) and a chrome layer specified with (0,0), the resulting transmission layer is (1,0) + (-1,0) = dark (0,0).</p>
atten factor	<p>Defines the layer to be an attenuated phase shifting layer and the optical transmission to be a percentage of incident light, as specified by factor.</p> <p>When Calibre LITHO tools encounter the atten factor command, they translate the factor into a real imaginary pair such that the attenuated background has a transmission value:</p> $\text{RE(layer)} = -\sqrt{\text{factor}}$ $\text{IM(layer)} = 0$ <p>The maximum allowed attenuation factor is 36.</p>
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.

- *mask_num dose*

An optional argument pair used with phase shifting masks to allow the simulators to support multiple masks. This argument is only used with the **visible** argument.

- `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/VTRE model should be the same dose setting used when running other Calibre tools with the model.

Examples

Declares two layers, one for the OPC layer (`SHAPES`) and one for the original target polygons (`ORIG`):

```
layer SHAPES visible dark
layer ORIG hidden dark
```

Declares an attenuated phase shifting layer:

```
layer OPC visible atten 0.06 0 1
```

Declares two alternating phase shifting layers:

```
layer PH0 visible clear 1 1.0
layer PH180 visible phase180 1 1.0
```

layer_properties

Setup Command

Specifies property names and property merge actions of an input layer.

Usage

```
layer_properties input_layer_name '{  
  prop1[:merge_action]  
  [prop2[:merge_action]]...]  
}'
```

Description

The layer_properties command explicitly declares properties from the input layer and can assign property merge actions to each property. You can define more than one property name and merge action in a command block enclosed in braces ({}).

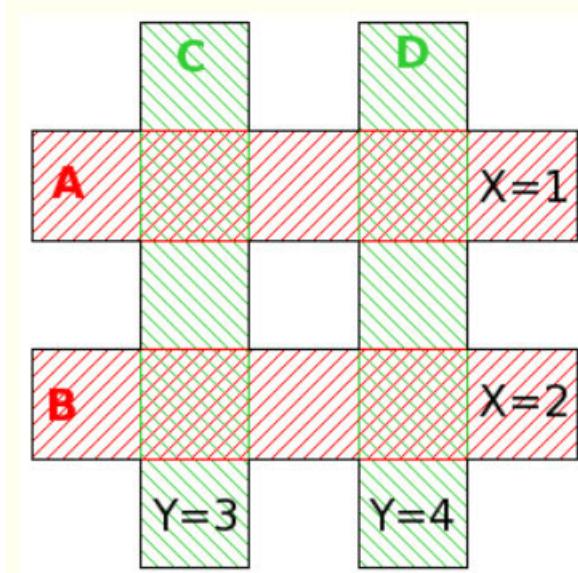
Property Merge Actions

The following is an illustration of a property merge action. Consider the following operation:

```
setlayer LAYER_INT = interact LAYER1 copy_props (X) LAYER2 copy_props (Y)  
>=1 max_extent 0.5 mutual
```

run on the following layout:

Figure 4-7. Merge Action Layout



Polygons A and B comprise LAYER1; polygons C and D comprise LAYER2; polygons of LAYER1 have a numeric property X, polygons of LAYER2 have a numeric property Y. The output of “interact ... mutual” is a polygon combining all four input polygons. According to the “copy_props” options, the output polygon should have properties X and Y, so Calibre

OPCVerify must decide which value of X (1 or 2) and which value of Y (3 or 4) to use for output.

When inputs for the interact operation are outputs from other setlayer operations, Calibre OPCVerify knows the semantics of their properties, so it can make the choice without requiring you to provide additional information. For example, if LAYER1 was a bridge output with min and target_cd properties, Calibre OPCVerify selects the smallest min and the corresponding target_cd; if it were a meefcheck output with a max property, Calibre OPCVerify selects the largest max, and so on.

When inputs for the interact operation are Calibre OPCVerify input layers, Calibre OPCVerify has no knowledge of the semantics of their properties, so you must specify how Calibre OPCVerify handles this situation by specifying a merge action for each property. The merge actions can be min (select the smallest value), max (select the largest value), and bind.

Note

 Not specifying layer_properties for an input layer is equivalent to specifying a list of names of all properties of the input layer without a merge action. You can omit layer_properties for input layers of a setlayer copy because a copy never merges polygons and does not require merge actions.

Arguments

- *input_layer_name*

A required argument that specifies the name of input layer. The layer must be declared by a layer statement.

- *merge_action*

An optional argument that specifies one of the following merge actions:

- min — Instructs Calibre OPCVerify to use the smallest property value of the input polygons for output polygons.
- max — Instructs Calibre OPCVerify to use the largest property value of the input polygons for output polygons.
- bind — Specifies a “bound property”, which is always selected from the same input polygon as its “master property”. The master property is the last non-bound property declared before this one. Thus, bind cannot be used for the first property in the list.

In the following example:

```
layer_properties in_layer {  
    X  min  
    A  bind  
    B  bind  
    Y  max  
    C  bind  
}
```

X and Y are master properties, A and B are bound to X, and C is bound to Y. Calibre OPCverify selects the smallest available X and the largest available Y. A and B is selected from the same polygon as X, and C is selected from the same polygon as Y.

- ***propN***

A required argument that specifies the name of a property. At least one property must be specified. The input layer must contain a property with this name or it will generate a runtime error. The name must be unique in the property list.

litho_model

Setup command

Defines an inline litho model. Used for encrypted setup files only.

Usage

litho_model name '{ inline_model_text }'

Description

This command permits inline definition of a litho model. It is intended only for use in creating encrypted setup files. Do not use this in standard Calibre OPCVerify or Calibre nmOPC setup files. The **litho_model** command is only permitted if either the setup file is encrypted, or if a special environment variable is set:

LITHO_PERMIT_INLINE_LITHO_MODEL_FOR_ENCRYPTED_SETUP_DEVELOPMENT
T=1

Arguments

- ***name***

A required argument that specifies the name of the litho model. Once defined, this name can be used anywhere in the setup file where litho models are supported.

- ***inline_model_text***

A required argument that specifies the litho model text in the same format as the contents of a *Lithomodel* file. However, instead of defining component models by specifying file names within the litho model directory, these must be replaced by the symbolic names of the models. These names must have previously been defined by “*_model_load” commands that load each component model. The individual “*_model_load” commands themselves can either be inline or file-based.

log_options

Setup command

Controls the output of various diagnostic messages in the output log.

Note

 No performance impact occurs if runtime messages are disabled. Performance can vary according to the log options of each particular command if runtime messages are enabled.

Usage

```
log_options '{  
    [parsing_messages [on | off]]  
    [runtime_messages [on | off]]  
    [report_cells [{on | off | min_area cell_area_microns_squared}]  
    [tile_warning_period period_seconds]  
    [tile_memory_warning_mb integer_mb]  
}'
```

Arguments

- **parsing_messages**
An option that suppresses parsing messages for each command when set to off. Default is on.
- **runtime_messages**
An optional argument that suppresses runtime messages for each command when set to off. Default is on.
- **report_cells**
An optional argument that controls cell reporting in the main cell loop.
 - **on** — Per-cell logging is done for every cell
 - **off** — Per-cell logging is disabled for every cell
 - **min_area *cell_area_microns_squared*** — Per-cell logging is done only for cells whose area (computed internally) is greater than or equal to the specified *cell_area_microns_squared* value. The default is *cell_area_microns_squared* set to 10000.
- **tile_warning_period *period_seconds***
An optional argument that reports a warning in the transcript when the tile processing is not less than the specified *period_seconds* of real time. The default value is 2400 seconds.

The reported warning is in a format similar to the following:

```
WARN TOP 3: LONG TIME FOR TILE: "TOP 3" TIME=3200s RSS=95
BBOX=(0.30, 0.54) (2.66, 2.91) um
```

The tile_warning_period option is not supported in Calibre FullScale (it has its own dedicated option) and is ignored.

- tile_memory_warning_mb *integer_mb*

An optional argument that reports a warning in the transcript if a tile uses more than the specified amount of memory in megabytes. The default value is 3700.

Note

 The tile_memory_warning_mb option sets the threshold *only* for the setlayer execution remote task, <Litho_Rtask_LHE_Tile>. Calibre runs can include many other remote tasks. For tasks other than <Litho_Rtask_LHE_Tile>, the threshold for memory warnings defaults to 1000MB; that global threshold is changed by setting the environment variable CALIBRE_EXEC_MEMORY. setlayer tiles do NOT use that environment variable or its default; therefore, tile_memory_warning_mb is the *only* way to set the tile execution threshold.

The reported warning is in a format similar to the following (this is all a single line of text that wraps):

```
Note: Remote task <Litho_Rtask_LHE_Tile> execute() at orw-calexample-r6
(pid=20315) used 4163 MB on cell TOP with tag 3. Remote LVHEAP = 4115/4920/
4942, ID = 5 TIMESTAMP = 8
```

The tile_memory_warning_mb option is not supported in Calibre FullScale (it has its own dedicated option) and is ignored.

mask_sample_grid

Setup command

Optionally specifies the mask sampling grid. In most cases, the default setting (rsm) is sufficient.

Usage

mask_sample_grid {rsm | uniform}

Arguments

- **rsm**

A required argument that specifies activation of rasterization with spectral matching (RSM) mode, which is an alternative method for mask rasterization. It is more accurate than conventional rasterization.

- **uniform**

A required argument that turns off RSM mode, and is designed for use with older versions of the software.

modelpath

Setup command

Specifies the directories to search for optical and resist models.

Note

 This command is order dependent. It must appear above any model load commands that use paths specified with this command.

Usage

modelpath *dir*

Arguments

- *dir*

A required argument that specifies a colon (:) separated list of directories to search, which can be absolute and relative paths. The default is the directory Calibre OPCVerify was invoked in.

Examples

Absolute path:

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

Relative path:

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

optical_model_load

Setup command

Loads an optical model.

At least one optical_load_command must be present unless a litho model is loaded.

Usage

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

Description

At least one optical_model_load command must be present. Multiple optical models may be loaded and used.

However, if [Litho Model Format](#) is being used, this command is no longer required; the optical model specified in the litho model file is used instead.

This command loads the optical model found in filename, or defined by specifying model text inside braces ({}). The optical model is given a user-defined optical_model_name which you use with the setlayer [image](#) operation.

Files ending in .fsk will be written to the optical model directory when creating models using the Calibre LITHOview **opticsgen** batch command. This file is used by OPCVerify to start up these optical models.

Tip

 If your optical model does not have a .fsk file (which can happen if it was created earlier than Calibre version 2005.1), regenerating your optical model using the current software (but using the original settings for consistency) may gain you a performance improvement for multiple, disk-based optical models. This is because the Fourier SOCS kernels are stored in the .fsk file, rather than re-derived for every optical model test.

For optical models with kernel diffusion or non-TCCS kernels, however, you *must* regenerate the optical model file using the 2005.4 version of the software in order to get the most recent optical model version .fsk file. If you regenerate your optical model and the previous optical model was earlier than version 6, you will also need to regenerate your resist model.

Arguments

- ***optical_model_name***

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

- ***filename***

An argument specifying a filename to load containing the optical model. If you do not specify a filename, an inline optical model must be supplied instead.

- ‘{’ *inline_model* ‘}’

An argument defining an optical model. The entire optical model specification must be enclosed in braces ({}).

optical_transform_size

Setup command

Overrides the default optical transform size setting.

This setup command is recommended for advanced users only, and can increase the simulation run time.

Usage

optical_transform_size *value*

Arguments

- ***value***

A required argument specifying the number of Nyquist simulation pixels in each simulation frame. If not specified, the optical transform size is selected automatically by the system. Permitted values are 128, 192, 256, 384, 512, 768, 1024, and 2048.

- Values less than 512 may be incompatible with some optical or resist models and can generate a parse-time error.
- Values larger than 512 may be incompatible with some simulation sampling schemes and can generate a parse-time error.

This command operates in conjunction with the [final_upsample](#) and [imagegrid](#) commands.

output_window

Setup command / setlayer command

Outputs a context-sensitive clip around specified errors.

Usage

Setup Command Version

```
output_window {context_layer | image_set_name} ...
  [halo halo_microns] [no_split]
  [compact compact_deviation_microns] [deangle deviation_microns]
  [filter_only]
  around layer_output_filter1 [layer_output_filter2]...
```

Setlayer Command Version

```
setlayer name = output_window {layer context_layer | filter}
  [halo halo] [no_split] [deangle deangle_deviation]
  around layer_output_filter1 [layer_output_filter2]...
```

Description

Tip

 **output_window** works only after all the layers are computed. It should be one of the last commands in the OPCverify setup file.

Outputs a context clip of an area based on geometries from one or more filter layers; the filter layers are usually error layers returned by Calibre OPCVerify setlayer commands.

When Calibre OPCVerify runs this command, an **output_window** filter is constructed using the following sequence:

1. All polygons of each **layer_output_filter** layer are deangled (replaced with slightly larger Manhattan-modified polygon versions) in such a way that:
 - Every **layer_output_filter** layer is not outside its deangled version.
 - The shortest distance between edges of the original and deangled polygons does not exceed **deangle_deviation** microns.
2. All deangled filter layers are sized up by halo microns.
3. All sized-up filter layers are merged into one **output_window** filter layer.
4. If the **filter_only** option is specified, the original context layer(s) are replaced by the single **output_window** filter; otherwise, each context layer is ANDed together with the **output_window** filter.

This command is intended to output only the context regions surrounding errors found by the Calibre OPCVerify run. You can use this function to reduce the output file size, because you are not generating the output for the whole contour.

Note

 It is strongly recommended that you use `output_window` on any contour intended to be output to SVRF, since full contours are very large and inefficient to write.

Note

 In the future, Siemens EDA reserves the right to modify the creation of the output windows by changing the method in which deangling is done.

Arguments

Setup Command Options

- ***context_layer***

A required argument specifying one or more layers to clip polygons from, subject to the following restrictions:

- ***context_layer*** cannot be a layer that was generated by a command that had any property, classification, or limit blocks.
- ***context_layer*** must also be output to SVRF. Otherwise, this command has no effect.
- Only a single ***context_layer*** layer is permitted when `filter_only` is also specified.

- ***image_set_name***

A required argument that can be specified as an alternative to a ***context_layer*** argument. All images in the specified `image_set` are as the output window context layers.

Any contour layer that is mapped back to SVRF and is part of an `image_set` used with a `pw_annotate` command must have an `output_window` command.

- **halo *halo_microns***

An optional argument specifying how much area around each polygon clip to include.
(Default is 0.2 microns)

- **no_split**

An optional argument specifying that the command should be run in `no_split` mode:

- Normally, `output_window` handles errors near a tile boundary (near being defined as when the halo for an `output_window` crosses the tile boundary) by simulating the error in both tiles, increasing simulation time and run time. In some minor cases, some tiles might not calculate exactly the same simulation contour as their neighbors, leading to tiles that might not generate an error (and no context) when their neighbor generates an error, leading to a split output window error.

- In no_split mode, Calibre OPCVerify temporarily saves all context layer geometries within halo distance of the tile boundary, regardless of there being an error present and uses them during post-processing to produce an error-free output_window for the context layer. This has the following effects:
 - i. Calibre OPCVerify simulates less contours in no_split mode, decreasing simulation time.
 - ii. Calibre OPCVerify stores more context layer geometries, increasing memory consumption; this means that it has to discard more geometries during the post-processing stage, which increases post-processing time.

Depending on your design, no_split mode may have different effects on runtime:

- The more complex the models used for simulation, the more significant the speedup is due to (i).
- The more errors Calibre OPCVerify finds (before classification and limiting), the less significant the slowdown is due to (ii).
- The larger the output_window halo argument is, the more significant the slowdown is due to (ii).
- The smaller tiles Calibre OPCVerify uses, the more significant the slowdown is due to (ii). This is why it is not recommended to use no_split in hierarchical processing mode, which uses many small tiles.
- compact *compact_deviation_microns*

An optional argument specifying that the output layer will be compacted; this option is primarily useful for contour layers. It reduces the number of vertices and memory requirements. The *compact_deviation_microns* value is used to set the maximum deviation of a compacted layer from the uncompacted layer.

(Default value is 0.00 microns (off); the maximum accepted value is 0.006 microns.)

- deangle *deviation_microns*

An optional argument specifying the maximum deangling distance to move an edge. (Default is 0.1 microns; if the specified value is less than 6dbu, it is replaced with 6dbu)

- filter_only

An optional argument that changes *context_layer* to contain only the halo area around the polygon regions defined by *layer_output_filters*.

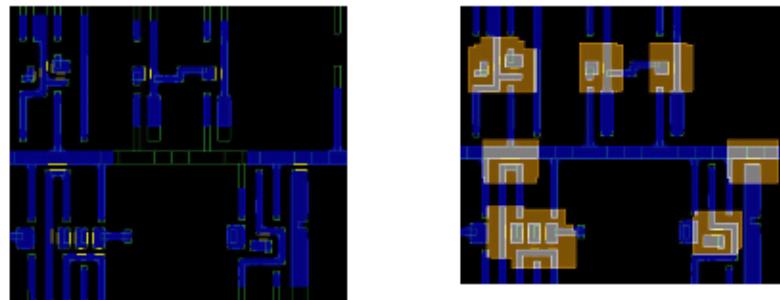
Only a single layer can be supplied when filter_only is specified, as the pre-output window content of the *context_layer* input layer is discarded.

This output can be used in subsequent SVRF operations outside the Calibre OPCVerify command file (for example, to filter non-OPCV layers in SVRF using boolean AND statements).

Note

 The **context_layer** input required by the filter_only mode of output_window must be previously created using an **empty_layer** command **before** the output_window command. (See the Example section below for more information.)

Figure 4-8. filter_only Mode in output_window



output_window, normal mode

filter_only mode (orange polygons)

- **around layer_output_filter1 layer_output_filter2 ...**

A required argument of a layer containing shapes to use as filters. Additional filter layer(s) are optional.

Note

 A layer that appears as a **context_layer** layer in any output_window command cannot be used as a layer_output_filter in any other output_window command.

The **layer_output_filter** layer must also be output to SVRF for the filter to have an effect. If it is not output, the **context_layer** will not be windowed around the filter_layer, and a note is added to the Calibre OPCVerify transcript.

Setlayer Version Arguments

- **layer context_layer**

A required argument that specifies layers to clip context from, similar to the setup command version.

- **filter**

A required argument that works similarly to the **filter_only** mode, generating the output window filter only.

Examples

Example 1

Given the following layer definitions in the OPCVerify setup command file:

```
optical_model_load f0 optical_model.  
layer TARG1 visible dark  
layer OPC1 hidden dark  
layer ISL hidden clear  
setlayer contour image optical f0 aerial .03  
setlayer ext_test = and contour OPC1
```

the following command filters out all the OPCVerify output layers (contour and ext_test, which are MAPped layers from the SVRF file) so that only the areas inside the ISL (island) layer are shown:

```
output_window ext_test around ISL
```

Example 2

The following commands result in an output layer that can be used in an SVRF boolean operation:

```
litho file setup [  
    layer PC_TARGET  
    layer PC_OPC  
    ...  
    output_window contour around error1 error2  
    setlayer output_window_filter = empty_layer  
    output_window output_window_filter filter_only around error1 error2  
]  
  
output_window_filter = LITHO OPCVERIFY PC_TARGET PC_OPC FILE setup map  
output_window_filter  
  
M1_windowed = M1 AND output_window_filter  
M2_windowed = M2 AND output_window_filter
```

processing_mode

Setup command

Sets the processing mode for Calibre OPCverify.

Usage

processing_mode [hierarchical | flat | hybrid]

Description

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

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

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

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

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

Arguments

- hierarchical | flat | hybrid

An optional argument that selects the type of processing (hierarchical, litho-flat, or hybrid). hierarchical processing is the default value for LITHO.

progress_meter

Setup command

Toggles the reporting of cell processing information in the output log.

Usage

progress_meter {on | off}

Description

Use this command to help estimate the progress of the LITHO operation in which it is used. It prints the completion percentage as tiles are run under the heading TIME FOR ALL CELLS in the console window. This option may have a negligibly small runtime impact due to the computation algorithm, depending on the size of the job. The cost of performing progress meter checks can be found in the logfile as “Workload Estimation”.

Arguments

- **on | off**

A required argument that toggles the progress meter status. Progress meter output is active by default.

promote_subframe_slivers

Setup command

Promotes small slivers geometries out of frames.

Usage

promote_subframe_slivers *size*

Description

Use this command to cause the sliver geometry to be promoted out of cells containing small slivers. By default, no geometry subframe slivers are promoted.

Arguments

- *size*

A required argument that determines the size of the geometry being promoted, in microns. Slivers that are larger than the specified *size* are promoted. The specified *size* must be a non-negative integer. Default is 0 (no slivers are promoted).

push

Setup command

Performs a light push on generated output layers.

Usage

push [output {light | no}]

Description

A light push is performed on output layers by default. An exception is made for output layers with DFM properties, which are not pushed. The “light push” is identical to the SVRF [Push](#) command with the LIGHT keyword specified. For some large runs, light push on output generated layers dramatically improves elapsed time performance. In other cases, the performance impact is small. Siemens EDA recommends not disabling the output light push.

Arguments

- output {light | no}

An optional argument to enable or disable a light push of generated output layers. The default state is light, which enables light push for output layers.

pw_annotate

Setup command

Modifies image set error markers with extra properties for errors in each component image (pw_image) of the image set.

Usage

```
pw_annotate error_layer [pw_image_constraint val][full_factorial]  
[default_property_value val]
```

Description

This setup file command instructs Calibre OPCVerify to run process window annotation. After process window annotation, all error markers on the *error_layer* have an extra set of properties, one property for each process window condition. The names of these properties are derived from the names of the [image_set](#) component images.

Specifying “pw_image_constraint” enables pw_annotate to find and report error properties even for pw_images which do not fail the original check. This constraint must be equal to or more relaxed than the constraint used in the main check. Specifying a more relaxed value means that more errors will be produced. For example, if the original check was “pinch < 0.040”, then a suitable “pw_image_constraint” value might be 0.045. The constraint specified should also be less than the minimum designed feature size of the contour to avoid generating excessive errors. Note that if a pw_image passes the check even with this relaxed constraint, then its property is reported as “default_property_value”.

The property values for process window conditions are values of the “base” property for error limiting. The “base” property is the one used for scoring in the classification and limit block (or by default, the first property in the property block). For example, for pinch and bridge, this is almost always “min”. For each process window condition, if there is an error for that process window condition, the error property value for that process window condition is produced. If there is no error for that process window condition, the value specified for default_property_value is produced.

Process window annotation is performed as a second, automatically-generated run of Calibre OPCVerify after the main run completes. This second run re-examines the region around every error marker created by the main run, taking into account all of the process window conditions. While the processing is quite efficient, there will be some increase in the total Calibre OPCVerify runtime, though typically less than 20%. The second run appears in the transcript as a LITHO PW_ANNOTATE run (not a LITHO OPCVERIFY run).

Note

 Because process window annotation is a two-run flow, there may be very small noise-level differences between the contours from the runs due to different tiling. The command tries to report properties that correspond exactly to the output contour context around each error. This means that for a given output contour, non-pw_annotated errors have context from the first run and pw_annotated errors have context from the second run. When context from both runs overlaps, Calibre OPCVerify prefers the results of the second run. In this case, occasionally very small jogs may be seen in the output context contours because of slight mismatches.

Restrictions

- pw_annotation is not supported and will generate a parse error in the flat Calibre engine (the engine invoked when you omit the “-hier” flag from the command line).
- The Calibre OPCVerify run must be done in TURBO FLEX or ULTRA FLEX mode.
- The error marker layer must be the .inner or .outer pvbnd of an [image_set](#). The component images of the set define the process window conditions to be annotated.
- For performance reasons, the setlayer check must specify error count limiting in a classify or limit block, with an error count limit that is not greater than 10000. If you need to increase the global limit, use the [pw_annotation_options](#) command.
- The process window image constraint is only supported if the main setlayer check has a single-sided constraint of the form “ $< val$ ”, “ $\leq val$ ”, “ $> val$ ” or “ $\geq val$ ”.
- Any contour layer that is mapped back to SVRF and is part of an image_set involved in pw_annotation must have an [output_window](#) command associated with it.
- For performance reasons, any individual pw_image component of the image_set that is mapped back to SVRF should have an output_window which is only around the pw_annotated checks.

Arguments

- **error_layer**
A required argument specifying the name of an error marker layer previously created by a setlayer check which supports process window annotation.
- **pw_image_constraint val**
An optional argument that is used to investigate errors for individual pw_images that may not be detected by the main check. This constraint must be equal to or more relaxed than the constraint used in the main check. If not specified, the constraint value specified in the main check is used.
- **full_factorial**
An optional argument that produces a full process window simulation. This option is only permitted if the image_set in the main check is a LithoModel range-based set which varies either two or three settings of the following settings: focus, dose, bias, aerial. The property

annotation includes all possible combinations of the settings, as well as the nominal values focus = 0, dose = 1.0, and bias = 0.0.

- **default_property_value val**

An optional argument describing the default property value (as an integer) applied to annotated process window conditions when there is no error. If not specified, the default is 1000.

Examples

Naming Conventions for Properties

Suppose the setup file contains these commands:

```
image_set imgs = full A focus (-40 0 40) dose (0.98 1.00 1.02)
setlayer pinch1 = pinch target imgs.inner ...
pw_annotate pinch1
```

The names of the component images created in the image_set are similar to “imgs_f-40_d1.02”. The annotated property for this pw condition is named “f-40_d1.02_min”.

More formally, pw_annotate uses the following naming conventions:

- *short-image-name* — The name of the image_set member image, with the prefix “*image-set-name_*”, if any, removed. (For “list” type image_sets, there is no such prefix and *short-image-name* is just the full image name.)
- *short-image-name_property-name* — The annotated property name, where *property_name* is the name of the property being annotated, typically “min”.

Related Topics

[pw_annotate_options](#)

[image_set](#)

pw_annotation_options

Setup command

Sets run options for the pw_annotation command.

Usage

pw_annotation_options [max_num_errors *n*]

Arguments

- max_num_errors *n*

An optional argument that specifies that all setlayer checks for which a **pw_annotation** command is coded must have error limiting with a maximum count not greater than this value. This limit is for performance reasons.

The default value is 10000.

Examples

```
pw_annotation_options max_num_errors 5000
```

rasterizer_upsample_factor

Setup command

Sets the rasterizer upsample factor relative to the optical (Nyquist) grid.

Usage

rasterizer_upsample_factor *num den* [coarse *num den*]

Description

This command changes the rasterizer upsample factor for the simulation mask sample grid.

Arguments

- ***num den***

A required argument pair of exact integers that sets the numerator and denominator of the upsample factor by which the mask is sampled, relative to the optical (Nyquist) grid. If the fractional value is greater than 1, the mask grid is sampled finer (smaller pixel size) than the optical grid.

The default values are 2 1.

Note

 Previous to the 2019.2 release, the default values were 4 1 when a DDM model was present and 2 1 when a DDM model was not present.

- **coarse *num den***

An optional argument that specifies a coarser upsampling from the optical grid to the mask grid. The values specified for the coarse numerator and denominator must be equal to or coarser than the values specified for the base ***num*** and ***den*** arguments (they may not specify smaller pixels than the base arguments).

This option is used with and required for multiple-grid OPC only.

Examples

Sets the mask sample pixel size to 1/2 of the Nyquist grid pixel:

```
rasterizer_upsample_factor 2 1
```

resist_model_load

Setup command

Loads a resist model into Calibre OPCVerify.

Usage

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

Description

Optional: A resist model is only required if you use the *resist_model* argument in the non-litho model version of the [image](#) command.

However, if [Litho Model Format](#) is being used, this command is no longer required; the resist model from the litho model file is used instead.

This command loads the resist model (VT5 or CM1) contained in the file *filename*, or is defined by specifying model text inline inside braces ({}). The resist model is assigned the user-defined *resist_model_name*, which you may later reference in the [image](#) command.

Note

 Siemens EDA recommends using CM1 models over VT5 models for improved performance.

Arguments

- ***resist_model_name***

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

- ***filename***

An optional argument specifying a filename to load containing the resist model. If you do not specify a filename, an inline optical model must be supplied instead.

- **{*inline_model*}**

An optional argument defining a resist model. The entire specification must be enclosed in braces ({}).

Related Topics

[VT5 Model File Description \[Calibre WORKbench User's and Reference Manual\]](#)

[CM1 Model File Format \[Calibre WORKbench User's and Reference Manual\]](#)

save_error_center_points

Setup command

Creates an output file containing the center points for error shapes.

Usage

```
save_error_center_points {'  
    layers [layer_name] [layer_name2...]  
    {... {'  
        {filename output_filename | file_prefix file_prefix [file_suffix file_suffix]}  
        [format {rdb | text}]  
        [duplicates_limit dup_count]  
        [duplicates_limit_per_class dup_count]  
        [layer_name {OPCV | SVRF}]  
    '}...  
'}
```

Description

This command creates files in RDB or text formats containing the error center point and property for each error shape.

- The center point of an error shape is the center of its bounding box. When the error shape is classified, it is given a CLASS_ID > 0, and any matching error shape is assigned the same CLASS_ID number and is considered a duplicate. When pm_classify (see [Classification Block](#)) is not used, error shapes that have any of the extents of their bounding box larger than max_size are unclassified and are assigned a CLASS_ID = 0. Additionally, setlayers which do not have a classify block but have a limit block are also assigned a CLASS_ID = 0.
- RDB center points are represented by 2x2, 2x3, 3x2, or 3x3 rectangles. The odd widths (3x3) are used to represent the center points of errors with odd bounding box dimensions. These boxes can be expanded by halo_value -1 dbu to visualize the halos used to clip unique and duplicate contexts.
- The center points are output and ordered using the following criteria:
 - a. By layer in the order they are invoked in the LITHO_OPVERIFY setup file.
 - b. By CLASS_ID groups in ascending order.
 - c. Within the CLASS_ID classify blocks:
 - i. If scoring is specified, by property value. Property values are sorted in worst-case descending order (min is sorted in ascending order; max is sorted in descending order). To satisfy the worst unique-CLASS_ID property value ordering, the center points with CLASS_ID==0 are interspersed among the CLASS_ID > 0 center_points.

- ii. By X/Y coordinate fields, in ascending order. If both limiting (worst) and keep duplicates options are specified in the classification block, output duplicate X/Y coordinates will also be limited by the worst unique duplicates and per_class specifications.
- Error shapes exceeding the max extent are considered unclassified. No attempt is made to compare their context and properties with other error shapes. All unclassified shapes are treated as unique shapes with no duplicates, and are assigned a CLASS_ID of 0.
- Center point coordinates are written out in the top cell coordinate system. The center points of the output file(s) are initialized and reset at Calibre OPCVerify parsing time. The center points are flattened and written one layer at a time in order to limit peak memory usage.

The text center points are in user units (dbu/precision). For classify blocks with the keyword ‘halo around extent’, the center coordinate has a non-integer value if the extent value is odd.

- Limiting of unique errors imposed by the [Classification Block](#) is honored. Only the limited unique center points will be output. However, all the corresponding duplicates will be output up to the duplicates_limit.

If both limiting (worst) and keep duplicates are specified in the classify block, the output duplicate x and y coordinates will also be limited by the “worst unique duplicates or per_class” specification.

- Multiple layers can be written to the same file. The x and y coordinates are grouped by output layer names. The setlayer output name is the same as MAPNAME in the SVRF LITHO OPCVERIFY invocations. The output layer order is determined by the order in which the corresponding SVRF LITHO OPCVERIFY MAP output is referenced by other SVRF operations. The order is usually set by SVRF CHECK MAPs referencing the LITHO OPCVERIFY output layer.
- The duplicate output is hierarchical if the run is hierarchical and flat if the run is flat (processing_mode flat), even if the coordinate is in the top cell.
- If the setlayer's classify block does not have an explicit or a default property, the property name/values fields of the center-points will not be written.
- The specified RDB output file must be unique with respect to other RDB files that are not produced by the save_error_center_point command.

Layer Name Globbing

A layer name string is a wildcard pattern if it contains one of the characters “?”, “*”, or “[”...“]”. Globbing is the operation that expands a wildcard pattern into the list of output layer names matching the pattern. Matching is defined as follows: A “?” (not between brackets) matches any single character. A “*” (not between brackets) matches any string, including the empty string. A “[”...“]” matches a single character from a group of characters specified between the brackets.

- **Wildcards** — The special meaning of “?”, “*” and “[”...“]” is removed by preceding them with a backslash. Thus, “[[*\”]” matches the four characters “[”, “?”, “*” and “\”].
- **Character Classes** — An expression “[”...“]” where the first character after the leading “[” is not an “!” matches a single character, namely any of the characters enclosed by the brackets. The string enclosed by the brackets cannot be empty; therefore “]” is allowed between the brackets, provided that it is the first character. Thus, “[![!]” matches the three characters “[”, “]” and “!”.)
- **Complementation** — If the first character after the leading “[” is a “!”, any single character not enclosed within the brackets matches. Thus, “[!a-]” matches any single character except “[”, “a” and “-”.)
- **Ranges** — Between the brackets, two characters separated by “-” denote a range. Thus, “[A-Fa-f0-9]” is equivalent to “[ABCDEFabcdef0123456789]”. The character “-” is included in its literal meaning by making it the first or last character between the brackets.

Arguments

- **layers** [*layer_name*][*layer_name2...*]

A required keyword that specifies an error center point output layer. Specifying names of one or more output layers is optional. If no *layer_names* are specified, all the setlayer outputs with at least one classify or limit block are written out. It also starts a specialized block of options, which must be enclosed in braces ({ }). Additional rules for handling layers are listed below:

- Duplicate layer name values are not allowed.
- Output layers that do not match the output layers of setlayer commands with at least one classify or limit block are ignored.
- Unreferenced setlayer outputs that are eliminated are ignored.
- Multiple layers statements are supported within a single save_error_center_points statement.

A layer name may be specified as a wildcard pattern using standard Unix glob-style pattern matching characters. For more information on wildcard pattern names, see “[Layer Name Globbing](#)” on page 219.

- **filename** *output_filename*

A required argument inside a layers block specifying the output filename. The formatted output file contains the output layers, class_id, x and y error center-point coordinate or rectangle, and the property value for each unique and duplicate error shape.

Only those output layers generated by setlayer statements with a classify block are output.

- **file_prefix *file_prefix***

A prefix used for generating multiple rdb or text output files, one file for each specified layer. The file paths are formed from the string *file_prefix layername file_suffix*. Each output file contains the output layer, class_id, x and y error center-point coordinate or rectangle, and property value for each unique and duplicate error shape. Directory delimiters ('/') are supported; directories will be created as needed. Only those output layers created by setlayers with a classify block will be output.

- **file_suffix *file_suffix***

A optional suffix that is appended to the *file_prefix* generated files. Embedded '/' directory delimiters are not supported in the suffix string. The default suffix value is ".txt" for format "text", and ".rdb" for format "rdb".

- **format {rdb | text}**

An optional argument that specifies the output type, in either the multilayer rdb format or the more readable text format. The default is rdb.

- **duplicates_limit *dup_count***

An optional argument that limits the total number of duplicate errors per layer to *dup_count*. If *duplicates_limit_per_class* is also specified, the *duplicates_limit* is applied to the duplicates that passed the *duplicates_limit_per_class* filter.

- **duplicates_limit_per_class *dup_count***

An optional argument that limits the number of duplicate errors per CLASS_ID within a layer to *dup_count*. The default is ULONG_MAX.

- **layer_name {OPCV | SVRF}**

An optional keyword that specifies what type of layer name is written to the output file; a Calibre OPCverify-mapped layer name (that appears to the right of the assignment operating in a LITHO OPCVERIFY command) or an SVRF layer name (that appears to the left of the assignment operator in a LITHO OPCVERIFY command). The option SVRF is not supported in flat mode.

Examples

Example 1 (RDB Output)

Given the following command:

```
save_error_center_points {
    layers {
        filename save_center_points.rdb
        format rdb
    }
}
```

The output contains the following information:

```

topcell 1000                                <--- Top cell, precision
LAYER_ONE                                    <--- OPCVERIFY/LITHO MAP name.
2 2 0 Feb 10 16:13:24 2014                  <--- 2 flat, 2 hierarchical polygons,
                                              0 text lines following, date.

p 1 4                                         <--- 1st polygon w/ 4 vertices.
min 0.19975                                   <--- property, value.
CLASS_ID 74                                    <--- class id.

60 85
62 85
62 88
60 88

p 2 4                                         <----- 2nd polygon w/ 4 vertices.
min 0.19875                                   <--- class id.
CLASS_ID 75

60 185
62 185
62 188
60 188

LAYER_TWO                                     <--- OPCVERIFY/LITHO MAP name.
1 1 0 Feb 10 16:13:24 2014                  <--- 1 flat, 1 hierarchical polygon,
                                              0 text lines following, date.

p 1 4                                         <--- 1st polygon w/ 4 vertices.
min 0.222                                      <--- property, value.
CLASS_ID 88                                    <--- class id.

60 85
62 85
62 88
60 88

```

Example 2 (Text Output)

Given the following command:

```

save_error_center_points {
    layers res2 res7 {
        filename save_center_points.txt
        format text
    }
}

```

The output contains the following information:

```
res2  
CLASS_ID X Y min  
1 1.355 0.585 0.21  
1 3.355 0.585 0.21  
2 5.355 2.085 0.23  
2 7.355 2.085 0.23  
3 7.355 3.585 0.25
```

```
res7  
CLASS_ID X Y min  
1 1.355 0.585 0.21  
1 3.355 0.585 0.21  
1 5.355 0.585 0.21  
1 7.355 0.585 0.21  
2 1.355 2.085 0.23
```

Related Topics

[Classification Block](#)

setlayer

Setup command

Creates and derives layers, depending on the options used.

Usage

`setlayer output_layer_name = setlayer_operation [setlayer_command_options]`

Description

You create Calibre OPCverify layers using the design layers you defined with the layer command, and you will eventually use one or more of them as output in your LITHO OPCVERIFY calls in your SVRF file.

Arguments

- ***output_layer_name***

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

- ***setlayer_operation***

A required option specifying one of the following operations:

- [Image Operations](#)

An image operation performs optical and resist simulation using geometry on visible layers. In the case of multiple exposures, different options can be specified for each exposure, allowing you to simulate a wider array of lithographic conditions.

- [DRC-type Operations](#)

Similar to Calibre DRC, you can use setlayer to work with the polygons on the source layers, outputting the result to the output layer.

- [Verification Control Operations](#)

Calibre OPCverify includes a number of special operations to perform activities on layers.

- ***setlayer_command_options***

A required argument specifying one or more options, which may be layers, values, constraints, flags, or other switches depending on the operation.

Tip

 The setlayer operations are all arguments to the setlayer command, but are described separately, starting with the section “[Setlayer Operations Reference](#).”

shadow_bias_model_load

Setup command

Loads a shadow bias model. Used when the shadow bias model is not contained in a litho model file.

Usage

```
shadow_bias_model_load name {file | '{' inline_model_text '}')}
```

Description

This command reads in the specified shadow bias model (.sbm) file and assigns the given *name* to it for later use with the [shadow_bias](#) command.

For best practices, shadow bias effects have been moved to an automatic biasing function that is activated by putting the shadowbias file inside the lithomodel. This command is retained for backwards compatibility.

Note

 Shadow bias models can only be used to apply mask corrections via the [shadow_bias](#) command. They are not supported in simulation operations such as image. Masks corrected by [shadow_bias](#) cannot be used in simulations, and produce invalid images.

A shadow bias model represents thick mask effects, and describes a polynomial equation for feature biases applied to the horizontal edges b_h , feature biases of the vertical edges b_v , shifts of the edges in the x direction s_x , and shifts of the edges in the y direction s_y as functions of the offset x .

$$\begin{aligned} b_h &= a_{h,0} + a_{h,1}|x| + a_{h,2}x^2 \\ b_v &= a_{v,0} + a_{v,2}x^2 + a_{v,4}x^4 \\ s_x &= a_{x,0} + a_{x,2}x^2 + a_{x,4}x^4 \\ s_y &= a_{y,0} + a_{y,1}|x| + a_{y,2}x^2 \end{aligned}$$

where

b_h — Bias applied to the horizontal edges, in nanometers

b_v — Bias applied to the vertical edges, in nanometers

s_x — Shift in the x-direction, in nanometers

s_y — Shift in the y-direction, in nanometers

x — Position of the edge of interest in relation to the center of the slit, in microns

$a_{h,0}, a_{h,1}, a_{h,2}, \dots, a_{y,2}$ — Modeling coefficients

The biases b are assumed to be positive when the edge under consideration moves outward in relation to the polygon, and negative otherwise.

A *.sbm* file uses the following syntax:

```

HTERM 0 aho
HTERM 1 ah1
...
VTERM 0 av0
VTERM 1 av2
...
XTERM 0 ax0
XTERM 1 ax2
...
YTERM 0 ay0
YTERM 1 ay1
...

```

where:

- **HTERM *order coefficient***
Describes a term for the horizontal edge bias polynomial equation. *order* is 0, 1, or 2, which identifies the degree for the modeling coefficient that follows.
- **VTERM *order coefficient***
Describes a term for the vertical edge bias polynomial equation. *order* is 0, 2, or 4, which identifies the degree for the modeling coefficient that follows.
- **XTERM *order coefficient***
Describes a term for the horizontal edge shift polynomial equation. *order* is 0, 2, or 4, which identifies the degree for the modeling coefficient that follows.
- **YTERM *order coefficient***
Describes a term for the vertical edge shift polynomial equation. *order* is 0, 1, or 2, which identifies the degree for the modeling coefficient that follows.

Arguments

- ***name***
A required user-defined name for the model.
- ***file***
A required argument specifying the file name containing the model. The model can be inside the current directory or in the model path.

- ***inline_model_text***

An inlined text file can be specified instead of the model file path. It must be contained inside braces ({}).

simulation_deangle

Setup command

Runs the deangler on all image contours.

Usage

simulation_deangle *deangle_tolerance*

Description

Note

 Siemens EDA strongly recommends that you remove all simulation_deangle commands from your Calibre nmOPC setup command files because the output is not aligned on tile boundaries and can lead to fragmentation errors. Use the **SVRF DEANGLE** command instead.

Calibre OPCverify users can continue to use this command as normal.

When active, this setting causes the output of all image commands to be passed through the deangler, changing all edges to 0, 45, or 90 degree angles, as long as the de-angling does not exceed the maximum deviation defined by ***deangle_tolerance***.

Note

 The smaller the value you specify for ***deangle_tolerance***, the more memory and runtime will be required, especially on designs with many non-manhattan edges. This is because each non-manhattan edge can be broken into up to length/***deangle_tolerance*** vertices.

Do not use simulation_deangle if you have memory usage issues.

Arguments

- ***deangle_tolerance***

A required argument that specifies the maximum deviation distance (in microns) from its original location an edge can have applied by the deangler. The value specified should be greater than 1 dbu.

Examples

For a 130 nm design, a value of:

```
simulation_deangle 0.03
```

should give noticeably de-angled contours.

stairstep

Setup command

Controls the aerial contour stairstep behavior.

Restriction

 This option is only used for aerial contour models, and is ignored for resist models.

Usage

stairstep {0 | 1}

Arguments

- 0
 - An argument that specifies that Calibre OPCVerify will use bilinear interpolation to intersect the image with the threshold. (The default is 0.)
- 1
 - An argument that specifies that Calibre OPCVerify represents the final contour as discrete pixels which are “on” or “off” for each pixel on the image grid.

stochastic_model_load

Loads a stochastic model from the specified location.

Usage

```
stochastic_model_load name {filename | '{' inline_model_text '}' }
```

Arguments

- *name*

A required option that specifies the name alias of the stochastic model to Calibre OPCVerify for later use.

- *filename*

A required argument specifying the location and name of the stochastic model file to load.

- *inline_model_text*

A required argument that can be used in place of a stochastic model input file. Inline file definitions must be enclosed in braces.

summary_report

Setup command

Creates an HTML page containing summary information and snapshots of selected errors.
Includes configuration output options for snapshots and histograms.

Note

 Summary report blocks can only be used in hierarchical mode. It returns an error if run in flat mode.

Usage

```
summary_report '{'
    path destination_path [with_datetime] [err_suffix e_suffix] [warn_suffix w_suffix]
        [clean_suffix c_suffix]
    [file_name name [with_datetime] [err_suffix e_suffix] [warn_suffix w_suffix]
        [clean_suffix c_suffix]]
    [title title_string]
    [errors layer1 [layer2 ...]]
    [warnings layer1 [layer2 ...]]
    [summary [top] key1 text1] ...
    [geometry_counts {hier | flat | both}]
    [snapshots num_per_layer] ...
    [snapshot_halo halo_microns] ...
    [snapshot_grid {step [line_width w] [color grid_color] [opacity grid_opacity] } | none] ...
    [snapshot_center_mark {mark_size [line_width w] [color mark_color]
        [opacity mark_opacity] } | none] ...
    [snapshot_scale {step ticks [steps#] [color scale_color] [opacity scale_opacity] } | none] ...
    [snapshot_background color] ...
    [snapshot_picture size] ...
    [snapshot_ids [sequential] [toc]] ...
    [snapshot_file_name template] ...
    [snapshot_world_view [world_view_options | none]
    [snapshot_bossung_table [bossung_table_options | none]
    [caption_units {\u | nm}] ...
    [show layer1 [layer2 ...] [ruler1 [ruler2 ...]][with context_layer1 [context_layer2 ...]]]...
    [ruler [color ruler_color] [opacity ruler_opacity] [width ruler_width]]...
    [histogram [scale {log | linear | both} ] [width width] [height height]
        [color bar_color] [background background_color]
        [from prop_min] [to prop_max] [step prop_step [ticks_per_step]]
        [files name1 [name2 ...]]]...
    [layer layer1 [fill fill_color [pattern pattern_name] [opacity fill_opacity]]
        [line line_color [line_opacity line_opacity] [line_width width]]]...
}'...
```

Note

 The *ruler* arguments are one or more repeated ruler placement syntax blocks, separated by spaces. See “Ruler Selection Statements” in the Arguments section of this command.

Description**Caution**

 **Placement-Sensitive Command** — In order to use this command properly, it must be placed after all setlayer commands (in other words, near the end of the command file). Failure to do so results in an error if summary_report refers to a layer that was not yet defined in a setlayer command.

Defining a summary_report block creates a directory containing an HTML page and associated files.

Tip

 Multiple summary_report blocks are allowed, each of which creates its own directory and HTML page. Unique directory names are dependent on the path and file_name arguments.

The summary report HTML page contains requested information from a completed Calibre OPCverify run:

- A summary report table containing:
 - The current directory.
 - The DRC file used.
 - The following statements copied from the DRC file:
 - LAYOUT PATH and LAYOUT PATH2
 - DRC RESULTS DATABASE
 - DRC SUMMARY REPORT
 - A list of the file-based models used.
- A table of the geometry counts returned by Calibre OPCverify:
 - Total: The total number of polygons output by OPCverify.
 - Filtered out: The number of polygons removed from the total by limiting or classification.
 - Unique: The number of unique error types, as a subset of the total.
 - Duplicate: The number of duplicate errors corresponding to unique polygons, whether they were output to Calibre OPCverify or not.

- Snapshot images of errors, plus any context layers specified using a show argument. The snapshot is a square $2 * \text{halo_microns}$ around the error, along with coordinates and timestamps. See the section “[Rules for Selecting Snapshot Locations](#)” for how errors are chosen.
- Histogram information in a graphic chart if the Calibre OPCVerify run produced any histograms (see “[Histogram Block](#)” on page 87).

Rules for Selecting Snapshot Locations

The rules for selecting snapshot locations are determined with classification and limit blocks in setlayer commands, and the snapshots *num_per_layer* argument using the following criteria:

- Layers with both classification and limiting:
Select *num_per_layer* unique errors in order by worst score, smallest cell_id, then smallest (x,y) vertex.
- Layers with limiting, but without classification:
Select *num_per_layer* non-unique errors in order by worst score, smallest cell_id, then smallest (x,y) vertex.
- Layers with classification, but without limiting:
Select *num_per_layer* unique errors in order by smallest cell_id, then smallest (x,y) vertex.
- Layers without classification AND without limiting statements:
Select *num_per_layer* errors in order by smallest cell_id, then smallest (x,y) vertex.

Note

 When the check does not have properties and is not classified, polygons are not hierarchically merged.

The snapshots are generated around the center of the extent of each chosen error polygon.

Special Usage for Syntax Arguments as Statements

Unlike most Calibre OPCVerify commands, summary_report defines some arguments as statements. Statements are arguments that can be repeated multiple times within the summary report block. summary_report arguments fall into the following groups:

Table 4-9. Summary Report Argument Scope

Argument or Statement	Notes
path, title, file_name, snapshot_ids, snapshot_file_name	These arguments apply to the whole report and can be used only once inside a summary_report block.

Table 4-9. Summary Report Argument Scope (cont.)

Argument or Statement	Notes
summary, geometry_counts	These arguments add information to the summary table. The summary argument can be used multiple times.
errors, warnings	These arguments specify a list of layers that are either error or warning layers. A layer may only be present in one errors or warnings list.
show, layer	These statements apply to specific layers. Using them for the same layer more than once is an error.
histogram	Histogram statements with a files sub-argument apply to specific histogram files. Using the same histogram file name twice in any histogram statement is an error. A histogram statement without a files sub-argument can be used only once.
snapshots, snapshot_halo, snapshot_grid, snapshot_center_mark, snapshot_world_view, snapshot_bossung_table, snapshot_scale, snapshot_background, snapshot_picture_size, caption_units, ruler	These statements change the snapshot formatting attributes. Processing of a summary_report block starts with a default set of formatting attributes (5 snapshots per error layer, halo of 1.5um, no grid, no scale, white background, picture size of 480px, captions in microns). If a statement changes an attribute, the new value applies to any show statements following the statement until that attribute is changed again. Thus, the statements in this group can be (and probably should be) used many times. If summary_report contains no show statements, the default list of snapshots is generated using the attributes that are in effect at the end of the summary_report block.

Arguments

- **path destination_path** [with_datetime] [err_suffix *e_suffix*] [warn_suffix *w_suffix*] [clean_suffix *c_suffix*]

A required argument that specifies the directory path for the output files. If this directory does not exist, Calibre OPCVerify attempts to create it. If this directory contains an older report it will be overwritten.

Tip

 Use of suffix options in the **path** and **file_name** arguments allows you to keep multiple summary_reports in the same directory. Their use is recommended only if you intend to manage (and delete) unneeded generated directories on a regular basis.

The optional arguments append additional suffixes to the ***destination_path***:

- with_datetime — Adds a timestamp suffix, in the form *YYYYMMDD-hhmmss*. The time when the summary_report generation process begins is the timestamp that is used.
- err_suffix — Adds the specified suffix when error snapshots are generated. Not used if no error snapshots are generated for the summary_report.
- warn_suffix — Adds the specific suffix when warning snapshots are generated. Not used if no warning snapshots are generated for the summary_report.
- clean_suffix — Adds the specified suffix when no error snapshots are generated. Not used if error snapshots are generated for the summary_report.

Suffixes are appended in the order shown, and any suffixes not specified are skipped.

- file_name *name* [with_datetime] [err_suffix *e_suffix*] [warn_suffix *w_suffix*] [clean_suffix *c_suffix*]

An optional argument that sets a user-specified base name and optional suffixes for the generated files. When one or more of the optional suffix arguments are included, they create a compound name that is used as the HTML file name and subdirectory that contains the snapshot and histogram files. Both the HTML file and subdirectory are always stored inside the directory specified in the path argument. The default name if file_name is not specified is report_files.

- title *title_string*

An optional argument that sets the title for the summary report file. If the title needs to contain spaces, either single quotes (‘) or double quotes (“) can be used as delimiters to enclose the string. Note that whichever quotes you use as a delimiter cannot be used inside the string, but the unused delimiter type is permitted. The default is “Summary Report.”

- errors *layer1* [*layer2* ...]

An optional argument that specifies a list of layers which are considered as error layers. If those layers have a geometry, the error_suffix is used for its path and file names. More than one errors statement is allowed. A *layer* may only be present in one errors or warnings list.

- warnings *layer1* [*layer2* ...]

An optional argument that specifies a list of layers which are considered as warning layers. If those layers have a geometry, the warn_suffix is used for its path and file names. More than one warnings statement is allowed. A *layer* may only be present in one errors or warnings list.

A more complex output matrix of path and file suffix results is applied if both the error and warnings list arguments are supplied in the same summary_report block. Layers can be added these separate lists to be designated as errors or warnings. The summary_report command picks the highest priority error found (error > warning > clean) and uses that

directory for the output. The suffix that the Calibre OPCVerify tool picks can be derived from the logic in the following table:

Table 4-10. Cross-Referencing Error and Warning Lists

Specialized Lists Specified	Errors Detected?	Warnings Detected?	Are there Errors or Warnings on Layers Not in the Specified Lists?	summary_report Chooses the Following suffix:
No lists (default basic mode)	N/A	N/A	No Yes	clean error
Errors list only specified	No	N/A	No Yes	clean warning (no errors, but warnings were present)
	Yes		No Yes	error error (an error file is created since errors were detected)
Warnings list only specified	N/A	No	No Yes	clean error (any layer not explicitly specified as a warning layer is considered an error layer)
			No Yes	warning error
Both Errors and Warnings lists specified	No	No	No Yes	clean
		Yes	No Yes	warning
	Yes		No Yes	error
		Yes	No Yes	

- **summary [top] *key1 text1***

An optional argument that specifies one or more additional comment rows to be added to the Summary table. The lines are added using the *key1* argument as the first column entry and the *text1* as the second column entry. Both arguments can either be a single word, or a string if the string is enclosed in quotes. Each line is added in the order they are defined in the summary_report block, either at the end of the table or at the top when the optional top keyword is specified.

- **geometry_counts {hier | flat | both}**

An optional argument that sets the display type for geometry counts in the summary table to be hierarchical (default), flat, or both.

- **snapshots *num_per_layer***

An optional argument that sets the maximum number of snapshots per result layer that will be written to the report. The number of results is either the specified value or the number of errors in the Calibre OPCVerify output for that layer, whichever is less. (Default: 5. Minimum: 1. Maximum: 100.)

- **snapshot_halo *halo_microns***

An optional argument that sets the radius of the halo for each snapshot in microns. (Default: 1.5 microns. Maximum: 5.0 microns)

- **snapshot_grid {*step* [*line_width w*] [*color grid_color*] [*opacity grid_opacity*] } | none**

An optional argument that draws a grid on the snapshots, with a grid size of *step* in microns in both vertical and horizontal directions on the top layer. The *step* argument is required unless the *snapshot_grid none* argument is specified. The grid uses a line width of *w*, and a grid color and opacity specified using the same color conventions as specified in the *layer* argument for this command.

The default line width is 2 dbu, default grid color is black, and the default opacity is 0.6.

Specifying “*snapshot_grid none*” cancels drawing a grid on subsequent snapshots.

- **snapshot_center_mark {*mark_size* [*line_width w*] [*color mark_color*] [*opacity mark_opacity*] } | none**

An optional argument that draws a mark in the center of the snapshot window with the given *mark_size* and *line_width* in microns. The default grid color is black, the default opacity is 1.0, and the default line width is 1 pixel. This is primarily intended as a visual aid for snapshots that do not show rulers. Selecting none cancels center mark drawing.

- **snapshot_scale {*step ticks* [*steps#*] [*color scale_color*] [*opacity scale_opacity*] } | none**

An optional argument that draws a scale (in nm or microns) in the lower left corner of snapshots. When a grid is also present, the origin of the scale is aligned with the grid. The scale can be adjusted by specifying its *step* in microns and the number of *ticks* per step (these are both required values with no default). An additional *steps#* parameter, if specified, sets the number of steps printed (the default is 2).

The scale is captioned using the *caption_units* argument.

By default, rulers are colored black, which can be changed with *scale_color*. They are set with a default opacity of 1, which can be changed by specifying a *scale_opacity*.

Specifying “snapshot_scale none” cancels drawing the scale for subsequent snapshots.

- **snapshot_background color**

An optional argument that changes the background color of the snapshot to the specified color. The default color is white.

- **snapshot_picture size**

An optional argument that sets the size of each snapshot section in HTML px units, where 1 px is about 1/96 of an inch, but the actual value is determined by the user’s browser. The default size is 480 px.

- **snapshot_ids [sequential] [toc]**

An optional argument that assigns every snapshot a unique id that is printed with the list of snapshot properties. By default, snapshot ids have the form *layername_num*, where *num* is the sequential number of the snapshot within that layer. For example, bridge_4 is the fourth snapshot for the bridge layer.

- If the sequential keyword is specified, each snapshot is numbered sequentially without layer names.
- If the toc keyword is specified, HTML links to the snapshots are added to the table of contents.

- **snapshot_file_name template**

An optional argument that specifies a filename template with custom format specifiers. The “.svg” extension will be appended to the filename. If template contains whitespace and or a hash sign ‘#’, it must be enclosed in single or double quotes.

Table 4-11. Format Specifiers Supported for *snapshot_file_name*

Format specifier	Description
%c	Cell name
%d	Class ID
%f	Flat count
%g	Global sequential snapshot number (unique for the whole summary report)
%i	Snapshot ID
%l	Layer name
%m	Layer unique internal number
%n	Local sequential snapshot number within this layer
%p	Score property name, or name of first property if no scoring

Table 4-11. Format Specifiers Supported for snapshot_file_name (cont.)

Format specifier	Description
%s	Setlayer command name (e.g. area_overlay, measure_cd)
%v	Score property value, or value of first property if no scoring
%x	Location X-coordinate
%y	Location Y-coordinate
%%	Literal ‘%’

If `snapshot_file_name` is not specified, the default template is “`img_%m_%n`”. The template parameter must contain at least one of the following format specifiers or combinations to guarantee that generated snapshot filenames are unique:

- %g
- %i
- %l, %n
- %m, %n

If the value for a format specifier does not exist (for example “%f” for a layer which is not classified), a placeholder value such as “0” or “(none)” is output.

Layer unique internal numbers (%m) are derived by numbering all inputs and outputs of the OPCVerify run in the order coded in SVRF, starting from 0 for the first input.

For example, specifying:

```
snapshot_file_name "img_{%l}_{%s}_{%x}_{%y}_{%v}_{%f}_{%n}"
```

results in filenames similar to the following:

```
img_[cdiff_layer]_[contour_diff]_[4.740, 94.625]_[0.0177433]_[12].svg
```

```
img_[cdiff_layer]_[contour_diff]_[4.740, 94.505]_[0.0167317]_[4].2.svg
```

```
img_[cdiff_layer]_[contour_diff]_[4.745, -1.135]_[0.0166963]_[7].3.svg
```

- `snapshot_world_view [world_view_options | none]`

An optional argument that controls printing of the full-chip (world view) of the snapshot. The world view is the rectangle representing the full chip, with a rectangular mark showing the approximate snapshot location.

- `world_view_options`

Starts printing the full-chip view (world view) of the snapshot and/or changes the output format for subsequent layers. The world view is customizable, where *world_view_options* has the following syntax:

[width *image_width*] [height *image_height*] [background *bg_color*]
[color *mark_color*]

The height and width are specified in pixels using *image_width* and *image_height*. The background color of the full-chip rectangle is specified using *bg_color*, and the color of the snapshot location mark is specified using *mark_color*.

Default image options are used if no options are specified. The default image width and height are 200 pixels, the default background color is “lightgray,” and the default snapshot mark color is “red.”

- none

Cancels the full-chip view of the snapshot for subsequent layers.

- **snapshot_bossung_table** [*bossung_table_options* | none]

An optional argument that controls the output of a Bossung table of properties for eligible pw_annotated layers. A pw_annotated layer is eligible for Bossung table generation if its setlayer check used a Litho Model range-based image_set with one, two, or three ranges chosen from focus, dose, bias, or aerial. If a layer is not eligible for Bossung tables, the specification is silently ignored. The pw_annotation property values that are printed in Bossung tables are not output in the snapshot properties table.

The Bossung table for a focus/dose image_set has a column for each focus value, and a row for each dose value. Each focus/dose table entry is the property for that pw_image. Similar tables are produced for a focus or bias or dose/bias image_set. For focus/dose/bias image_sets, separate focus/dose tables are produced for each bias value. If the image_set involves aerial ranges, the dose is replaced by the aerial threshold. For single-range image_sets, the produced tables have a column with range values and another column with corresponding pw_annotation properties.

- *bossung_table_options*

Starts printing the bossung table and/or changes the table options for subsequent layers. Optional parameters can be used to customize the color-coding of the Bossung table cells, where *bossung_table_options* has the following syntax:

[color_error {*clr_error* | none}] [color_fair {*clr_fair* | none}]
[color_good {*clr_good* | none}]

clr-error specifies the background color for the cells containing errors, which are produced by property values that satisfy the original setlayer constraint. The default color is “lightpink.”

clr-fair specifies the background color for the cells containing property values that don't satisfy the setlayer constraint, but are within the “relaxed” pw_annotation constraint. The default color is “powderblue.”

clr-good specifies the background color for the cells containing property values that do not satisfy the pw_annotation constraint. The default color is “palegreen.”

If any color is specified as “none,” the corresponding cells are not color-marked. If no options are specified, the defaults are used.

- none

Cancels the output of Bossung tables for subsequent layers.

- **caption_units {u | nm}**

An optional argument that specifies the units used for the snapshot_scale and ruler (mark_inside) output as either microns (u) or nanometers (nm). The default is u (microns).

- **show layer1 [layer2 ...] [ruler1 [ruler2 ...]] [with context_layer1 [context_layer2 ...]]**

An optional argument that requests snapshots of the worst errors from the list of supplied layers.

If the show statement is not specified, summary_report generates snapshots for all layers that have classification or limiting statement blocks.

rulers can be added (by defining one or more *ruler* blocks, as described in “Ruler Selection Statements”) to the snapshot showing approximately where the measurements were taken that created the error marker.

If a “with” sub-argument is specified, the error layer is printed on top, layered on only the requested context layers in the specified order. If a “with” sub-argument is not specified, all the Calibre OPCverify input layers are shown as the snapshot context.

Only input and final output OPCverify layers can be used as *layer* and *context_layer* values for the show statement, since all intermediate layers are discarded before summary_report is run. To include a contour layer in the context, use the **output_window** command to copy it to an output layer.

- **ruler [color *ruler_color*] [opacity *ruler_opacity*] [width *ruler_width*]**

An optional argument that configures the appearance of rulers that were requested by the show statement. For information on the “ruler” statement, see the “Ruler Appearance Arguments” section of this command.

- **histogram [scale {log | linear | both}] [width *width*] [height *height*] [color *bar_color*] [background *background_color*] [from *prop_min*] [to *prop_max*] [step *prop_step*] [*ticks_per_step*]] [files *name1* [*name2* ...]]**

An optional argument that specifies histogram formatting for the histograms listed in the files sub-argument, or defines default histogram formatting if the files sub-argument is omitted.

- scale {log | linear| both}

- Indicates the histogram scale to use. If “both” is specified, the summary report will contain two histograms.
- *width width*
Sets the width of histogram images in HTML px units. Default is 800 px.
- *height height*
Sets the height of histogram images in HTML px units. Default is 400 px.
- *color bar_color*
Sets the color of the histogram bars. The default color is steelblue.
- *background background_color*
Sets the background color behind the histogram graph. The default color is linen.
- *from prop_min*
Sets a lower bound for the histogram values. If specified, property values lower than *prop_min* are not shown.
- *to prop_max*
Sets an upper bound for the histogram values. If specified, property values higher than *prop_max* are not shown.
- *step prop_step [ticks_per_step]*
Sets the step to use for X-axis labels and the number of ticks to print between labels. If step is not specified, both values are selected automatically.
- *files name1 [name2 ...]*
Specifies a list of histogram input files that this histogram block applies its settings to. If this keyword is not specified, all histogram files will be written, excluding histogram files explicitly named by other histogram commands.
- *layer layer [fill fill_color [pattern pattern_number] [opacity fill_opacity] [line line_color [line_opacity line_opacity] [line_width width]]]*
An optional argument that specifies that each layer statement sets the appearance of the specified *layer* in the snapshot images:
 - *fill_color* and *line_color* can be:
 - An RGB value in any format permitted by the CSS2 specification:
<http://www.w3.org/TR/CSS2/syndata.html#value-def-color>
#rgb and #rrggbb formats are allowed, but only if they are enclosed in single or double quotes.

- Any of the color keyword names recognized by SVG as documented in the W3C specification:

<http://www.w3.org/TR/SVG/types.html#ColorKeywords>

or their RGB code equivalents. If the latter are used, no spaces are allowed in the string (correct: “rgb(135,206,235)” incorrect: “rgb (135, 206, 235)”).

For additional information on the SVG format, see “Notes on SVG” later in this command description.

- *pattern* can be one of the following predefined names:

- diagonal_right_wide
- diagonal_right (or diagonal_1)
- diagonal_left_wide
- diagonal_left (or diagonal_2)
- carets
- light_speckle
- speckle
- alt_light_speckle
- alt_speckle
- triangle_small
- wave_small
- wave
- right_slope
- left_slope
- plus
- brick
- circles
- carpet (or carpet_1)
- solid (the default value)
- clear (this is same as pattern solid opacity 0.0)

All pattern names are same (or similar) as the Calibre WORKbench pattern names.

- *fill_opacity* and *line_opacity* set the transparency for the shapes and lines on the layer, respectively. They can be values ranging from 0.0 (invisible) to 1.0 (opaque).
- *width* sets the width of the lines in the drawn layer. The default is 2 dbu.

If no layer statements are specified for an output layer, the output is a 2dbu-wide black line for shapes on the layer.

If the *fill_color* is specified but the *line_color* is not, there is no outline. If the *line_color* is specified but the *fill_color* is not, then the layer is not filled.

Ruler Selection Statements

As a visual aid, *summary_report* has the capability to add rulers to the snapshot images once they have been rendered.

Note

 Due to the way that rulers are added to the snapshot file image after Calibre OPCVerify has completed its run, rulers are not directly associated with the error markers. Instead, they are placed according to the search parameters set in the *mark_inside* sub-argument. Incorrectly configured rulers can result in rulers appearing in different locations than the markers, or no rulers appearing at all. In some cases, no ruler appearing is normal behavior, such as for hard bridging, since hard bridge rulers have a measurement width of zero.

Two statements specify how rulers are added to the snapshot images: the *mark_inside* sub-argument described in this section and the ruler appearance configuration statement (described in the “Ruler Appearance Arguments” section of this command).

mark_inside uses the following syntax:

```
mark_inside ruler_halo ruler_method ruler_args \
[[not] {inside|outside} filter_layer]
```

Multiple instances of a ruler (*mark_inside*) can be specified per snapshot. Each ruler statement is separated from the next by spaces.

- *ruler_halo*

A required argument that specifies the search distance to attempt to place a ruler near a marker. The ruler search region is a square with a radius of *ruler_halo* in microns, starting from the center of the snapshot. Regardless of the ruler type selected, *summary_report* only selects edges inside this region. It is important to select a value large enough to find relevant edges, but smaller than the *output_window* or *snapshot_halo* values.

- *ruler_method ruler_args*

A required argument that sets the ruler measurement as one of the following four types, with related arguments:

- *distance* {internal | external | enclosure} *layer* {separation *s* | *to_layer*}

Finds the minimum internal, external, or enclosure distance between objects on the specified *layer(s)*, and connects the points with a ruler.

Most commonly used with **bridge**, **pinch**, and **measure_distance** commands.

One-layer distance rulers must specify an “*s*” separation argument (see the description of separation in the **measure_distance** command).

Enclosure must specify a *to_layer*.

- **diff** *layer1 layer2*

Finds the maximum distance between the nearest points of *layer1* and *layer2*, connecting the points with a ruler.

Most commonly used with the **contour_diff** and **meefcheck** commands.

- **cd** *layer ref_layer {external | internal} {min | max} cd_max x [cd_min y]*

Finds either the minimum or maximum external / internal CD and connects the points. See the description of **measure_cd** for more information on these arguments.

Most commonly used with the **measure_cd** and **measure_cdv** commands.

- **epe** *contour target {min | max} [magnitude]*

Find either the minimum or maximum distance from the edges of the *target* layer to the nearest properly oriented edge of the *contour* layer. The sign of the distance is the same as the **measure_epe** command you associate with this ruler, and is only valid when associated with the **measure_epe** command. However, if the magnitude argument is specified, the ruler uses the absolute value of the distance instead.

- [not] {inside | outside} *filter_layer*

An optional argument that sets a filter layer for ruler creation.

If a ruler specification includes [not] {inside | outside} *filter_layer*, the filter is applied inside the ruler search region. There are two possible strategies of using this sub-argument:

- If the operation that produced the error layer has a filtering sub-argument, that filtering sub-argument can and should be repeated in the ruler specification.
- Many Calibre OPCVerify operations (such as **measure_cd** and **bridge** with a large enough **output_extent**) produce error markers that enclose the edges that have caused the trouble. In such a case the output layer itself can be used as the filter in conjunction with a relatively large *ruler_halo*.

Ruler Appearance Arguments

A ruler’s appearance can be configured with the following syntax:

```
ruler [color ruler_color] [opacity ruler_opacity] \
[width ruler_width]
```

where:

- **color ruler_color**
Sets the color of the ruler. The default color is black (see the preceding [color specification](#) with the *fill_color* and *line_color* options for more information).
- **opacity ruler_opacity**
Sets the opacity of the ruler. The default opacity is 1.0 (solid).
- **width ruler_width**
Sets the width of the ruler line in HTML px units. The default is 0.5 px.

Notes on SVG

Summary reports are HTML files, which can be viewed in any Web-capable browser. The snapshots and histograms are rendered in Scalable Vector Graphics (SVG) format. Users of older browsers may need to download the SVG plug-in from Adobe Corporation:

<http://www.adobe.com/svg/viewer/install/main.html>

Each summary report snapshot can be clicked to view the image in a larger size in a separate window. Navigation controls for standard SVG viewers are shown in the following table. (ASVG, used with Internet Explorer, has different viewer controls.)

Table 4-12. SVG View Controls

Keys	Function
+, Numpad +	Zoom in
-, Numpad -	Zoom out
0, Numpad 0, Numpad 5	Reset pan/zoom controls
Left arrow, Numpad 4	Pan left
Up arrow, Numpad 8	Pan up
Right arrow, Numpad 6	Pan right
Down arrow, Numpad 2	Pan down
Esc	Close the window

Examples

The first part of this example establishes the layers that will be output and performs various Calibre OPCverify operations.

```
background clear

layer poly    hidden
layer active   hidden
layer poly_opc visible atten 0.06
```

```
// contour layers for three different dose conditions
setlayer cntr1 = image optical f0 dose 0.98 resist_model cm1
setlayer cntr2 = image optical f0 dose 1.00 resist_model cm1
setlayer cntr3 = image optical f0 dose 1.02 resist_model cm1

setlayer cntr_diff = contour_diff >0.012 poly cntr1 cntr3 \
    max_search 0.05 property {max} classify {
        context poly poly_opc
        halo 0.5
        score bin_size 0.0005 largest property max
    }

setlayer bridge = bridge poly outside poly_opc cntr1 <0.028 separation 0.1 \
    property {min} classify {
        context poly poly_opc
        halo 0.5
        score bin_size 0.0005 smallest property min
    }

setlayer pinch = pinch poly inside poly_opc cntr3 <0.05 separation 0.1 \
    property {min} classify {
        context poly poly_opc
        halo 0.5
        score bin_size 0.0005 smallest property min
    }

output_window cntr1 halo 0.5 around cntr_diff bridge
output_window cntr2 halo 0.5 around cntr_diff bridge pinch
output_window cntr3 halo 0.5 around cntr_diff pinch
```

Each of the following result layers takes one or more of the layers above and writes a raw ASCII histogram file (cntr_diff.hist, bridge.hist, and pinch.hist) as its output.

```
setlayer cdiff_hist = contour_diff >0.012 poly cntr1 cntr3 max_search 0.05
property {max} histogram {
    file cntr_diff.hist property max bin_size 0.0005
}

setlayer br_hist = bridge poly outside poly_opc cntr1 <0.028 separation
0.1 property {min} histogram {
    file bridge.hist property min bin_size 0.002
}

setlayer pn_hist = pinch poly inside poly_opc cntr3 <0.05 separation 0.1
property {min} histogram {
    file pinch.hist property min bin_size 0.001
}
```

Finally, the summary report block is configured for outputs:

```
summary_report {
    path ./html
    snapshots 15
    snapshot_halo 0.5
    snapshot_picture_size 480
    snapshot_background linen
    snapshot_grid 0.02
    // select which layers to show:
    // contour diff with a diff-style ruler
    show cntr_diff mark_inside 0.06 diff cntr1 cntr3 with poly cntr1 cntr3
    cntr2
    // bridge with a distance-external style ruler
    show bridge mark_inside 0.15 distance external cntr1 separation 0.5
        with poly cntr1 cntr2
    // pinch with a distance-internal style ruler
    show pinch mark_inside 0.2 distance internal cntr3 separation 0.5
        with poly cntr3 cntr2
    // layer color configuration
    layer bridge line limegreen line_width 0.02
    layer pinch line cyan line_width 0.02
    layer cntr_diff line yellow line_width 0.02
    layer poly fill silver
    layer poly_opc line navy
    layer cntr_norm fill palegreen opacity 0.4 line black
    layer cntr_over fill powderblue opacity 0.4 line black
    layer cntr_undr fill red opacity 0.4 line black
    // histogram selection and configuration
    histogram height 300 scale both
    histogram height 300 scale both from 0.02 files pinch.hist bridge.hist
```

svrf_var_import

Setup command

Imports a variable from the SVRF rule file.

Usage

svrf_var_import varname

Description

Imports an SVRF rule file variable into the Calibre OPCVerify setup file. Variables must be declared in the SVRF rule file using the VARIABLE command. Once imported, variable names can be substituted using \$varname references, and are treated as if they are Tcl variables in behavior.

- If a listed variable is not declared in the SVRF rule file, the Calibre OPCVerify run halts with an error.
- If a listed variable is declared in the Calibre OPCVerify setup file, it will be overwritten by the imported variable.

Arguments

- *varname*

A required argument specifying the variable name to import, as declared in the SVRF rule file.

Examples

Assume the following definitions in the SVRF rule file.

```
VARIABLE dose1 1.0
VARIABLE dose2 0.9
```

The following command inside the Calibre OPCVerify setup file imports the variables into the Calibre OPCVerify environment:

```
svrf_var_import dose1
svrf_var_import dose2
```

The new variables can then be used in commands:

```
setlayer contour1 = image optical f0 dose $dose1 aerial .3
setlayer contour2 = image optical f0 dose $dose2 aerial .3
```

tilemicrons

Setup command

Specifies the size of an Calibre OPCVerify tile.

Usage

tilemicrons *tile_size* [adjust | exact]

Arguments

- ***tile_size***

A required argument that specifies a size for tiles in microns. Large designs are broken into tiles of *tile_size*. The default is 100 um, though this size should be reduced because it is too large for current design nodes.

- **adjust**

An optional argument used when the run performs dense simulation. The default behavior adjusts the tilemicrons setting slightly to make it closer to a full multiple of simulation frames if possible. A simulation frame is an FFT frame, which is usually about 6 to 10 microns in size. Each frame has a part around the edge that will be invalid; Calibre OPCVerify only keeps the valid part of the frame.

Note



The adjust algorithm has been changed in Calibre 2015.4 for Calibre OPCVerify and Calibre nmOPC. It now gives different adjusted tile sizes compared to 2015.3 and previous releases. The new results should have improved simulation performance.

- **exact**

An optional argument that uses the *tile_size* as specified.

Examples

```
tilemicrons 120
```

topo_model_load

Configuration command

Loads a topographical (“topo”) model into Calibre OPCVerify.

Usage

```
topo_model_load name {file | '{' inline_model_text '}'}
```

Description

This command is used only when a [Litho Model Format](#) file is not present.

The topo model can then be used with [image](#) commands.

For more information on using topo models in Calibre OPCVerify, see the section “[Topographical Models in Calibre OPCVerify](#)” on page 56.

Note

 This command is ignored if a litho model is in use; the topo_model data entry is read from the *Lithomodel* file instead.

Arguments

- ***name***

A required argument specifying the user name for the model that is referenced from the [image](#) command.

- ***file* | '{' *inline_model_text* '}'**

A required argument either pointing to a separate file located in the current working directory or in the [modelpath](#), or inlined topographical model text enclosed in braces.

zplanes_model_load

Setup command

Loads a zplanes model into Calibre OPCVerify.

Usage

zplanes_model_load *name* {*file* | '{' *inline_model_text* '}'}

Description

This command is used only when a [Litho Model Format](#) file is not present.

The zplanes model can then be used with [image](#) commands.

Arguments

- *name*

A required argument specifying the user name for the model that is referenced from the [image](#) command.

- *file* | '{' *inline_model_text* '}'

A required argument either pointing to a separate file located in the current working directory or in the [modelpath](#), or inlined zplanes model text enclosed in braces.

Using Setlayer Options as Operations

Calibre OPCVerify uses setlayer options (sometimes referred to as setlayer operations, setlayer subcommands, or setlayer commands) to create and derive layers.

You create Calibre OPCVerify layers using the design layers you defined with the [layer](#) command, and you will eventually use one or more of the Calibre OPCVerify layers as output in your LITHO OPCVERIFY calls in your SVRF file.

All setlayer operations consist of a single *case-sensitive* line that uses the syntax:

```
setlayer output_layer_name = setlayer_operation
```

A *setlayer_operation* is one of the following types:

- [Image Operations](#)
- [DRC-type Operations](#)
- [Verification Control Operations](#)

Image Operations

An image operation performs optical and resist simulation using geometry on visible layers defined by the layer command, subject to the command options.

It outputs a polygon layer with the aerial contour at threshold *value* or a printimage contour defined by the resist model *model_name*. In the case of multiple exposures, different options can be specified for each exposure, allowing you to simulate a wider array of lithographic conditions.

Image operations are described in the [image](#) syntax page, in this chapter.

DRC-type Operations

The DRC-type operations are implemented for use with Calibre OPCVerify setlayer commands. They must be entered in case-sensitive text.

Table 4-13. Calibre OPCVerify Setlayer Operations (DRC-type)

Operation	Description
and	Performs multi-layer AND on inputs.
copy	Copies the layer to a new layer.
external	Performs an external check within the specified region.
internal	Performs an internal check within the specified region.
not	Performs a multi-layer NOT of all inputs.
or	Performs a multi-layer OR of all inputs.
size	Performs the sizing operation on the input layer, expanding or shrinking polygons on the input layer by the specified amount.
width	Performs a width check on the specified region.
xor	Performs a multi-layer XOR of all inputs.

Note

 These DRC commands are only equivalents to their DRC counterparts; you cannot use DRC on Calibre OPCVerify-generated layers. Use the commands shown above with Calibre OPCVerify layers.

Verification Control Operations

The customized verification control operations are implemented for use with Calibre OPCVerify setlayer commands. They must be entered in case-sensitive text.

Table 4-14. Calibre OPCVerify Setlayer Operations (Verification Control)

Operation	Description
annotate	Annotates an input layer with shapes matching the selection criteria.
area_compute	Checks for shapes that match a certain area constraint.
area_overlay	Performs a contact / via alignment check.
area_ratio	Checks for shapes that meet a certain area ratio constraint.
bandcheck	Checks CD accuracy for inner and outer tolerance band violations.
bridge	Checks for bridging problems.
bridge_tolerance	Checks for outer tolerance band violations.
build_tolerance	Generates a tolerance zone.
center_shift	Measures projections of contacts and vias from a target.
circularity_compute	Measures the circularity of simulated contours.
cornerchop	Chops out a triangular section off all corners.
contour_diff	Calculates the error factor between two contours, similar to meefcheck .
dofcheck	Calculates the depth of focus (DOF) for target shape edges under different defocus conditions.
empty_layer	Creates an empty layer.
end_cap	Checks endcap coverage.
extra_printing	Checks for extra printing features.
filter_generate	Generates a filter layer for various commands.
gate_stats	Computes statistics for a gate area.
gauges	Adds properties to an output layer based on user-defined gauges.
holes	Returns holes inside layer shapes.
identify_corner	Identifies line fragments of edges near corners.
identify_edge	Identifies line ends or jogs.

Table 4-14. Calibre OPCVerify Setlayer Operations (Verification Control)

Operation	Description
<code>imax_check</code>	Analyzes the aerial image intensity for an image contour against an Imax constraint.
<code>imin_check</code>	Analyzes the aerial image intensity for an image contour against an Imin constraint.
<code>interact</code>	Checks for shapes that interact with each other.
<code>measure_cd</code>	Checks width and space CD accuracy.
<code>measure_cdv</code>	Checks CD between two contours versus a target CD.
<code>measure_cross_section</code>	Checks the cross section of a contour versus a target shape.
<code>measure_distance</code>	Checks the distance between an edge and the nearest edge.
<code>measure_epe</code>	Checks if a layer's EPE fails a specified constraint.
<code>meefcheck</code>	Checks if a polygon's MEEF values fail a specified constraint.
<code>nilscheck</code>	Checks the Normalized Image Log-Slope (NILS) for two or more contours for a specified range constraint.
<code>not_printing</code>	Checks for features that do not print.
<code>notchfill</code>	Fills in notches on polygons.
<code>pinch</code>	Checks for pinching problems.
<code>pinch_tolerance</code>	Checks for inner tolerance band violations.
<code>polygon_extent</code>	Checks the area of polygons in x and y dimensions.
<code>pvband</code>	Generates a process variation band from multiple contours.
<code>pwcheck</code>	Checks the common process window versus a process window.
<code>shadow_bias</code>	Shifts a layer according to a shadow bias model.
<code>shift</code>	Shifts the polygons on a layer a distance in x and y.
<code>show_annotation</code>	Writes annotation markers to an output layer.
<code>veb_simulate</code>	Applies an etch bias based on an etch model to a contour layer.
<code>window</code>	Outputs context clips based on an error layer.

Setlayer Operations Reference

All commands that create or modify layers are known as setlayer operations.

The following sections list all [setlayer](#)-related commands.

Table 4-15. Calibre OPCVerify Setlayer Operations List

Command	Description
and	Performs a multi-layer AND of all inputs.
annotate	Creates an annotated copy of a bridge, pinch, or area_overlay layer.
apa_check	Computes and checks the average printability analysis (APA) function value in relation to specified regions.
area_compute	Checks area coverage. This command is mainly intended for contact layers.
area_overlay	Performs a contact alignment check, reporting possible shift issues.
area_ratio	Checks the area ratio of a shape versus a constraint.
asraf_print_check	Detects under-printing around ASRAF features.
bandcheck	Checks for tolerance band violations.
bridge	Checks for bridging conditions.
bridge_tolerance	Checks for outer tolerance band violations on a contour layer.
build_tolerance	Generates a tolerance zone (outer or inner) around polygons on a layer.
center_shift	Measures displacement from the centers of contour polygons to a corresponding target.
circularity_compute	Computes the circularity of layer shapes as a method to check contact areas.
contour_diff	Compares the difference between two contours.
copy	Copies the specified layer to a new layer.
cornerchop	Chops convex and concave corners on polygon layers.
curvature_check	Checks if the radius of curvature for a specified layer (usually a curvilinear mask or an image) is larger than a user-specified value.
dofcheck	Tests the depth of focus value for multiple contours. Used for process window checks.

Table 4-15. Calibre OPCVerify Setlayer Operations List (cont.)

Command	Description
<code>empty_layer</code>	Creates an empty derived layer that can be output to SVRF or used in other operations.
<code>enclosure</code>	Performs an enclosure check within the specified region.
<code>end_cap</code>	Finds end caps on edges compared to a contour layer.
<code>external</code>	Checks a region for external distance violations.
<code>extra_printing</code>	Detects extra printing features (SBARs and sidelobes).
<code>filter_generate</code>	Generates filter layers to use with other verification commands.
<code>gate_stats</code>	Computes specified statistics for gate structures.
<code>gauges</code>	Calculates properties on gauges supplied as a layer.
<code>holes</code>	Returns polygons that fit in holes found in the input layer.
<code>identify_corner</code>	Identifies line fragments of edges near corners.
<code>identify_edge</code>	Identifies line ends and jogs.
<code>image</code>	Creates an image contour.
<code>imax_check</code>	Analyzes the aerial image intensity from an image command and compares it against an imax constraint.
<code>imin_check</code>	Analyzes the aerial image intensity from an image command and compares it to an imin constraint.
<code>intensity_ilsccheck</code>	Computes and checks the image log slope (ils) for an image previously defined by setlayer image.
<code>intensity_meefcheck</code>	Computes and checks Mask Error Enhancement Factor (MEEF) for an image contour.
<code>intensity_nilscheck</code>	Computes and checks the normalized image log slope (nils) for an image previously defined by setlayer image.
<code>interact</code>	Checks for interactions between polygons.
<code>internal</code>	Performs the internal check for the specified region.
<code>maskgen</code>	Generates a mask layer for debugging purposes.
<code>measure_cd</code>	Checks accuracy of printed space or width CDs.
<code>measure_cdv</code>	Measures the CD variation between two contour layers relative to a target layer.
<code>measure_cross_section</code>	Measures the cross section of target polygons versus a constraint.
<code>measure_distance</code>	Measures distances between edges on a layer.

Table 4-15. Calibre OPCVerify Setlayer Operations List (cont.)

Command	Description
measure_epe	Measures the EPE (edge placement error) between target and contour edges.
meefcheck	Checks the target's MEEF values versus a specified constraint.
nilscheck	Checks the NILS or ILS value versus a specified constraint.
not	Performs a multi-layer NOT by subtracting each input from <i>input1</i> , starting with <i>input2</i> .
not_printing	Detects non-printing features on a layer.
notchfill	Attempts to fill in notches or nubs on the specified input layer. This command is designed to reduce irregularities in polygon edges.
or	Performs a multi-layer OR of all inputs.
pinch	Checks for pinching errors that meet the specified constraint.
pinch_tolerance	Checks for inner tolerance band violations.
polygon_extent	Returns polygons based on their extents.
pvband	Generates a process variation band from multiple contours. It forms the OR of the XOR of each unique pair of input image layers.
pwcheck	Checks if the calculated common process window fits inside the process window (PW) in the user-specified location.
shadow_bias	Defines a shadow bias model options block.
shift	Shifts the shapes on the specified layer by the specified amounts. It is designed to be used on misaligned masks.
show_annotation	Shows markers on annotated layers.
size	Performs the sizing operation on the input layer, expanding or shrinking polygons on the input layer by the specified amount.
sraf_apa	Computes the average printability analysis (APA) for SRAF regions.
sraf_print_flux	Computes intensity flux for SRAFs on an image layer.
tilegen	Creates an annotated layer to display tile and cell boundaries with properties.
veb_simulate	Performs etch bias simulation.

Table 4-15. Calibre OPCVerify Setlayer Operations List (cont.)

Command	Description
width	Performs the equivalent of the WITH WIDTH operation in Calibre DRC, returning the polygons on the specified layer that meet the specified constraint.
window	Creates a window context region.
xor	Performs an exclusive OR of the inputs.

and

DRC-type operation

Performs a multi-layer AND of all inputs.

Note

 That this command works only on Calibre OPCverify layers, and its results should not be used outside of Calibre OPCverify.

Usage

and *input1* . . . *inputN*

Arguments

- *input1* . . . *inputN*

A required argument, specifying the layers to operate on.

Examples

The following example selects all polygon areas common to both layer2 and layer3 polygons:

Figure 4-9. Two-layer Boolean AND Operation



If you rewrote the two-layer Boolean and operation from Figure 4-9 as:

`and layer3 layer2`

the operation does not produce a different geometric output because Boolean and statements are commutative.

The following is a common type of layer derivation:

```
setlayer gate = poly and diff
```

annotate

Verification control

Creates an annotated copy of a bridge, pinch, or area_overlay layer.

Usage

```
annotate input_layer annotation_type [options ...]
```

Description

This command makes a copy of the specified *input_layer*, annotating each geometry on the layer based on the qualification criteria. The annotation is not visible (no additional shapes or properties are added) but the result layer can then be used with commands that support annotation (currently only [bridge](#), [pinch](#), and [area_overlay](#)).

The following annotation type-related options are available:

- double_via

layer_lower [*layer_upper*] *max_space* *max_double_via_space*

Annotates as double-via polygons two polygons on the input layer that meet all of the following criteria:

- Electrically connected by a segment in *layer_lower* (or the optional *layer_upper*)
- Separated by a distance less than *max_double_via_space*

- orientation *horiz_edge* *horiz_min_len_microns* *vert_edge* *vert_min_len_microns*

The orientation annotation type marks target edges.

- Horizontal edges with length $\geq \text{horiz_min_len_microns}$ are annotated as “*horiz_edge*.”
- Vertical edges with length $\geq \text{vert_min_len_microns}$ are annotated as “*vert_edge*”.
- Angled and short edges are annotated as “*default*”.
- An edge pair where both edges are “*horiz_edge*” is given the class “*horiz_edge*”. An edge pair where both edges are “*vert_edge*” is given the class “*vert_edge*”.
- All other edge pairs are given the class “*default*”.

The main intended use for “orientation” annotation is to allow specification of different error criteria for soft bridge/pinch defects between long target lines in the horizontal and vertical direction, such as situations where a dipole light source is used. The minimum edge lengths are supported so that line-ends may be considered separately, rather than as horizontal/vertical lines. This permits a second practical usage, in which different error criteria may be applied to defects involving line-ends rather than between pairs of long lines. If this distinction is not required, the minimum lengths may be set to 0.

Arguments

- ***input_layer***
A required argument specifying the input layer. For double-via annotation, this should be the via layer.
- ***annotation_type***
A required argument specifying the annotation type (described below). Only one ***annotation_type*** can be specified per **annotate** command.
- ***options***
An optional argument specifying options related to the chosen annotation type. See the Description for more information on available options.

Examples

Example 1

```
annotate contact1 double_via poly1 active1 max_space 0.07
```

Detects double-vias on the contact1 layer using the poly1 and active1 layers as a context.

Example 2

Assume an x/y asymmetric source such that vertical lines (with horizontal line-ends) are tightly resolved with a target CD of 50 nm, whereas horizontal lines are less well resolved and their minimum design CD is 80 nm. Also assume it is desired that soft defects involving line-ends will have their own separate (default) error constraints, or maybe it is simply desired to create an error layer containing only soft line-end defects. Then it might be reasonable to set the minimum lengths to 2 times the CD in each direction as follows:

```
setlayer xytarget=annotate target orientation \
    horiz_edge 0.10 vert_edge 0.16
```

Then if the bridge error criterion between tight vertical lines is 40 nm, between loose horizontal lines is 60 nm, and where line-ends are involved is 50 nm, then typical bridge/pinch operations might be as follows:

```
setlayer all_bridge = bridge xytarget cntr default <= 0.05 \
    horiz_edge <= 0.06 vert_edge <= 0.04 separation 0.1

setlayer hrz_bridge = bridge xytarget cntr default ignore \
    horiz_edge <= 0.06 vert_edge ignore separation 0.1

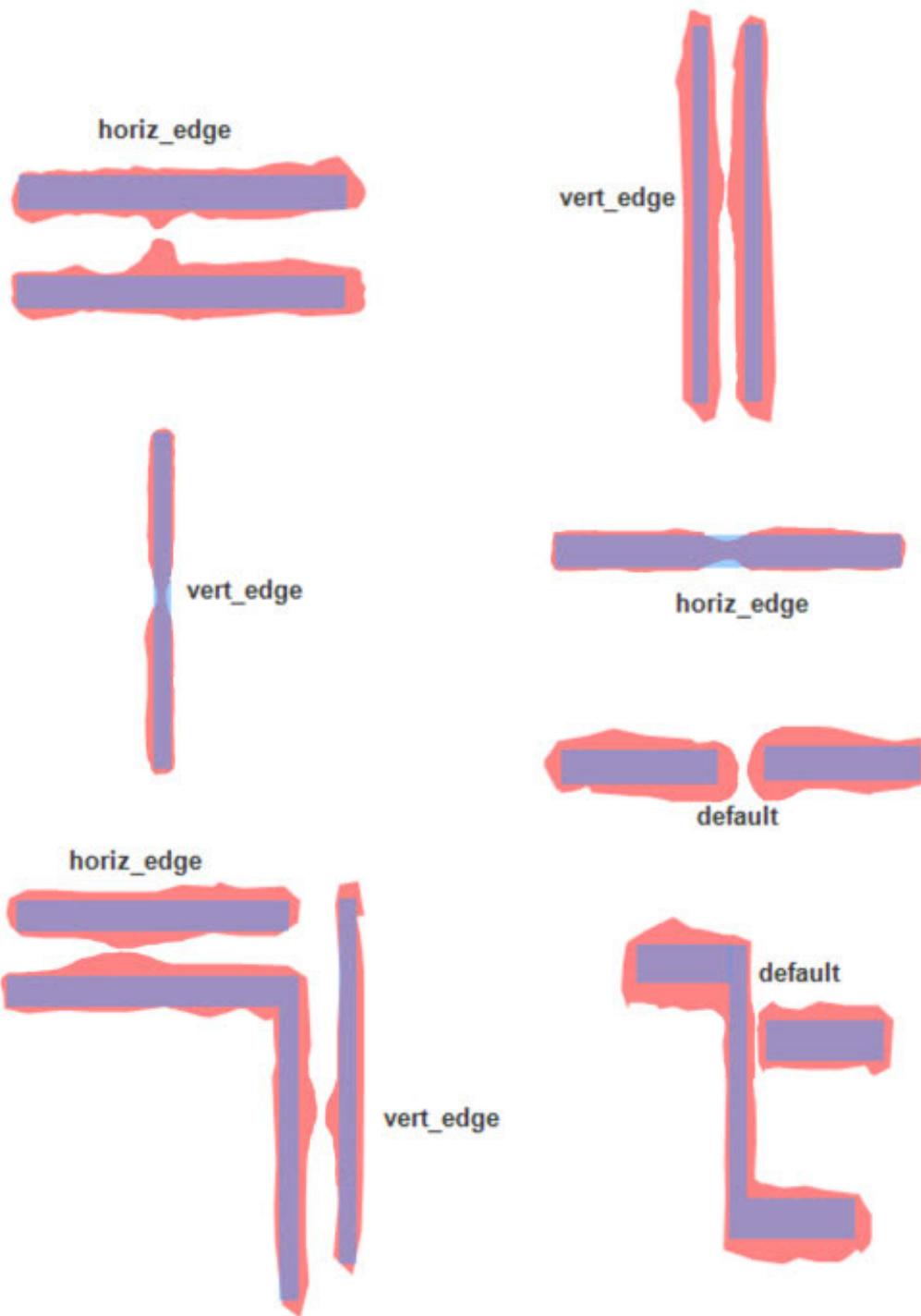
setlayer vrt_bridge = bridge xytarget cntr default ignore \
    horiz_edge ignore vert_edge <= 0.04 separation 0.1

setlayer end_bridge = bridge xytarget cntr default <= 0.05 \
    horiz_edge ignore vert_edge ignore separation 0.1
```

Note that all hard bridge/pinch defects will be output to all error layers. The annotated constraints apply only to soft defects.

Figure 4-10 illustrates how annotation classes would be applied to some soft bridge and pinch defects for this example. Blue is target and pink is the image contour.

Figure 4-10. annotate Results for Orientation



apa_check

Verification control

Computes and checks the average printability analysis (APA) function value in relation to specified regions.

Usage

```
apa_check image_layer constraint
  [[not] {inside | outside} filter_layer]
  {{include_layer inc_layer [size_by val_um]}} |
  {{exclude_layer exc_layer include_condition constraint
    [include_condition_over_under value]}}
stochastic_model modelName
# error-centric section
[property '{'
  apa [apa_actual] [apa_area] [apa_pixel_count]
'}']
[classify, limit, or histogram block]
```

Description

The `apa_check` command computes and checks the average printability analysis (APA) function value of the specified regions in the image layer. The value is between 0 and 1.0 and is normalized by the region area.

This command has two modes:

- include (selected with the `include_layer` argument) for which areas under the specified layer regions are checked. It is designed to work on SRAFs.
- exclude (selected with the `exclude_layer` argument) for which areas not under the specified layer regions are checked. Checked areas must also meet a secondary include condition constraint. It is designed to work on pixels.

This command also requires a stochastic model to have previously been loaded with a [stochastic_model_load](#) setup command.

Arguments

- ***image_layer***

A required argument specifying a previously-generated [image](#) layer to be checked.

- ***constraints***

A required argument specifying a standard Calibre OPCVerify constraint to be applied to the computed APA value. Two-sided constraints are supported. A computed APA value satisfying the constraint is output as an error marker.

- **include_layer *inc_lay* [size_by *val_um*]**

A required argument instructing the command to perform the APA computations on areas covered by regions in the specified layer. The layer can be pre-scaled using the optional size_by argument.

- **exclude_layer *exc_lay* include_condition *constraint* [include_condition_over_under *value*]**

A required argument instructing the command to perform the APA computations on areas NOT covered by regions in the specified layer AND whose pixels layout region meets the constraint specified in the required include_condition keyword.

The constraint applies to the resist intensity grid value after subtracting for the CM1 resist model constant term. (Note that the stochastic flow is available only for CM1 resist models.)

Note

 Use the resist threshold (THRESHOLD and CONST_TERM parameters in the resist model) as the comparison value for the **constraint**. For example, if the THRESHOLD is 2.050000e-01, and the CONST_TERM is 0, the comparison value is 0.205-0 = 0.205.

The optional include_condition_over_under keyword oversizes the include regions by the specified value. The default is 0.0.

- **stochastic_model *modelName***

A required argument specifying the name of the stochastic model to use for computations. The model must have been previously loaded using the [stochastic_model_load](#) command.

- **property '{' apa [apa_actual] [apa_area] [apa_pixel_count] '}'**

An optional argument defining a property block, with property values enclosed in braces.

- apa — A required keyword when a property block is declared, the APA property value contains the result of the APA computation normalized over the area of the SRAF regions. Its value is dimensionless and is normalized by the SRAF region.
- apa_actual — An optional keyword that returns the raw computed APA value without any normalization.
- apa_area — An optional keyword that returns the area of the layout region from which the APA computation was done.
- apa_pixel_count — An optional keyword that returns the number of simulation grid points used in the APA computation.

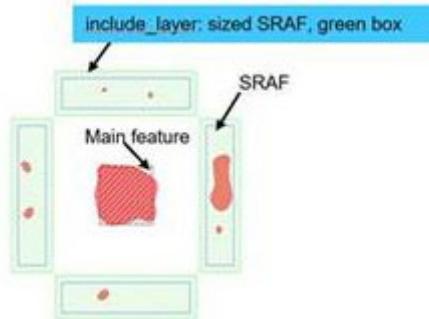
- **classify, limit, or histogram block**

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

Examples

Example 1: include_layer Covering SRAF Regions

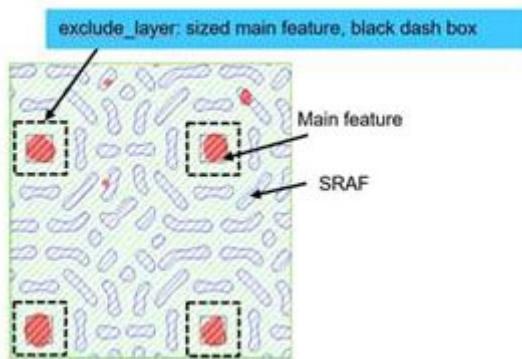
The following example checks APAs for an SRAF layer:



```
setlayer size_sraf_apa = apa_check cntr_0.1.0 > 0.00 \
include_layer sraf size_by 0.003 \
stochastic_model stcm4 property {
apa
apa_actual
apa_area
apa_pixel_count
}
```

Example 2: exclude_layer Excluding Main Features

The following example checks APAs for pixels outside of the main features specified on the exclude_layer:



```
setlayer relative_apa = apa_check cntr_0.1.0 > 0.001 \
exclude_layer excl_layer include_condition >= 0.34 \
include_condition_over_under 0.002 \
stochastic_model stcm4 \
property {
apa
apa_actual
apa_area
apa_pixel_count
}
```

area_compute

Verification control

Checks area coverage. This command is mainly intended for contact layers.

Usage

```
area_compute simulation_layer [reference reference_layer]
  [[not] {inside | outside} filter_layer]
  {[not] constraint}
  max_extent value
  [output_type [reference | simulation {extents | centers size_microns}]
   [exception [also [property_value prop_value]]]
   # error-centric section
   [property '{'
    area
    area_ref
   '}'] [classify, limit, or histogram block]]
```

Description

The area_compute function checks the area of each shape in the specified *simulation_layer* and outputs those shapes which meet the constraint, after filtering away shapes exceeding the value in either X or Y.

This function (and the [area_ratio](#) function) are good for checking area coverage, and are designed primarily to be used for contact layers.

Concurrency Support

Concurrency is supported for multiple **area_compute** calls that differ only in the *constraint* argument.

Arguments

- ***simulation_layer***
A required argument, specifying the simulation layer to check.
- reference *reference_layer*
An optional argument specifying a reference layer, usually the target layer.
- [not] {inside | outside} *filter_layer*
An optional argument specifying a filter layer. The filter is applied against either the reference layer or the simulation layer (default), depending on the setting used for the *output_type* parameter.
Only output polygons that also satisfy the filter (are completely “inside” or “outside” *filter_layer*, with the optional “not” inverting the selection) are output.

- [not] **constraint**

An argument defining the constraint on the areas of the shapes in the specified simulation layer. It uses the user units that you set (default is square um).

Specifying the “not” parameter returns shapes that do not pass the constraint.

This argument interacts with other arguments:

- It is not allowed if “exception” without “also” is specified.
- It is optional if a histogram block is specified.
- In all other cases, it is a required argument.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- **max_extent value**

A required argument defining a filter for the maximum extent of the input shapes to check. Any shape which exceeds the maximum extent in X or Y will be skipped.

Tip

 The [critical_dimension](#) setup command can be used in place of this required argument. The *value* used for **max_extent** is $3 * \text{critical_dimension_value}$.

- **output_type [reference | simulation] {extents | centers size_microns}**

An optional argument specifying the output type. By default, shapes on the simulation layer are output for this command. If a reference layer is supplied, specifying “output_type reference” outputs the relevant shapes on the reference layer instead.

The output shape can be replaced by specifying the following modifiers:

- extents — Uses minimum bounding boxes around the shape.
- centers *size_microns* — Uses squares with sides of *size_microns* in the centers of the bounding boxes (the specified size value must not exceed **max_extent**)

- **exception [also [property_value *prop_value*]]**

An optional keyword that outputs exceptions.

When a *reference_layer* is specified, the area is computed and checked only for shapes on the *simulation_layer* that interact with shapes on the *reference_layer*. Non-interacting polygons on the *reference_layer* and *simulation_layer* are treated differently depending on which **output_type** parameter is specified:

- **output_type simulation** — All shapes on the *simulation_layer* are processed, but those that do not interact with *reference_layer* are considered exceptions. Extra shapes on *reference_layer* not interacting with *simulation_layer* are ignored.

- output_type reference — All shapes on the *reference_layer* are processed, but those that do not interact with the *simulation_layer* are considered exceptions. Extra shapes on the *simulation_layer* not interacting with the *reference_layer* are ignored.

In this mode, constraint and property specifications are not allowed.

If “exception also” is specified, a constraint is required and the property specification is allowed. In this mode, area_compute outputs exceptions in addition to regular results. If property_value is specified, exceptions get a property value equal to *prop_value*. The default *prop_value* is -0.0001.

- property {
 area
 area_ref
}

An optional argument that specifies a property operation for the returned output of this command. The area of the simulation layer (using the area argument) and the area of the target shape (using the area_ref argument) can be computed. The braces ({{}}) around the property keyword(s) are required syntax.

When a *reference_layer* is also specified and the interaction between shapes on the simulation layer and reference layer is not one-to-one, the corresponding area property of the output polygon is the sum of areas of interacting polygons on another layer, depending on the output_type argument:

- output_type reference — If the output reference polygon interacts with multiple simulation polygons, its area property is the total area of interacting simulation polygons.
- output_type simulation — If the output simulation polygon interacts with multiple reference polygons, its area_ref property is the total area of interacting reference polygons.

Note

 Using the property option may cause a noticeable runtime impact, especially when there are large amounts of result markers generated through the use of this function. The SVRF DRC CHECK MAP rule file command will not retrieve properties from a layer output by this command; you must add an additional DFM RDB check as detailed in the section “[Creating the SVRF Rule File](#)”.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

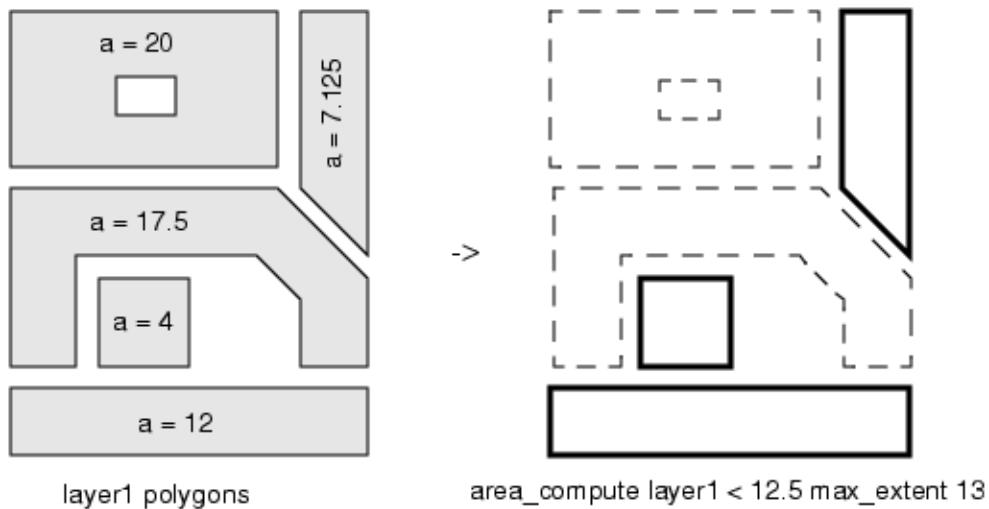
Examples

Example 1

The area_compute operation shown in [Figure 4-11](#) outputs all layer1 polygons with an area less than 12.5 user units squared, after throwing out any shapes greater in X or Y than 13 user units.

```
setlayer over_area = area_compute layer1 < 12.5 max_extent 13
```

Figure 4-11. area_compute operation



Example 2

The area_compute operation shown following limits the output on layer a14 to the 50 error shapes (in a hierarchical sense) found by area_compute that have the largest areas.

```
setlayer a14 = area_compute i1 > 0.25 max_extent 0.75 property
{ area
} limit { score largest worst 50 }
```

If the last error bin has more than 50 error shapes, all the shapes in the bin are included.

area_overlay

Verification control

Performs a contact alignment check, reporting possible shift issues.

Usage

```
area_overlay layer_contact_simulation
  [[not] {inside | outside} filter_layer]
  [drawn layer_contact_drawn]
  layer_poly_contour
  [{min | max}] [not] {constraint | annotated_constraint} [{ratio | absolute}]
  shift shift_microns [circular]
  max_extent max_extent_microns
  [output_type worst]
  # error-centric section
  [property {'min
  max
  min_absolute
  max_absolute
  '} [classify, limit, or histogram block]]
```

Description

area_overlay performs a contact (or via) alignment check where the user specifies the misalignment amount and coverage constraint. Eight different misalignments (shifts) and the original position are checked. If the minimum (or maximum) of all the shift coverage values meets the constraint, either the original contact (default output) or the shifted contact(s) with the minimum (or maximum) coverage (if the output_type worst argument is specified) is returned.

Concurrency Support

area_overlay operations that have the same input layers, shift [circular], and max_extent arguments will be run concurrently.

Arguments

- ***layer_contact_simulation***
A required argument that specifies the layer containing the contacts that are to be checked for coverage after being alignment-shifted.
- [not] {inside | outside} *filter_layer*
An optional argument specifying a filter layer. Only output polygons that also satisfy the filter (are completely “inside” or “outside” *filter_layer*, with the optional “not” inverting the selection) are output.

- drawn *layer_contact_drawn*

An optional keyword specifying the drawn contact layer. This is required only for annotated constraints.

- *layer_poly_contour*

A required argument that specifies a generated contour layer that must at least partially cover the contacts.

- [min | max] [not] {**constraint** | **annotated_constraint**} [ratio | absolute]

Note



If specified, min or max must appear immediately before the constraint.

If specified, ratio or absolute must appear immediately after the constraint.

A required argument that specifies a constraint (in the range of 0.0 to 1.0) applied as either:

- the percentage of the contact's area that is covered by the poly divided by the entire area of the contact (ratio mode, which is the default)
- the percentage of the contact's area in microns squared that is covered by the poly (absolute mode)

If the contact's minimum (or maximum if max is specified) percent coverage meets this constraint (or fails to meet it if the 'not' qualifier is also specified), either the unshifted contact or the contact with the minimum/maximum coverage is returned.

Using an **annotated_constraint** also creates the additional restriction that the contact under consideration must also be overlapped by a contact in the *layer_contact_drawn* layer to meet the annotated constraint. If there is no overlapping contact, or there are multiple possible associations for the contact, the default constraint case is used.

The default behavior for this command is to check for the minimum percentage.

- **shift** *shift_microns* [circular]

A required argument that specifies the amount of translation shift to be applied to the contacts in microns. The translations applied are:

- default mode: (0 0), (-S,-S), (-S,+S), (+S,-S), (+S,+S), (0, -S), (0, +S), (-S, 0), and (+S, 0)
- circular mode: (0, 0), (-R,-R), (-R, +R), (+R, -R), (+R, +R), (0, -S), (0, +S), (-S,0), and (+S, 0)

where S is *shift_microns* and R = *shift_microns*/square_root(2)

- **max_extent** *max_extent_microns*

A required argument. Only contacts (polygons on the *layer_contact_simulation* layer) whose maximum dimensions are less than or equal to **max_extent_microns** are checked for coverage. Contacts larger than this value are skipped.

Siemens EDA recommends that you set this value larger than the biggest contact you want to check for overlay problems. However, avoid setting max_extent to overly large values, as this may impact overall performance.

Tip

The [critical_dimension](#) setup command can be used in place of this required argument. The *value* used for **max_extent** is $3 * \text{critical_dimension_value}$.

- output_type worst

An optional argument that returns the translated contact with the worst (minimum or maximum) value. If multiple translated contacts have the same worst value, then all the translated worst case valued contacts are merged together and returned.

- property {min | max | min_absolute | max_absolute}

An optional argument that attaches the percentage of minimum (min) ratio coverage, maximum (max) ratio coverage, minimum area in microns squared (min_absolute), or maximum area in microns squared (max_absolute) among all nine positions to each output polygon. The braces ({{}}) around the property keyword(s) are required syntax. Multiple keywords can be specified, one per line.

- *classify, limit, or histogram block*

An optional argument that specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

Notes

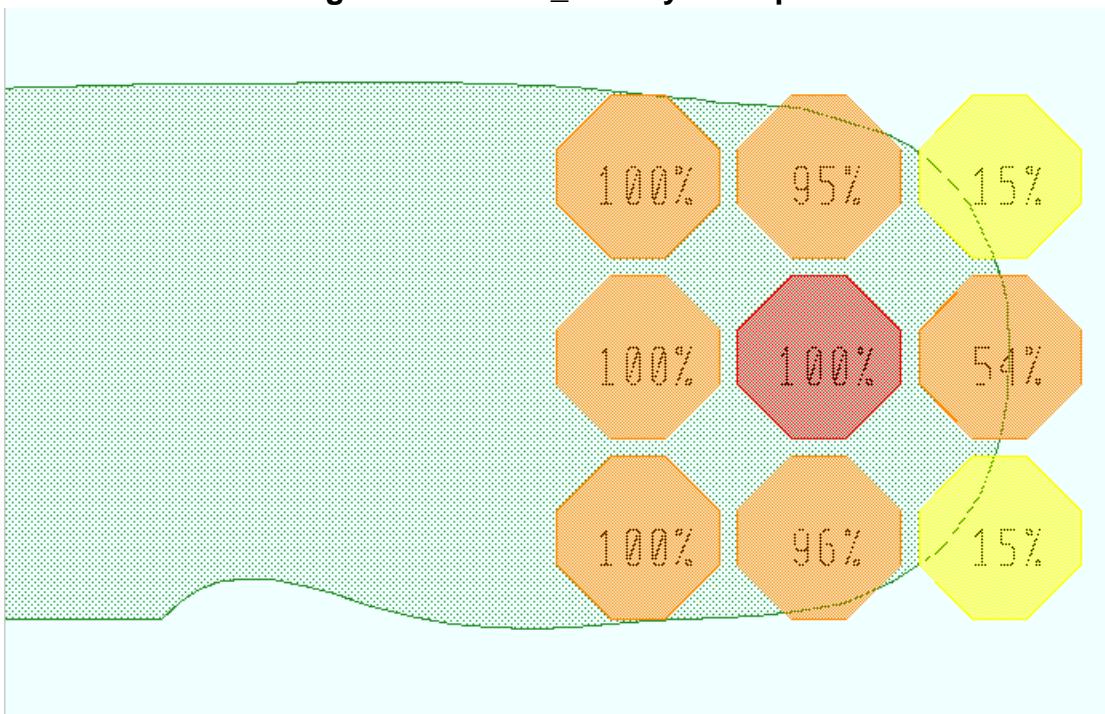
The **max_extent** and **shift** arguments set a lower bound for the interaction distance used in hierarchical processing. If the interaction distance set by these arguments is large, it can increase the runtime and memory usage.

Examples

Example 1 — Contact Alignment Check

```
setlayer out = area_overlay contact_simulation poly_contour < 0.6 \
shift .02 max_extent 0.15 output_type worst property {min}
```

Figure 4-12. area_overlay Example



- Green - input poly_contour layer
- Red - input contact_simulation layer
- Orange/Yellow - shifted contacts
- Yellow - is the output result of the check. Two shifted contacts are returned, since they have the same minimum value.
- Value of property min = 0.15

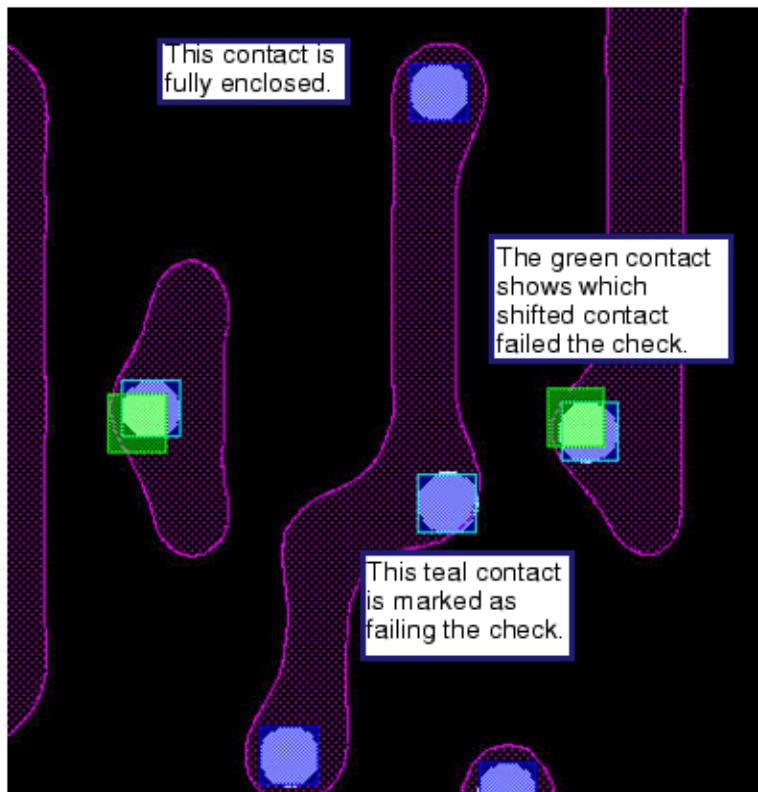
Example 2 — Max Extent and Shifted Contacts

Given the following **area_overlay** commands:

```
setlayer bad_ct_a1 = area_overlay m1_ct img_m2_ctr < 0.90 shift 0.20 \
max_extent 0.082 property {min}
setlayer bad_ct_a2 = area_overlay m1_ct img_m2_ctr < 0.90 shift 0.20 \
max_extent 0.081 output_type worst property {min}
setlayer bad_ct_a3 = area_overlay m1_ct img_m2_ctr < 0.90 shift 0.20 \
max_extent 0.080 output_type worst property {min}
```

Calibre OPCVerify outputs the following results:

Figure 4-13. area_overlay With Shifted Contacts



- **Purple contours** — Input metal contour layer
- **White ovals** — Input contact contour layer
- **Dark blue squares** — Original drawn contacts
- **Teal square** — Return value from bad_ct_a1, showing the contact that failed the check
- **Green squares** — Return value from bad_ct_a2, showing the shifted contact that failed the check (output_type worst argument supplied)

Note that bad_ct_a3 does not return any values, because all of the contacts are larger than 0.080 microns (max_extent value was less than all contacts)

area_ratio

Verification control

Checks the area ratio of a shape versus a constraint.

Usage

```
area_ratio sim_layer ref_layer
[[not] {inside | outside} filter_layer]
[[not] constraint] max_extent value
[output_type [reference | simulation] [extents | centers size_microns]]
[exception [also [property_value]]]
# error-centric section
[property {'area_ratio'} [classify, limit, or histogram block]]
```

Description

The `area_ratio` command checks for each filtered shape on `sim_layer` and `ref_layer` with an extent in X and Y that are both smaller than the maximum extent value. For those shapes on the `sim_layer` that interact one-to-one with polygons on the `ref_layer`, the command finds the area ratio (`sim_layer` area / `ref_layer` area) and compares it against the specified `constraint`. When the `constraint` is satisfied, the corresponding shape from either `sim_layer` or `ref_layer` is copied to the output, depending on the output parameters.

Concurrency Support

Concurrency is supported for multiple `area_ratio` calls that differ only in the `constraint` argument.

Notes

This function (and the `area_compute` function) are good for checking area coverage, and are designed primarily to be used for contact layers.

Interacting shapes on the `sim_layer` and `ref_layer` that do not interact one-to-one with each other are considered exceptions. Non-interacting polygons on the `ref_layer` and `sim_layer` are treated differently depending on the `output_type` parameter specified:

- `output_type simulation` — Shapes on the `sim_layer` that do not interact with the `ref_layer` are considered exceptions. Extra shapes on the `ref_layer` not interacting with the `sim_layer` are ignored.
- `output_type reference` — Shapes on the `ref_layer` that do not interact with the `sim_layer` are considered exceptions. Extra shapes on the `sim_layer` not interacting with the `ref_layer` are ignored.

To find instances where zero or multiple shapes intersect, use the [interact](#) command. For example, given two contours M1 and C1, you would use the following code:

```
setlayer covered = and M1 C1
setlayer zerotouch = interact C1 covered == 0 max_extent 0.1
setlayer multitouch = interact C1 covered > 0 max_extent 0.1
```

An area_ratio command to detect poor coverage along with this example code would be as follows:

```
setlayer poor_coverage = area_ratio covered C1 < 0.95 max_extent 0.1
```

Arguments

- ***sim_layer***

A required argument, specifying the simulation layer to check.

- ***ref_layer***

A required argument, specifying the reference (target) layer.

- [not] {inside | outside} *filter_layer*

An optional argument, specifying a filter layer. The filter is applied against either the reference layer or the simulation layer (default), depending on the setting used for the output_type parameter.

Only output polygons that also satisfy the filter (are completely “inside” or “outside” *filter_layer*, with the optional “not” inverting the selection) are output.

- [not] *constraint*

A conditional argument defining the constraint on the ratio of area (area1 from *sim_layer*, and area2 from *ref_layer*) of appropriate shapes, which is described mathematically as area1/area2.

To specify a constraint as a ratio, the second value is used as a ratio of the first, using 1.0 as ‘equal’. For example, the constraint “< 1.5” is read as “50 percent greater than.”

This argument interacts with other arguments:

- It is not allowed if “exception” without “also” is specified.
- It is optional if a histogram block is specified.
- In all other cases, it is a required argument.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- **max_extent value**

A required argument defining a filter for maximum extent, in X and Y, of the input shapes to check in percent mode. This acts as a filter, because any shape which exceeds the maximum extent in X or Y will be skipped.

Tip

The `critical_dimension` setup command can be used in place of this required argument. The *value* used for **max_extent** is $3 * \text{critical_dimension_value}$.

- **output_type {reference | simulation} [extents | centers size_microns]**

An optional argument that controls the output.

Note

 When this option is used, specifying either reference or simulation is required; extents and centers are always optional.

When a one-to-one pair is found of reference to simulation shapes and the constraint is satisfied, specifying this argument outputs a copy of the shape from the `ref_layer` or simulation layer. The default is reference, which takes less memory and outputs the shapes.

The output shape can be replaced by specifying the following modifiers:

- extents — Uses minimum bounding boxes around the shape.
- centers *size_microns* — Uses squares with sides of *size_microns* in the centers of the bounding boxes (the specified size value must not exceed **max_extent**)

- **exception [also [property_value]]**

An optional argument that outputs exception markers depending on which arguments are specified:

- When only exception is specified, only exceptions are output. In this mode, **constraint** and property specifications are not allowed.
- When exception also is specified, **constraint** is required and property specification is allowed. In this mode, `area_ratio` outputs exceptions along with regular results. If a property block is specified, exceptions are assigned a property value equal to *property_value*. The default prop_value is -0.0001. This is useful in limiting and histogram generation.

- **property {area_ratio}**

An optional argument that specifies a property operation for the returned output of this command. Only the `area_ratio` (`sim_layer area / ref_layer area`) can be computed for this command. The braces ({})) around the keyword(s) are required syntax.

Note

 Using the property option can cause a noticeable runtime impact, especially when there are large amounts of result markers generated through the use of this function. The SVRF DRC CHECK MAP rule file command will not retrieve properties from a layer output by this command; you must add an additional DFM RDB check as detailed in the section “[Creating the SVRF Rule File](#)”.

- *classify, limit, or histogram block*

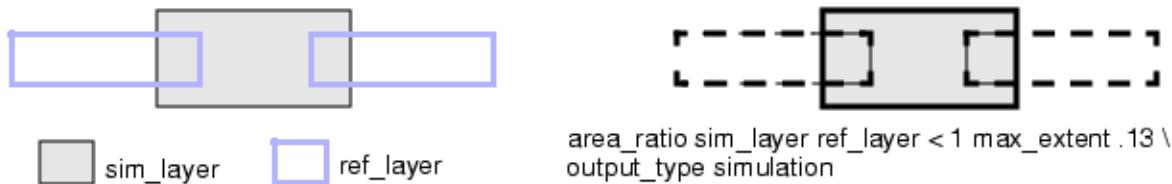
An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

Examples

The following command discards any potential shapes with a side larger than .13 um before calculating anything. It then outputs the layer1 shapes that have a coverage in X and Y of less than with a ratio of less than 1.

```
setlayer a1 = area_ratio layer1 layer2 < 1 max_extent .13 \
    output_type simulation
```

Figure 4-14. area_ratio Operation



asraf_print_check

Verification control

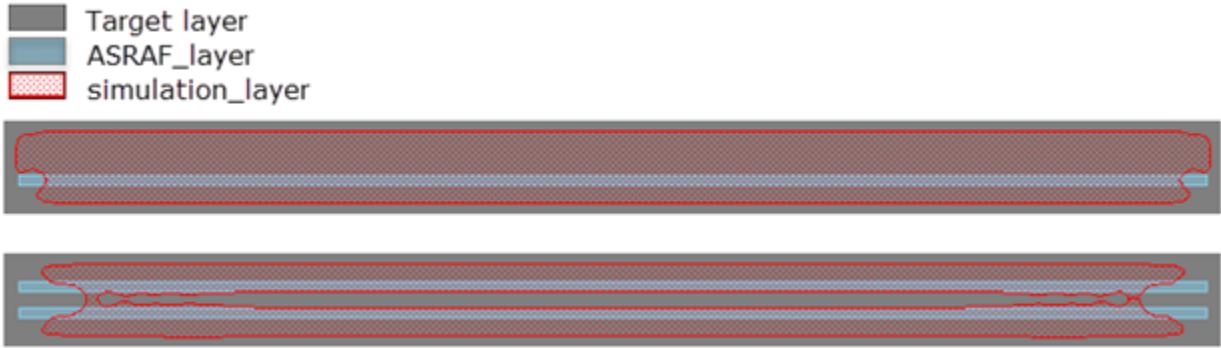
Detects under-printing around ASRAF features.

Usage

```
asraf_print_check ASRAF_layer simulation_layer
  [[not] {inside | outside} filter_layer]
  max_asraf_space max_asraf_space_microns
  [area area_constraint]
  [max_width max_width_constraint]
  [max_extent extent_microns]
  [output_type {extents | as_is}]
  # error-centric section
  [property {'{
    max_width
    area
  '}' [classify, limit, or histogram block]]]
```

Description

The asraf_print_check command checks for non-printing effects caused by negative SRAF shapes (also referred to as ASRAFs) affecting the image contours. Two input layers, an ASRAF layer and an image contour (simulation_layer), are shown in the following figure.



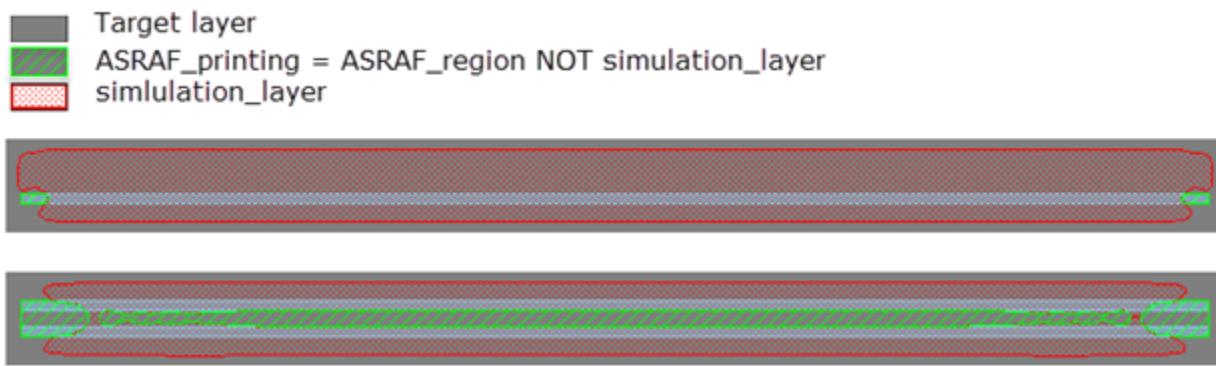
The asraf_print_check command first constructs a temporary layer “ASRAF_region” by applying the operations

```
SIZE ASRAF_layer BY max_asraf_space_microns/2 OVERUNDER
```

which merges all ASRAF shapes separated by less than *max_asraf_space_microns* into single features, as illustrated in the following figure.

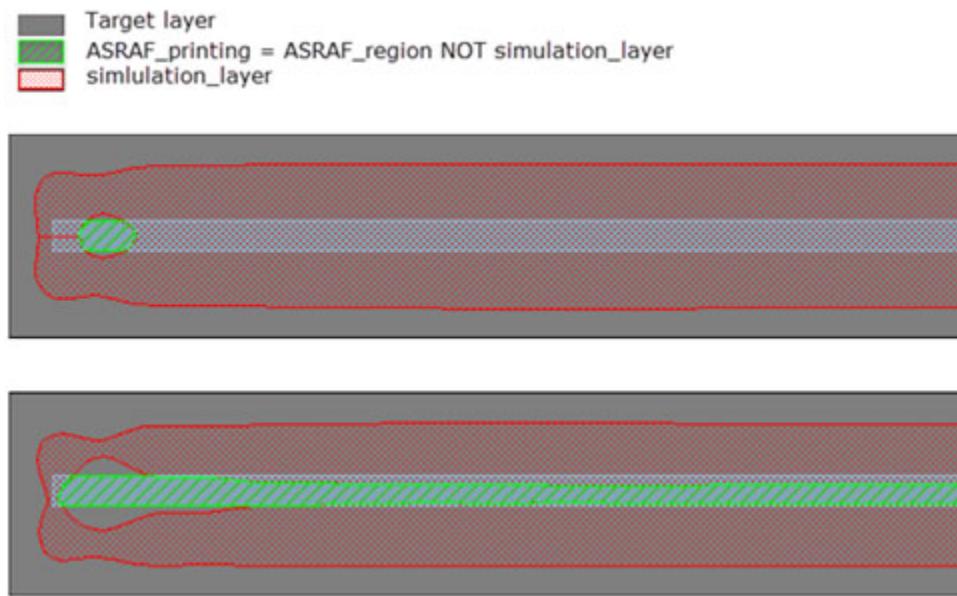


Calibre OPCVerify then detects ASRAF printing by selecting all parts of "ASRAF_region" not covered by "simulation_layer" (by computing ***ASRAF_region NOT simulation_layer***), as shown on the following figure.



Please note that Calibre OPCVerify computes the properties of `asraf_print_check` output shapes, not of the real ASRAF-induced errors in the image. This can lead to Calibre OPCVerify

underreporting both area and maximum width when an ASRAF-induced hole is wider than the corresponding ASRAF itself, as shown in the following figure.



In this case the `max_width` property will be equal to the width of ASRAF, and the `area` property is derived by applying an AND operation to the hole and the ASRAF area. This result is satisfactory, since a hole that is as wide as an ASRAF shape is already a significant error.

Concurrency Support

Two `asraf_print_check` operations will run concurrently if their `ASRAF_layer`, `simulation_layer`, [not] {inside | outside} `filter_layer`, `max_asraf_space`, and `max_extent` argument options are the same.

Arguments

- **`ASRAF_layer`**
A required argument specifying a layer containing ASRAFs.
- **`simulation_layer`**
A required argument specifying a target contour layer to compare the ASRAFs to.
- **`max_asraf_space` `max_asraf_space_microns`**
A required argument specifying the maximum distance between two ASRAFs that should be considered a single feature. The distance is given in microns. See the Description section for how this value is used.
- [not] {inside | outside} `filter_layer`
An optional argument specifying a filter layer. The filter is applied to the simulation layer. Only ASRAFs that satisfy the filter constraint (are completely “inside” or “outside” `filter_layer`, with the optional “not” inverting the selection) are checked.

- **area *area_constraint***

An optional argument specifying an area constraint for the ASRAF printing (error) regions. Any error shape not meeting the constraint is discarded. For example: “area > *value*”, where *value* is an area in square microns.

- **max_width *max_width_constraint***

An optional argument specifying a constraint for the maximum width of ASRAF printing. Any ASRAF printing shapes that do not satisfy the width constraint are discarded.

The maximum width is defined as the smaller of the largest vertical and horizontal cross sections of the ASRAF printing shapes.

If the max_width property calculation is requested in the property block for this command, the max_width argument is required, and the *max_width_constraint* must include an upper bound. In other words, a constraint such as “< *x*” is allowed, but “> *x*” is not allowed because there is no upper bound.

- **max_extent *extent_microns***

Note

 max_extent is an optional argument, but is required when an area constraint or property block are specified.

An optional argument that is used by the area and max_width properties output. Given an ASRAF non-printing object with a bounding box of length X and height Y, Calibre OPCVerify considers max (X,Y) the long dimension of the error and min (X,Y) its short dimension. It then uses the max_extent argument as follows:

- When the area property is requested in the property block, the area is calculated only for errors with a long dimension not larger than max_extent. Areas larger than max_extent are set to 100.
- When the max_width property is requested in the property block, the max_width property is returned only for errors in the short dimension not larger than max_extent. Widths greater than max_extent are set to 100.
- When an area constraint is present, errors with a long dimension larger than max_extent are discarded.
- If a max_width constraint is specified without an area constraint, asraf_print_check keeps all ASRAF non-printing objects satisfying the max_width constraint, even if their extent is larger than the *extent_microns* value.
- When an error with a long dimension larger than max_extent crosses a tile boundary:
 - If the output_type argument is set to extents, the markers in different tiles may be slightly misaligned. This is expected behavior.

- When specifying a max_width constraint, an error can get split at the tile boundary, in cases where the scanning from one tile does not see the part of the error which satisfies the constraint.

extent_microns is added to the interaction distance for the operation, so large *extent_microns* values can slow down Calibre OPCVerify performance.

- **output_type {extents | as_is}**

An optional argument that sets the output for the command. Specifying extents (the default) prints a bounding box for found errors. Specifying as_is outputs the error shapes.

- **[property '{' max_width area '}]**

An optional argument that specifies a property operation for the returned output of this command. The max_width and area for ASRAF non-printing shapes can be computed for this command, and are modified by the max_extent argument as described previously. The braces ({{}) around the property keyword(s) are required syntax, and the initial brace is required on the same line after the property keyword. Additionally, each property must be on its own line.

- ***classify, limit, or histogram block***

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

bandcheck

Verification control

Checks for tolerance band violations.

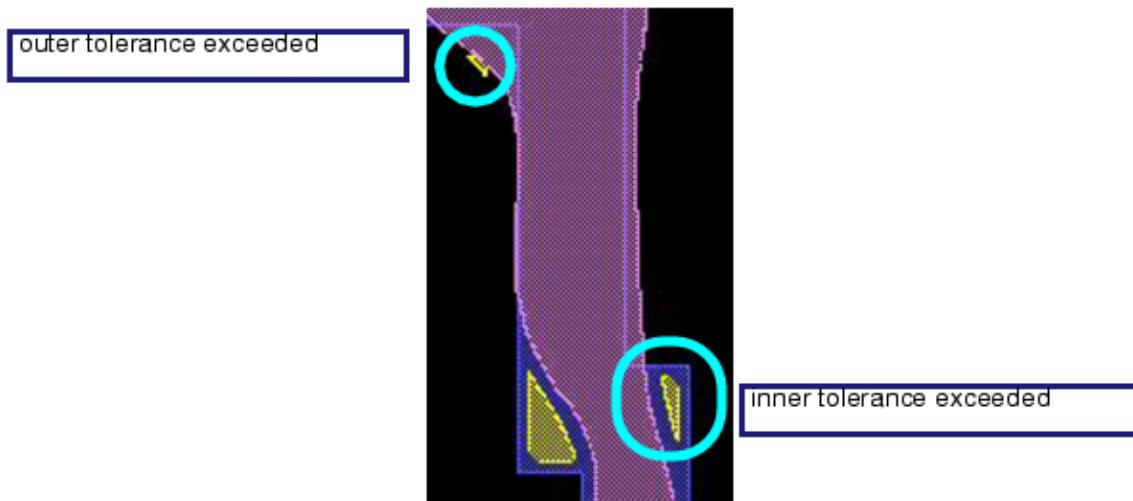
Usage

bandcheck *contour inner outer*

Description

Checks for inner and outer tolerance band violations. It compares the given contour layer to the specified inner and outer tolerance zones and returns locations where the CD is violated.

Figure 4-15. bandcheck Example



Arguments

- ***contour***

A required contour layer to check. You can generate a contour layer using the [image](#) command.

- ***inner***

A required context layer containing the inner tolerance zone. You can generate the inner tolerance zone using a [build_tolerance](#) layer operation.

- ***outer***

A required context layer containing the outer tolerance zone. You can generate the outer tolerance zone using a [build_tolerance](#) layer operation.

Examples

The following example code sets up the build tolerance zones (inner first, then outer):

```
setlayer contour = image optical f0 aerial .07
setlayer it1 = build_tolerance inner TARGET max .01 stepsize .005
setlayer itc = cornerchop .01 .01
setlayer ot1 = build_tolerance outer TARGET max .03 stepsize .01
setlayer otc = cornerchop ot1 .01 .01
```

The following command then generates the bandcheck:

```
setlayer build_out = bandcheck contour itc otc
```

bridge

Verification control

Checks for bridging conditions.

Usage

```
bridge target_layer [[not]{inside | outside} filter_layer [filter_contour_also]]  
{contour_layer | {  
{overlay | cutmask | keepmask} max_error contour_layerA contour_layerB} }  
[{|simple_constraint| |double_constraint| |annotated_constraint|}  
separation sep_microns  
[runlength {> | >= } length] [max_runlength rl_max [exception rl_ex]]  
[projecting [projection_length]]  
[max_tolerance tol_microns]  
[max_edge edgelen_microns]  
[output_expand width_microns [width_microns2]]  
[pinpoint [markers | gauges] [size]]  
[slivers {off | size [ignore] | {blots size1 [ignore] holes size2 [ignore]} [manhattan | skew} }]  
[small_extra_printing max_extent [extents [centers size] | suppress | manhattan]]  
[cd_search_dist cdist]  
# error-centric section  
[property {'min  
[target_cd]  
[shift_dir]  
[runlength]  
'} [classify, limit, or histogram block]  
[add_properties block]  
[pinpoint_output block]]
```

Description

Checks for topological errors in printed contours that occur when two contour edges bridge (touch where they should not; this is referred to as hard bridging). This command can also check for near-bridging conditions (soft bridging).

The bridge check always captures [extra_printing](#) errors. This command is the complement to the [pinch](#) command.

Arguments

- *target_layer*

A required target layer to check.

- [not] {inside | outside} *filter_layer* [filter_contour_also]

An optional argument that outputs results that are inside or outside the specified filtering layer. Specifying the “not” qualifier reverses the filter. “Extra printing” type errors are output only if all edges satisfy the filter condition.

Adding the filter_contour_also qualifier causes the command to exclude contour edges that are not assigned to any filtered target edges.

Note

 Exercise caution when using filter layers with the bridge/pinch commands. The performance of Calibre OPCverify may vary depending on the nature of the filter: Filters consisting of relatively large polygons over small parts of the design can improve performance over filterless bridge and pinch commands. Filters consisting of many small polygons over large parts of the design can slow down performance and are not recommended over filterless bridge and pinch commands. The best approach in this situation is to use bridge/pinch without filters, and then apply the Calibre OPCverify Boolean [and](#) operation to the results with your intended filter layer.

- *contour_layer*

A required printed contour layer to check. You can generate a contour layer using the [image](#) command.

- **overlay | cutmask | keepmask** *max_error* *contour_layerA* *contour_layerB*

An alternative to *contour_layer* (single contour) mode, these checks are designed to enhance the basic bridging check to include a check between two contour layers (*contour_layerA* and *contour_layerB*) for use with double patterning designs.

- The **overlay** argument adds an additional check for bridging conditions in a contour defined as the union (A OR B) of the two contour layers. **overlay** is typically used for LELE designs.
- The **cutmask** argument adds an additional check for bridging conditions in a contour defined as the difference (A NOT B) between the two contour layers. **cutmask** is typically used for cutmask designs.
- The **keepmask** argument adds an additional check for bridging conditions in a contour defined as the intersection (A AND B) between the two contour layers. **keepmask** is used in keepmask designs.

The **overlay**, **cutmask**, and **keepmask** arguments require a *max_error* distance in microns, which is used as the maximum expected shift of an individual mask (and hence of an individual contour) from its nominal position, assuming that each mask can randomly shift by *max_error* / 2 in any direction. In effect, in addition to checking for bridge conditions on the nominal contour (*contour_layerA*), this command checks a second, shift-sensitive contour created for all possible unions (**overlay**), differences (**cutmask**), or intersections (**keepmask**) which is produced by shifting the masks from their nominal positions.

Tip

 Error markers resulting from an overlay, cutmask, or keepmask bridge command may be the result of either the normal bridge check, or the overlay, cutmask, or keepmask bridge check.

- $\{simple_constraint \mid double_constraint \mid annotated_constraint\}$ separation *sep_microns* [runlength {> | \geq } *length*] [max_runlength *rl_max* [exception *rl_ex*]]

An optional argument that sets a constraint condition for the bridging check.

Either a simple constraint, double constraint, or annotated constraint can be specified as the bridging constraint.

- *simple_constraint*

Takes the form:

$<[=] distance_microns$

Checks for soft bridging conditions, defined as a contour that is less than (or equal to if \leq is specified) *distance_microns* from the nearest contour. *distance_microns* must be a positive value.

Note

 If a constraint is not specified, the bridge command only detects hard bridging. Using $<=0$ for *simple_constraint* will cause an error.

- *double_constraint*

Takes the form:

$>[=] lower_bound <[=] upper_bound$

A double-sided constraint is used to separate out soft bridging and soft pinching errors into bins. However, because this method can return false positives around space ends, a *double_constraint* must be accompanied by a *filter_layer*. The filter should exclude space ends according to the separation value.

- *annotated_constraint*

Note

 *annotated_constraint* arguments require the *target_layer* input argument to be the output from an [annotate](#) command.

Takes the form:

default $<[=] distance_microns$
annotation_type1 { $<[=] distance_microns2 \mid ignore\}$
[*annotation_type2* { $<[=] distance_microns3 \mid ignore\}]$

When *annotation_type* arguments are supplied, they use that constraint for qualifying shapes that have the matching annotation type(s). The annotate command “double_via” and “orientation” types are supported, but only one type is allowed per command.

Only annotation types supported by the specific annotation command that created the layer can be detected (you cannot check for both double_via and orientation on the same layer). In other words, “annotate ... double_via” can have at most two constraints (default and double_via), and “annotate ... orientation” can have at most three (default, horiz_edge, and vert_edge).

The default constraint is used for any shapes that do not have a specified *annotation_type*, and is the equivalent of a *simple_constraint*.

If an *annotation_type* constraint is set to ignore, shapes that contain an instance of that *annotation_type* are skipped.

The *sep_microns* argument is used as a minimum separation distance between measurement points to avoid detecting part of the same contour.

Tip

 The [critical_dimension](#) setup command can be used in place of the separation argument. The value used for *sep_microns* is $3 * \text{critical_dimension_value}$.

If runlength is also specified, the command ignores soft bridges that have a run length (the length along the contour border that violates the error constraint) smaller than the specified *length*. In other words, only errors that have a run length that is greater than (or equal to if \geq is specified) the specified *length* are reported.

runlength cannot be used with the [overlay](#) or [cutmask](#) arguments.

If the *max_runlength* argument is specified, errors with a run length larger than *rl_max* are considered exceptions, and *rl_ex* is used as the output value. In order to report consistent results on tile boundaries, *rl_max* is added to the interaction distance, so using too large a value for *rl_max* can degrade performance.

max_runlength is only used if the runlength property is requested in the property block. The default value for *rl_ex* is 100 microns.

- *projecting [projection_length]*

An optional argument that limits the command to check only those contour edges with associated target edges that are parallel and project on each other. The target edges must project for a minimum *projection_length* in microns (default is [1.0 dbu](#)) for the command to check the distance between the contour edges. Otherwise, the contour edges are not considered in the check.

- *max_tolerance tol_microns*

An optional argument specifying the maximum search range that this command will attempt to find a matching contour in. If there are no contours within *tol_microns* of a target edge,

the edge is marked as an error. Siemens EDA recommends setting the `max_tolerance` argument to be equal (or nearly equal) to the minimum feature size (default is 0.15 microns):

Tip

 The `critical_dimension` setup command can be used in place of this argument. The `tol_microns` value used for `max_tolerance` is $1 * critical_dimension_value$.

- Setting `max_tolerance` to a value smaller than the minimum feature size will start generating false errors; setting it to less than half of the CD is definitely too small.
 - Setting `max_tolerance` to a value greater than the minimum feature size will increase the CPU computation time; setting `max_tolerance` larger than a feature extent is not recommended.
 - The output shape parameters `max_edge` and `output_expand` take their default values from `max_tolerance`, and may give unusable results if you set `max_tolerance` too large.
- `max_edge edgelen_microns`
An optional argument that specifies the maximum length of a contour edge or target edge before it is subdivided. Smaller values of `max_edge` give a finer granularity at the cost of run time. Default is the value set for `max_tolerance`.
 - `output_expand width_microns [width_microns2]`
An optional argument that specifies the maximum width an error marker can expand from the marked error contour. If only one `width_microns` value is specified, it sets how far the marker expands outside the target. If `width_microns2` is also specified, it sets how far the marker expands inside the target.
Values far smaller than the CD for `width_microns` mark the error more precisely; larger values (slightly larger than the CD) produce fewer, larger markers that are easier to see. Default is the value set for `max_tolerance`.
For contact or via layers only, set the additional `width_microns2` argument to 0 to avoid merging together markers at the corner of contacts. Otherwise, leave the value at the default (1dbu).
 - `pinpoint [markers | gauges] [size]`
An optional argument that replaces the normal output with markers showing the exact location of the minimum distance measured for the operation.

Note

 This option has been superseded by the `pinpoint_output block`. It is currently retained for backwards compatibility. It is not the same as the `pinpoint_output block`. If you use the `pinpoint` argument, it must appear before the error-centric section starts or it causes an error.

- slivers {off | size [ignore] | {blots size1 [ignore] holes size2 [ignore]} [manhattan | skew]}

An optional argument that filters out all narrow contour polygons and holes before testing for bridging. By default, sliver detection is off. The parameters determine how the slivers are handled.

- `size [ignore]` — Calibre OPCVerify builds a bounding box around every contour polygon and hole. If the smaller side of this bounding box is less than `size` microns, the polygon or hole is considered to be a sliver and is removed from the contour layer.

When the ignore option is specified, the slivers are discarded. Otherwise, sliver polygons are marked by bridge commands.

- `blots size1 [ignore] holes size2 [ignore]` — Different `size` criteria can be specified for blots (polygons) and holes.

In the default manhattan mode, the sliver bounding box is aligned with the X and Y axes.

When the mode is skew, Calibre OPCVerify builds and tests two bounding boxes; one axes-aligned and one rotated by 45°. It returns the smaller bounding box.

- `small_extra_printing max_extent [extents | {extents centers size} | suppress | manhattan]`

An optional argument that sets how to mark extra printing contour polygons with extents less than or equal to the `max_extent` value:

- `extents` — Output minimum bounding boxes of each polygon rather than polygons themselves.
 - `extents centers size` — Output marker squares in the centers of the bounding boxes.
 - `suppress` — Does not output anything for extra printing if the extent area is <= `extent`.
 - `manhattan` — Convert to 0 or 90 degree angles and output the contour polygons.

Default is the value set for max_tolerance.

- `cd_search_dist cdist`

An optional argument that specifies an optional maximum search distance in microns used when computing values for the “min” with “target_cd” property. If no nearby edges are detected within `cdist` of the contour, the target_cd value is set to 0.0.

Excessively large values for this argument will affect runtime performance. Default is `2*max_tolerance`.

Tip

The `critical_dimension` setup command can be used in place of this argument. The `cdist` value used for `cd_search_dist` is $2 * \text{critical_dimension_value}$.

- `property {'min [target_cd]}`

[shift_dir]

[runlength] ‘{’

An optional argument that specifies a property operation for the returned output of this command.

The minimum difference between contours can be computed by specifying the “min” property.

If the target_cd qualifier is added after the min keyword (it must also be on its own line), the target spacing is recorded in the target_cd property as the closest distance from the target edge to any other target edge, measured normal to the error edge. The exceptions are as follows:

- For errors marked on image contours, target_cd is reported as 0.0.
- For error edges which are not 45 degree skew lines, target_cd is reported as 0.0.

If the shift_dir qualifier is added after the min keyword (it must also be on its own line), shift_dir contains the direction in degrees that *contour_layer2* must be shifted to achieve the reported min property. The shift_dir qualifier can only be used when overlay or cutmask mode is active. The values range from -180 degrees to +180 degrees, starting at 0 degrees indicating a shift of *contour_layer2* rightward, +90 is a shift upward, and -90 is a shift downward. If a shift in multiple directions can result in the min value, an arbitrary shift_dir value is returned. If the min value is possible in all directions, the shift_dir property contains a value of 360.

The optional runlength property returns the length of the contour that violates the distance constraint. A hard bridge returns a runlength value of zero. Using runlength also requires the runlength argument to be used as part of the constraint argument.

If one error marker corresponds to several separate bridges, its runlength property is the maximum run length of the bridges inside the marker.

The braces ({}) around the property keyword are required syntax. Property operations require a distance constraint except in the case of hard bridge-only checks.

- *classify, limit, or histogram block*

Specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Note

 This command also has a pinpoint argument, only retained for backwards compatibility; it is a separate argument and is not the same as the pinpoint_output block.

Pinpoint output has special behavior for the bridge command. For a soft bridge, the min property is always a distance between two points, pt1 and pt2, located on edges of the contour layer.

- When the output mode is markers (the default), the marker is a tiny square, placed exactly midway between points pt1 and pt2.
- When the output mode is gauges, the marker is a rhombus, connecting pt1 and pt2. The gauges mode is recommended only for small layouts.

The size argument specifies the side of the squares (for markers) or the smaller diagonal or the rhombus (for gauges).

For a hard bridge, there is no obvious point to pinpoint. A square marker is placed in the center of the bounding box of the error shape in both modes. For L-shaped or n-shaped errors, the marker can be outside the error shape.

Examples

A hard bridging check:

```
setlayer short = bridge m1 img_m1_ctr max_tolerance 0.05 \
    output_expand 0.005 property {min}
```

A soft bridging check:

```
setlayer soft_bridge = bridge m1 img_m1_ctr < 0.060 \
    separation 0.20 max_tolerance 0.05 max_edge 0.050 \
    output_expand 0.005 property {min}
```

Tip

 Remember to check your max_tolerance value. The value specified for max_tolerance is highly dependent on your minimum CD values; the prior examples are specific to the internal design used to create the example and may not work as expected if your design CD is not 0.05 microns.

An annotated bridging check:

```
setlayer a1 = annotate contact1 double_via poly active max_space 0.07
setlayer double_60 = bridge a1 contour1 < 0.060 double_via < 0.055 \
    separation 0.060 max_tolerance 0.040 output_expand 0.005 property {min}
```

The example code above enhances a standard bridge check by using a different constraint for any shapes marked as double-vias on the input layer (a1), which is the output of the [annotate](#) command run on the contact layer (contact1).

An overlay (two contour) check between the contours ctrl1 and ctr2:

```
setlayer dp_b bridge tgt overlay 0.005 ctrl1 ctr2 <= 0.01 separation 0.08 \
    max_tolerance 0.030 property { min }
```

A double-sided constraint check is as follows:

```
setlayer se_fltr = filter_generate trgt expand 0.0015 space_end 0.02 0.012
setlayer p1 = bridge tgt outside se_fltr cntr <= 0.04 separation 0.08
setlayer p2 = bridge tgt outside se_fltr cntr >0.04 <= 0.042 \
    separation 0.08
```

In this example, p1 is a standard bridge check, and p2 is a double-sided constraint. It is important to understand that without the filter created with the [filter_generate](#) command, p2 returns a false positive result for the space end.

Related Topics

[pinch](#)
[extra_printing](#)
[not_printing](#)

bridge_tolerance

Verification control

Checks for outer tolerance band violations on a contour layer.

Tip

i The **bridge_tolerance** command is not the recommended method to find bridging errors. Use the **bridge** command instead. See the section “[Replace Older OPCVerify Bridge and Pinch Detection Methods With Bridge and Pinch](#)” on page 498 for related best practice information.

Usage

bridge_tolerance *contour tolerance_zone*

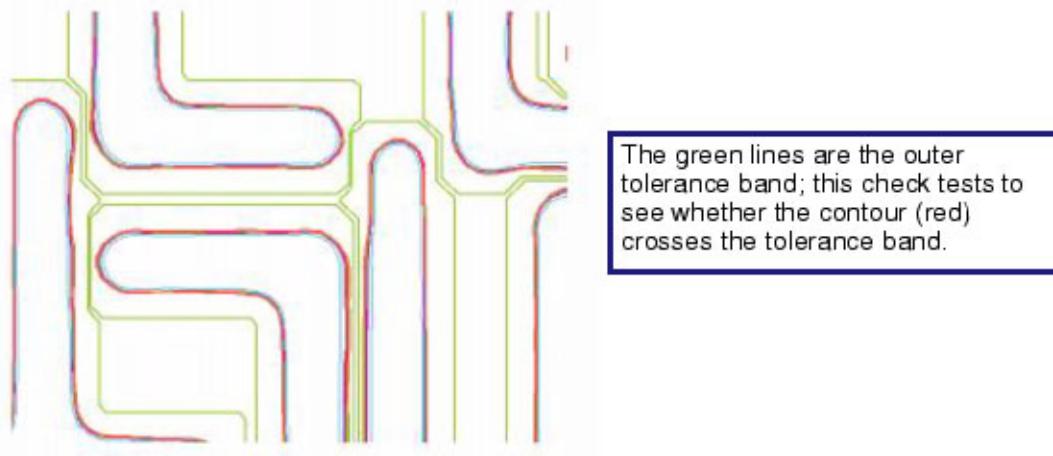
Description

Checks for outer tolerance band violations for the specified contour layer. This function can be performed as an embedded operation (using setlayer). It requires a precomputed outer tolerance zone to be supplied.

Note

 For designs that use visible assist features (such as SRAFs and dummy fill), it is important that your build tolerance zone include those features inside the build tolerance zone. Otherwise, the features will be detected as false bridging errors, even if they are located on a separate layer.

Figure 4-16. Bridge Tolerance Example



Arguments

- ***contour***

A required argument specifying a contour layer to check. You can generate a contour layer using the [image](#) command.

- ***tolerance_zone***

A required argument specifying a layer with a generated outer tolerance zone to check for bridging. You can generate the outer tolerance zone on an input layer using a [build_tolerance](#) layer operation.

Examples

Given the following layer definition code:

```
layer SHAPES visible dark
layer ORIG hidden darkoptical_model_load o1 mymodel.opt
setlayer tol = build_tolerance outer ORIG max .08 stepsize .01
setlayer to = cornerchop tol .02 .02
setlayer i1 = image optical o1 dose 0.96 aerial .3
```

The following operation creates a bridge tolerance result using the specified layers:

```
setlayer bridge1 = bridge_tolerance i1 to
```

build_tolerance

Verification control

Generates a tolerance zone (outer or inner) around polygons on a layer.

Note

 This command was originally intended for use in constructing bridging and pinching checks. You should use the [bridge](#), [pinch](#), [extra_printing](#), and [not_printing](#) checks instead for that purpose.

Usage

build_tolerance {outer | inner} target max max stepsize step

Description

Generates a tolerance zone (a scan area around the polygons in the area being verified). You supply the maximum amount to try to bias target shapes to generate the tolerance zone. Calibre OPCverify biases the tolerance zone in multiples of the stepsize argument.

- For outer tolerance, the target shape edge will be outward, biased by the maximum amount possible, while avoiding merging with other shapes.
- For inner tolerance, the target shape edges will be inward, biased by the maximum amount possible to avoid self-intersection.

Note that building inner tolerance bands for small features (such as contacts) may result in the collapse of the tolerance band for large (approaching 1/2 the contact feature size) values of **max**. If more aggressive inner tolerance bands are required, use the [interact](#) command to check if smaller features are printing (this is the equivalent of an aggressive inner tolerance band).

The runtime of this operation is proportional to the number of steps, which is roughly **max** divided by stepsize. A smaller stepsize translates to a longer runtime.

Note

 If you are building tolerance bands on 45-degree angle layouts, you may run into some difficulties in accurately representing the geometry. Siemens EDA advises that you use multiple smaller, alternating increments of **build_tolerance** and [cornerchop](#) until you achieve a satisfactory result. This command should not be used on contours.

Arguments

- **outer | inner**

A required argument selecting whether outer or inner tolerance is to be tested for.

- ***target***

A required layer name, containing the polygon shapes that Calibre OPCVerify modifies in order to generate the outer tolerance.

- ***max max***

A required maximum value in user units (default is microns) to bias out the target layer's shapes when generating the tolerance. The target will be biased by this amount only when it is far enough away from other edges on the target not to generate collisions.

- ***stepsize step***

A required distance in user units (default is microns). Calibre OPCVerify uses this as the stepsize for bias movement step attempts.

Examples

```
setlayer tol = build_tolerance outer ORIG max .080 stepsize .010
```

center_shift

Verification command

Measures displacement from the centers of contour polygons to a corresponding target.

Usage

```
center_shift contour_layer target_layer
  [[not] {inside | outside} filter_layer]
  [constraint]
  [max_extent max_extent_microns]
  [output_type type]
  [exception [also [property_value property_value]]]
  # error-centric section
  [property {'}
    [shift]
    [shiftX]
    [shiftY]
  {'} [classify, limit, or histogram block]]
```

Description

The center_shift command is intended to be used on contact or via layers. It measures the centers displacement of contour polygons from their target pairing in both absolute and X/Y projections. The target and contour polygons make a pair if they interact with each other and have one-to-one mapping. If a target or contour has no associated polygon from the contour (target) layer or have multiple associations, it is an exception.

Arguments

- ***contour_layer***
A required argument specifying a contour layer from a contact or via layer.
- ***target_layer***
A required argument specifying the drawn target layer.
- **[not] {inside | outside} *filter_layer***
An optional filter layer. If the *output_type* argument “target” is specified, the filter is applied to *target_layer*. By default, the filter is applied to *contour_layer*.
- ***max_extent max_extent***
An argument specifying that any target and contour polygons with extent larger than ***max_extent*** are ignored. This argument is required unless the [critical_dimension](#) command was specified, in which case the default value for *max_extent* is $3 * \text{critical_dimension_value}$.

- *constraint*

An optional argument specifying that output is generated only when the absolute value of the displacement satisfies this constraint.

- *output_type type*

An optional argument that selects the shape for error markers. The *type* argument can be one of the following:

- target — The entire target polygon is marked. This is the default.
- center *size* — A square of the specified *size* in microns is placed at the center of the contour.

- *exception* [also [*property_value value*]]

An optional argument that changes the output.

- When “exception” is specified alone, the center_shift command marks only target polygons that do not interact with exactly one contour polygon. In this mode, *constraint* and *property* arguments are not allowed.
- When “exception also” is specified, exception markers are output with normal error markers. All exception marker properties are set to *value*. The default *value* is -0.0001.

- *property {*

[shift]

[shiftX]\

[shiftY]

}

An optional argument that applies properties to the error markers depending on which property keywords are specified. The braces ({}) are part of the required syntax, and each property keyword must be on its own separate line.

- shift — Absolute contour displacement.
- shiftX — X projection of the contour displacement.
- shiftY — Y projection of the contour displacement.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

circularity_compute

Verification check

Computes the circularity of layer shapes as a method to check contact areas.

Usage

```
circularity_compute simulation_layer
[reference reference_layer]
[[not] {inside | outside} filter_layer]
[not] circularity_constraint
max_extent max_extent_microns
[output_type [reference | simulation] [extents |{centers size_microns}]]
[exception [also [property_value prop_value]]]
# error-centric section
[property '{'
    area
    area_ref
    circularity
'}'[classify, limit, or histogram block]]
```

Description

This command analyzes the input layer shapes and computes their circularity. It outputs those shapes which meet the constraint, after filtering away any shapes exceeding **max_extent_microns** in either the X or Y dimensions. The circularity of a 2D figure is defined as $4\pi \text{Area}/\text{Perimeter}^2$. Circularity is always less than or equal to 1; it is 1 for a perfect cycle and < 1 for anything else.

Arguments

- ***simulation_layer***
A required argument specifying the simulation layer to scan for shapes.
- **reference *reference_layer***
An optional argument specifying a reference (target) layer.
- **[not] {inside | outside} *filter_layer***
An optional argument that specifies that only polygons lying completely inside or completely outside *filter_layer* are selected from the ***simulation_layer***. Specifying “not” inverts the selection. If the **output_type** argument is “reference”, the filter is applied to the reference layer; if it is “simulation” then it is applied to the simulation layer.
- **[not] ***circularity_constraint*****
A required argument setting a constraint on the circularity of appropriate shapes from ***simulation_layer***, in dimensionless units. A circularity constraint must always be greater than 0 and less than or equal to 1.

This argument interacts with other arguments:

- It is not allowed if “exception” without “also” is specified.
- It is optional if a histogram block is specified.
- In all other cases, it is a required argument.
- **max_extent max_extent_microns**
A required argument defining a filter for the maximum extent of the input shapes to check. Any shape which exceeds the maximum extent in X or Y will be skipped.
- **output_type [reference | simulation] [extents | {centers size_microns}]**
An optional argument specifying the output type. When “reference” is specified, the polygons from the *reference_layer* are used for output; when “simulation” is specified, it uses polygons from the *simulation_layer*. (The default is simulation.)
Using “output_type reference” requires a reference layer to be specified.
 - The default is to output the polygons themselves.
 - If extents is specified, the output polygons are replaced with their minimum bounding boxes.
 - If centers *size_microns* is specified, the output polygons are replaced by squares with the side *size_microns* in the centers of the bounding boxes. When centers *size_microns* is used, *size_microns* must not exceed **max_extent_microns**.
- **exception [also [property_value prop_value]]**
An optional keyword that controls the output of exceptions. If “exception” is specified without the “also” keyword, only exception polygons are output. In this mode, constraint and property specification are not allowed.
If “exception also” is specified, a constraint is required and the property specification is allowed. In this mode, area_compute outputs exceptions alongside regular results.

When *reference_layer* is specified, circularity is computed and checked only for shapes on *simulation_layer* that interact with shapes on *reference_layer*. Non-interacting polygons on *reference_layer* and *simulation_layer* are treated differently depending on the *output_type* parameter specified:

- *output_type simulation* — All shapes on *simulation_layer* are processed, but those that do not interact with *reference_layer* are considered exceptions. Extra shapes on *reference_layer* not interacting with *simulation_layer* are ignored.
- *output_type reference* — All shapes on *reference_layer* are processed, but those that do not interact with *simulation_layer* are considered exceptions. Extra shapes on *simulation_layer* not interacting with *reference_layer* are ignored.

If *property_value* is specified, exceptions get a property value equal to *prop_value*. The default *prop_value* is -0.0001.

- property '{
 - area
 - area_ref
 - circularity}'

An optional argument that specifies properties to attach to the output layer. The opening brace ({}) is required on the same line after the property keyword. The following properties are supported:

- “area” reports the area of the polygon from *simulation_layer*.
- “area_ref” is the area of the polygon from *reference_layer*. This option requires the *reference_layer* argument to also be specified.
- “circularity” reports the circularity of the polygon from *simulation_layer*.

When *reference_layer* is also specified, and the interaction between shapes on the simulation layer and reference layer is not one-to-one, the corresponding area property of the output polygon is the sum of areas of interacting polygons on another layer, depending on the *output_type* argument:

- *output_type* reference — If the output reference polygon interacts with multiple simulation polygons, its area property is the total area of interacting simulation polygons. However, the circularity of the simulation polygon cannot be computed in this case, and the circularity property is assigned the exception value.
- *output_type* simulation — If the output simulation polygon interacts with multiple reference polygons, its area_ref property is the total area of interacting reference polygons.
- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

contour_diff

Verification check

Compares the difference between two contours.

Usage

contour_diff

```
constr
layer_target layer_im1 layer_im2 [[not] {inside | outside} layer_target_filter]
[exception_filter layer_exception_filter] [trim_ends trim_microns]
[expand expand_microns] max_search ms_microns
[contour_diff_spacing cds_microns]
[contour_layer_spacing {layer_im1_im2_spacing_microns}]
[jog_filter jog_length_microns jog_extend_microns]
[exception [also [property_value prop_value]]] |
# error-centric section
[property '{'
max
diff
'}' [classify, limit, or histogram block] [add_properties block] [pinpoint_output block]]
```

Description

This command compares the difference between two contours (defined as the Euclidean distance between them), and returns shapes that meet the constraint. Two contour_diff commands will run concurrently if they differ only in **constr**, exception, and property arguments.

Arguments

- **constr**

A required argument (unless exception is specified) that specifies the range of contour difference values for which output shapes will be generated.

Note

 The constraint condition is always checked against the absolute value of the contour difference, even when the signed difference (diff property) is requested in the properties block.

- **layer_target**

A required argument that specifies the drawn target shapes layer.

- **layer_im1 layer_im2**

A required argument that specifies two contours.

- [not] {inside | outside} *layer_target_filter*

An optional argument defining a filter layer. Only *layer_target* edges inside or outside *layer_target_filter* will be considered if the filter is specified. Otherwise, all *layer_target* edges will be checked.

- exception_filter *layer_exception_filter*

An optional argument specifying that any measurement location inside or touching the filter is marked as an exception.

- trim_ends *trim_microns*

An optional argument that trims output boxes around each gauge. The boxes are trimmed by *trim_microns* along the target edge if the output box is longer than $2 * \text{trim_microns} + 2\text{dbu}$ along the target edge. This implies that $2 * \text{trim_microns}$ must be smaller than *contour_diff_spacing* - 2dbu.

- expand *expand_microns*

An optional argument that specifies the halfwidth of the output markers; the marker is expanded from the edge in each direction. Defaults to 10 dbu when the trim_end argument is not also specified. Otherwise, it defaults to $\max(1\text{dbu}, \min(10\text{ dbu}, \text{trim_microns} - 1\text{dbu}))$.

- **max_search ms_microns**

A required argument that specifies the maximum Euclidean search distance from each measurement point on *layer_target* in order to find closest points on *layer_im1* and *layer_im2*.

Tip

 The **critical_dimension** setup command can be used in place of this required argument. The *ms_microns* value used for **max_search** is $0.5 * \text{critical_dimension_value}$.

- contour_diff_spacing *cds_microns*

An optional argument that specifies the maximum distance between measurement locations along a single contour. Default is 0.01 microns.

Tip

 The **critical_dimension** setup command can be used in place of this argument. The *cds_microns* value used for **contour_diff_spacing** is $0.2 * \text{critical_dimension_value}$.

- contour_layer_spacing {*layer_im1_im2_spacing_microns*}

An optional argument that specifies the maximum distance between measurement locations on contours *layer_im1* and *layer_im2* during scanning (default: contour_diff_spacing*1.5).

- jog_filter *jog_length_microns jog_extend_microns*

Tip

 Siemens EDA strongly recommends the use of jog_filter, especially when comparing contour layers. Not specifying jog_filter may cause contour_diff to detect false errors.

An optional argument that specifies that no measurements are performed around jogs on *layer_target*. Jogs are defined using the following arguments:

- *jog_length_microns*

Specifies the maximum length of a jog, in microns.

- *jog_extend_microns*

Specifies how far the jog-adjacent edges are filtered in microns.

- exception [also [property_value *prop_value*]]

An optional argument that outputs exception markers.

- When exception is specified, output is generated only for those measurement points where a difference could not be computed, due to point B or point C not being found within **max_search** of the measurement point. The exception option used alone is mutually exclusive with the **constr** and property arguments.

- When exception also is specified, a *constraint* is required and property specification is allowed. In this mode, **contour_diff** outputs both exception shapes and constraint-related results. If a property block is specified, exception polygons have a property value attached equal to *property_value*. The default value is -0.0001. This is useful for limiting and histogram generation.

- property '{' max | diff '}'

An optional argument that specifies a property operation for the returned output of this command.

- max — Returns the maximum contour difference (distance between the contours).
- diff — Returns the maximum contour difference in distance as a signed value. The sign corresponds to the relative position of the contours at the measurement location:
 - *diff > 0* — *layer_im1* is inside *layer_im2*
 - *diff < 0* — *layer_im1* is outside *layer_im2*

The constraint condition is always checked against the *contour_diff* magnitude as reported in the *max* property. If *max* was not specified in the property block, it is added automatically.

The braces ({{}}) around the property keyword(s) are required syntax when both properties are specified.

For more information on property and limits, see the section “[Error-Centric Section Blocks](#)” on page 69.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see the section “[Pinpoint Output Block](#)” on page 102.

Examples

```
contour_diff > 0.003 poly im0 im1 inside active max_search 0.05
```

copy

DRC-type operation

Copies the specified layer to a new layer.

Usage

```
copy layer [property '{' property_name1  property_name2....'}']  
[classify, limit, or histogram block]
```

Arguments

- *layer*

A required argument, specifying the layer to copy.

- property '{' property_name1 property_name2....'}

An optional argument that allows Calibre OPCVerify to access the specified DFM properties of its input layers for use in classification or with the [interact](#) command.

The property {...} clause is allowed only if *layer* is a Calibre OPCVerify input layer containing polygons; it cannot be a derived layer (from a setlayer operation) or an edge layer.

Property names inside the property list must be valid DFM property names on the input layer. Both numeric and string DFM properties are supported, though string properties cannot be used for limiting or as a scoring property for classify.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69. However, copy supports the full set of error-centric options only when the property block contains at least some numeric properties.

Without an explicit property block, the only supported error-centric option is classify.

Examples

Copies the previously-defined layer t1 to the new layer tx:

```
setlayer tx = copy t1
```

cornerchop

Verification control

Chops convex and concave corners on polygon layers.

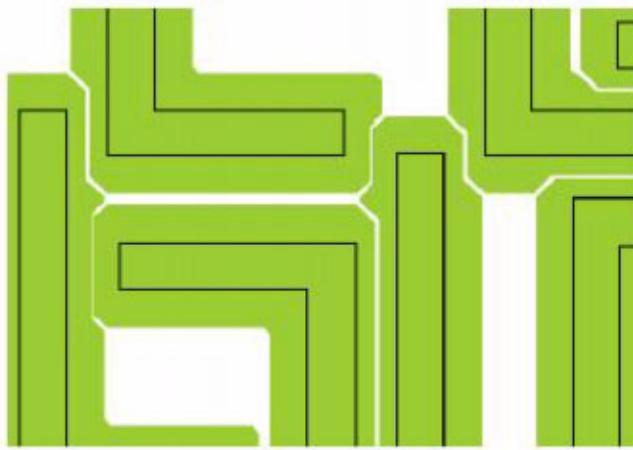
Usage

```
cornerchop layer {{convex_chop concave_chop} | chop_table {'table'}}  
[right_angles_only] [chop_full_edge 0 | 1] [chop_notches {diag | rect | all | none}]
```

Description

Chops out a triangular section off all corners. It returns the modified polygon layer.

Figure 4-17. cornerchop Example



Note

 If you are building tolerance bands on 45-degree angle layouts, you may run into some difficulties in accurately representing the geometry. Siemens EDA advises that you use multiple smaller, alternating increments of [build_tolerance](#) and cornerchop until you achieve a satisfactory result.

Arguments

- ***layer***

A required argument, specifying a layer containing polygons. This layer is useful for operating on [build_tolerance](#) data.

- ***convex_chop***

A required argument, specifying the amount of space in user units (default is in microns) to chop out of convex corners.

- ***concave_chop***

A required argument, specifying the amount of space in user units (default is in microns) to chop out of concave corners.

- ***chop_table table***

An alternative argument to the required ***convex_chop*** and ***concave_chop*** arguments specifying a more detailed lookup table for chop lengths, enclosed in braces. Each chop table entry is a set of three values in microns; an edge length, a convex corner chop value, and a concave corner chop value.

Each corner has its shorter edge measured, and its length is used to reference a table entry.

- If no entry with an exact match to the edge's length is found, linear interpolation is done between the two nearest entries. In the example table below, a length of 0.035 would have its values interpolated to 0.09 for convex and 0.0085 for concave corners.
- If a corner's edge length is smaller than the minimum edge length in the table, the corner uses the minimum value in the table.
- If a corner's edge length is longer than the maximum edge length in the table, the corner uses the maximum value in the table.

An example chop table might look similar to the following:

```
setlayer target_cornerchop = cornerchop target chop_table {\  
 0.020 0.004 0.003 \  
 0.030 0.008 0.007 \  
 0.040 0.010 0.010 }
```

- ***right_angles_only***

An optional argument that sets the command to only chop right angle corners.

- ***chop_full_edge 0 | 1***

An optional argument that specifies the maximum size of a chopped corner. A value of 0 chops a corner that exceeds the half length of an edge on either side. A value of 1 chops a corner that exceeds the full length of an edge on either side.

- ***chop_notches {diag | rect | all | none}***

An optional argument that specifies how existing chopped corners and notches are handled before corner chop calculations.

- **diag** — Existing chopped corners smaller than the CC or CX values (these values are defined in the litho model) are removed before chopping.
- **rect** — Existing orthogonal notches at corners are removed if they have both edge lengths smaller than the corner chop length (CC or CX) and an area less than the area of the corner ($CC^2/2.0$ or $CX^2/2.0$) multiplied by the area limit scale factor set by the user environment variable LITHO_CHOP_NOTCHES_AREA_LIMIT_SCALE.

Note

 The default value of LITHO_CHOP_NOTCHES_AREA_LIMIT_SCALE is 0.88.

- all — Both existing chopped corners and orthogonal notches are removed.
- none — Removes nothing before performing corner chop calculations.

Examples

Given the following definitions:

```
layer SHAPES visible dark  
setlayer tol = build_tolerance outer SHAPES max .08 stepsize .01
```

the following command cornerchops the supplied layer (tol), returning the result layer **to**:

```
setlayer to = cornerchop tol .02 .02
```

curvature_check

Checks if the radius of curvature for a specified layer (usually a curvilinear mask or an image) is larger than a user-specified value.

Usage

```
curvature_check layer
  [[not] {inside | outside} filter_layer]
  radius radius_um length length_um [{all | convex | concave}]
  [{ {expand | markers} [half-width_microns] } | {extents [max_extent] }]
  # error-centric section
  [property '{'
    radius
    signed_radius
    curvature
    fit_error
  '}']
  [classify, limit, or histogram block]
```

Arguments

- ***layer***
A required argument specifying a layer to check the curvature on. The layer must contain relatively smooth objects, such as curvilinear masks or images.
- [[not] {inside | outside} *filter_layer*]
An optional argument specifying a filter layer. Only polygons [not] inside or outside of the *filter_layer* are checked.
- ***radius radius_um***
A required argument specifying a radius constraint. All regions on the input layer where the curvature radius is less than ***radius_um*** are marked.
- ***length length_um***
A required argument specifying a contour length constraint. Calibre OPCVerify estimates the curvature at the center of each edge of layer by fitting a parabola to a piece of contour of ***length_um***. This is an important argument, because too large or too small a length can lead to poor estimates of the curvature:
 - If ***length_um*** is too small, Calibre OPCVerify might use too few edges (in the worst case, just one edge) to get a reliable estimate of the curvature. Actual contours are not parabolas, even though any smooth line can be locally approximated by a parabola.
 - If ***length_um*** is too large, the piece of contour used by Calibre OPCVerify is unlikely to be a good approximation of the contour.

These requirements push the ideal value for **length_um** in opposite directions. If the typical edge length in **layer** is δ , then theoretically it is necessary to have radius \ll length $\ll \delta$, but usually there is not enough room between the radius value and δ to satisfy strong inequalities. Use the following guidelines:

- length > radius is bad, length > 3*(radius) is worse.
- length should be at least 5 δ .

A best practice is that in most cases **length_um** should be approximately equal to **radius_um**. You should use curvature_check on layers with as small δ as possible. For a layer with $\delta \approx 2$ nm, there is no way to check that its radius of curvature does not exceed 5 nm.

- **all | convex | concave**

An optional keyword that controls which parts of the input layer are checked (all, convex parts only, or concave parts only).

- **{ {expand | markers} [half-width_microns] } | {extents [max_extent]}**

An optional keyword that sets the appearance of error markers.

- expand — Each edge that violates the curvature restriction is expanded *half-width_microns* and written to the output. This is the default mode.
- markers — Each edge that violates the curvature restriction is marked at its center with a square marker with a side of $2 * \text{half-width_microns}$. This can be used as a debugging method; with a small enough *half-width_microns* value, the markers do not merge, allowing you to examine their curvature values separately.
- extents — Consecutive edges that violate the curvature restriction are joined into an error region, after which the bounding box for the error region is output. If the size of the bounding box is larger than the specified *max_extent*, it is subdivided until the bounding box is smaller than the *max_extent*.

If *half-width_microns* is not specified, it defaults to 2 dbu.

- **property '{
radius
signed_radius
curvature
fit_error
'}**

An optional argument specifying a properties block, which must be enclosed in braces.

The curvature_check command supports four properties:

- **radius** — The radius of the curvature in microns. This property is always positive.

When several curvature_check errors are merged together, the properties of the combined error are the properties of the individual error with the smallest radius property.

Note

 Because Calibre OPCVerify uses the radius property for the property merge operation, it silently adds radius to the list of properties if you specify a property list without it.

- signed_radius — The radius of the curvature in microns, but it is positive for convex regions and negative for concave regions.
- curvature — The signed curvature, $1/\text{signed_radius}$.
- fit_error — The maximum distance from a vertex of the contour to the approximating parabola in microns. A large value of fit_error indicates a poor fit, which typically happen when the length argument is too large or when a contour has some really bad kinks.

Note

 While a small value of fit_error does indicate a good fit, it does not guarantee a good value of the curvature. When the length argument is too small, fitting is always good, but the curvature is usually wrong.

- *classify, limit, or histogram block*

An optional argument that specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

dofcheck

Verification check

Tests the depth of focus value for multiple contours. Used for process window checks.

Usage

dofcheck

```
[constr]  
layer_target  
{image_set image_set_name [focus_units {nm | um}] |  
    {images i1 i2 i3 [i4 ... i8] defocus f1 f2 f3 [f4 ... f8]} }  
max_search max  
[tolerance tol]  
[[not] {inside | outside} layer_target_filter]  
[{width | space}]  
[dof_spacing micron_spacing]  
[trim_ends trim_microns]  
[separation sep]  
[exception [also [property_value prop_value]]]  
# error-centric section  
[property {'min'} [classify, limit, or histogram block] [add_properties block]  
[pinpoint_output block]]
```

Description

Tests the DOF (depth of focus) value for a specified constraint. In this operation, DOF is defined as follows:

The target shape edges are filtered, and internal or external CD sections (similar to the **measure_cd** operation) are identified.

It uses the following flow:

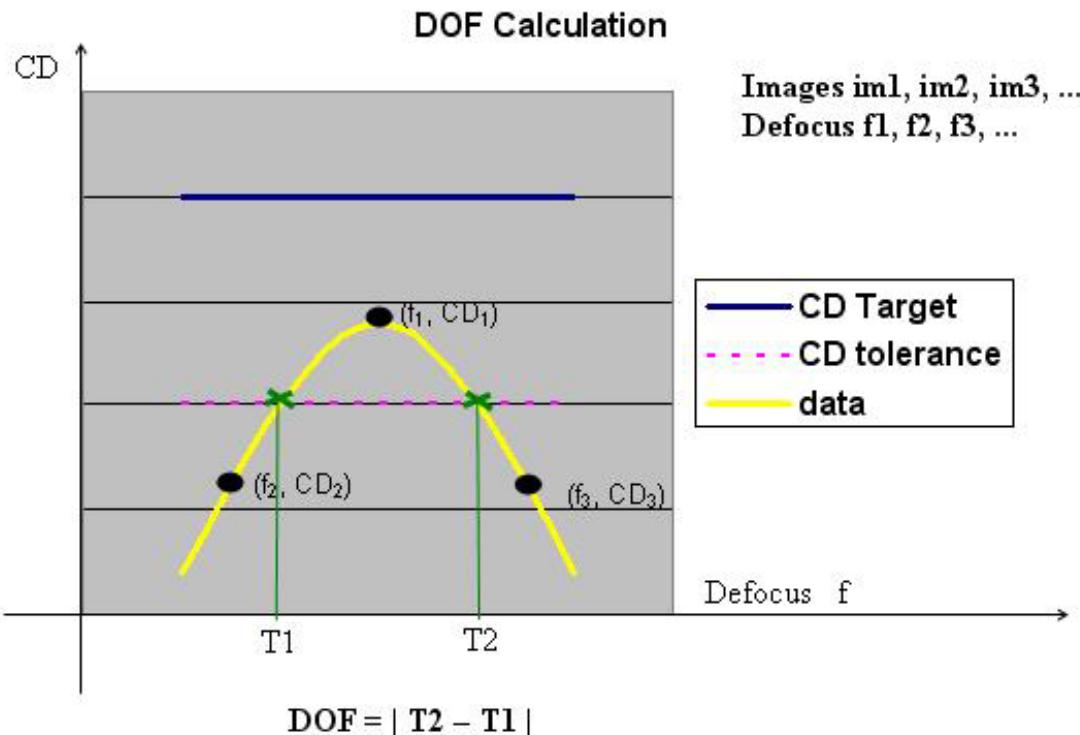
1. Each identified CD section is automatically subdivided into smaller sections.
2. Each of those sections are fitted with several measurement gauges placed perpendicular to the CD, spaced approximately *dof_spacing* apart along the CD.
3. For each measurement gauge, **dofcheck** determines the CDtarget values CD1, CD2, CD3, (and so on) as long as the contour images are less than **max_search** away from the target layer edges.
If one of the CD values cannot be determined, an exception is generated for that section of the CD.
4. If all the CD values are measurable, they are solved as a parabola.
 - o T is defined as the internal or external width of a given CDtarget section.

- The CD tolerances CDtolerance_max and CDtolerance_min are defined as:
 - CDtolerance_max = T*(1+tolerance)
 - CDtolerance_min = T*(1-tolerance)

At each CD gauge, **dofcheck** finds a parabolic least square fit to a set of (f_1, CD_1) , (f_2, CD_2) , (f_3, CD_3) , and (f_i, CD_i) points (see Figure 4-18) to compute coefficients a, b and c for the equation:

$$CD = a * \text{defocus} * \text{defocus} + b * \text{defocus} + c$$

Figure 4-18. dofcheck Diagram



Three main cases are possible:

- Case 1:

The parabola cuts CDtolerance_min or CDtolerance_max at two places. Assuming the parabola cuts CDtolerance_min, then DOF is defined as follows:

$$\begin{aligned} T2 &= (-b + \sqrt{b^2 - 4*a*(c - CDtolerance_min)}) / 2*a \\ T1 &= (-b - \sqrt{b^2 - 4*a*(c - CDtolerance_min)}) / 2*a \\ \text{dof} &= | T2 - T1 | \end{aligned}$$

CDtolerance_max works using a similar formula.

- Case 2:

If the parabola does not cut CDtolerance_min or CDtolerance_max, DOF is defined to be 0 in this case.

- Case 3:

The parabola cuts both CDtolerance_min and CDtolerance_max. Then it has two separate sections within the tolerance band, while its apex is outside the band. Suppose the defocus intervals corresponding to satisfying sections of the parabola are $[T1_a, T2_a]$ and $[T1_b, T2_b]$. The defocus of the nominal condition F_{nom} is used to determine DOF:

- If F_{nom} lies between $T1_a$ and $T2_a$, then $DOF = |T2_a - T1_a|$.
- If F_{nom} lies between $T1_b$ and $T2_b$, then $DOF = |T2_b - T1_b|$.
- Otherwise, $DOF = 0$.

5. If the DOF value for a given gauge satisfies the constraint, then the part of the CD around the gauge will be written into the output layer.

Concurrency Support

Two dofcheck commands will run concurrently if they differ only in *constr*, exception, “width | space”, “trim_ends”, and “property” arguments.

Arguments

- *constr*

A required argument (unless the exception argument is specified) specifying the DOF values for which output shapes will be generated. This must be the first argument when used.

- *layer_target*

A required argument that specifies the drawn target layer. Must be the first argument if the exception argument is specified, or the second argument otherwise.

- **image_set** *image_set_name* [focus_units {nm | um}]

A required argument that can be used in place of the **images** and **defocus** arguments, shown following. It supplies a previously-constructed set of at least three image contours at different focus conditions, as created by [image_set](#). The image_set command must appear before the dofcheck command.

The focus_units parameter can be used to set the output for properties, common dof, and sgd output in the specified unit type (default is um). **image_set** is presumed to supply focus in nm.

Note

 The **image_set** argument is intended to replace the images and defocus arguments; the images and defocus arguments may be removed in a future release.

- **images *i1 i2 i3 [i4 ... i8]***

A required argument that specifies at least three required image contours (corresponding defocus values are specified following). The first image must be created at the nominal focus condition. Up to eight different image contours with their corresponding defocus values are accepted, improving the robustness of the parabolic fit.

- **defocus *f1 f2 f3 [f4 ... f8]***

A required argument that specifies the defocus values associated with the images arguments. This keyword must immediately follow the images keyword and is required.

- **max_search *max***

A required argument that specifies the maximum search distance in microns from the target to an image contour to compute contour CDs. A CD width or space must be $\leq 2*\text{max_search}$ for this operation to be performed. All CD widths larger than $2*\text{max_search}$ are not measured.

Tip

 The **critical_dimension** setup command can be used in place of this required argument. The *max* value used for **max_search** is $0.5*\text{critical_dimension_value}$.

- **tolerance *tol***

An optional argument specifying an acceptable CD tolerance value, represented as a percentage between 0 and 1. The default is 0.1, which means that a 10% CD variation is acceptable.

- **[not] {inside | outside} *layer_target_filter***

An optional filter argument. Only CDs formed by target layer edges inside or outside *layer_target_filter* will be considered if the filter is specified. Otherwise, CDs formed by all *layer_target* edges will be checked.

- **width | space**

An optional argument specifying whether the operation is detecting internal (width) or external (space) CDs. Default is width.

- **dof_spacing *microns_spacing***

An optional argument that specifies the maximum gauge spacing distance in microns. Default is 0.01 microns.

Tip

 The **critical_dimension** setup command can be used in place of this argument. The *microns_spacing* value used for dof_spacing is $0.2*\text{critical_dimension_value}$.

- *trim_ends trim_microns*

An optional argument that specifies that the output boxes from this command are trimmed by the specified value along the target edge if the output box is longer than $2*trim_microns+2dbu$ along the target edge. This implies that *trim_microns* cannot be larger than the *dof_spacing* argument (*microns_spacing*)-2dbu.

- *separation sep*

An optional argument that specifies the minimum target separation required to measure a CD. Target separation is defined as the distance along the target polygon between the ends of the target CD. Locations with a target separation less than *sep* in microns are excluded from the measurement. This option allows you to exclude line and space ends from CD measurements without using a separate filter layer. Default is = 0 (no separation).

- *exception [also [property_value prop_value]]*

An optional argument that outputs exception markers.

- When exception is specified, output is generated only around the gauges that were part of a CD, but that did not intersect at least two image contours. The exception option used alone is mutually exclusive with the *constr* and property arguments.
- When exception also is specified, *constr* is required and property specification is allowed. In this mode, **dofcheck** outputs both exception shapes and constraint-related results. If a property block is also specified, exception polygons have the user-specified *prop_value* attached. The default value is -0.001. This is useful for limiting and histogram generation.

- property ‘{’min‘}’

An optional argument that specifies an optional property operation for the returned output of this command. Only the minimum dof value (specified as “min”) can be computed for this command. The braces ({{}}) around the property keyword(s) are required syntax.

- *standard error-centric options*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. The full syntax is described in the section “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Examples

```
dofcheck < 0.05 poly images im0 im1 im2 defocus 0 0.05 -0.05 inside \
active max_search 0.05 width
```

empty_layer

Verification operation

Creates an empty derived layer that can be output to SVRF or used in other operations.

Usage

empty_layer

Arguments

No arguments.

Examples

```
setlayer X = empty_layer
```

enclosure

DRC-type operation

Performs an enclosure check within the specified region.

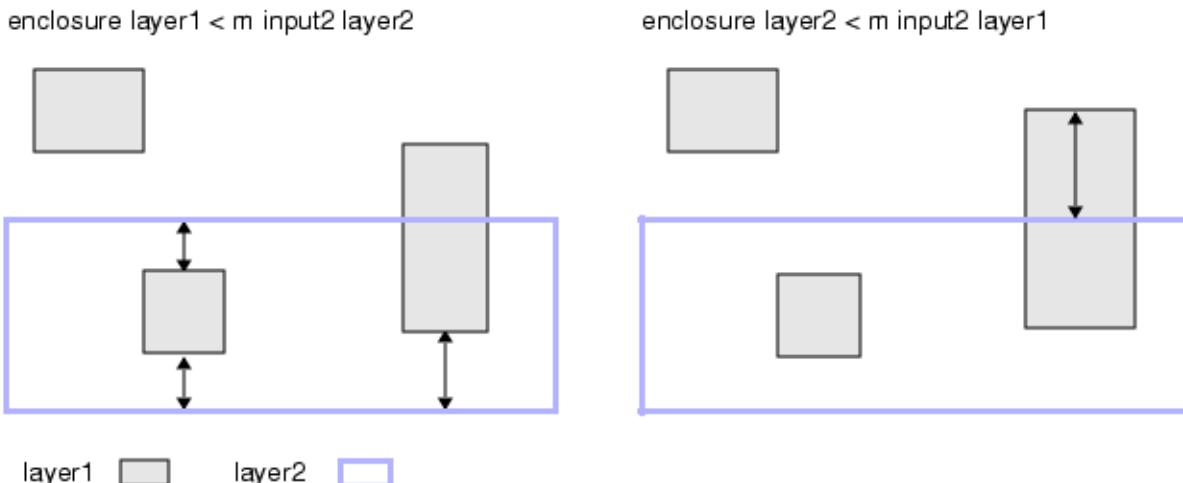
Usage

```
enclosure layer1 constraint input2 layer2 [inside layer3][parallel only][extents]  
[[not] projecting [proj_constr]]  
{ {{expand expand_value_microns} | region} [manhattan | nonmanhattan]}  
[property '{'  
{min | max}  
'}']
```

Description

Performs an enclosure check within the specified region. This uses the OPPOSITE metric, and outputs the REGION of the violation, as defined by the Calibre DRC OPPOSITE and REGION options.

Figure 4-19. Basic Enclosure Rule Checks



Concurrency Support

Concurrency is supported for multiple enclosure calls that differ only in the *constraint* argument.

Arguments

- *layer1*

A required argument, specifying a layer containing the polygon shapes to measure enclosure on. Usually used with layers generated with the [image](#) command.

- ***constraint***

A required argument specifying a bounded constraint, which can include comparison operators such as $<$, \leq , $>$, and \geq .

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- ***input2 layer2***

A required argument, specifying a layer used only to compute enclosure measurement; Calibre OPCverify computes the enclosure of the outside of *layer1* edges by the inside of *layer2* edges. (This means that the order of input is important.)

- ***inside layer3***

An optional argument, specifying that the check is only performed on edges which are from *layer1* not outside *layer3*.

- ***parallel only***

An optional argument specifying that the command only measures parallel edges. This option is used with the property argument.

- ***extents***

An optional argument that when specified causes Calibre OPCverify to form extent rectangles on the input layer before performing the check. This argument is primarily for contact layers.

- ***[not] projecting [proj_constr]***

An optional argument, specifying that the external distance is measured only where one edge projects onto another edge. If a constraint is specified in *proj_constr*, the projection length must also pass the constraint to pass the enclosure check.

Use the *not* command to measure only when neither edge projects onto the other edge.

- **{expand *expand_value_microns* | region} [manhattan | nonmanhattan]**

A required argument, specifying the output type from one of the following choices.

- expand and nonmanhattan — Expands each matching edge and outputs the expanded edges.
- expand and manhattan — Expands each matching edge by the specified amount and outputs the bounding box of the expanded edges.
- region and nonmanhattan — Outputs regions potentially containing skew edges similar to the SVRF [Enclosure](#) REGION statement.
- region and manhattan — Computes regions similar to the REGION statement, but orthogonalizes the regions before output.

manhattan is the default choice for *manhattan* or *nonmanhattan*.

- property '{
min
max '}'

An optional argument that outputs properties attached to the result markers. The “min” property calculates the smallest distance measurement covered by the marker; the “max” property calculates the largest distance measurement covered by the marker.

The property argument also requires the parallel_only argument to be specified. The braces ({{}}) are required, and “min” and “max” must be the only argument on a line.

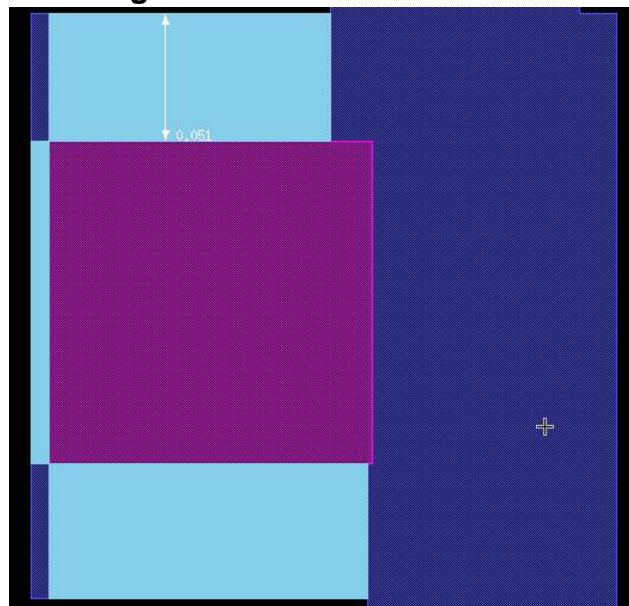
Examples

The following code tests whether or not a contact layer is enclosed by the target shapes by more than 0.06 um:

```
setlayer enc1 = enclosure contact < 0.06 input2 target region
```

In the result shown in [Figure 4-20](#), the enclosure output is shown in light blue, for the contact layer (purple) enclosed by the target layer (dark blue):

Figure 4-20. enclosure Result



end_cap

Verification control

Finds end caps on edges compared to a contour layer.

Usage

```
end_cap layer_active [layer_activeContour]  
    inside layer_poly_target [cut_by cutmask] layer_poly_sim  
    distance_constraint  
    [max_search max_search_microns]  
    [{exceptions_only | exceptions_also}]  
    [max_edge max]  
    [output_expand expval [to_edges_also] [manhattan | nonmanhattan]]  
    # error-centric section  
    [property '{'  
        max  
        min  
    '}' [classify, limit, or histogram block]]
```

Description

This operation is used to find a minimum cap from the edges on *layer_active* to an enclosing edge on *layer_poly_sim*. The operation starts by selecting the edges of active target lying inside *layer_poly_target*. For selected each edge a search region is created, bounded by the active target edge and a parallel segment offset by the maximum constraint. (Note that all constraints must have an upper bound.) The search uses an opposite metric.

It can be used to check the end cap enclosure, and should only be used for poly over active overlap checks.

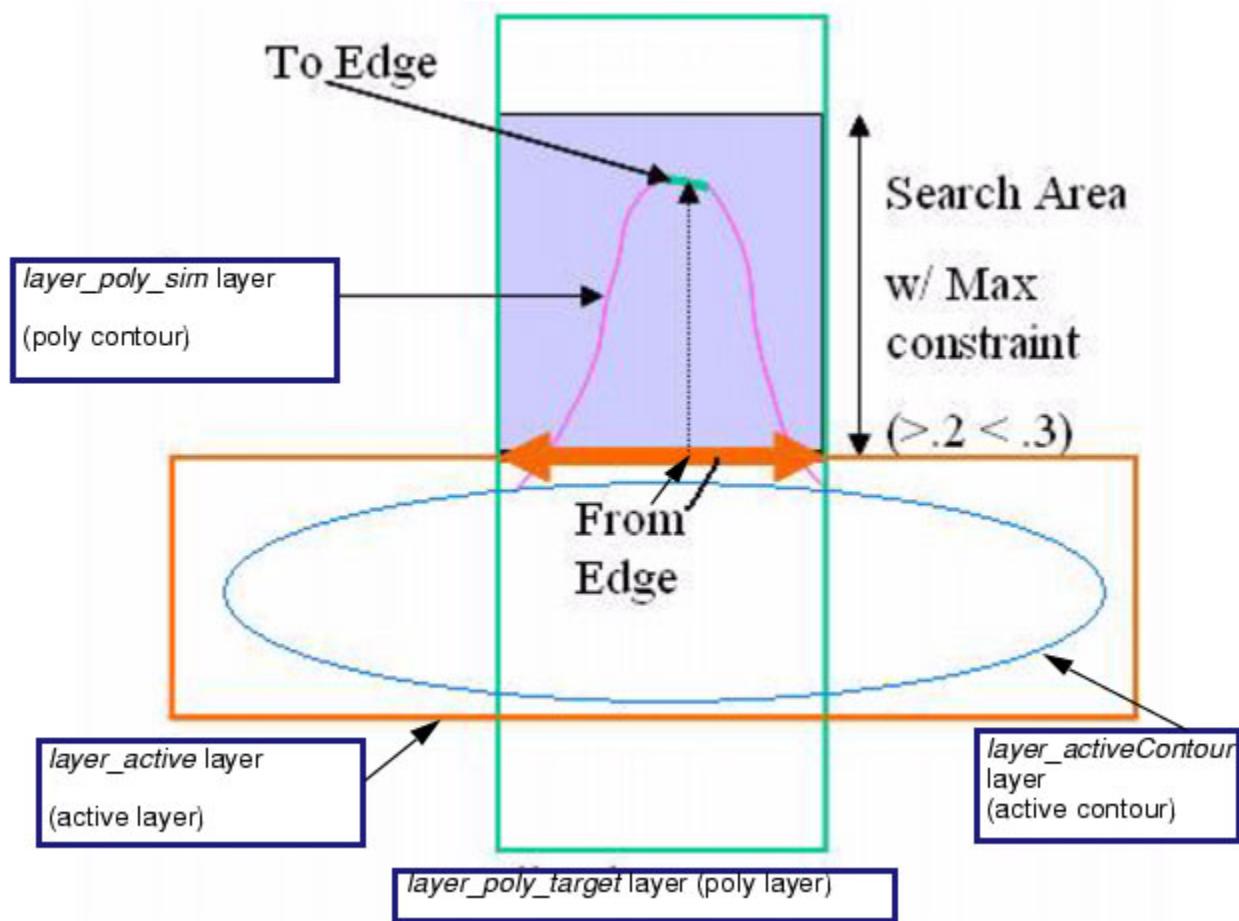
The operation outputs markers for the region between the *layer_active* edges and *layer_poly_sim* edges that meet the distance constraint.

Concurrency Support

Concurrency is supported for multiple end_cap calls that adhere to the following restrictions.

- The input layers (*layer_active*, *layer_activeContour*, *layer_poly_target*, *layer_poly_sim*) must be the same.
- The *max_edge* and *max_search* arguments must be the same.

Figure 4-21. end_cap Measurement



Arguments

- **layer_active**
A required argument specifying the measurement-from layer (unless *layer_activeContour* is also specified). This is the layer that is going to be enclosed.
- **layer_activeContour**
An optional argument specifying an active contour to measure from.
- **inside layer_poly_target**
A required argument that specifies a filter layer; only *from_layer* edges inside the filter layer are considered in the measurement function.

Note

For cut_by cutmask operations, *layer_poly_target* must be the poly target *after the cutmask is applied*. Otherwise, Calibre OPCVerify reports false errors.

- **cut_by** *cutmask*

An optional argument used for the cutmask process. The *cutmask* layer must be the cut mask used to produce the *layer_poly_target* layer. This option instructs end_cap to measure only to *layer_poly_sim* edges that were created as the result of cutmask application. If *cut_by* is specified, both the max and min property values are tested versus the *distance_constraint*. If either property value satisfies the constraint, end_cap outputs a marker.

- **layer_poly_sim**

A required argument that specifies the enclosing layer. This is usually a contour.

- **distance_constraint**

A required argument. Specifies a *distance_constraint* range, which must follow the standard SVRF constraint syntax rules and must be bounded with an upper bound. This constraint is applied to the maximum enclosure distance found within the search region for each *layer_active* edge.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- **max_search** *max_search_microns*

An optional argument that determines the maximum search distance in microns from *layer_active* edges to *layer_activeContour*. The default value is 0.05 microns.

Tip

 The [critical_dimension](#) setup command can be used in place of this argument. The value used for *max_search_microns* is $1 * \text{critical_dimension_value}$.

- **exceptions_only | exceptions_also**

An optional argument accompanying *layer_activeContour* that causes a 2dbu wide box to be output around each *layer_poly_target* edge from which no *layer_activeContour* was found. (Under normal operations, any edge that does not have a *layer_activeContour* associated with it is not output.)

If *exceptions_only* is specified, only the exception markers are output. If *exceptions_also* is specified, both exception and constraint-selected markers are output.

This option can only be specified when *layer_activeContour* is specified. *exceptions_only* cannot be specified with an error-centric properties block. If *exceptions_also* is specified with a property block, the exceptions will have their property value set as 100.

The size of the output boxes is affected by the *output_expand* option.

- **max_edge** *max*

An optional argument. Any filtered *layer_active* edge or *layer_poly_sim* edge longer than *max* distance will be fragmented into edges of a length smaller than *max*. It is important to make sure that filtered *layer_active* edges are shorter than *max*. Otherwise, the maximum enclosure distance might be computed incorrectly.

The default value is 0.2 microns.

Tip

 The [critical_dimension](#) setup command can be used in place of this argument. The value used for *max* is $3 * \text{critical_dimension_value}$.

- **output_expand** *expval*[**to_edges_also**] [[manhattan](#)] [nonmanhattan](#)]

An optional argument that changes the output style. If **output_expand** is not specified, the command outputs the bounding box of the to- and from- edges that match the constraint.

The operation supports the following modes:

- **output_expand** *expval*

Changes the command output to be just the bounding box of the **layer_active** expanded by *expval*. Larger values create fewer, larger error markers.

- **to_edges_also**

An optional additional argument that causes the matching to-edge to be output inside a bounding box.

- [manhattan](#)

The default option. The command outputs the bounding boxes of the expanded edges, instead of the expanded edges.

- [nonmanhattan](#)

An optional argument specifying the output consists of just the expanded edges and not their bounding boxes. Requires the **output_expand** option to also be specified.

- **property** {'**max**
 min
 '}

An optional argument that specifies optional property operations for the returned output of this command.

- **max** — Returns the maximum distance measured between the qualifying **layer_active** edges and **layer_poly_sim** edges.
- **min** — Returns the minimum distance measured between the qualifying **layer_active** edges and the outside edges within the **layer_poly_target**. It is intended for use with the **cut_by** option. The distance is negative if the cutmask edges lie within the active area (**layer_active** or **layer_activeContour**) and **layer_poly_target**. If no cutmask edges are found within the search distance specified by the **distance_constraint**, the minimum distance is set to 100 if the active_target edge segment is outside the cutmask, or set to -100 if the **layer_active** edge segment is inside the cutmask.

The braces ({{}}) around the property keyword(s) are required syntax. Properties will be attached to all of the error shapes.

Note

 Using the property option can cause a noticeable runtime impact, especially when there are large amounts of result markers generated through the use of this function. The SVRF DRC CHECK MAP rule file command will not retrieve properties from a layer output by this command; you must add an additional DFM RDB check as detailed in the “Add Database and Output Reporting” in the “[Creating the SVRF Rule File](#)” topic.

- *classify, limit, or histogram block*

An optional argument that specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

Examples

The following example code flags an end cap that does not enclose the active region by at least 150 nm. The check is for the contour layer enclosing the active layer, and it uses the poly layer as a filter layer.

```
setlayer uncapped = end_cap ACTIVE inside POLY CONTOUR < 0.150 max_edge \
0.2 output_expand .004 to_edges_also
```

external

DRC-type operation

Checks a region for external distance violations.

Usage

```
external layer1 constraint [inside layer2] [parallel only] [extents]
  [[not] projecting [proj_constr]]
  {{expand expand_value_microns} | region} [manhattan | nonmanhattan]
  [property '{'
    {min | max}
  '}']
```

Description

Performs the external check within the specified region. This function uses the OPPOSITE metric, and outputs the REGION of the violation, as defined by the Calibre DRC OPPOSITE and REGION options.

Concurrency Support

Concurrency is supported for multiple external calls that differ only in the *constraint* argument.

Arguments

- ***layer1***

A required argument, specifying a layer containing the polygon shapes to perform external measurements on. Usually used with layers generated with setlayer commands.

- ***constraint***

A required argument, specifying a bounded constraint, which can include <, <=, >, >=, etc.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- *inside layer2*

When this optional argument is specified, the check is only performed on edges which are from *layer1* not outside *layer2*.

- *parallel only*

An optional argument that limits the operation to parallel edges only.

- *extents*

An optional argument that when specified causes Calibre OPCVerify to form extent rectangles on the input layer before performing the check. This argument is primarily for contact layers.

- [not] projecting [*proj_constr*]

An optional argument, specifying that the external distance is measured only where one edge projects onto another edge. If a constraint is specified in *proj_constr*, the projection length must also pass the constraint to pass the enclosure check.

Use the not command to measure only when neither edge projects onto the other edge.

- {expand *expand_value_microns* | region} [manhattan | nonmanhattan]

A required argument, specifying the output type from one of the following choices.

- expand and nonmanhattan — Expands each matching edge and outputs the expanded edges.
- expand and manhattan — Expands each matching edge by the specified amount and outputs the bounding box of the expanded edges.
- region and nonmanhattan — Outputs regions potentially containing skew edges similar to the SVRF [External](#) REGION statement.
- region and manhattan — Computes regions similar to the REGION statement, but orthogonalizes the regions before output.

manhattan is the default choice for manhattan or nonmanhattan.

- property ‘{’

min

max ‘}’

An optional argument that outputs properties attached to the result markers. The “min” property calculates the smallest distance measurement covered by the marker; the “max” property calculates the largest distance measurement covered by the marker.

The property argument also requires the parallel_only argument to be specified. The braces ({{}) are required, and “min” and “max” must be the only argument on a line.

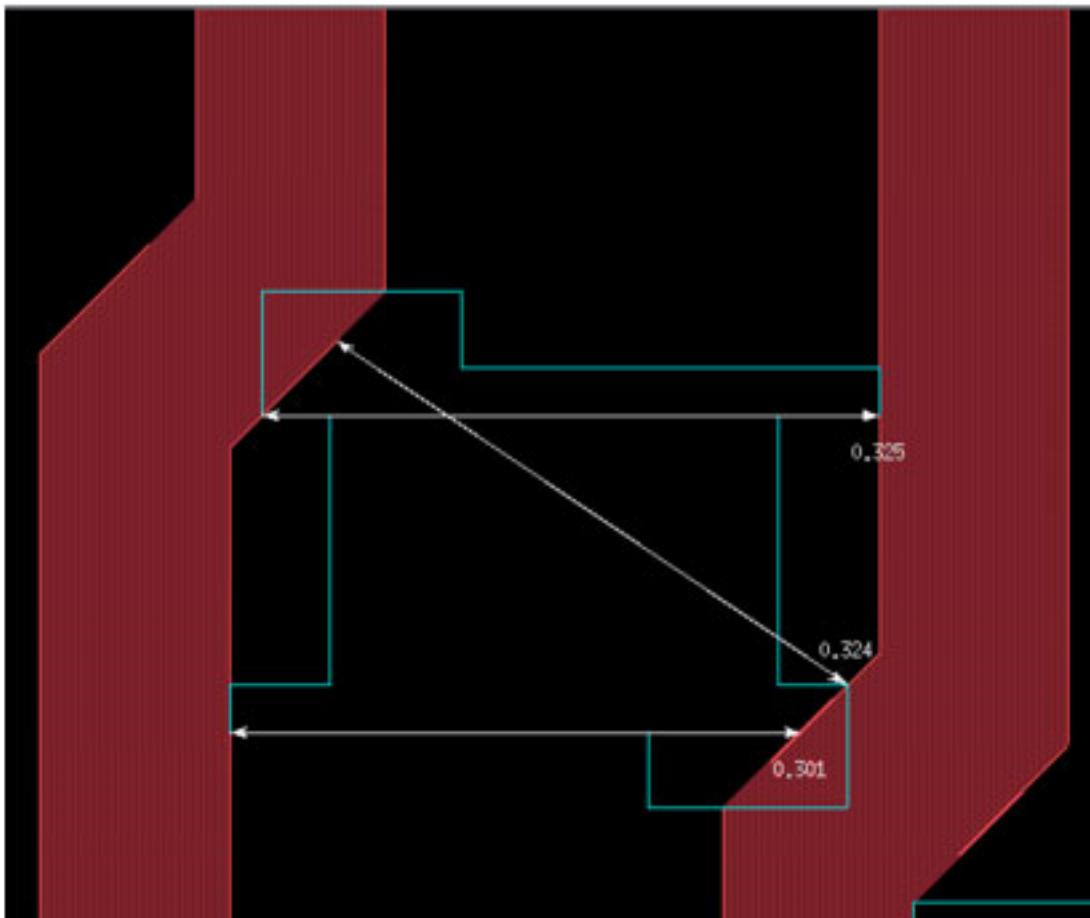
Examples

The following example checks for external distances of the edges on the target layer that are greater than or equal to 0.3 nm, and less than 0.325 nm apart:

```
setlayer ext_test = external target >= .3 < .325 region extents
```

The following figure shows the output of this command:

Figure 4-22. external Example



extra_printing

Verification control

Detects extra printing features (SBARs and sidelobes).

Usage

```
extra_printing target [[not] {inside | outside} filter_layer] contour [max_extent value]
    slivers {off | size [ignore] | {blots size1 [ignore] holes size2 [ignore]} [manhattan | skew]}
    [adjust_small {manhattan | extents | {extents centers size} | suppress}]
    # error-centric section
    [property '{' min
    '}' [classify, limit, or histogram block]]
```

Description

This command detects extra printing features (SBARs and sidelobes) by checking the drawn (**target**) layer versus the specified **contour** layer. Any contour shapes that do not interact with a shape on the target layer are returned.

Arguments

- ***target***
A required argument specifying the target layer.
- [not] {inside | outside} *filter_layer*
An optional argument that performs the specified check using the filter (not inside, inside, not outside, or outside) as an additional selection criteria. Only errors that have at least one edge satisfying the filter will be selected.
- ***contour***
A required argument specifying a generated contour layer.
- ***max_extent value***
An optional argument (default: 0.15 microns) that Calibre OPCVerify uses to distinguish between gross EPE errors at tile/hierarchical boundaries and extra printing defects.
Setting *max_extent* to a very small value will produce false positive extra_printing errors at tile/hierarchical boundaries where the target layer shape is not within *max_extent* distance of the contour. Setting *max_extent* to a very large value will make Calibre OPCVerify run slower due to a flattening of the hierarchy.

Tip

The **critical_dimension** setup command can be used in place of this argument. The *value* used for *max_extent* is $1 * \text{critical_dimension_value}$.

- slivers {off | size [ignore] | {blots *size1* [ignore] holes *size2* [ignore]} [manhattan | skew]}
By default, sliver detection is off.

An optional argument that filters out all narrow contour polygons and holes before checking for extra_printing. By default, sliver detection is off. The parameters specified determine how the slivers are handled.

- *size* [ignore] — Calibre OPCVerify builds a bounding box around every contour polygon and hole. If the smaller side of this bounding box is less than *size* microns, the polygon or hole is considered to be a sliver and is removed from the contour layer.

When the ignore option is specified, the slivers are discarded. Otherwise, sliver polygons are marked by the extra_printing command.

- blots *size1* [ignore] holes *size2* [ignore] — Different *size* criteria can be specified for blots (polygons) and holes.

In the default manhattan mode, the sliver bounding box is aligned with the X and Y axes. When the mode is skew, Calibre OPCVerify builds and tests two bounding boxes; one axes-aligned and one rotated by 45°. It returns the smaller bounding box.

- **adjust_small {manhattan | extents | {extents centers *size*} | suppress}**

An optional argument that specifies how the command marks extra printing contour polygons with extents \leq max_extent.

- manhattan — Manhattanize and output the contour polygons.
- extents — Output minimum bounding boxes of each polygon rather than polygons themselves.
- extents centers *size* — Output marker squares in the centers of the bounding boxes at the specified *size*.
- suppress — Does not output anything for extra printing with extents \leq max_extent.

- **property {'min'}**

The “min” property block must be specified to use the [pw_annotation](#) command with this check. However, note that the property “min” will always be zero.

The braces ({{}}) around the property keyword are required syntax.

- ***classify, limit, or histogram block***

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

Examples

```
setlayer exp = extra_printing poly contour1
```

Related Topics

[bridge](#)

pinch

not_printing

filter_generate

Verification control

Generates filter layers to use with other verification commands.

Usage

filter_generate layer

expand halfwidth

[line_end *le_length* [side *le_side_length*] *le_extend* [*le_jog*]]
[space_end *se_length* [side *se_side_length*] *se_extend* [*se_jog*]]
[jog *jog_length* *jog_extend*]
[convex *convex_extend*]
[concave *concave_extend*]

Description

This operation generates filters intended to be used with commands that can reduce the possible selection area with those filters, such as [measure_cd](#) or [measure_epe](#). It detects line ends, space ends, jogs, concave, and convex corners on the input layer, and generates filter shapes around those features using the following algorithm:

1. Line ends, space ends, and jogs are detected (only options which are explicitly specified are checked for).
 - o A line end is defined by an edge smaller than or equal to *le_length* with two adjacent, parallel edges, larger than or equal to *le_side_length* (or *le_length* if the side option is not specified), with convex corners between the smaller edge and the adjacent edge.
 - o A space end is defined by an edge smaller than or equal to *se_length* between two concave corners with two adjacent edges larger than or equal to *se_side_length* (or *se_length* if the side option is not specified).
 - o A jog is defined by an edge smaller than or equal to *jog_length*, with adjacent edges having one convex and one concave corner, with lengths larger than or equal to *jog_length*.
 - o By default, line ends and space ends are calculated only as unbroken edges. However, you can handle “edge” jogs present on line ends, space ends, and adjacent side edges by specifying the optional *le_jog* and *se_jog* arguments.

When *le_jog* is specified, the line end must be either a single edge or a continuous sequence of edges, adjacent on both ends to parallel properly facing edges and forming convex corners with them. The line end must have a projecting spacing between the ends less or equal to *le_length*, and total lengths of edges less or equal to the sum of the projected spacing and *le_jog*.

Additionally, each of two parallel edges adjacent to a line end must either have length greater or equal to *le_side_length*, or be followed by edges mostly parallel to it. Thus, the total length of parallel edges in the sequence is greater or equal to *le_side_length*, and the total length of non-parallel edges is less or equal to *le_jog*. In the latter case, the entire edge sequence is treated as if it were a single “virtual” edge broken by jogs.

Lastly, the pair of edges adjacent to a line end (either real edges or “virtual” edges with jogs), must have a common projection length greater or equal to *le_length*.

For example, setting *le_jog* to 3 allows one 3 dbu jog, three 1 dbu jogs, or one 1 dbu and one 2 dbu jog on a line end and each of the two adjacent sides (provided that all other requirements are satisfied).

A space end with jogs is similarly defined when *se_jog* is specified. However, the corners between space end and adjacent edges must be concave.

2. The edge or edges for the line end, space end, or jog is expanded by *halfwidth*. Adjacent edges up to their *_extend* values or the edge length +*halfwidth* (whichever is smaller), are also expanded by *halfwidth*. The *_extend* values must be greater than zero.
3. Corners (which are not a line end, space end, or jog) are checked for convexity/concavity, and their adjacent edges (up to *convex_extend/concave_extend* or their length + *halfwidth*, whichever is smaller) are expanded by *halfwidth* as well. After this, everything is merged.

Tip

i One of the more effective uses of this command is to remove unwanted false pinching errors near line ends and false bridging errors near space ends; by creating the filter layer based on the reference layer, you can filter out areas where line end pullback or space end curve contours appear.

The results of this command are placed context layer named in the [setlayer](#) command.

Arguments

- ***layer***

A required argument, specifying a layer containing polygon shapes. Normally, you use the reference layer that you will later supply as the *ref_layer* argument to [measure_cd](#), and the output of this command as the *filter_layer* argument to [measure_cd](#).

- ***expand halfwidth***

A required argument, specifying a command-wide standard distance to expand inside and outside adjacent edges of a detected feature (line end, space end, jog, or corner) by. filter_generate also uses the *halfwidth* value as part of the computation (along with the feature length) for the length along adjacent edges to expand (unless the *_extend* values specified for the feature is smaller).

Note



Although line_end, space_end, jog, convex, and concave are all listed as optional arguments, at least one of them must be specified for this command.

- **line_end *le_length* [side *le_side_length*] *le_extend* [*le_jog*]**

Specifies the handling for line ends on the layer.

- *le_length*

A required argument to line_end, specifying the maximum length of a line end, in microns.

- side *le_side_length*

An optional argument to line_end, specifying the length of parallel edges to *le_length*. The value of *le_length* is used if *le_side_length* is not specified.

- *le_extend*

A required argument to line_end, specifying how far the line end adjacent edges are filtered.

- *le_jog*

An optional argument to line end, specifying the amount of line end jogs that will be allowed while still classifying the fragment as a line end. Default is *0*.

- **space_end *se_length* [side *se_side_length*] *se_extend* [*se_jog*]**

An optional argument that specifies the handling for space ends on the layer.

- *se_length*

A required argument to space_end, specifying the maximum length of a space end, in microns.

- side *se_side_length*

An optional argument to space_end, specifying the length of parallel edges to *se_length*. The value of *se_length* is used if *se_side_length* is not specified.

- *se_extend*

A required argument to space_end, specifying the how far the space end adjacent edges are filtered.

- *se_jog*

An optional argument to space_end, specifying the amount of jog length allowed in the fragment while still classifying the fragment as a space end. (Default is *0*).

- **jog *jog_length* *jog_extend***

An optional argument that specifies the handling for jogs on the layer.

- *jog_length*
A required argument, specifying the maximum length of a jog, in microns.
- *jog_extend*
A required argument, specifying how far the jog adjacent edges are filtered.
- *convex convex_extend*
An optional argument, specifying how far away from a convex corner edges are filtered.
- *concave concave_extend*
An optional argument, specifying how far away from a concave corner edges are filtered.

Examples

The following command creates a filter layer that has the following attributes:

- defines line ends as 0.10 um or less
- defines space ends as 0.10 um or less
- defines jogs as 0.075 um or less

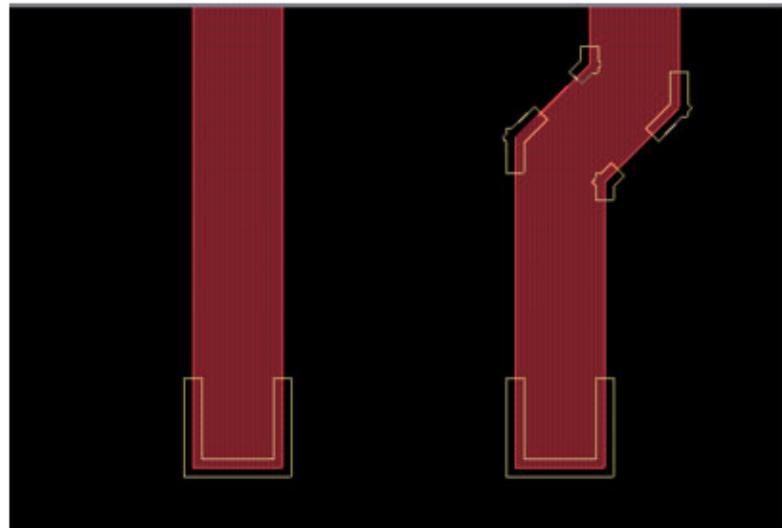
After finding the line ends, space ends, and jogs, it creates filters using the following sizes:

- filters line ends out to 0.1 um
- filters space ends out to 0.01 m
- filters convex corners out to 0.04 um
- filters concave corners out to 0.02 um
- filters jogs out to 0.004 um

```
setlayer fill = filter_generate TARGET expand 0.02 \
    line_end 0.10 0.1 \
    space_end 0.10 0.01 \
    jog 0.075 0.004 \
    convex 0.04 \
    concave 0.02
```

The following figure shows the output of this command:

Figure 4-23. filter_generate Example Output



gate_stats

Verification control

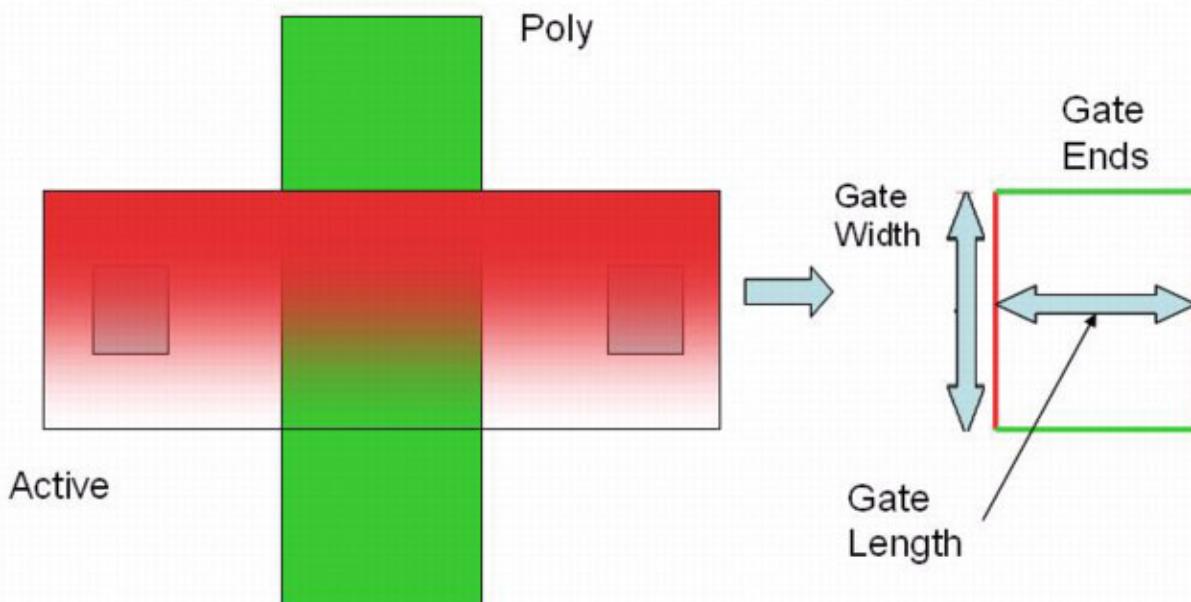
Computes specified statistics for gate structures.

Usage

```
gate_stats layer_poly_sim layer_poly_target
    inside layer_active [active_down_size size_val_microns]
        [poly_filter [not] {inside | outside} layer_poly_filter]
        [active_contour layer_active_contour]
        {cdeffective | min | max | range | average | sigma | center | exceptions_only
            | min_w | max_w | range_w | average_w | sigma_w | center_w | exceptions_only_w}
        [not] constr [absolute | ratio | relative]
        [exceptions_also]
    cd_min min
    cd_max max
        [cd_w_min cd_w_min_microns]
        [cd_w_max cd_w_max_microns]
    max_extent [strict] extent
        [no_split_marker]
        expand_error half_width
        # error-centric section
        [property '{']
        [target_cd | cdeffective | min | max | average | center | range | sigma |
            cdeffective_ratio | min_ratio | max_ratio | average_ratio | center_ratio | range_ratio |
            cdeffective_relative | min_relative | max_relative | average_relative | center_relative |
            range_relative | target_cd_w | min_w | max_w | average_w | center_w | range_w | sigma_w |
            min_ratio_w | max_ratio_w | average_ratio_w | center_ratio_w | range_ratio_w |
            min_relative_w | max_relative_w | average_relative_w | center_relative_w |
            range_relative_w
        '}] [classify, limit, or histogram block]]
```

Description

Computes various statistics over entire gates (defined as polygons generated with a boolean AND of poly and active layers).

Figure 4-24. Constructing Gates With `gate_stats`

The default output consists of regions defined by `layer_poly_sim` and `layer_poly_target`, which can be affected by the ‘active_down_size’ mode. Only polygons satisfying the specified constraint ***constr*** (or failing the constraint, if ‘not’ is specified) are output.

This command is similar to `measure_cd`, except that the gate is not broken up due to its topology (such as notches).

For each pair of appropriately facing parallel edges in the reference layer (`layer_poly_target`), which:

- are inside of `layer_active`
- are separated by at least `cd_min`, and at most `cd_max`
- have mutually projecting edge lengths of at least `cd_min` length

the edges on the check `layer_poly_sim` are searched.

Concurrency Support

The **gate_stats** command will be run concurrently for multiple `gate_stats` commands where the following arguments are the same:

- `layer_poly_sim`
- `layer_poly_target`
- `layer_active`
- `layer_active_contour`

- `active_down_size`
- `layer_poly_filter` and the filter type
- **cd_min**
- **cd_max**
- `cd_min_w`
- `cd_max_w`
- **max_extent**

Arguments

- **`layer_poly_sim`**

A required argument specifying the simulation contour layer to operate on. The simulation contour layer should correspond to the poly layer contour.

- **`layer_poly_target`**

A required argument specifying the reference poly layer to operate on.

- **inside `layer_active`**

A required argument specifying the reference active layer to operate on.

- `poly_filter [not] {inside | outside} layer_poly_filter`

An optional argument that restricts gate length measurements only to locations where both sides of the gate meet the specified condition. If only one side of a gate is included inside the filter, the side not included in the filter is not considered during the constraint calculation or property collection phases.

- `active_down_size size_val_microns`

An optional argument to `layer_active`. Specifying `active_down_size` undersizes the `layer_active` polygons before using it to filter the input contour layer (*layerI*).

You would typically use this option to exclude a portion of the gate ends in the length dimension, such as if a nearby feature on the contour layer was interfering with the measurements.

- `active_contour layer_active_contour`

An optional argument that specifies an active contour layer. Gate width will be measured using `layer_active_contour` perpendicular to gate ends over the interval defined by the projecting gate ends. `layer_active_contour` will be searched `max_search` microns away from the `layer_active` target gate ends.

- cdeffective | min | max | range | average | sigma | center | exceptions only | min_w | max_w | range_w | average_w | sigma_w | center_w | exceptions_only_w

A set of optional arguments that specify how to measure the gate area against the constraint. They can be used concurrently and their functionality differs depending if you select the additional absolute, ratio, or relative modifiers to the constraint.

- cdeffective

Outputs rectangular gates that have a length equal to the effective length of the simulated gate.

It uses the following formula to calculate the gate length:

where w represents the gate width and $L(x)$ represents the gate length ratios measured across its width. For example, the following figure represents one gate:

Figure 4-25. Gate Equation

$$CD_{\text{effective}} = \frac{w}{\int_0^w \frac{1}{L(x)} dx}$$

The result is the total width divided by the total ratios. In this case, both ratios (40:40 and 20:20) are one, so the CD effective result is $60 / 1 + 1 = 30$.

For non-rectangular gates, the cdeffective algorithm is run on both the contour and the target, and the results are compared.

- min

Absolute constraint: Outputs the gate polygon if the minimum gate length meets the constraint.

Ratio constraint: Minimum (measured/target) gate length must meet the constraint.

Relative constraint: Minimum (measured-target) gate length must meet the constraint.

- max

Absolute: Outputs the gate polygon if the maximum gate length meets the constraint.

Ratio: Maximum (measured/target) gate length must meet the constraint.

Relative: Maximum (measured-target) gate length must meet the constraint.

- range

Absolute: Outputs a gate polygon if the maximum gate length minus the minimum gate length meets the input constraint. For example, if the constraint is $0.01 >< 0.03$, the following maximum and minimum gate length values will all result in output:

0.06 and 0.04, 0.1 and 0.08, and 0.22 and 0.20.

“range” can be considered as a tolerance factor for gate length, instead of a specific range constraint.

Ratio: Maximum (measured/target) gate length minus minimum (measured/target) gate length must meet the input constraint.

Relative: Maximum (measured-target) gate length minus minimum (measured-target) gate length must meet the input constraint.

- o average

Absolute: Outputs the gate polygon if the average gate length meets the constraint. This is just the total gate area divided by the width.

Using the same shape above, gate_stats in average mode calculates an L of $(40*40) + (20*20) / 60 = 30.33$.

Ratio: Average measured gate length/average target gate length must meet the constraint.

Relative: Average measured gate length - average target gate length must meet the constraint.

- o sigma

Absolute mode only. Computes the average length and standard deviation from this average for each gate that meets the cd_min/cd_max criteria. The measurement constraint is compared against the standard deviation and gates that meet the constraint are output.

- o center

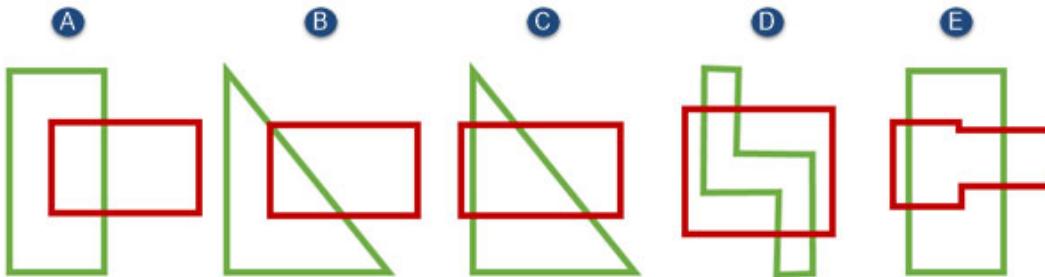
Takes the length measurement at the middle of the width dimension (measure left to right in the figure above). The command uses the cutline at that coordinate for constraint measurements. Note that non-rectangular gates might result in arbitrary results, as some gate centers may land on gate bends, while others will not.

- o exceptions_only

Outputs the gate polygon only if the contour exceeds the search area, or is pinched/bridged within the search area, or if two appropriate gate ends cannot be found. The

following shape configurations would generate an exception (green is poly, red is active):

Figure 4-26. gate_stats exceptions_only Samples



- A generates an exception due to three edges being coincident with the gate area.
 - B generates an exception due to only one edge being coincident with the gate area.
 - C generates two gate edges that are not parallel.
 - D generates edges that are not projecting on each other.
 - E generates an exception because gates must have exactly two end edges (no jogs are allowed).
- min_w

Absolute mode — Outputs the gate polygon if the minimum measured gate width meets the input constraint.

Ratio mode — Outputs the gate polygon if the minimum (measured /target) gate width meets the input constraint.

Relative mode — Outputs the gate polygon if the minimum (measured - target) gate width meets the input constraint.

- max_w

Absolute mode: Outputs the gate polygon if the maximum measured gate width meets the input constraint.

Ratio mode — Outputs the gate polygon if the maximum (measured /target) gate width meets the input constraint.

Relative mode — Outputs the gate polygon if maximum (measured - target) gate width meets the input constraint.

- range_w

Absolute mode — Outputs the gate polygon if the maximum measured gate width minus the minimum gate width meets the input constraint.

Ratio mode — Outputs the gate polygon if the maximum (measured/target) gate width minus the minimum (measured/target) gate width meets the input constraint.

Relative mode — Outputs the gate polygon if the maximum (measured - target) gate width minus the minimum (measured - target) gate width meets the input constraint.

- o average_w

Absolute mode — Outputs the gate polygon if the average measured gate width meets the input constraint.

Ratio mode — Outputs the gate polygon if the (average measured gate width)/(average target gate width) meets the input constraint.

Relative mode — Outputs the gate polygon if the (average measured gate width) - (average target gate width) meets the input constraint.

- o sigma_w

sigma constraint is allowed in absolute mode only. Outputs the gate polygon if the standard deviation of the measured gate width meets the input constraint.

- o center_w

Takes the extent of each polygon in **layer_target** and **layer_active**; takes the shorter dimension of the extent and cuts it in half. Uses the resulting cutline for constraint measurements. Non-rectangular gates might result in arbitrary results as some gate centers land on gate bends and others do not.

- [not] **constr** [ratio | absolute | relative]

A required argument that specifies the constraint to be applied to the statistical measurement. This option is incompatible with the exceptions_only mode. By default, the constraint is absolute; you can use a ratio or relative constraint in conjunction with the min, max, range, cdeffective, or center constraint types.

To specify a constraint as a ratio, the second value is used as a ratio of the first, using 1.0 as ‘equal’. For example, the constraint “< 1.5” is read as “50 percent greater than.”

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- exceptions_also

An optional argument that is mutually exclusive with exceptions_only and exceptions_only_w. When specified, exception gates are output along with constraint-selected gates. Exception gates are assigned property values depending on the property type requested:

- o -100 for cdeffective, min, average, center, sigma, and target_cd
- o 100 for max and range

- **cd_min min**
cd_max max

A pair of required arguments that instructs that the command only performs measurements if the following are true:

The distance between the *layer_poly_target* edges is:

- greater than or equal to cd_min
- less than or equal to cd_max

cd_max is also used in the search distance criteria; if a contour is not within *max/2* of the reference layer edge, it will not be found.

Tip

 The [critical_dimension](#) setup command can be used in place of these required arguments. The *min* value used for **cd_min** is $0.5 * \text{critical_dimension_value}$, and the *max* value used for **cd_max** is $1.5 * \text{critical_dimension_value}$.

- **cd_w_min cd_w_min_microns**
cd_w_max cd_w_max_microns

A pair of optional arguments that instructs the command to only perform measurements if the following are true:

The distance between active edges is:

- greater than or equal to cd_w_min
- less than or equal to cd_w_max

- **max_extent [strict] extent**

A required argument (unless no_split_marker is specified) specifying the maximum gate extent (gate width) in microns that will be processed. Gates wider than this value might be broken into smaller sections with statistics calculated on each section, *unless the strict keyword is specified*, in which case only gates with an extent less than or equal to the specified value are processed. This argument is used with tiling and hierarchy to determine how much context is needed while processing the command. A max_extent value that is too small will cause gates on the edge of a tile or hierarchy instance to be broken up.

Siemens EDA recommends the use of strict mode for max_extent.

A gate broken because of a too small value of max_extent:

- Might cause a subset of the gate to be output in cases where one side of the gate meets a constraint, but the other side does not. Set the max_extent value to be larger than the largest gate length in order to avoid a false positive for a gate that is broken up in this way.
- Assigns broken gate parts separate properties which are merged when the gate error shapes are merged to resolve cross-tile gates, and the property that is retained is based on the maximum or minimum property value (depending on the mode selected)

for the command). Properties that use the ‘average’ or ‘center’ modes may be incorrect.

- no_split_marker

An optional argument that specifies that all gates crossing hierarchical or tile boundaries are processed after the main tile and cell loop in order to get rid of split error markers caused by image inconsistency. Additionally, gates larger than max_extent will always be processed correctly. For this reason, the max_extent argument is optional when no_split_marker is specified.

Note

 no_split_marker can be specified only if the layer produced by this command is not used as input to another setlayer command or as a filter to the [output_window](#) command.

This option is not supported when Calibre OPCVerify is called from inside the MDP EMBED command.

- expand_error *half_width*

An optional argument that can only be used with the min and max constraints.

Similar to the output_expand option of measure_distance and measure_cd, this option outputs a narrow highlight band across the measured gate CD instead of the entire gate area. The *half_width* parameter defines the width of the highlight region as $2 * \text{half_width}$. An error will be issued if the constraint is not “min” or “max”.

- property {*keywords*

}

An optional argument that specifies an optional property operation for the returned output of this command. The braces ({{}) around the property keyword(s) are required syntax. One or more of the keywords shown in the following table can be specified, one per line:

Table 4-16. gate_stats Property Keywords

cdeffective	min	max
range	average	sigma
center	target_cd	cdeffective_ratio
min_ratio	max_ratio	average_ratio
range_ratio	center_ratio	cdeffective_relative
min_relative	max_relative	average_relative
range_relative	target_cd_w	min_w
max_w	average_w	center_w
range_w	sigma_w	min_ratio_w

Table 4-16. gate_stats Property Keywords (cont.)

max_ratio_w	average_ratio_w	center_ratio_w
range_ratio_w	min_relative_w	max_relative_w
average_relative_w	center_relative_w	range_relative_w
center_relative		

Note

 Using the property option can cause a noticeable runtime impact, especially when there are large amounts of result markers generated through the use of this function. The SVRF DRC CHECK MAP rule file command will not retrieve properties from a layer output by this command; you must add an additional DFM RDB check as detailed in the section “Add Database and Output Reporting” in the “[Creating the SVRF Rule File](#)” topic.

- *classify, limit, or histogram block*

An optional argument that specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

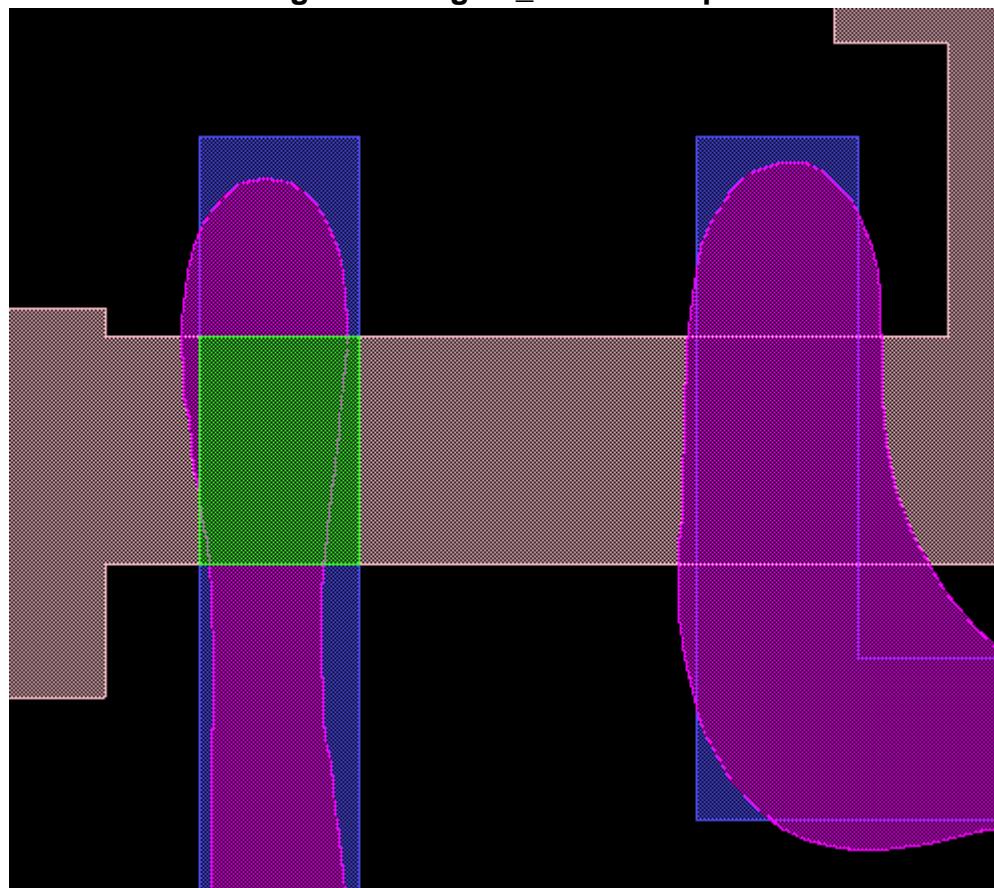
Examples

The following command outputs the shape for gates with a minimum width of less than .06 nm:

```
setlayer gs1 = gate_stats contour Poly_target inside Active \
min <=0.06 cd_min 0.050 cd_max 0.080 max_extent 0.06
```

Notice that the output ([Figure 4-27](#), green box) is only output for the contour on the left, because the gate on the right is never less than .06 nm at any point during the gate crossing.

Figure 4-27. gate_stats Example



The following example applies multiple properties to the results of the command:

```
setlayer gs1 = gate_stats contour Poly_target inside Active min <=0.06
cd_min 0.050 cd_max 0.080 max_extent 0.06 property {
max
min }
```

gauges

Verification command

Calculates properties on gauges supplied as a layer.

Usage

```
gauges input_layer
    max_size max_gauge_length
    [property {in_prop ...}]
    add_properties {'
        prop_name = properties_command
        ...
    '}
```

Description

The gauges operation calculates certain properties on user-provided gauges on an *input_layer*. The layer can be specified using the **RET INPUT** command. The gauges operation can be used, for example, in conjunction with libraries of “hard to print” patterns.

Three possible interactions of a gauge and input layer are supported:

- **Two Intersections** — cd, dof, ldof, image_shift, and nils with the auto_cd option require two intersections between a gauge and the target. If there are fewer intersections, they report an exception.
- **One Intersection** — contour_diff, meef, and nils without auto_cd option require one intersection between a gauge and the target. If there are no intersections, they report an exception. If there are two intersections, they select the worst of the two.
- **More Than Two Intersections** — If a gauge intersects more than two target edges, this command selects two intersections closest to the gauge center and treats this gauge as having two intersections.

Gauges must be placed at (for width) or between (for space) the correct target edges. Gauges are ideally placed when the gauge center is located between the target edges. The gauge should cross both edges for polygon gauges; edge gauges are allowed to cross just one edge.

Note

 Gauges can be oriented in Manhattan, 45 degree, or arbitrary (skew) angles starting with the 2017.3 release. Skew edges only worked for cd, ldof, and image_shift operations previous to the 2017.3 release.

The output layer of this operation is a copy of *input_layer*, with the following additions:

- If a property *{...}* block is specified, the requested properties on the input layer are copied to the output layer.

- Each properties_command line inside the required **add_properties** block adds one property to the output layer. All the property names (created and copied) inside one gauge operation must be unique.

Table 4-17. gauges properties_commands

Command	Runs
<code>cd contour target {internal external auto_cd} [with '{' rel ratio target other '}']</code>	measure_cd
<code>contour_diff target contour1 contour2</code>	contour_diff
<code>meef target contour1 contour2 delta_mask dm (or) meef target image_set image_set_name [delta_mask dm]</code>	meefcheck
<code>dof target {image_set image_set_name {images i1 i2 i3 ... defocus f1 f2 f3 ...}} [focus_units {nm um}] [width space]</code>	dofcheck
<code>ldof target image_set image_set_name [focus_units {nm um}] [width space auto_cd] [ellipse rectangle] [model_poly8 model_poly13] [target_cd {nominal_contour drawn}] dose_latitude lat [common_ldof_file_name [dose_tol dose_tol] [focus_tol focus_tol]] [sgd_output sgd_file_name] [gauge_id GaugeID [sort]][exceptions_also] [dof_max dof_center] [with '{'focus_nom / dose_nom/ cd_type'}'...]</code>	pwcheck
<code>nils target images i1 i2 ... levels f1 f2 ... [width space both auto_cd disregard_cd] (or) nils target image_set image_set_name [...] [width space both auto_cd disregard_cd]</code>	nilscheck
<code>image_shift target {contour1 image_set1}... {width space auto_cd} [with '{'shiftX shiftY'}...']</code>	N/A
Common add_properties (distance, epe, intensity_ils, intensity_nils, imin, imax, contrast)	See “ add_properties Block ” on page 91

Supported **add_properties** subcommands are as follows:

- **cd** — Runs a CD distance check for the specified gauges.

`prop_name = cd contour target {external | internal | auto_cd}
[with '{'rel | ratio | target | other '}']`

This command is equivalent to running the following setlayer operation:

[measure_cd](#) contour target direction inside gauge_layer

The measure_cd required argument cd_max is set to **max_gauge_length** and the required argument max_search is set to **max_gauge_length/2**. measure_cd's minimum property value is also attached to every gauge.

The expected gauges for this property command are strips just crossing the target polygons for internal CD, or just connecting the neighboring target polygons for external CD.

If the auto_cd argument is specified, the type of measurement (external or internal) is detected by how the gauge crosses the reference polygon.

If the “with” argument is specified, additional properties are added to every gauge. This property will have the output layer name as the first part of its name, then a period (“.”) and then the specified postfix: rel, ratio or target corresponding to one of the following **measure_cd** properties:

- rel — The equivalent of “min_relative”
- ratio — The equivalent of “min_ratio”
- target — The equivalent of “min_target”
- other — Writes the specified string as a property, but this property cannot be used for scoring or limiting.
- **contour_diff** — Checks the contour difference across the gauges.

prop_name = contour_diff layer_target layer_im1 layer_im2

This command is the equivalent of the **contour_diff** command, which functions the same as meef (above). The required contour_diff delta_mask argument is set to 1.

- **meef** — Runs a MEEF check on the specified gauges. Both standard contour and **image_set** formats are supported.

prop_name = meef layer_target layer_im1 layer_im2 delta_mask dm

prop_name = meef layer_target image_set image_set_name [delta_mask dm]

This command is equivalent to running the **meefcheck** operation on the provided gauges, using **max_gauge_length/2** as the value for the meefcheck **max_search** argument.

When a gauge intersects a single target edge, MEEF is calculated at the intersection point and assigned to gauge properties. When a gauge intersects two or more target edges, two MEEF values are calculated for a pair of properly selected intersection points, and the maximum of two values is assigned to the gauge property.

The delta_mask keyword is optional when an image set is specified with images that have a different bias value. In this situation, delta mask is calculated as the difference between biases of the first and second images in the image set. Expected gauges for this command are strips just crossing the edges of target polygons, marking places where the

measurements are to be taken. If the same gauges are used for MEEF and CD-based checks, each gauge gets two MEEF measurements at both ends of the gauge, and the reported MEEF is the maximum of two MEEF values.

- **dof** — Runs a DOF check at the specified gauges.

```
prop_name = dof layer_target {image_set image_set_name |  
    {images i1 i2 i3 ... defocus f1 f2 f3 ...}} [focus_units {nm | um}] [width | space]
```

This command is equivalent to running the [dofcheck](#) operation on the user-supplied gauges. The **max_search** value used for the dofcheck call is set to **max_gauge_length**/2. The calculated gauges are assigned to gauge properties.

An [image_set](#) can be used in place of the **images** and **defocus** arguments.

- **ldof** — Runs a process window-based DOF check using the specified gauges.

```
prop_name = ldof target image_set image_set_name  
[focus_units {nm| um}] [width | space | auto_cd]  
[ellipse | rectangle] [model_poly8 | model_poly13]  
[target_cd {nominal_contour | drawn}]  
[dose_latitude lat [tolerance [absolute] tol]  
[common_ldof file_name [dose_tol dose_tol] [focus_tol focus_tol]]  
[sgd_output sgd_file_name]  
[gauge_id GaugeID [sort]] [exceptions_also] [dof_max | dof_center]  
[with {'focus_nom / dose_nom/ cd_type'}...]
```

This command is equivalent to running the [pwcheck](#) operation.

Note

 Due to the 2019.2 default change from `model_poly8` to `model_poly13`, `pwcheck` now requires a minimum of 14 images, not 9 images as it used to be before the 2019.2 release, unless the `model_poly8` option is specified explicitly.

The **max_search** value used for the pwcheck call is set to **max_gauge_length**/2. The LDOF parameters are calculated at every gauge, and corresponding values are assigned to gauge properties. If for a particular gauge the LDOF cannot be found or an exception occurs, property values for that gauge are set to -0.001.

If `dof_center` is specified, the primary DOF property is computed with the center of the rectangle or ellipse fixed at nominal dose and focus. If `dof_max` (the default) is specified, the primary DOF is computed with a floating center point, and the maximum possible DOF is located within the process window.

Note

 For more information on super gauge data files, see the section “Super Gauge Data File Format” in the [Calibre WORKbench User’s and Reference Manual](#).

The target and contour CDs are calculated and stored for each process condition (dose and focus).

Dose_latitude specifies an alternate method for setting min_dose and max_dose.

- $\text{min_dose} = 1 - \text{dose_latitude}/2$
- $\text{max_dose} = 1 + \text{dose_latitude}/2$

If the sgd_output keyword is specified, the gauge, target CD and contour CD are stored to a super gauge data (.sgd) format file.

- Target and contour CDs are calculated and stored for each process condition (dose and focus). If a CD cannot be determined for a particular dose and focus, it is set to -1 in the .sgd file.
- Each gauge may be identified by its ID, which is passed as a property with the value specified in *GaugeID*. This ID is passed to the Struct name field in the .sgd file. The ID may have a numeric or string value. String values have an “s” prepended to them for the string ID value.
- If the gauge_id argument is omitted, the Struct name field is set to “NA”, but the gauge can be identified by its number in the Row column.
- If the sort keyword is specified, gauges in the .sgd file will be sorted by the GaugeID column in lexicographic order.
- The command also attaches the dof property to every gauge. The dof property is set to the minimum value for that gauge.
- sgd_output and common_ldof cannot be specified simultaneously.

When the exceptions_also option is specified, the gauges that produced exceptions are included to the .sgd file output along with normal gauges. The gauges with exceptions have the “On” flag in the .sgd file set to “0”.

sgd_output outputs -1 when the CD cannot be calculated.

The expected gauges for this keyword are strips just crossing the polygons for width measurements, or just touching the edges for space measurements.

- **nils** — Runs the [nilscheck](#) command using the specified gauges. The **max_search** value used for the nilscheck call is set to **max_gauge_length**/2. When a gauge intersects a single target edge, its NILS value is calculated at the intersection point and assigned to a gauge property. When a gauge intersects two or more target edges, two NILS values are calculated for a pair of properly selected intersection points, and the minimum of the two values is assigned to the gauge property.

prop_name = nils layer_target images i1 i2... levels f1 f2...
[width | space | both | auto_cd | disregard_cd]

or the image_set equivalent:

```
prop_name = nils layer_target image_set image_set_name [...]  
[width | space | both | auto_cd | disregard_cd]
```

The optional parameter width | space | both | auto_cd | disregard_cd can be used to specify the type of CD to be calculated. All keywords except auto_cd have the same meaning as in setlayer nilscheck. The default is width. When auto_cd is specified, the type of CD for NILS calculation is detected for each gauge automatically based on how the gauge intersects target polygons: when it crosses a polygon, a “width” CD is used; when it crosses an empty space between two polygons, a “space” CD is used. If a gauge intersects more than two target edges, the type of CD is defined by the mutual position of the particular pair of edges that is used for gauge measurement. The auto_cd option can be used only for gauges intersecting at least two target edges; it is an exception otherwise. The expected gauges for this keyword are strips just crossing the target polygons for width measurements, or just connecting the neighboring target polygons for space measurements.

- **image_shift** — Calculates the image shift for one or more contours. The resulting property values are attached to the gauges.

```
prop_name = image_shift target {contour1 | image_set1} ...  
{width | space | auto_cd} [with '{' {shiftX | shiftY} ... '}']
```

Image shift is a shift of a linear feature on the image from the target along the gauge. It is defined as a distance between centers of corresponding target and contour CDs, which are both measured along the gauge. The gauge should be oriented transversely to the feature. The calculated image_shift is either a positive value or equal to zero.

The input contours are specified as contour layer names or image_sets, which can be mixed. Specifying an image set includes all the images in the set in the calculation. Any number of contours and image sets can be specified for a single measurement within the global limits on the number of input layers.

If multiple contours are specified, the output property is the worst (maximum) value of all contours at each gauge. This keyword searches for target and contour CDs along the gauges, using **max_gauge_length/2** for the **max_search** parameter (see [measure_cd](#)). All CDs required for the calculation must exist; if a target CD or any of contour CDs along the gauge cannot be determined, the corresponding gauge is assigned the exception value -0.001.

The {width | space | auto_cd} parameter specifies the type of CD for measurement: “width” for internal, “space” for external measurements. When ‘auto_cd’ is specified, the type of measurement is determined based on how the gauge crosses the target polygon.

If the with clause is specified, the X and Y components of the image_shift vector are added as a property to the gauge. The vector orientation is from the contour CD center to the target CD center. A positive shiftX value indicates that the contour center is located to the left of the target center; a positive shiftY value indicates that the contour center is

below the target center. This follows the same convention on shift sign as used by setlayer center_shift.

The expected gauges for this command are strips just crossing the target polygons for width measurements, or just connecting the neighboring target polygons for space measurements.

- Common add_properties — The gauges command supports the generic common [add_properties Block](#) commands (distance, epe, intensity_ils, intensity_nils, imin, imax, and contrast), with the following caveats:
 - In the cases where the add_properties block and gauges have commands with the same name (cd, contour_diff, dof, meef, and nils), use the gauges version of the command described on this reference page.
 - Previous to the 2019.2 release, the user-specified gauge for the gauges command was used in place of the automatically-detected gauge that add_properties would normally calculate values from.

The add_property values were calculated at both ends of the gauge, and the worst value of the two for each property was assigned to the gauge property.

Starting with the 2019.2 release, the internal gauge used for add_properties calculations may not exactly coincide with the gauge specified on the input layer. The add_properties internal gauge is now defined by point(s) of intersection between the user-specified gauge and the property's "reference layer".

- The reference layer depends on the add_properties operation, and uses the following argument to that operation:
 - cd, dof, epe, ils, nils: **target**
 - contrast, imin, imax: **target** (in the specified **gauge_set**)
 - contour_diff, meef: **contour1**
 - distance: **layer**
 - intensity_ils, intensity_nils: the layer specified using the measure_at and target_layer parameters
- Measurement rules for common setlayer gauge checks are used to select proper intersection points to define a corresponding add_property gauge.
- When multiple intersections are found, the pair of intersection points that are closest to the center of the user gauge are selected.
- If there are no intersections between the gauge and the reference layer, the added property is assigned the exception value.

- The intensity_nils CD is calculated differently for setlayer gauges:
 - CD for ILS normalization is calculated strictly in the direction of each gauge.
 - The auto_cd choice is added to the normalize_to parameter, which selects width or space CDs for each gauge, individually based on the gauge placement on the measure_at layer. If the gauge does not cross two edges, intensity_nils normalize_to auto_cd returns an exception.

Arguments

- ***input_layer***

A required argument specifying an input layer containing gauges. It is copied to the output layer.

- ***max_size max_gauge_length***

A required argument that sets the cd_max and max_search parameters for the property functions.

- **property {*in_prop* ...}**

An optional argument that names a list of properties on the ***input_layer*** that are also copied to the output layer.

- **add_properties '{**

prop_name = *properties_command*

...

}

Note

 Although it looks similar, this parameter is not the same as the error-centric [add_properties Block](#).

A required command block, enclosed by braces ({}), that contains one or more ***prop_name*** and ***properties_command*** pairs. The ***prop_name*** is added to the output if its associated ***properties_command*** completes with results. Each individual ***prop_name*** command line must start on a new line; you can extend commands across multiple lines with the continuation character ()�.

Notes on Gauge Placement

Calibre OPCVerify does not actually check that input gauges make any sense. If the gauges are made too wide, OPCVerify starts calculating properties with a default step (0.01 microns for most operations) alongside target edges and then selects either the minimum or maximum of them as the gauge property.

Ideally, each gauge should correspond to exactly one measurement of each requested property. Badly placed gauges can cause OPCVerify either to make multiple measurements per gauge or to make no measurements for a gauge at all.

If the gauge placement is completely wrong (such as if gauges placed completely inside target polygons are used for space measurements) Calibre OPCverify will not be able to associate any properties with the gauges. In this case, the gauges have a property with a value of 0 and Calibre outputs warnings to the transcript.

Examples

Example 1 (Polygon Gauges)

```
layer user_gauges
setlayer xmas_tree = gauges user_gauges max_size 0.02 property { ID }
add_properties {
    cd1 = cd contour1 target internal
    cd2 = cd contour2 target external {with rel}
    diff = contour_diff target contour1 contour2
}
```

This command creates the output layer xmas_tree, copies the input layer user_gauges and its ID property into it, and adds four new properties: cd1, cd2, cd2.rel, and diff. Polygons on the input layer (the gauges) are expected to be narrow strips (roughly 1-dbu wide) long enough to cover a line or a space, but not much longer.

Example 2 (Edge Gauges)

Edge gauges require additional handling in the SVRF rule file, because the input gauge and resulting output gauge from the Calibre OPCverify call both must be edge gauges.

In the SVRF section, this section of the code converts the polygon gauges into edge gauges:

```
LAYER space_gauges 1071
LAYER width_gauges 1072
user_gauge = space_gauges OR width_gauges
user_gauge_e = INT user_gauge <= 0.0012 opposite
user_gauge_edge = DFM COPY user_gauge_e CENTERLINE
```

The LITHO OPCVERIFY call uses the edge gauge layer as input (notice the EDGE keyword added to the MAP statement):

```
'pw_result' = LITHO OPCVERIFY m1.opc m1.sraf m1.target
crv_tgt user_gauge user_gauge_edge width_gauges space_gauges
cntr_nom cntr_nom_hi FILE m1.orc.gauges MAP 'pw_result' EDGE
'pw_result' {
    COPY 'pw_result'
}
DRC CHECK MAP 'pw_result' 3000 PROPERTIES
```

The setlayer command inside the Calibre OPCVerify setup command file is otherwise unchanged:

```
setlayer pw_result = gauges user_gauge_edge max_size 0.1 add_properties {  
    cd = cd cntr_nom crv_tgt auto_cd  
    meef = meef crv_tgt cntr_nom cntr_bias1nm delta_mask 0.001  
    ils = nils crv_tgt images cntr_nom cntr_nom_hi levels 0.268971 \  
        0.27 disregard_cd  
    ldof = ldof crv_tgt image_set Contours auto_cd dose_latitude 0.05 \  
        sgd_output ldof_final.sgd exceptions_also with dose_nom  
    contour_diff = contour_diff crv_tgt cntr_nom cntr_bias1nm }
```

Related Topics

[Application 6: Viewing ldof Gauges in the Process Window Analysis Tool](#)

holes

Verification control

Returns polygons that fit in holes found in the input layer.

Usage

```
holes input_layer
  [[not] {inside | outside} filter_layer]
  [constraint] max_extent max_extent
  [output_type {extents | centers size_microns}]
  [inner] [empty]
  # error-centric section
  [property {'{'area
  '}' [classify, limit, or histogram block]]]
```

Description

This operation returns polygons that fit inside holes in the input layer. Note that holes within holes are merged out by default. It is similar to the SVRF **HOLES** command.

Concurrency Support

Up to 32 **holes** operations that have the same *input_layer* can be run concurrently.

Arguments

- ***input_layer***
A required original layer or derived polygon layer for input to the command.
- **[not] {inside | outside} *filter_layer***
An optional argument specifying a filter layer for the command. Only holes that also satisfy the filter conditions (those completely inside or outside the filter layer with the optional “not” inverting the selection) are output.
- ***constraint***
An optional argument used as an area filter. Holes are only output if their area meets the constraint.
- ****max_extent** *max_extent***
A required argument specifying a maximum extent distance; holes exceeding this x or y distance are ignored.
- ***output_type* {extents | centers *size_microns*}**
An optional argument specifying that the output shapes are replaced with:
 - extents — Minimum bounding boxes
 - centers *size_microns* — Squares of *size_microns* located in the centers of the bounding boxes (the specified size value must not exceed **max_extent**)

- inner

An optional argument that changes the behavior of the command such that holes that contain other holes are not returned. This keyword is useful in isolating metal slots, since it returns the innermost hole.

- empty

An optional argument that changes the behavior of the command such that holes that are not completely outside the *input_layer* (enclosed or coincident holes) are not output. This can be used to find holes that have no polygons from the input layer inside them (empty holes).

- property {area}

An optional property block that attaches the area of the hole to resulting output. The braces ({{}}) around the property keyword(s) are required syntax.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

Examples

The following example checks for holes that have an extent of .45 nm and an area of less than 10 nm. The hole must also not contain a hole inside itself.

```
setlayer holes_check = holes has_holes <=10 max_extent 0.45 inner
```

identify_corner

Verification control

Identifies line fragments of edges near corners.

Usage

```
identify_corner layer [length1 constraint]  
    [length2 constraint]  
    corner {convex | concave | angle_constr}  
    extend_dir1 length_from_corner  
    extend_dir2 length_from_corner  
    expand value | {value_in value_out}
```

Arguments

- ***layer***
A required argument, specifying the layer containing the shapes to analyze.
- ***length1 constraint***
An optional argument, specifying a minimum length constraint for the first edge of the pair. If an edge being tested does not pass the given constraint, the edge fails and is not included in the result set.
If only *length1* is specified, the edge passes if either adjacent edge passes the constraint.
- ***length2 constraint***
An optional argument, specifying a minimum length constraint for the second edge of the pair. Both adjacent edges must meet their constraints for the edge to pass. Note that the algorithm tests the adjacent edges twice (given the endpoints A, B of an edge, Calibre OPCVerify tests A/*length1*, B/*length2*, and A/*length2*, B/*length1*). If either permutation passes both constraints, the edge passes.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- ***corner* {convex | concave | *angle_constr*}**
A required argument, specifying the type of corner to identify.
- ***extend_dir1 length_from_corner***
A required argument, specifying how far along the edge to extend the corner marker for a matching first edge.
- ***extend_dir2 length_from_corner***
A required argument, specifying how far along the edge to extend the corner marker for a matching second edge.

Note

 Because of the way corners are detected, a corner that passes both length1 and length2 constraints has both the extend_dir1 and extend_dir2 extensions applied; this results in the longer of the two extensions taking precedence.

- **expand value | {value_in value_out}**

A required argument, specifying the amount to expand the marked edges by.

- Specifying a single value expands the corner inside and outside of the polygon by the same amount.
- You can specify different values for the expansion inside and outside the polygon using the second argument format (*value_in value_out*).

Examples

The following code identifies corners that have the following attributes:

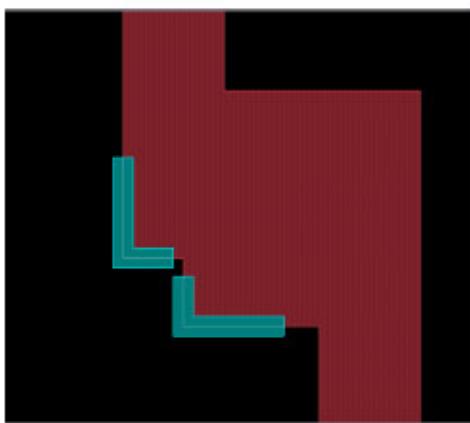
- convex corner
- one edge shorter than .1 um
- second (adjacent) edge greater than .1 um

Identified edges are expanded by .01 nm for .05um in the actual edge direction, and .1 nm in the adjacent edge direction.

```
setlayer idc = identify_corner TARGET length1 < .1 length2 > .1 \
corner convex extend_dir1 .05 extend_dir2 .10 expand .01
```

Figure 4-28 shows the output of the command (notice how the corners are asymmetrical, since our detection check was not symmetrical):

Figure 4-28. identify_corner Example Output



identify_edge

Verification control

Identifies line ends and jogs.

Usage

```
identify_edge layer length length_constraint
  [length1 constraint]
  [length2 constraint]
  [corner1 {convex | concave | angle_constr}]
  [corner2 {convex | concave | angle_constr}]
  [complement]
  [trim_ends trim [min_length]]
  [extend [by_factor] ext_value]
  expand {value | value_in value_out}
```

Description

This operation can be used to identify line ends or jogs. It is analogous to the DRC operation CONVEX_EDGE. It then performs end trimming and span complementation, followed by edge expansion in the normal direction and the parallel direction to create the final output shape.

Arguments

- ***layer***
A required argument, specifying the layer containing the shapes to analyze.
- ***length length_constraint***
A required argument signifying a length constraint for the edge to test.

Note



Avoid using large ***length_constraint*** values, which increase interaction distance.
This can result in flattening of data and slower run times.

- ***length1 constraint***
An optional argument, specifying a minimum length constraint for the first edge adjacent to the edge being tested. If an edge being tested does not pass the given constraint, the edge fails and is not included in the result set.
If only length1 is specified, the edge passes if either adjacent edge to the edge being tested passes the constraint.
- ***length2 constraint***
An optional argument, specifying a minimum length constraint for the second edge adjacent to the edge being tested. Both adjacent edges to the edge being tested must meet their constraints for the edge to pass. Note that the algorithm tests the adjacent edges twice (given

the endpoints A, B of an edge, Calibre OPCVerify tests A/length1, B/length2, and A/length2, B/length1). If either permutation passes both constraints, the edge passes.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- corner1 {convex | concave | *angle_constr*}

An optional argument, specifying a constraint on the corner for the adjacent edge to the edge being examined.

If only corner1 is specified and either corner meets the constraint, the edge passes.

If length1 is also specified, an adjacent edge must pass the combined length1 and corner1 constraints to pass.

- corner2 {convex | concave | *angle_constr*}

An optional argument specifying a constraint for the corner for the second adjacent edge. Both adjacent corners must meet their constraints for the edge to pass. Calibre OPCVerify tests the adjacent corners using both permutations (endpoint A/corner1 with endpoint B/corner2, and endpoint A/corner2 with endpoint B/corner1). If either permutation passes both constraints, the edge passes.

If length2 is also specified, both adjacent edges must pass the set of combined constraints (length1 (and if specified, corner1) and length2, corner2) for this edge to pass.

- complement

An optional argument that identifies edges that do not meet the specified constraints.

- trim_ends *trim* [*min_length*]

An optional argument that enables edge trimming. The parameter *trim* specifies the length (in microns) that will be cut off each end of the output edges. The parameter *min_length* specifies the minimum edge length (in microns) after trimming.

For short edges, the trim length is adjusted to preserve the central portion of *min_length* in ums. If *min_length* is not specified, the default value is 3 dbu. The trim_ends option cannot be specified together with the extend keyword.

- extend [by_factor] *ext_value*

An optional argument, specifying how far along the identified edge to extend or shrink the marker. Without the optional by_factor keyword, *ext_value* specifies a positive or negative extension in microns.

If the optional by_factor flag is specified, then *ext_value* must be greater than -0.5 and cannot be zero. The extension length is computed individually as the product of an edge’s length and *ext_value*.

- If the extension length is positive, the edge is extended from its endpoints by the computed value.

- If the extension length is negative, the edge is trimmed by the absolute value of the extension length on both ends. However, the edge trimming is adjusted if needed to preserve a minimum edge length of 3 dbu.
- **expand *value | value_in value_out***
A required argument, specifying the amount to expand the marked edges by.
 - Specifying a single value expands the corner inside and outside of the polygon.
 - You can specify different values for the expansion inside and outside the polygon using the second argument format (*value_in value_out*).

Examples

The following code selects any line edges less than .11 um, expanding them by .01 um:

```
setlayer ide1 = identify_edge TARGET length < .11 expand .01
```

The following more restrictive version of the code example detects mainly line ends and some jogs:

```
setlayer ide2 = identify_edge TARGET length < .11 length1 > .1 \
length2 > .1 expand .01
```

The following even more restrictive version will detect only line ends:

```
setlayer ide3 = identify_edge TARGET length < .11 length1 > .1 \
length2 > .1 corner1 convex corner2 convex expand .01
```

The following alternate version detects space ends:

```
setlayer ide4 = identify_edge TARGET length < .11 length1 > .1 \
length2 > .1 corner1 concave corner2 concave expand .01
```

Note

 The *Calibre MPCverify User's and Reference* reference page for [identify_edge](#) contains additional usage examples for the [SEM Box Correction](#) feature (described in the *Calibre nmMPC User's and Reference Manual*).

image

Calibre OPCverify operation

Creates an image contour.

Usage

Default (implicit) mode (optical setup specified by Calibre OPCverify [background](#) and [layer](#) keywords):

setlayer layername = image

```
[mask0 | mask 0] {optical modelname [dose d] [xshift x] [yshift y] [size sz]}
[ {mask1 | mask 1} {optical modelname [dose d] [xshift x] [yshift y] [size sz]}]
[filter filter_layer]
{aerial value | {resist_model resist_model_name [model_bounds_sitecenters]}}}
[etch_model etch_model_name]
```

Explicit mode (explicit specification of optical setup that overrides the default):

setlayer layername = image

```
[mask0 | mask 0] {optical modelname
[background bg_trans
[layer layername {asraf | {trans [ddm ddm_label]}}
[bias x_bias_microns y_bias_microns]]] ...
[dose d] [xshift x] [yshift y] [size sz]}
[ {mask1 | mask 1} {optical modelname [flare name]
[background bg_trans
[layer layername {asraf | {trans [ddm ddm_label]}}
[bias x_bias_microns y_bias_microns]]] ...
[dose d] [xshift x] [yshift y] [size sz]}]
[filter filter_layer]
{aerial value | {resist_model resist_model_name [model_bounds_sitecenters]}}}
[etch_model etch_model_name]
[topo_model topo_model_name]
[underlying {active | poly | finfet} [layer_name] ...]
```

Litho model mode (requires an [image_options](#) block to be defined):

setlayer layername = image image_options_name

```
[[maskN | mask N] [focus fhm] [dose d] [bias b] [xshift x] [yshift y]]...
[aerial {value | model} [thr_delta val[%]]]
[plane {zval_um | SRAF_POSITIVE | SRAF_NEGATIVE | TOPLOSS_PLANE}]
[no_etch] [no_shadow_bias]
[no_n2r] [no_n2e]
[no_flare | flare name] [slit_fixed_x val_um | slit_delta_x val_um]
[no_black_border | bbm_options name]
[stochastic val][stochastic_sigma positive_val][stochastic_seed val]
```

Description

The image command performs optical and resist simulations to generate contours.

- In implicit (default) mode, Calibre OPCverify “background” and “layer” keywords determine which layers get simulated, their optical transmission values, and the background optical transmission values.
- In explicit mode, the background transmission and all the layers to be simulated with their corresponding transmission values for each exposure must be specified.

The image command in default and explicit mode use the arguments listed under the section .

Note

 If there are no implicit image commands at all in the command file, Calibre OPCverify may generate warning messages when the [layer](#) command contains arguments other than the layer name.

- In litho model mode, the user must specify an [image_options](#) block, and also must use a set of files in [Litho Model Format](#).

The image command in litho model mode uses the arguments listed under “Default and Explicit Mode Arguments” in the Arguments section of this command.

Tip

 **Use as an Intermediate Command Only** - The image command’s contour output covers the full chip by default. It should not be used as SVRF input or saved as a GDS/OASIS file “as is” outside of Calibre OPCverify without some additional reduction of the results (such as using the [window](#) command to isolate an area, or other filtering-style commands that return only contours that meet certain criteria), as the large amount of data used by simulation contours can cause noticeable performance slowdowns.

Concurrency Behavior

Image commands that use the same optical models, background transmission, and simulated layers with the corresponding transmissions, xshift, yshift, and size values for every exposure are performed concurrently. This means that the default and explicit modes are not performed concurrently.

Image commands that use different [image_options](#) blocks might also be performed concurrently if the mask setups specified by the [image_options](#) blocks are sufficiently similar. Concurrency with different [image_options](#) blocks requires that the [image_options](#) blocks have the same filter, optical models, backgrounds, layers, layer transmissions, layer ddm models, layer shifts, and biases for every mask exposure.

If multiple image commands with flare options are set, the flare options (in the [flare_model_load](#) definition) must match exactly to be run concurrently. A single image command with flare options can be run concurrently with image commands without flare options.

CM1 contours are never run concurrently with VT5 contours.

For efficiency purposes, aerial contours are grouped with VT5 contours if no CM1 models are present. When CM1 models are present, aerial contours are grouped with CM1 contours for concurrency processing.

Arguments

Default and Explicit Mode Arguments

- mask0 | mask 0 | mask1 | mask 1

An optional argument declaring a mask. For double mask exposures, mask1 is a required separator indicating the beginning of mask1 options. Optical model names are required for each mask.

Starting with the 2013.2 release, both “maskn” (no space between the mask keyword and the mask number argument) and “mask n” (a space between the mask and mask number) syntax are accepted for the image command.

- Mask image options

- **optical modelname**

A required argument specifying the name of an optical model to be used for this image simulation. The modelname must have previously been defined using an [optical_model_load](#) setup command.

- **flare name**

An optional argument that specifies an EUV flare model to use in simulation. The *name* must have been previously defined using the [flare_model_load](#) command.

Note

 Flare model use requires you to include the EUV keyword in the [LITHO OPCVERIFY](#) statement (so that it appears as LITHO EUV OPCVERIFY). It also requires an additional license (caleuv) to function. Previous versions of the image command included the longrange flare layers at the end of the image command definition using the “flare_model” keyword. This layer list is now specified as part of the [flare_model_load](#) command, and the “flare” keyword is used to load the model instead, positioned right after the “optical” keyword. Using the old “flare_model” syntax now causes an error.

- **background bg_trans [layer layername {asraf | {trans [ddm ddm_label]}}] [bias x_bias_microns y_bias_microns]...**

An optional argument and keywords for explicitly overriding background transmission, the layers being simulated, and the corresponding layer transmission values. The ddm keyword indicates an optional domain decomposition model (previously loaded using the [ddm_model_load](#) setup command) to be associated with the layer.

Note

 If multiple layers are defined in an explicit syntax declaration for image and a DDM model is used, each visible layer must include a ddm keyword as part of its layer parameter declaration.

If the bias keyword is specified, vertical edges are biased by *x_bias_microns*, horizontal edges are biased by *y_bias_microns*, and all other edges are biased by $(x_bias_microns + y_bias_microns)/2$. Layer biasing is performed after the mask sizing specified by the size keyword.

If the asraf keyword is specified instead of the *trans* and *ddm* options, the layer is designated as an asraf layer, which is a negative SRAF layer. Negative SRAF layers contain positive features specifying the positions of negative SRAFs, which are holes cut out of the main mask features before simulation. If an asraf layer is specified, at least one positive feature layer must also be specified.

Note

 For asraf layers, positive values for bias or size make the shapes defining the negative SRAFs larger. This means that the holes created by the asraf cutouts will also be larger. However, if the negative SRAFs are included as holes in the main feature layer, increasing the bias value makes the holes smaller.

Specifying “bias” will cause Calibre OPCverify to clone all transformed placements of all cells.

- dose *d*

An optional keyword that specifies a dose for this image operation. It overrides the default dose specified in the layer command, if any.

- xshift *x* yshift *y*

An optional keyword pair that shifts the geometric data on the mask by the given *x* and *y* coordinates. If these parameters are specified, Calibre OPCverify will clone all rotated or reflected cell transforms. The default value for these parameters is 0.

- size *sz*

An optional parameter that sizes the geometric data on the mask by the specified amount. The default value is 0.

One of either **aerial** and **resist_model** is required.

- **aerial value**

An argument that generates an aerial image threshold at *value* to produce the output contour. Use this command to perform critical feature detection.

Note

 The deprecated argument “aerial_contour” is also allowed as a synonym for “aerial”.

- **resist_model resist_model_name**

An argument that generates a printimage contour using the specified resist model. Use this argument to perform CD-checking applications only.

resist_model_name should be the name of a resist model specified earlier with the **resist_model_load** command.

- **filter filter_layer**

An optional argument that limits the image contours to be inside the specified filter layer only.

- **model_bounds_sitecenters**

An optional argument that can be specified accompanying the **resist_model** argument. When specified, Calibre OPCVerify draws 2 dbu boxes around the centers of sites when the center of the site value falls outside of the calibrated model bounds. Only used for VT5 models.

- **etch_model etch_model_name**

An argument that generates a printimage contour using the specified etch model. This argument is only used with variable etch bias (VEB) models as part of the model-based retargeting flow.

Note

 The use of VEB models containing visibility kernels in the **image** command and in **veh_simulate** can increase runtime significantly. The larger the visibility kernel diffusion length, the longer the runtime. The recommended diffusion length of a visibility kernel is less than 0.8μm. Only one visibility kernel should be run at a time.

- **topo_model topo_model_name**

An optional argument that can only be used in explicit mode.

This argument specifies a model previously loaded using the **topo_model_load** command or defined inside a litho model (see [Litho Model Format](#)). Using this option instructs Calibre OPCVerify to use topo model calculations from the calibrated topo model in the image generation. Using this option requires you to also specify the “underlying” argument on the same line.

Using topo models adds a noticeable runtime increase to image contour generation.

- underlying {active | poly | finfet} [*layer_name*] ...

An optional argument associated with topo models, specifying the type of underlying feature layer and a list of one or more layers to assign the underlying type to.

Litho Model Mode Arguments

Note

 If you use litho model mode, the [layer](#) commands defined earlier in the file disregard any commands other than the layer and layer name defined. Calibre OPCVerify generates a warning messages for other options specified in the layer command.

- *image_options_name*

A required argument specifying the name of the *image_options* block defining the masks to be used.

- mask*N* | mask *N*

An optional argument specifying a mask number as defined in the litho model file to which the following parameters apply. If not specified, mask 0 is assumed. The layers which comprise this mask are defined by layer statements in the [image_options](#) block.

Note

 A space between the “mask” keyword is the preferred syntax in order to match the litho model syntax.

- focus *fnm*

An optional argument defining a focus condition to be used for this mask. This argument maps to the optical model defined by “optical *file* mask *n* focus *fnm*” in the litho model.

- dose *d*

An optional argument specifying the dose to be used for this mask. The default is 1.0.

Note

 If a base dose for this mask was specified in the litho model file, the actual dose used is treated as a relative dose, and is formed by *multiplying* this value with the base dose in the litho model.

- bias *b*

An optional argument specifying a bias in microns to be used for sizing this mask before using it in simulation.

Note

 If any biases were specified for individual *mask_layer*'s in the litho model, those biases are treated as a relative bias, and are *added* to the value specified here.

- `xshift x`
`yshift y`

An optional argument pair in microns causing all the geometric data on a given mask to be shifted by x,y. If specified, these options will cause Calibre to clone all rotated or reflected cell transforms. The default is 0.

Note

 If any shifts were specified for individual mask_layers in the litho model file, they are treated as relative shift values, and are *added* to the values specified here.

Tip

 See the description under [Litho Model Format](#) for the order in which geometry preprocessing is applied before simulation.

- `aerial value | model`

An optional argument that specifies an aerial image threshold value to produce the output contour. This option overrides the resist model specified in the Lithomodel file. If “model” is specified, the Lithomodel file keyword “image_threshold” value is used for *value*.

Note

 The deprecated keyword “aerial_contour” is also allowed as a synonym for “aerial”.

- `thr_delta val[%]`

An optional argument which is only used with “aerial model”. It specifies a signed offset to modify the “image_threshold” parameter in the Lithomodel file. If the % sign is present, the offset is used as a relative percentage. If the % sign is not present, the offset is used as an absolute delta.

- `plane {zval_um | SRAF_POSITIVE | SRAF_NEGATIVE | TOPLOSS_PLANE}`

An optional keyword used for 3D resist imaging and specifies the z-position within the resist film where the image is computed. This keyword is permitted only if the optical model is 3D-enabled (using the imageplanes keyword). This also requires either a CM1-3D resist model or an aerial threshold. The value is relative to 0.0 at the bottom of the resist, and must be ≥ 0.0 and \leq the resist film thickness. If the plane keyword is not specified, the default is the usual non-3D convention (use the base focus plane from the optical model).

The keywords SRAF_POSITIVE and SRAF_NEGATIVE specify special planes.

SRAF_POSITIVE is used to model printing of positive SRAFs. SRAF_NEGATIVE is used to model printing of negative SRAFs. If these keywords are used, the Litho Model must contain a ZPlanes model, and the actual plane value is extracted from that model.

The keyword TOPLOSS_PLANE is used to model resist toploss, and also uses a ZPlanes model.

For more information on ZPlanes, see “[ZPlanes Model Format](#)” on page 118.

- **no_etch**
An optional argument. If the litho model file contains an etch model, this keyword disables using the etch model for this image command.
- **no_shadow_bias**
An optional argument. If the litho model file contains a shadow bias model file, this keyword disables using the shadow bias model for this image command.
- **no_n2r**
An optional keyword that disables the N2R (Neural Network Resist) model in the litho model if one is present. It has no effect if the litho model does not contain an N2R model.
- **no_n2e**
An optional keyword that disables the N2E (Neural Network Etch) model in the litho model if one is present. It has no effect if the litho model does not contain an N2E model.
- **no_flare**
An optional argument. If the litho model contains an EUV flare model, this keyword disables using the flare model for this image command.
- **flare *name***
An optional argument that specifies that a named [flare_longrange](#) specification for this litho model should be used instead of the default unnamed specification. This option provides the capability to vary long-range flare settings on a per-image basis.
- **slit_fixed_x *val_um***
An optional argument. If the litho model contains an EUV through-slit model, this parameter can be specified to indicate that all imaging should use a fixed set of models across the entire design. The models used are those which would normally be selected for the slit position *slit_fixed_x* (in microns relative to the center of the EUV field). This parameter can not be specified with *slit_delta_x*.
- **slit_delta_x *val_um***
An optional argument. If the litho model contains an EUV through-slit model, this parameter can be specified to indicate that all through-slit model locations should be displaced by *val_um* (in microns) from the slit locations specified in the litho model. This parameter cannot be specified with *slit_fixed_x*.
- **no_black_border**
An optional argument. If the litho model contains a black border model, this keyword disables using the black border model for this image command.
- **bbm_options *name***
An optional argument. If a named *bbm_options* block is specified, the black border model parameters are overridden inside the litho model. This option provides the capability to vary black border effect simulation on a per-image basis.

For more information, see the [bbm_options](#) command.

- **stochastic {low | high | rand}**

An optional parameter used with an EUV stochastic model. The accepted values are “low” or “high” corresponding to lower or higher stochastic bounds, respectively, and “rand”, which generates a random value from possible stochastic noise values.

- **stochastic_sigma *positive_val***

An optional parameter used with an EUV stochastic model, specifying the confidence interval. It must be a multiple of the standard deviation (sigma). This option is ignored when “stochastic rand” is specified.

- **stochastic_seed *val***

An optional parameter that sets the random seed for the “stochastic rand” parameter in order to ensure the same results across runs. If not specified, a true random seed is used.

Examples

Default Mode

The following example creates an aerial image contour using the optical model named o1, an override dose of 0.96, and a threshold value of 0.3:

```
setlayer i1 = image optical o1 dose 0.96 aerial 0.3
```

The following example code creates a double exposure aerial contour usable with an alternating phase shift mask:

```
setlayer i2 = image optical o1 mask1 optical o2 aerial 0.18
```

Explicit Mode

Given the following setup definitions:

```
modelpath ./models:$env(DESIGN_DIR)/models
#####
psm models
optical_model_load    o1      pc30d
optical_model_load    o2      pc30d2nd
resist_model_load     r1      resist
#####
single exposure
optical_model_load   mo1   ment_45nm_attPSM
resist_model_load    r2    cm1_d1f1
resist_model_load    r3    cm1_d1f2
```

and the following input layers from the SVRF rule file:

```
background clear dark
layer PH0OPC    visible clear  1 1.0
layer PH18OPC   visible phase180 1 1.0
layer BLOCKOPC  visible dark   0 0.86
# short layer syntax. M1 and M1OPC default to hidden.
layer M1
layer M1OPC
##### imagegrid base setting
imagegrid 0.02
```

and the following intermediate layer:

```
setlayer bsz = size BLOCKOPC by 0.1
```

the following command uses the default optical setup (no background statement is specified):

```
# default optical setup
setlayer dpi = image optical o1 mask1 optical o2 resist_model r1
```

but the following commands use different optical setups (background statement is specified):

1. This image command uses the intermediate layer bsz (the sized up BLOCKOPC) as declared previously, along with the other phase shift masks as the mask1 parameters.

```
#redefine input layer
setlayer dpi_cc = image optical o1 background clear layer \
    bsz dark dose 0.86 mask1 optical o2 background dark \
    layer PH0OPC clear layer PH18OPC phase180 resist_model r1
```

2. This image command uses the input layer M1OPC, but with new transmission settings (using the real and imaginary syntax).

```
#redefine optical setup to do a different physical layer
set sim2 "optical m01 background clear layer M1OPC -1.245 0.0"
# sim2 can be reused in multiple image commands (see below)
eval setlayer c1 = image $sim2 resist_model mr1
eval setlayer c2 = image $sim2 resist_model mr2
eval setlayer c3 = image $sim2 aerial 0.12
```

In this example, the dpi operation can not run concurrently with the dpi_cc operation or the c3 operation, because their input layers are not the same. The c1 and c2 operations are run concurrently with the c3 operation because the c1 and c2 operations use a CM1 resist model, and the c3 operation uses an aerial contour.

DDM Models in Explicit Mode

The following example shows the correct way to explicitly specify multiple layers when DDM models are part of the optical model. Notice that each layer statement has its own ddm keyword.

```
setlayer img1 = image mask0 optical optcname background dark
layer layer1 clear ddm ddm_name
layer layer2 clear ddm ddm_name
resist_model resistname
```

Flare Models

The following example shows concurrency with flare models:

```
setlayer f1_ac = image optical o1 flare fm1 aerial 0.2
setlayer f2_ac = image optical o1 aerial 0.2
setlayer f3_ac = image optical o1 flare fm2 aerial 0.2
```

The layers f1_ac (a flare layer) and f2_ac (a non-flare layer that has matching optical and aerial contour settings) will be run as a concurrent group. The f3_ac layer is run separately because it has a different number of flare layers than f1_ac in its flare definition (not shown).

Litho Model Example 1

A simple image command using a basic litho model file. The [image_options](#) block is defined first, and includes a filter layer and two visible layers. It looks for the litho model file in the directory 32nmPOLY.

```
image_options opt1 {
    layer poly.main visible mask_layer 0
    layer poly.sraf visible mask_layer 1
    layer filter_lay filter
    litho_model 32nmPOLY # This is the litho model
}
```

The image command uses the opt1 image_options block for the layers to simulate. Layer b applies a dose and focus value; layer c applies a bias value.

```
setlayer a = image opt1
setlayer b = image opt1 dose 1.1 focus -20nm
setlayer c = image opt1 bias 0.002
```

Litho Model Example 2

This example shows how to shift one mask in double exposure:

```
image_options A {
    layer mainfeature visible mask 0 mask_layer 0
    layer mainfeature2 visible mask 1 mask_layer 0
    litho_model 32nmPOLY
}
setlayer x = image A
setlayer a = image A mask0 xshift .002
```

imax_check

Verification command

Analyzes the aerial image intensity from an image command and compares it against an imax constraint.

Usage

```
imax_check {image_layer | image_set_name}...
    constraint
    gauge_set set_name
    [markers {gauge [width width_microns] | trim_ends trim_microns}]
    [invalid_gauges {keep | suppress}]
    [invalid_property prop_val]
    # error-centric section
    [property {'imax [image]
    '}' [classify, limit, or histogram block]
    [add_properties block]
    [pinpoint_output block]]]
```

Description

The imax_check analyzes the aerial image intensity from an [image](#) command.

- For dark-field exposures, the check investigates regions inside the narrow lines.
- For clear-field exposures, the check investigates the narrow spaces.

This check investigates regions between facing pairs of parallel Manhattan or 45 degree target edges, where the edges are closer than a certain user-specified distance (for example, 1.5 * [critical_dimension](#)). The check does not investigate corner-to-corner interactions or skew edges.

When an appropriate edge pair is identified, a set of gauges is generated. The gauges are perpendicular to the edges, and spaced at user-specified intervals (by default 0.015 um) along the centerline of the edge pair. The interval is specified in the [gauge_set](#) block with the “gauge_spacing” argument. The length of each gauge is the spacing between the edges plus a small overlap amount.

The check places a set of gauges that span the lines or spaces in the regions of interest. For each gauge, the maximum intensity Imax is computed.

- If Imax matches the check constraint, an error marker with an optional Imax property is generated at that location. The valid range of the Imax property is greater than or equal to 0, up to a maximum of approximately 1.0, and is dependent on the simulation transmissions, optical models and dose values.
- If a valid Imax cannot be determined (such as when Imax is at the end of the gauge), Imax is set to the “invalid_property” value (default is -1.0) and an error marker with that property will always be generated, irrespective of the constraint.

Arguments

- ***image_layer / image_set_name***

A required argument specifying one or more images whose intensity will be analyzed. The **image** commands used to generate the images must have been specified with an aerial threshold. The actual threshold value is ignored for this check. If multiple images are specified, the check reports the worst errors among all images. The **image_layer** names and **image_set_names** can be mixed freely. Specifying an **image_set_name** is the same as specifying all images in the set.

- ***constraint***

A required argument specifying a constraint. The constraint may be a single constraint or a bounded (2-sided) one. For an imax_check, a typical error constraint would be “ $\leq value$ ”. The valid range of imax values is 0.0 to approximately 1.0.

A small imax value is considered a bad value, so constraints closer to 0.0 that give some errors but not too many errors are ideal.

(See “[Constraints](#)” on page 67 for more information.)

- ***gauge_set set_name***

A required argument specifying a **gauge_set** definition block. The command uses the parameters specified in the gauge_set block to place gauges.

Note



A gauge_set definition does not specify whether the gauges are internal (across the lines) or external (across the spaces). For imax_check, dark field images are analyzed with internal gauges and clear field images are analyzed with external gauges.

- **markers {gauge [width *width_microns*] | trim_ends *trim_microns*}**

An optional argument specifying the style of error markers to generate.

The default is “trim_ends 0”. In this case, the error markers generated are rectangles bounded by the edge pair, and spanning the entire space between gauges. Markers are merged into a single block when adjacent gauges have errors.

- **gauge** — Sets the markers to be a narrow rectangles whose width is *width_microns* and length is the extended length of the gauge. These markers provide a visual representation of exactly how the gauges were placed. If the width keyword is omitted, the default width is 12 dbu. If *width_microns* is specified less than 12 dbu, it will be ignored and 12 dbu will be used.
- **trim_ends** — Sets the markers as rectangles bounded by the edge pair, trimmed by *trim_microns* along the target edge when the marker box is longer than $2 * trim_microns + 2$ dbu along the target edge. (This implies that $2 * trim_microns$ must be smaller than *gauge_spacing - 2dbu*.) This option causes each gauge to have its own separate marker and property value.

Tip

 To see how all gauges are placed, run a check with an unrealistically tight constraint, so that every gauge will be an error, and use “markers gauge”.

- invalid_gauges {keep | suppress}

An optional argument that sets how invalid gauges are handled. Invalid gauges occur when no valid Imax value could be found, such as when the intensities along the gauge do not pass through a maximum.

keep — Invalid gauges produce an error marker, and are assigned the special Imax property value specified in the invalid_property argument. This is the default setting.

suppress — Invalid gauges are ignored and do not produce markers.

- invalid_property *prop_val*

An optional argument that specifies an arbitrary value to be assigned to invalid Imax results. Invalid results always generate markers with this property, even if the specified **constraint** was met.

The default value is -1.0, and *prop_val* must be ≤ -1.0 .

- property {imax | [image]}

An optional keyword that, if imax is specified, places an Imax property on each error marker. If image is specified, an image property is generated, specifying the name of the input image with the worst value.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Related Topics

[gauge_set](#)

[imin_check](#)

imin_check

Verification command

Analyzes the aerial image intensity from an image command and compares it to an imin constraint.

Usage

```
imin_check {image_layer | image_set_name}...
    constraint
    gauge_set set_name
    [markers {gauge [width width_microns] | trim_ends trim_microns}]
    [invalid_gauges {keep | suppress}]
    [invalid_property prop_val]
    # error-centric section
    [property {'imin
    '}' [classify, limit, or histogram block]
    [add_properties block]
    [pinpoint_output block]]]
```

Description

The imin_check analyzes the aerial image intensity from an [image](#) command.

- For clear-field exposures, the check investigates regions inside the narrow lines.
- For dark-field exposures, the check investigates the narrow spaces.

The imin_check investigates regions between facing pairs of parallel Manhattan or 45-degree target edges, where the edges are closer than a certain user-specified distance (by default, 1.5 * [critical_dimension](#)). The check does not investigate corner-to-corner interactions or skew edges.

When an appropriate edge pair is identified, a set of gauges is generated, using the definitions specified in a [gauge_set](#) block.

The generated gauges are perpendicular to the edges, and are spaced at intervals along the centerline of the edge pair. The interval is specified by the [gauge_set](#) “[gauge_spacing](#)” parameter. The length of each gauge is the space between the edges plus a small overlap amount.

A set of gauges is placed to span the lines or spaces in the regions of interest. For each gauge, the minimum intensity Imin is computed.

- If Imin matches the check constraint, an error marker with an optional imin property is generated at that location. The valid range of imin property values is 0 up to a maximum of approximately 1.0 and depends on the simulation transmissions, optical models, and dose values.

- If a valid Imin cannot be determined (such as when Imin is at the end of the gauge), it is set to the “invalid_property” value (default is 10.0) and an error marker with that property is generated, irrespective of the constraint.

Arguments

- ***image_layer / image_set_name***

A required argument specifying one or more images whose intensity will be analyzed. The **image** commands used to generate the images must have been specified with an aerial threshold. The actual threshold value is ignored for this check. If multiple images are specified, the check reports the worst errors among all images. The ***image_layer*** names and ***image_set_names*** can be mixed freely. Specifying an ***image_set_name*** is the same as specifying all images in the set.

- ***constraint***

A required argument specifying a constraint. The constraint may be a single constraint or a bounded (2-sided) one. For an imin_check, a typical error constraint would be “ $\geq value$ ”. The valid range of imin values is 0.0 to approximately 1.0.

A large imin value is considered a bad value. Use constraint values closer to 1.0 that give some errors but not too many.

(See “[Constraints](#)” on page 67 for more information.)

- ***gauge_set set_name***

A required argument specifying a **gauge_set** definition block. The command uses the parameters specified in the gauge_set block to place gauges.

Note



A gauge_set definition does not specify whether the gauges are internal (across the lines) or external (across the spaces). For imin_check, clear field images are analyzed with internal gauges and dark field images are analyzed with external gauges.

- **markers {gauge [width *width_microns*] | trim_ends *trim_microns*}**

An optional argument specifying the style of error markers to generate.

The default is “trim_ends 0”. In this case, the error markers generated are rectangles bounded by the edge pair, and spanning the entire space between gauges. Markers are merged into a single block when adjacent gauges have errors.

- **gauge** — Sets the markers to be narrow rectangles whose width is *width_microns* and length is the extended length of the gauge. These markers provide a visual representation of exactly how the gauges were placed. If the width keyword is omitted, the default width is 12 dbu. If *width_microns* is specified less than 12 dbu, it will be ignored and 12 dbu will be used.
- **trim_ends** — Sets the markers to be rectangles bounded by the edge pair, trimmed by *trim_microns* along the target edge when the marker box is longer than $2 * trim_microns + 2$ dbu. (This implies that $2 * trim_microns$ must be smaller than

gauge_spacing - 2dbu.) This option causes each gauge to have its own separate marker and property value.

Tip

 To see how all gauges are placed, run a check with an unrealistically tight constraint so every gauge will be an error, and use “markers gauge”.

- invalid_gauges {keep | suppress}

An optional argument that sets how invalid gauges are handled. Invalid gauges occur when no valid Imin value could be found, such as when the intensities along the gauge do not pass through a minimum.

keep — Invalid gauges produce an error marker, and are assigned the special Imin property value specified in the invalid_property argument. This is the default setting.

suppress — Invalid gauges are ignored and do not produce markers.

- invalid_property *prop_val*

An optional argument that specifies an arbitrary value to be assigned to invalid Imin results. Invalid results mean that a minimum was not found within the gauge. Invalid results always generate markers with this property, even if the specified **constraint** was met.

The default value is 10.0, and *prop_val* must be greater than or equal to 10.0.

- property {imin}

An optional argument that specifies that an Imin property is placed on each error marker.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Related Topics

[gauge_set](#)

[imax_check](#)

intensity_ilscheck

Verification command

Computes and checks the image log slope (ils) for an image previously defined by setlayer image.

Usage

```
intensity_ilscheck {image_layer | image_set_name}...
    constraint
        [measure_at {target | curved_target | contour}]
        [target_layer target_lay [[not] {inside | outside} filter_lay
            [max_search search_dist_um]]]
        [curved_target curved_lay]
        [use_intensity {aerial | cm1}]
        [gauge_spacing dist_um]
        [markers [{gauge [width width_um] | trim_ends trim_um}] [height height_um]]
    # error-centric section
    [property {ils [image]} [classify, limit, or histogram block]
        [add_properties block]
        [pinpoint_output block]]]
```

Note

 The image name(s) and constraint must be coded in the order shown; the remaining arguments can be specified in any order.

Description

The intensity_ilscheck command computes and checks the image log slope (ils) for an image previously defined by setlayer image. In one dimension, ils at point x is defined in terms of the intensity I(x) as $\text{ils} = \text{abs}((1/I) * (dI/dx)) = \text{abs}(d(\ln(I))/dx)$. The standard convention is to measure ils at target edges. It is desirable that ils be as large as possible. There is no standard definition of ils for arbitrary 2D images. The intensity_ilscheck command defines ils as $\text{abs}((1/I) * G_{\text{max}})$, where G_{max} is the steepest-descent intensity gradient at the point being measured. The range of I is from 0.0 to some maximum value on the order of 1.0 depending on image dose. The distance units for the gradient are 1/microns. Results are computed at every measurement point.

Arguments

- ***image_layer|image_set_name***

A required argument specifying one or more images whose intensity ils are checked. The image layers must be unique direct outputs of setlayer image. The [image](#) commands used to generate the images must be specified with an aerial threshold or resist threshold; an etch model can not be specified. Additionally, any non-CTR VT5 resist model is not permitted in the setup file, as it will result in a runtime error.

If multiple images are specified, the check reports the worst errors among all images. The *image_layer* names and *image_set_names* can be mixed freely. Specifying an *image_set_name* is the same as specifying all images in the set.

Note



If “measure_at contour” is specified, only a single image can be checked.

- **constraint**

A required argument specifying a constraint. The constraint may be a single constraint or a bounded range. For intensity_ilscheck, a typical error constraint would be “ $\leq value$ ”.

- **measure_at {target | curved_target | contour}**

An optional argument specifying where the ils measurements are made. The default keyword is “target”, which specifies that the measurements are made at intervals along the target layer edges. The “curved_target” keyword specifies that measurement are made at intervals along a curved target layer. “contour” specifies the measurements are made at intervals along the edges of the input contour for the setlayer check.

- **target_layer target_lay [[not] {inside|outside} filter_lay [max_search search_dist_um]]**

An argument specifying the target layer for the image being analyzed. Specifying a target layer is required if “measure_at target” is specified, or if a filter layer is used; otherwise it is not permitted.

Note



By default, measure_at is “target”, which requires specifying target_layer.

A filter layer can be specified so that ils measurements are only taken within the region permitted by the filter. If “measure_at curved_target” or “measure_at contour” is specified, the filter layer is first applied to the original target and then the target edges, which are then mapped to the edges of the “measure_at” layer. Thus, ils measurements are taken within the regions specified by the “measure_at” layer, which were mapped to the target. The maximum distance from a target edge to a contour edge for which they can be mapped to each other is specified by *search_dist_um*. This value is not permitted when “measure_at target” is specified; otherwise, it is required.

- **curved_target curve_lay**

An argument specifying a curved version of the target layer. Specifying a curved target layer is required if “measure_at curved_target” is specified; otherwise it is not permitted. Generating the curved target layer from the target using setlayer curve is recommended.

Note



Curved target layers must be generated in the setup file before any image commands are specified to avoid negative impacts on performance.

- `use_intensity {aerial | cm1}`

An optional argument specifying the intensity grid on which intensity gradients are computed. The default keyword is “aerial”, which specifies to use the aerial image intensity for the input contour of the setlayer check regardless if the setlayer image command includes a resist model. The “cm1” keyword is only permitted if the setlayer image command specifies a resist model, and specifies to use the intensity grid output by the CM1 resist model.

Note

 If “use_intensity cm1” is specified, the “measure_at” setting must be “contour”, as the CM1 intensity is only physically meaningful close to the contour.

- `gauge_spacing dist_um`

An optional argument set that defines the spacing interval between the measurement gauges on the target or contour. The value must be ≥ 0.005 and ≤ 0.015 , and the default is 0.0125.

- `markers [{gauge [width width_um] | trim_ends trim_um}] [height height_um]`

An optional argument that specifies the type of error markers to generate. The default is “trim_ends 0”, which specifies that the error markers generated are rectangles whose width (along the edge) is `dist_um` specified by the `gauge_spacing` argument, and whose height (perpendicular to the edge) is `height_um` (default is 0.015 μm). When adjacent measurement points have errors, markers are merged into a single block. If a `trim_ends` value is > 0 , the error markers around each point are trimmed by `trim_um` microns (on each side) along the edge, given that the marker box is longer than $2 * \text{trim_um} + 2\text{dbu}$ along the edge. This implies that $2 * \text{trim_um}$ must be smaller than `dist_um - 2dbu`. This option results in each measurement point having its own marker.

If “gauge” is specified, the markers are narrow rectangles. Their width is `width_um` microns (default 12 dbu, and is forced to 12 dbu if specified as anything less) and their length is `height_um` (default 0.015 um). The markers are oriented along the steepest-descent direction of the intensity at each measurement point, which provides a visual representation of exactly how ILS was computed.

Tip

 To see how all measurement points are placed, use “markers gauge” and run a check with an unrealistically tight constraint that will result in an error for every point.

- `property {ils [image]}`

An optional argument that attaches a property to the error markers. If “property ils” is specified, an “ils” property is placed on each error marker. If multiple images are being analyzed, the “ils” property is the worst (smallest) value over all images.

If “image” is also specified, the name of the input image with the worst ils value is attached to the error marker. The “image” property must be specified after “ils”.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

intensity_meefcheck

Computes and checks Mask Error Enhancement Factor (MEEF) for an image contour.

Usage

```
intensity_meefcheck image_layer constraint
  [delta_mask delta_mask_um] [cntr_max_search search_dist_um]
  [exceptions_also [exception_meef value]]
  [target_layer target_layer [not] {inside | outside} filter_layer max_search search_dist_um]
  [gauge_spacing dist_um]
  [markers [{gauge [width width_um] | trim_ends trim_um}] [height height_um]]
  #error-centric section
  [property '{' meef '}' [classify, limit, or histogram block]]
  [add_properties block]
  [pinpoint_output block]]
```

Description

The intensity_meefcheck command measures MEEF by comparing two intensity grids, one for the original image and one created by biasing the mask by a specified amount. MEEF is dimensionless, and is defined as (contour edge delta)/(mask edge delta) with respect to a small specified displacement of the mask edges. It is positive if the contour edges move in the same direction as the mask edges, and negative otherwise. Negative MEEF is very rare, and usually indicates a bad resist model.

The command makes measurements along gauges perpendicular to the ideal contour of the main image, and finds the threshold crossings for both intensity grids.

The intensity_meefcheck command is expected to be more accurate than the [meefcheck](#) command, because it is based entirely on intensity interpolation. Its accuracy is not affected by contour edge segment approximations or snapping of contour edges to the design dbu.

Arguments

- ***image_layer***

A required argument specifying an image contour layer to check MEEF on. The image layer has the following requirements:

- The image layer must be a direct output of an [image](#) command; it may not be a copy.
- The image must have been generated using either an aerial threshold or a resist model. It must not have been generated using an etch model.
- intensity_meefcheck is not supported if a non-CTR VT5 resist model is used in the setup file. This will cause a runtime error.

Note

 The image layer is used only as a guide to where to make measurements. The accuracy of the contour does not have a strong effect on the MEEF accuracy. However, the accuracy of the intensity interpolation is important. This command therefore ignores the user's [contour_options](#) interp_algo setting (if any; the default is cubic), and uses a mixture of a 5th and 7th degree Lagrange interpolation instead.

- **constraint**

A required argument specifying a standard constraint. This can be a single constraint or a bounded (2-sided) one. For intensity_meefcheck, a typical error constraint would be “ \geq value”.

- **delta_mask delta_um**

An optional argument specifying the positive mask edge displacement (bias) to be used to create the second intensity grid. The default is 0.002, and *value* may not be less than 0.001. Setting excessively large values may degrade performance and accuracy.

- **cntr_max_search search_dist_um**

An optional argument specifying the maximum distance from the main contour that Calibre OPCVerify can search to find an edge of the second contour. The default is $10 * \text{delta_mask_um}$, which supports MEEF values up to 10.

cntr_max_search should be substantially less than the minimum feature size of the image; otherwise incorrect results can occur.

- **exceptions_also [exception_meef value]**

An optional argument that toggles the inclusion of exceptions in the output. By default, when MEEF cannot be measured because the second image is outside *cntr_max_search*, that measurement point is suppressed (no results are generated). If “*exceptions_also*” is specified, then those measurement points are reported with an artificial MEEF property *value*. The default value is 10000.

- **target_layer target_layer [not] {inside | outside} filter_layer max_search search_dist_um**

An optional argument specifying a target layer for the image being analyzed. For intensity_meefcheck, *target_layer* is used solely as a means of specifying a filter. Using this argument therefore also requires a filter layer.

The filter layer is first applied to the target, then the target edges that were not filtered out are mapped onto the contour of the main image. MEEF measurements are done only for the regions of the contour which were successfully mapped to the target.

search_dist_um specifies the maximum search distance from a target edge to find a nearby contour edge that can be mapped to the target.

- **gauge_spacing *dist_um***
An optional argument that defines, in microns, the spacing interval between the measurement points on the target or contour. Must be ≥ 0.005 and ≤ 0.015 . The default is 0.0125.
- **markers [{gauge [width *width_um*] | trim_ends *trim_um*}] [height *height_um*]**
An optional argument that sets the appearance of generated error markers as either gauge markers or trimmed boxes, with the height parameter as an option to either mode.
 - If gauge is specified, the markers appear as narrow rectangles with a width of *width_um* (default is 12 dbu, with an enforced minimum of 12dbu) and a length of *height_um* (default is 0.15um). This output option places markers oriented along the measurement direction.
 - If trim_ends is larger than zero, error markers around each point are trimmed by the specified value in microns along each edge whenever the marker box is longer than $(2 * \text{trim_um} + 2\text{dbu})$ along the edge. This output option causes each error to have its own separate marker and property.
 - The default is trim_ends 0, which creates rectangles with a width (along the edge) of gauge_spacing, and a height (perpendicular to the edge) of *height_um*. This output option merges multiple adjacent errors into a single block.
- **property '{' meef '}'**
An optional argument that places a property “meef” on the error markers with the MEEF value for that marker.
- **classify, limit, or histogram block**
An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.
- **add_properties block**
An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.
- **pinpoint_output block**
An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Examples

The following example tests the MEEF values for cntr0, returning any parts of the contour with a large MEEF. It writes gauge markers using the default size and adds a “meef” value to the error markers. If a MEEF value was not found for a marker due to a missing second contour, an artificial value of 10000 is stored for meef, because “exceptions_also” has been specified.

```
setlayer meef1 = intensity_meefcheck cntr0 > 2.5 markers gauge
exceptions_also property { meef }
```

intensity_nilscheck

Verification command

Computes and checks the normalized image log slope (nils) for an image previously defined by setlayer image.

Usage

```
intensity_nilscheck {image_layer | image_set_name}...
    constraint
        [measure_at {target | curved_target | contour}]
        [target_layer target_lay [[not] {inside|outside} filter_lay
            [max_search search_dist_um]]]
        [curved_target curved_lay]
        [use_intensity {aerial | cm1}]
        [gauge_spacing dist_um]
    normalize_to {width | space}
        [max_local_cd val_um]
        [fix_small_cds {yes | no}]
        [critical_dimension val_um]
        [min_separation_fac fac]
        [cd_angle_range_degs degrees]
        [[markers [{gauge [width width_um] | trim_ends trim_um}] [height height_um]]]
    # error-centric section
        [property {nils [local_cd] [image]} [classify, limit, or histogram block]
        [add_properties block]
        [pinpoint_output block]]]
```

Note

 The image name(s) and constraint must be coded in the order shown; the remaining arguments can be specified in any order.

Description

The intensity_nilscheck command computes and checks the normalized image log slope (NILS) for an image previously defined by setlayer image. In one dimension, NILS at point x is defined in terms of the intensity I(x) as $\text{NILS} = w * \text{abs}((1/I) * (dI/dx)) = w * \text{abs}(d(\ln(I))/dx)$, where w is the 1-dimensional half-pitch (the line or space CD). The normalization by CD is essentially heuristic and its only purpose is to emphasize that small features are more important than large ones. The standard convention is to measure nils at target edges. It is desirable that nils be as large as possible.

There is no standard definition of NILS for arbitrary 2D images. The intensity_nilscheck defines nils as $w * \text{abs}((1/I) * G_{\text{max}})$, where G_{max} is the steepest-descent intensity gradient at the point being measured. The range of I is from 0.0 to some maximum value on the order of 1.0 depending on image dose. The NILS value is dimensionless (units $\mu\text{m}/\mu\text{m}$). The variable "w" is

the local line width or space width at each measurement point (either definition may be used). The width is defined as the distance from the measurement point to the closest facing polygon edge, measured across a small angular range around the intensity gradient steepest descent direction. If a local CD is not found at a measurement point, or exceeds “max_local_cd”, or is invalid with respect to the “distance around loop” criterion described below, the measurement point is skipped (no result is generated).

Issues with Local CD Computation on Contours and Curved Targets

Since an optimal method for computing a local CD on a 2D contour is not well-defined, the additional arguments fix_small_cds, min_separation_fac and cd_angle_range_degs are available for tuning the CD computation. These are advanced features and Siemens EDA support can advise on their use.

The min_separation_fac argument is used to prevent computation of unrealistically small CDs when the angular range being investigated hits the same polygon contour very close to the measurement point (separation-on-loop filtering). If this parameter is set too small, unrealistically small CDs may be reported. If it is set too large, it results in valid measurement points being dropped because no acceptable CD is found. For example, in the case of a circular contact contour, setting min_separation_fac > 1.57 ($\pi/2$) rejects all valid CDs. The default value of 1.3 is reasonable for most usage.

The cd_angle_range_degs argument controls searching for local CD in an angular range around a line along the intensity steepest descent direction, as opposed to just looking along this line. This can be important when computing CDs between pairs of facing line ends or space ends, because in this case the line along the gradient may not be pointing in exactly the correct direction to intersect the contour edges of the facing feature. Therefore it is necessary to look over a wider angular range to have reasonable confidence that a valid CD will be found. Increasing this from the default of 7.5 degrees may help to resolve missing data points on low-quality images.

In general, it is recommended that line ends are excluded from nils width checks (using filter layers), and that space ends are excluded from nils space checks, because in these cases the computed CDs are measured along the “long direction” of the feature and are not useful.

Arguments

- ***image_layer | image_set_name***

A required argument specifying one or more images whose intensity nils are checked. The image layers must be unique direct outputs of setlayer image. The [image](#) commands used to generate the images must be specified with an aerial threshold or resist threshold; an etch model can not be specified. Additionally, any non-CTR VT5 resist model is not permitted in the setup file, as it will result in a runtime error.

If multiple images are specified, the check reports the worst errors among all images. The *image_layer* names and *image_set_names* can be mixed freely. Specifying an *image_set_name* is the same as specifying all images in the set.

Note



If “measure_at contour” is specified, only a single image can be checked.

- **constraint**

A required argument specifying a constraint. The constraint may be a single constraint or a bounded range. For intensity_nilscheck, a typical error constraint would be “ $\leq value$ ”.

- **measure_at {target | curved_target | contour}**

An optional argument specifying where the nils measurements are made. The default keyword is “target”, which specifies that the measurements are made at intervals along the target layer edges. The “curved_target” keyword specifies that measurement are made at intervals along a curved target layer. “contour” specifies the measurements are made at intervals along the edges of the input contour for the setlayer check.

- **target_layer *target_lay* [[not] {inside|outside} *filter_lay* [max_search *search_dist_um*]]**

An argument specifying the target layer for the image being analyzed. Specifying a target layer is required if “measure_at target” is specified, or if a filter layer is used; otherwise it is not permitted.

Note



By default, measure_at is “target”, which requires specifying target_layer.

A filter layer can be specified so that nils measurements are only taken within the region permitted by the filter. If “measure_at curved_target” or “measure_at contour” is specified, the filter layer is first applied to the original target and then the target edges, which are then mapped to the edges of the “measure_at” layer. Thus, nils measurements are taken within the regions specified by the “measure_at” layer, which were mapped to the target. The maximum distance from a target edge to a contour edge for which they can be mapped to each other is specified by *search_dist_um*. This value is not permitted when “measure_at target” is specified; otherwise, it is required.

- **curved_target *curve_lay***

An argument specifying a curved version of the target layer. Specifying a curved target layer is required if “measure_at curved_target” is specified; otherwise it is not permitted. Generating the curved target layer from the target using setlayer curve is recommended.

Note



Curved target layers must be generated in the setup file before any image commands are specified to avoid negative impacts on performance.

- **use_intensity {aerial | cm1}**

An optional argument specifying the intensity grid on which intensity gradients are computed. The default keyword is “aerial”, which specifies to use the aerial image intensity for the input contour of the setlayer check regardless if the setlayer image command includes a resist model. The “cm1” keyword is only permitted if the setlayer image

command specifies a resist model, and specifies to use the intensity grid output by the CM1 resist model.

Note

 If “use_intensity cm1” is specified, the “measure_at” setting must be “contour”, as the CM1 intensity is only physically meaningful close to the contour.

- **gauge_spacing *dist_um***

An optional argument that defines the spacing interval between the measurement gauges on the target or contour. The value must be ≥ 0.005 and ≤ 0.015 , and the default is 0.0125.

- **normalize_to {width | space}**

A required argument which specifies how the local CD used in normalization is computed at each measurement point. The “width” keyword specifies to measure the local CD inside the target or contour polygons. The “space” keyword specifies to measure the CD as the local spacing between polygons.

- **max_local_cd *val_um***

An optional argument that specifies a maximum value, in microns, for the local CD value. If the local CD is greater than this value, no results are generated at the measurement point. The default values is $3 * \text{critical_dimension}$, and its value must be greater than `critical_dimension`.

Note

 Excessively large values for `max_local_cd` can substantially degrade performance.

- **fix_small_cds {yes | no}**

An optional argument that forces measured CDs that are smaller than the `critical_dimension` to assume the value of the `critical_dimension`. The default is “no” which disables the option. If “yes” is specified, the forced value is used for the nils calculation and is reported as the `local_cd` property. This is useful when measuring CDs on low-quality contours, where pinching and bridging may cause some very small local CDs to be measured, leading to wide variations in the reported nils properties.

- **critical_dimension *val_um***

An argument that specifies the critical dimension for taking nils measurements. If a global `critical_dimension` has been specified, this argument is optional and defaults to the specified global value; otherwise, the argument is required. If a width check is being done, `critical_dimension` should be set to the smallest designed line width in the target geometry. If a spacing check is being done, it should be set to the smallest designed spacing in the target geometry. The `critical_dimension` is used as a tolerance for rejecting certain unrealistically small local CD values. A computed local CD is rejected (the measurement point is skipped) if the endpoints of the CD measurement are on the same “`measure_at`” polygon and are closer together than $\text{min_separation_fac} * \text{critical_dimension}$ when measured along the perimeter of the polygon. This is designed to eliminate unrealistic “corner-cutting” measurements.

Note

 If critical_dimension is not set correctly, measurement points may be dropped or unrealistic local CDs may be reported.

- min_separation_fac *fac*

An optional argument that filters any measured CD whose endpoint is on the same polygon contour as the measurement point and which is separated from the measurement point by less than min_separation_fac * critical_dimension when measured along the contour. The default value is 1.3, and must be ≥ 0.0 and ≤ 3.0 .

- cd_angle_range_degs *degrees*

An optional argument that specifies the angle on either side of a line along the intensity steepest descent direction. The value must be ≥ 1.0 and ≤ 30.0 , and the default is 7.5. The smallest local CD found within this angle range around the steepest descent direction is used as the local CD. This argument is not permitted when “measure_at target” is specified.

- markers [{gauge [width *width_um*] | trim_ends *trim_um*}] [height *height_um*]

An optional argument that specifies the type of error markers to generate. The default is “trim_ends 0”, which specifies that the error markers generated are rectangles whose width (along the edge) is *dist_um* specified by the gauge_spacing argument, and whose height (perpendicular to the edge) is *height_um* (default is 0.015 μm). When adjacent measurement points have errors, markers are merged into a single block. If a trim_ends value is > 0 , the error markers around each point are trimmed by *trim_um* microns (on each side) along the edge, given that the marker box is longer than $2*trim_um + 2$ dbu along the edge. This implies that $2*trim_um$ must be smaller than *dist_um* - 2 dbu. This option results in each measurement point having its own marker.

If “gauge” is specified, the markers are narrow rectangles. Their width is *width_um* microns (default 12 dbu, and is forced to 12 dbu if specified as anything less) and their length is *height_um* (default 0.015 um). The markers are oriented along the steepest-descent direction of the intensity at each measurement point, which provides a visual representation of exactly how NILS was computed.

Tip

 To see how all measurement points are placed, use “markers gauge” and run a check with an unrealistically tight constraint that will result in an error for every point.

- property {nils [local_cd] [image]}

An optional argument that attaches a property to the error markers. If “property nils” is specified, an “nils” property is placed on each error marker. If multiple images are being analyzed, the “nils” property is the worst (smallest) value over all images.

If “local_cd” is also specified, an additional “local_cd” property is generated specifying the local CD in microns which is used for normalizing. If “image” is specified, the name of the input image with the worst nils value is attached to the error marker. The “local_cd” and “image” properties must be specified after “nils”.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

interact

Verification control

Checks for interactions between polygons.

Tip

i This command is not meant to check for matching between contours and target layers. Use the [extra_printing](#) and [not_printing](#) commands instead.

Usage

```
interact layer1 [copy_props [‘(‘property_list1‘)’]]  
         layer2 [copy_props [‘(‘property_list2‘)’]]  
[[not] constraint] max_extent value [mutual]
```

where

```
property_list = prop_definition [...]  
prop_definition = [.] {[new_name =] prop_name | *}
```

Description

Checks to see if the polygons on the specified layers interact with each other (share any area, touch, or overlap). This function prefilters all **layer1** and **layer2** shapes to only keep the shapes smaller than the maximum extent value in both the **layer1** and **layer2** bounding boxes.

By default, the resulting shapes returned are only the shapes on **layer1**. Specifying the mutual option runs interact twice, with the second run swapping **layer1** and **layer2**. Both results are output.

Arguments

- **layer1** [copy_props [property_list1]]

A required argument specifying one of the input layers to test for interaction.

If the copy_props parameter is specified, properties on specified in *property_list1* are copied to the output layer. The “.” and “*” characters have special meaning.

Table 4-18. copy_props Syntax

Syntax	Description
*	Copies all properties “as is”. Cannot be combined with other <i>prop_definitions</i> in a <i>property_list</i> .
:	Copies all properties, but also prepends properties by the layer name and a period. Cannot be combined with other <i>prop_definitions</i> in a <i>property_list</i> . This is the default behavior if you specify “copy_props” with no arguments.

Table 4-18. copy_props Syntax (cont.)

Syntax	Description
<i>prop_name</i>	Outputs the named property.
<i>.prop_name</i>	Outputs the named property with the layer name and a period prepended to it.
<i>newname=propname</i>	Outputs the named property with the new name.
<i>.newname=propname</i>	Outputs the named property with the new name, prepended by the layer name and a period.

Note

 A *property_list* must be enclosed in parentheses (), and each property list item is separated by spaces.

When no *prop_list* is specified, all properties of the input layer are copied and their names are prefixed by the name of the input layer followed by a period. In other words, the default *prop_list* is “(.*)”.

When more than one polygon from *layer2* interacts with the same polygon from *layer1*, Calibre OPCverify selects *layer2* properties for output according the same rules used when multiple error markers are merged in the output. For example, if *layer2* has a property “min”, the smallest “min” of all *layer2* polygons interacting with *layer1* polygons is used.

Note

 Properties on input layers passed in by LITHO OPCVERIFY are not directly supported. Instead, use the setlayer copy command to access the properties for input layers passed in by LITHO OPCVERIFY. These properties do not have an associated merge_action. The default merge_action value for these properties is “ANY” which means that the interact-merge process randomly selects one of the multiple interacting secondary layer properties. No computation of min, max, or sum is performed on these passed-in properties.

- *layer2* [copy_props [*property_list2*]]

A required argument specifying the second input layer to test for interaction.

If the copy_props parameter is specified, properties specified in *property_list2* are copied to the output layer as shown in [Table 4-18](#).

When more than one polygon from *layer2* interacts with a single polygon from *layer1*, Calibre OPCverify selects the *layer2* properties from the polygon with either the smallest or largest value, depending on the property type, to attach to the output result.

- [not] *constraint*

An optional argument that controls the selection of *layer1* polygons according to the number of *layer2* polygons that interact with each *layer1* polygon.

When “not” is specified, the interact command becomes “not interact”, filtering for polygons that do not interact with each other.

The constraint should contain non-negative integer numbers. You should also not use a constraint of ==0; use a constraint of “not > 0” instead.

The default constraint is >=1.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- **max_extent value**

A required argument that specifies the maximum extent of the *layer1* polygons to consider. Any *layer1* polygons that exceeds the maximum extent are ignored.

- mutual

An optional argument. If specified, Calibre OPCVerify runs an interact operation on *layer2* *layer1* and *layer1 layer2* in same run.

Examples

The following example checks for simulated contact shapes that have exactly one contact layer (constraint of == 1) interaction, but first throws out any shapes (in either the contact or simulated contact layers) larger than .13 um in any direction:

```
setlayer int1 = interact SIMCONTACT CONTACT == 1 max_extent .13
```

internal

DRC-type operation

Performs the internal check for the specified region.

Usage

```
internal layer1 constraint [inside layer2] [parallel only] [extents]
  [[not] projecting [proj_constr]]
  {{expand expand_value_microns} | region} {manhattan | nonmanhattan}
  [property '{'
    {min | max}
  '}']
```

Description

Performs the internal check within the specified region. This uses the OPPOSITE metric, and outputs the REGION of the violation, as defined by the Calibre DRC OPPOSITE and REGION options.

Concurrency Support

Concurrency is supported for multiple internal calls that differ only in the **constraint** argument.

Arguments

- ***layer1***

A required argument specifying a layer containing the polygon shapes to perform internal measurements on. Usually used with layers generated with setlayer commands.

- ***constraint***

A required argument specifying a bounded constraint, which can include <, <=, >, >=, etc.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- inside *layer2*

An optional argument that when specified, checks only edges which are from *layer1* not inside *layer2*. If this argument is not used (only one layer specified), the constraint is applied to the interior facing edges of the *layer1* polygons.

- parallel only

An optional argument that limits the operation to parallel edges only.

- extents

An optional argument that when specified causes Calibre OPCverify to form extent rectangles on the input layer before performing the check. This argument is primarily for contact layers.

- [not] projecting [*proj_constr*]

An optional argument specifying that the internal distance is measured only where one edge projects onto another edge. If a constraint is specified in *proj_constr*, the projection length must also pass the constraint to pass the internal check.

Use the not option to measure only when neither edge projects onto the other edge.

- {expand *expand_value_microns* | region} [manhattan | nonmanhattan]

A required argument, specifying the output type from one of the following choices.

- expand and nonmanhattan — Expands each matching edge and outputs the expanded edges.
- expand and manhattan — Expands each matching edge by the specified amount and outputs the bounding box of the expanded edges.
- region and nonmanhattan — Outputs regions potentially containing skew edges similar to the SVRF Internal REGION statement.
- region and manhattan — Computes regions similar to the REGION statement, but orthogonalizes the regions before output.

manhattan is the default choice for manhattan or nonmanhattan.

- property '{'
min | max '}'

An optional argument that outputs properties attached to the result markers. The “min” property calculates the smallest distance measurement covered by the marker; the “max” property calculates the largest distance measurement covered by the marker.

The property argument also requires the parallel_only argument to be specified. The braces ({{}) are required, and “min” and “max” must be the only argument on a line.

Examples

Finds areas inside shapes that have edges less than .11 um apart:

```
setlayer int1 = internal TARGET < .11 region
```

maskgen

Verification command

Generates a mask layer for debugging purposes.

Usage

maskgen *image_name* [mask *n*] [rounded {0 | 1}]

Description

This command generates a debug layer for the mask layer used in simulation for the specified [image](#) command. The generated mask layer includes the effects of all mask layer geometry modifications. These geometry modifications (such as bias and cornerchop) may come from the mask_layer specifications in the litho model, or by explicit arguments to the image command.

Arguments

- ***image_name***

A required argument that specifies the name of an image layer generated by the image command.

- **mask *n***

An optional argument specifying the mask number in the image command or litho model to generate. The default is mask 0. If the specified mask number does not exist in the image command, an empty layer is returned.

- **rounded {0 | 1}**

An optional argument that specifies whether or not the chopped corners in the layout are converted to rounded corners with the same area. The rounded layout is a rendering of the mask simulated with a DDM model built with the corner_type rounded setting (as described in the [Calibre WORKbench User's and Reference Manual](#)). The default is 0.

Examples

The following example generates a debug layer for the image output from the image command shown (img_pv is an image set containing a litho model):

```
setlayer img_nom_sz = image img_pv bias 0.002
setlayer mask_test = maskgen img_nom_sz
```

measure_cd

Verification control

Checks accuracy of printed space or width CDs.

Usage

```
measure_cd layer ref_layer {external | internal}
  [[not] {inside | outside} filter_layer]
  cd_max cd_max_dist
  max_search max_search_distance
  tol [not] constraint [ratio | absolute | relative]
  [cd_min cd_min_dist]
  [cd_min_length min_length]
  [separation sep]
  [overunder dist]
  [output_expanded_edges halfwidth]
  [exceptions_only | exceptions_also]
  [runlength runlength_constraint]
  # error-centric section
  [property '{'
    [min]
    [max]
    [min_relative]
    [max_relative]
    [min_ratio]
    [max_ratio]
    [min_target]
    [max_target]
  '}']
  [classify, limit, or histogram block]
  [add_properties block]
  [pinpoint_output block]]
```

Description

This operation checks the accuracy of printed space or width CDs. The edges on the printimage layer (*layer*) that qualify must have a pair of appropriately facing parallel edges in the reference layer (*ref_layer*) that satisfy the following conditions:

- must be separated by at least *cd_min_dist*
- must be separated by at most *cd_max_dist*
- must have a mutually projecting edge length is at least *min_length*
- Optionally, meet an additionally specified condition of (not) outside | inside of *filter_layer*

Qualifying edges (within **max_search**) will then be checked for distance matching within the user-supplied absolute or percent tolerance. If the tolerance is achieved, a shape will be generated. The output shapes are drawn on the output layer.

The command works for manhattan and 45-degree edges.

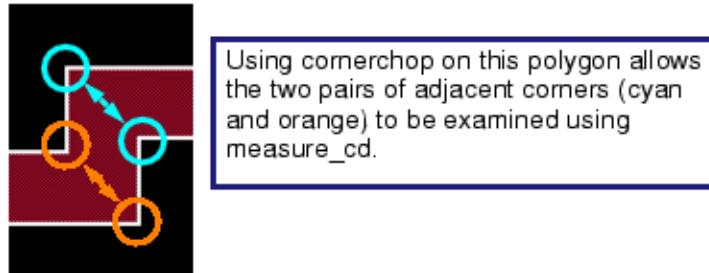
The following modes of output are supported:

- region of the error (default)
- edges bordering the error (`output_expanded_edges` option)
- regions where no CD could be determined (`exceptions_only` option)

Handling 90-Degree Angle Corners

In the case of two parallel 90-degree corners, `measure_cd` will normally not check for accuracy (since it requires two parallel edges). To use `measure_cd` on edges with 90 degree angles, you use the [cornerchop](#) function to give the two corners edges that face each other ([Figure 4-29](#)).

Figure 4-29. Using measure_cd on 90 Degree Corners

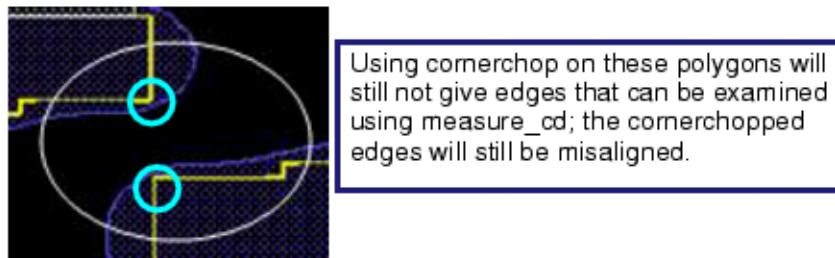


You perform the `cornerchop` function on the reference layer, creating concave and convex corners of the same value, such as in the following command:

```
cornerchop reflayer .02 .02
```

For misaligned 90-degree angle corners, however, this method may not provide the minimum CD measurement. Use the [external](#) function instead.

Figure 4-30. Misaligned Corners Exception



Concurrency Support

Concurrency is supported for up to 256 measure_cd calls that differ only in one or more of the following arguments:

- **tol** [not] **constraint** [ratio | absolute | relative]
- output_expanded_edges *halfwidth*
- exceptions_only
- cdeffective (obsolete)

Arguments

- **layer**

A required argument that specifies the original or derived polygon layer to check. You normally supply the printimage or aerial contour as this layer.

- **ref_layer**

A required argument that specifies the reference layer containing original or derived polygons defining the target CD values.

Tip

 When possible, use original layers for **ref_layer**, because derived polygon layers used for input may be missing edges to match to the **layer** argument.

- **external | internal**

A required argument specifying that the check is for internal or external edges.

- [not] {inside | outside} *filter_layer*

An optional argument specifying that only **ref_layer** edges interacting with *filter_layer* shapes as given are checked. Otherwise, all edges on **ref_layer** are checked.

Calibre OPCVerify selects check edges identically to the SVRF INSIDE EDGE and OUTSIDE EDGE commands:

- inside selects edges that are inside, not coincident to *filter_layer* shapes
- not inside selects edges that are outside or coincident to *filter_layer* shapes
- outside selects edges that are outside, not coincident to *filter_layer* shapes
- not outside selects edges that are inside or coincident to *filter_layer* shapes

- **cd_max cd_max_dist**

A required filter for the maximum cd for the pair of **ref_layer** edges to include in the check.

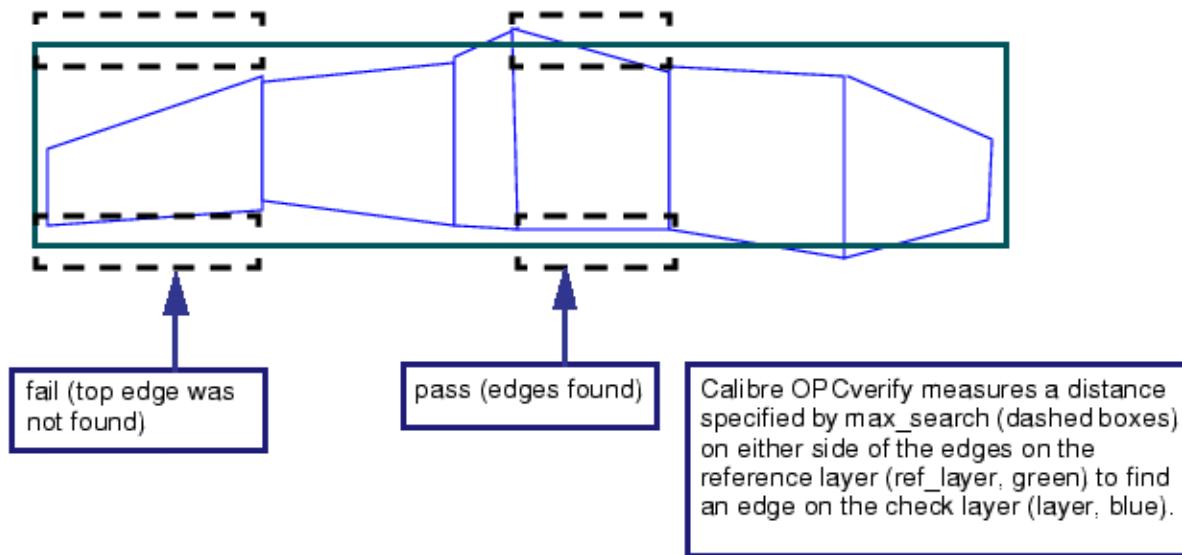
Tip

i The `critical_dimension` setup command can be used in place of this required argument. The `cd_max_dist` value used for `cd_max` is $1.5 * critical_dimension_value$.

- **max_search max_search_distance**

A required argument specifying the maximum distance to search outside of the space between the two relevant `ref_layer` edges for layer edges to be checked.

Figure 4-31. max_search Diagram



Note

i Setting `max_search` larger than half of your CD value may result in output shapes being rejected if the search encounters the contour of the next nearest polygon edge.

- Values that are too small may miss contours, creating exception-style errors. This is a known source of false positive errors; `max_search` should be at least greater than your minimum CD value.
- Values that are too high may find contours from the next edge beyond the nearest edge, creating false positive errors.

Siemens EDA recommends using a value that is half the CD width for this argument.

Tip

i The `critical_dimension` setup command can be used in place of this required argument. The `max_search_distance` value used for `max_search` is $0.5 * critical_dimension_value$.

- **tol** [not] **constraint** [ratio | absolute | relative]

A required argument that defines the tolerance. In “ratio” or “relative” (default) modes, the tolerance is a function of the CD on the relevant **ref_layer** edges. If the constraint is satisfied, an output shape is generated. For “relative” mode, the constraint is relative to the CD in question. In “ratio” mode, the constraint is specified as a ratio of the measured/reference CD in question. For the “absolute” mode, the cd constraint is in user units (default is microns). The default tolerance mode is relative.

To specify a constraint as a ratio, the second value is used as a ratio of the first, using 1.0 as ‘equal’. For example, the constraint “< 1.5” is read as “50 percent greater than.”

Note

 The **max_search** argument is processed before the **tol** argument; setting a **max_search** value too small can cause a **layer** edge to be skipped before it is considered by the **tol** constraint. Always ensure your **max_search** value is sufficient to identify areas of interest.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- **cd_min cd_min_dist**

An optional filter for the minimum cd for the pair of **ref_layer** edges to include in the check. Default value is 0.

- **cd_min_length min_length**

An optional filter for the minimum mutually projecting overlap length on two **ref_layer** edges which are to be included in the check. Default value is **cd_max + 1dbu**.

- **separation sep**

An optional argument that specifies the minimum target separation required to measure a CD. Target separation is defined as the distance along the target polygon between the ends of the target CD. Locations with a target separation less than *sep* in microns are excluded from the measurement. This option allows you to exclude line and space ends from CD measurements without using a separate filter layer. Default is = 0 (no separation).

- **overunder dist**

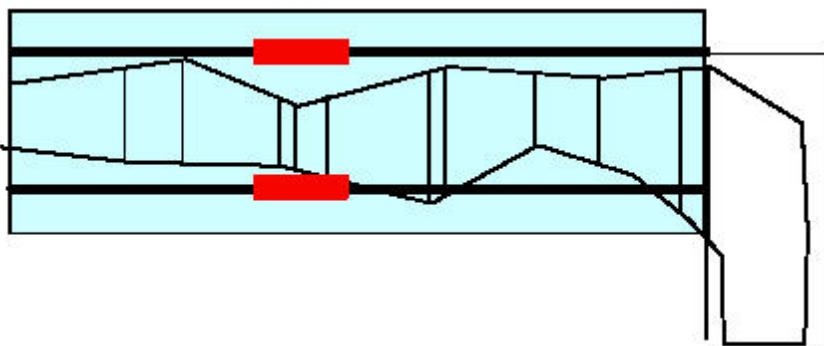
An optional argument specifying that within each CD region, if multiple error rectangles are closer than $2 * dist$ from each other, Calibre OPCverify merges them into one rectangle, and fills the space between them. This is done by performing an sizing “overunder” operation on the original error rectangles. The default value is 0.

- **output_expanded_edges halfwidth**

An optional argument specifying that the output of **measure_cd** will be thinly expanded edges surrounding the **ref_layer** CD edges (shown as red bars in [Figure 4-32](#)). This style of output is useful because the sum total length of these edges represents the length of CD

which matches the selected constraint. By default, this option is not used. This option is also incompatible with cdeffective.

Figure 4-32. measure_cd, output_expanded_edges Example



- exceptions_only | exceptions_also

exceptions_only and exceptions_also are mutually exclusive options. exceptions_only cannot be specified with output_expanded.

An optional argument that changes the output of measure_cd to be the regions where the image cd cannot be accurately determined (for example, no print) if exceptions_only is specified; exceptions_also outputs both constraint-selected markers and exception markers.

If a property block is also specified, exception markers are set to the following values depending on which property is requested:

- -100 for min, min_rel, and min_ratio
- 100 for max, max_rel, and max_ratio

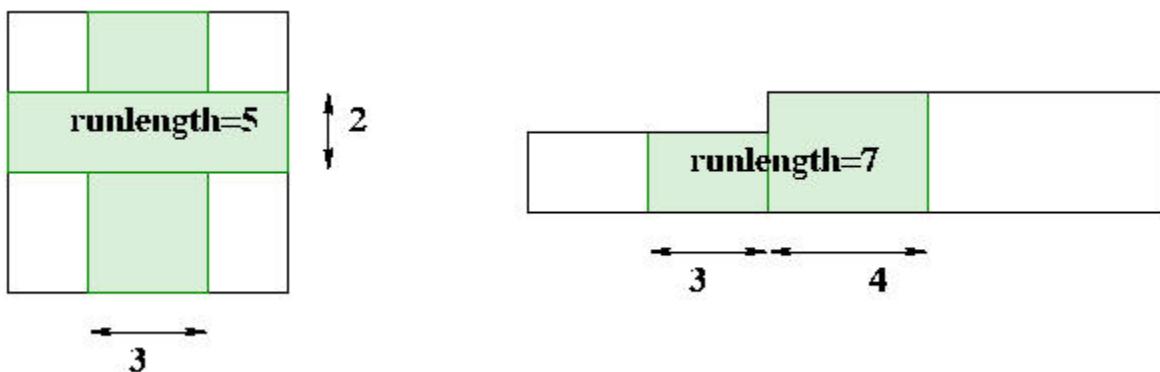
Note

 For runs using exceptions_only and exceptions_also, setting the **max_search** value less than the measured contour width will result in shapes being output, because measure_cd cannot determine the printing validity without the contour.

- runlength *runlength_constraint*

An optional argument that if specified, limits the output only to markers that meet the *runlength_constraint*. A run length is defined as the length of the marker across the CD with the following caveats:

- If a marker's edges along the target are of unequal length (such as for 45 degree CDs) the average of the two marker edge lengths are used as the run length.
- If a marker includes multiple CDs (such as a contact that has measure_cd markers in both directions or two markers that touch), the marker lengths will be summed as shown in the following figure.

Figure 4-33. runlength Argument in measure_cd

- property {'keywords'}

An optional argument that specifies an optional property operation for the returned output of this command. The braces ({{}}) around the property keyword(s) are required syntax. Multiple keywords can be specified, one per line. The allowed *keywords* are shown in the following table.

Table 4-19. measure_cd Property Keywords

min	max	min_relative
max_relative	min_ratio	max_ratio
min_target	max_target	

The min and max properties return absolute measurements. The other four property names are used for min and max measurements that have relative or ratio values. The min_target and max_target properties correspond to the maximum and minimum CD found within the target polygon.

Note

 Using the property option can cause a noticeable runtime impact, especially when there are large amounts of result markers generated through the use of this function. The SVRF DRC CHECK MAP rule file command will not retrieve properties from a layer output by this command; you must add an additional DFM RDB check as detailed in the section ““Creating the SVRF Rule File” on page 49”.

The following example attaches three different property values (min, max, and max_relative) to each output error shape.

```
setlayer mcd = measure_cd ...
property {
  min
  max
  max_relative
}
```

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Examples

In standard mode, the following command measures the CD for a contour:

```
setlayer mcd1 = measure_cd contour TARGET internal inside ISLAND \
    cd_min .09 cd_max .11 cd_min_length .001 max_search .03 \
    tol >= .080 < .082 absolute
```

Chaining multiple measure_cd commands together and outputting the results to different layers can be used as a binning tool, as shown in the following figure (each colored band represents a different tolerance range):

Figure 4-34. measure_cd Example Output (Standard Mode)

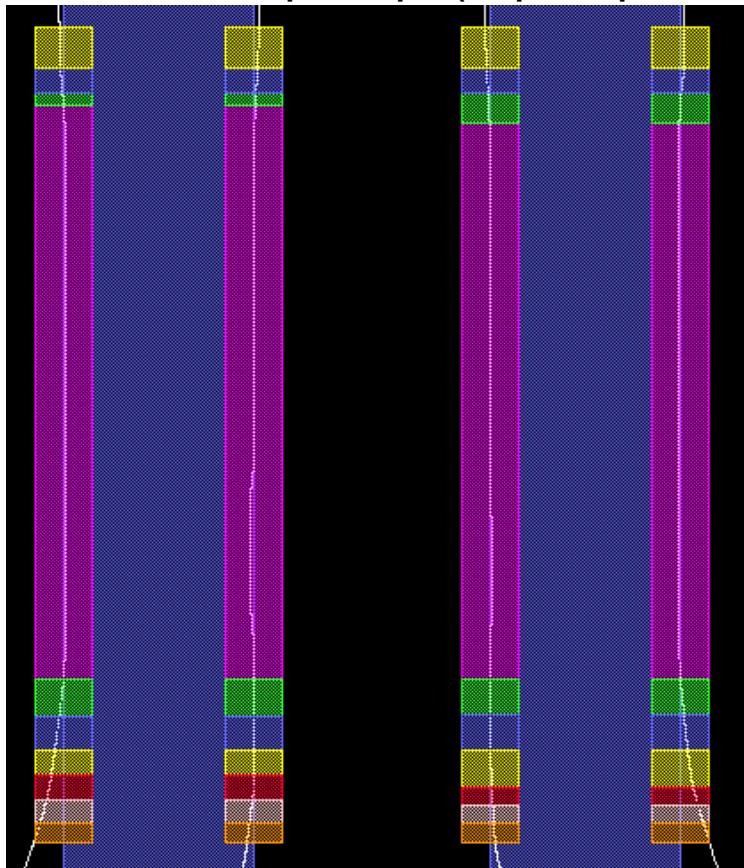


Specifying the output_expanded_edges option for the same example code:

```
setlayer mcd1 = measure_cd contour TARGET internal inside ISLAND \
    cd_min .09 cd_max .11 cd_min_length .001 max_search .03 \
    tol >= .080 < .082 absolute output_expanded_edges .02
```

results in the output shown in [Figure 4-35](#):

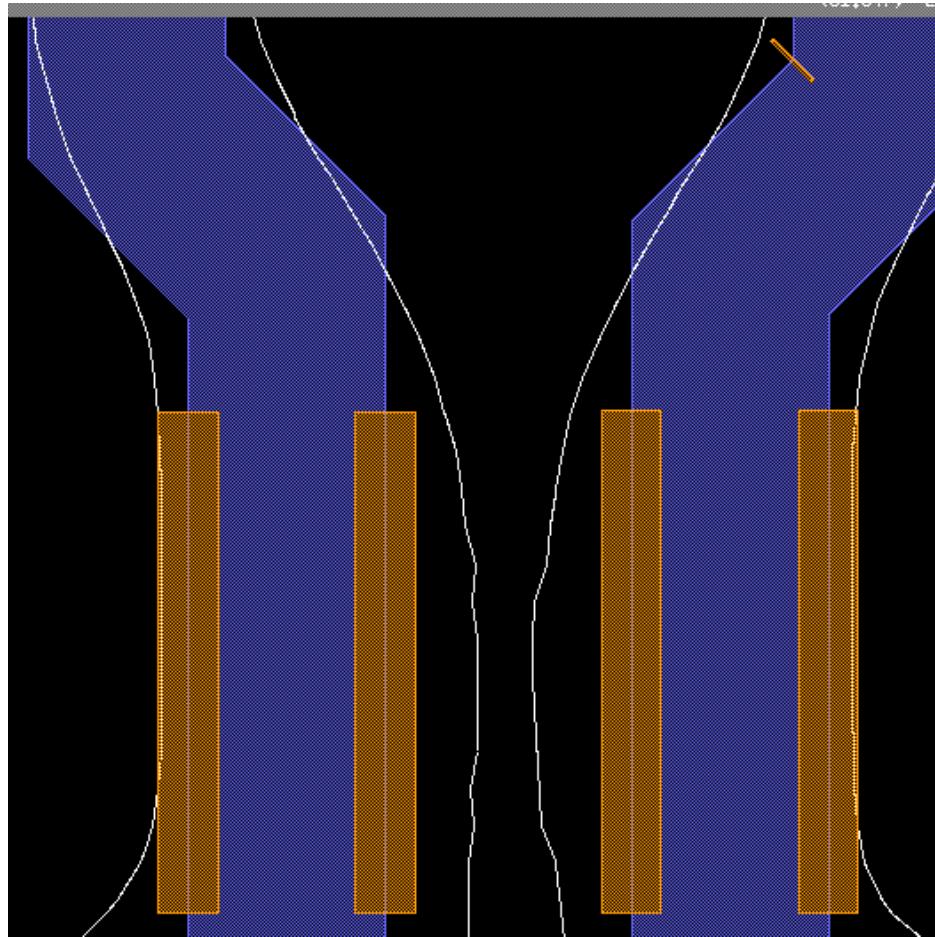
Figure 4-35. measure_cd Example Output (output_expanded_edges Mode)



The following example uses the exceptions_only option to highlight areas of the design where the CD could not be measured:

```
setlayer mcd1 = measure_cd contour TARGET internal inside ISLAND \
    cd_min .09 cd_max .11 cd_min_length .001 max_search .03 \
    tol >= .080 < .082 absolute exceptions_only
```

Figure 4-36. measure_cd Example Output (exceptions_only Mode)



measure_cdv

Verification control

Measures the CD variation between two contour layers relative to a target layer.

Usage

measure_cdv

```
layer_contour1 layer_contour2 layer_target
{external | internal}
[[not] {inside | outside} filter_layer]
tol [not] constr [absolute | ratio]
max_search max_search
cd_max cd_max
[cd_min cd_min]
[cd_min_length length]
[separation sep]
[overunder size]
[contour_cd_ratio]
[output_expanded_edges halfwidth]
[exceptions_only | exceptions_also]
# error-centric section
[property '{'
max
max_ratio
max_target
min_target '}' [classify, limit, or histogram block]
[add_properties block]
[pinpoint_output block]]
```

Description

This command measures the CD variation between *layer_contour1* and *layer_contour2* wherever *layer_target* has a CD that meets the specified constraint.

Concurrency Support: Up to 256 measure_cdv operations can run concurrently provided that they differ only in the following arguments: **tol** [not] **constr** [ratio | absolute], **output_expanded_edges halfwidth**, and **exceptions_only**.

Arguments

- **layer_contour1 layer_contour2 layer_target**
A set of required arguments specifying input layers.
- **external | internal**
A required argument specifying whether external or internal CDs are to be checked.

- [not] {inside | outside} *filter_layer*

An optional filter layer which can be used to exclude target corners and jogs for better results.

- **tol** [not] **constr** [absolute | ratio]

A required constraint to apply to either absolute CD variation (absolute mode, which is the default) or CD variation divided by the target_cd (ratio mode). However, if the contour_cd_ratio argument is also specified, ratio mode uses the layer_contour1 CD instead of the target_cd.

- **max_search** *max_search*

A required argument specifying a search distance. If contour edges are not found within max_search microns of the target edge for both contour layers, an exception is generated

- Values that are too small may miss contours, creating exception-style errors.
- Values that are too high may find contours from the next edge beyond the nearest edge, creating false positive errors.

Siemens EDA recommends using a value that is half the CD width for this argument.

Tip

 The **critical_dimension** setup command can be used in place of this required argument. The **max_search** value used for **max_search** is $0.5 * \text{critical_dimension_value}$.

- **cd_max** *cd_max*

A required argument specifying a maximum CD selection criteria. Only cds formed by parallel target edges where $\text{cd_min} < \text{cd_width} < \text{cd_max}$ and projecting length $\geq \text{cd_min_length}$ are checked.

Tip

 The **critical_dimension** setup command can be used in place of this required argument. The **cd_max** value used for **cd_max** is $1.5 * \text{critical_dimension_value}$.

- **cd_min** *cd_min*

An optional argument specifying a maximum CD selection criteria. Default is 0.

- **cd_min_length** *length*

An optional argument specifying a minimum projecting length for a target edge. Default: **cd_max + 1 dbu**

- **separation** *sep*

An optional argument that specifies the minimum target separation required to measure a CD. Target separation is defined as the distance along the target polygon between the ends of the target CD. Locations with a target separation less than *sep* in microns are excluded

from the measurement. This option allows you to exclude line and space ends from CD measurements without using a separate filter layer. Default is = 0 (no separation).

- overunder *size*

An optional argument that executes a size overunder command on error markers to merge any markers separated by less than $2 * \text{size}$ into one marker to reduce the error count. Default is 0 (do not merge markers).

- contour_cd_ratio

An optional mode switch that changes the reference CD from the standard target CD (*layer_target*) used for ratio calculation. If specified, the CD for *layer_contour1* is used as the reference CD for the max_ratio property and ratio constraint mode.

- output_expanded_edges *halfwidth*

An optional argument that changes the output of error markers to boxes of *halfwidth* microns along the layer target edges for greater visibility. Default is 0 (no boxes).

- exceptions_only | exceptions_also

An optional argument that changes the mode of the command to mark CD sections where one of the contours was not found within **max_search** distance of the target edge.

exceptions_only and exceptions_also are mutually exclusive options. exceptions_only cannot be specified with property keywords.

exceptions_also outputs both exception markers and constraint-selected markers. If a property block is also specified, exception markers are set to a value of 100 for max and max_ratio.

- property '{'keywords'
'}'

An optional argument that specifies a property operation for the returned output of this command. The braces ({{}}) around the property keyword(s) are required syntax. Multiple keywords can be specified, one per line. Multiple values can be computed for this command:

- max is the maximum variation detected in a given output polygon.
- max_ratio is the maximum (cd variation/target_cd, or cd variation/layer_contour1 cd if contour_cd_ratio was specified) detected in a given output polygon.
- max_target is the maximum target_cd in a given output polygon.
- min_target is the minimum target_cd in a given output polygon.

Note

 Using the property option can cause a noticeable runtime impact, especially when there are large amounts of result markers generated through the use of this function. The SVRF DRC CHECK MAP command will not retrieve properties from a layer output by this command; you must add an additional DFM RDB check as detailed in the section “Add Database and Output Reporting” in the “[Creating the SVRF Rule File](#)” topic.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Examples

```
setlayer cdv_layer = measure_cdv contour_a contour_b \
    target tol >= 0.03 cd_max 0.07 max_search .05
```

measure_cross_section

Verification control

Measures the cross section of target polygons versus a constraint.

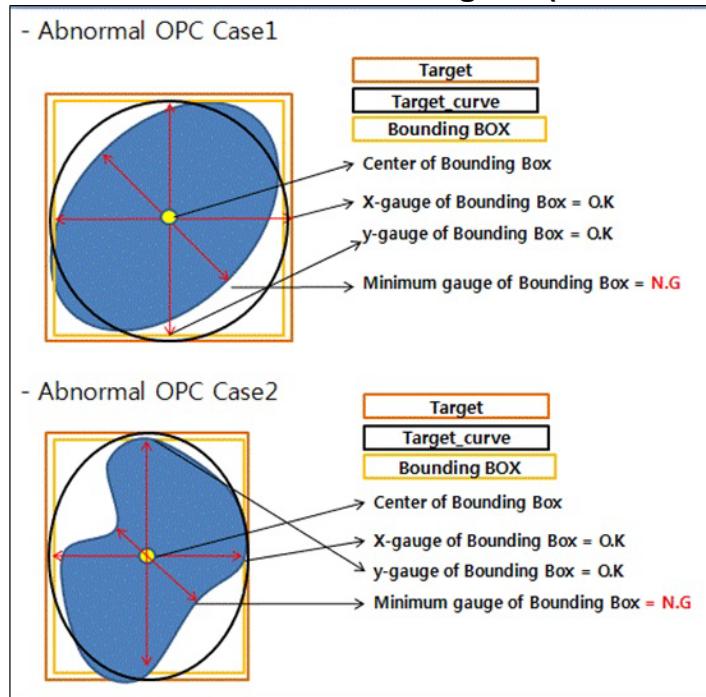
Usage

```
measure_cross_section contour_layer
  [target target_layer]
  [[not] {inside | outside} filter_layer]
  constraint [min | max | min_relative | max_relative | min_ratio | max_ratio]
  max_extent max_extent_um
  [{through {target_center | contour_center | target_bbox_center |
    contour_bbox_center}} | 
   horizontal | vertical | manhattan]
  [output_type {target | center output_size_um | gauge [gauge_width_um] }]
  [exception [also [property_value property_value]]]
  # error-centric section
  [property '{'
   min
   max
   min_relative
   max_relative
   min_ratio
   max_ratio
  '}'] [classify, limit, or histogram block]]
```

Description

Measures the minimum or maximum cross section of a contour polygon. The cross section may be measured in one of two modes:

- **Rotating scan mode** (“through” argument) — Selects one point at the center of a contour polygon, or its associated target, or the center of a bounding box. It measures the lengths of all cross-sections of the shape by lines going through that point, and compares the shortest or longest cross section with the user-specified constraint.
This mode is designed primarily to test coverage of non-square contacts.
- **Cartesian scan** (“horizontal”, “vertical”, or “manhattan” arguments) — Measures all cross sections of the contour polygon by lines parallel to the X or Y axis (or both). It calculates only the maximum cross section, since the minimum length of such a cross section calculation can be zero.

Figure 4-37. measure_cross_section Diagram (OK versus Not Good)

Arguments

- **contour_layer**
A required argument specifying a contour layer input.
- **target target_layer**
An optional argument specifying the target layer input.
- **[not] {inside | outside} filter_layer**
An optional argument, specifying a filter layer. The filter is applied against either the reference layer or the simulation layer (default), depending on the setting used for the output_type parameter.
Only output polygons that also satisfy the filter (are completely “inside” or “outside” filter_layer, with the optional “not” inverting the selection) are output.
- **constraint [min | max | min_relative | max_relative | min_ratio | max_ratio]**
A required argument specifying the constraint used to measure the gauge length (see Table 4-20). For rotating scan mode, the minimum gauge is tested by default (min argument). Specify max to check a maximum gauge constraint. “max”-type variables are the only allowed values for Cartesian scan mode.
- **max_extent max_extent_um**
A required argument that sets the maximum extent of target polygons to test. This restriction is necessary to limit the interaction distance of the operation.

Tip

 The default is $3 \times CD$ if `critical_dimension` is also defined in this rule file, and makes `max_extent` optional.

- `through {target_centroid | contour_centroid | target_bbox_center | contour_bbox_center}`
An optional argument that selects rotating scan mode. It indicates how Calibre OPCVerify selects the central point (CP) through which all the cross sections will go. It can be one of the following:
 - `target_centroid` — CP is the centroid (also known as the geometric center, or barycenter) of the target polygon.
 - `contour_centroid` — CP is the centroid of the contour polygon.
 - `target_bbox_center` — CP is the center of the bounding box of the target polygon.
 - `contour_bbox_center` — CP is the center of the bounding box of the contour polygon.
- Using `target_centroid` and `target_bbox_center` require `target` to be specified. The default is `target_centroid` when `target_layer` is specified, or `contour_centroid` otherwise.
- `horizontal | vertical | manhattan`
An optional argument that specifies Cartesian mode, and is used in place of the “through” argument. Cartesian mode scans only parallel to the specified axes, and returns the longest cross section in the scan.
 - `horizontal` — Sets Cartesian mode for horizontal cross sections only.
 - `vertical` — Sets Cartesian mode for vertical cross sections only.
 - `manhattan` — Sets Cartesian mode for both horizontal and vertical cross sections.
- `output_type {target | center output_size_um | gauge [gauge_width_um]}`
An optional argument that sets the shape to output when an error is found. The setting can be one of the following:
 - `target` — Marks the entire target polygon.
 - `center output_size_um` — Draws a square placed at the central point as selected by the `through` argument. Not available in Cartesian mode.
 - `gauge [gauge_width_um]` — Outputs the gauge. The width of the gauge is specified in microns. Default is 4.0 dbu and cannot be less than 4.0dbu.
- The default is `target` when `target_layer` is set, or no default otherwise.
- `exception [also [property_value value]]`
An optional argument. When only “exception” is specified `measure_cross_section` marks only target polygons that do not have a one-to-one interaction with contour or fail the `max_search` criterion. In this mode constraint and property specification are not allowed.

When “exception also” is specified exceptions are output alongside with normal error markers and all their properties are set to property_value. The default property_value is -0.0001.

- property ‘{’
min
max
min_relative
max_relative
min_ratio
max_ratio
‘}’

An optional argument that specifies a property operation for the returned output of this command. The braces ({}) around the property keyword(s) are required syntax. Note that “min”, “min_ratio”, “min_relative”, and “min_target” are only valid for rotating scan mode.

Table 4-20. measure_cross_section Properties

Property Argument	Description
min	Contains the length of the shortest cross section of the polygon (available only in the rotating scan line mode).
min_relative	Contains the difference between the length of the shortest cross section and the corresponding target reference length (available only in the rotating scan line mode).
min_ratio	Contains the ratio of the shortest cross section length to the corresponding target reference length (available only in the rotating scan line mode).
max	Contains the length of the longest cross section of the polygon.
max_relative	Contains the difference between the length of the longest cross section and the corresponding target reference length.
max_ratio	Contains the ratio of the longest cross section length to the corresponding target reference length.

- *classify, limit, or histogram block*

An optional argument that specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

measure_distance

Verification control

Measures distances between edges on a layer.

Tip

i This command is not intended to detect bridging and pinching conditions; use the [bridge](#), [pinch](#), [extra_printing](#), and [not_printing](#) commands instead.

Usage

```
measure_distance from_layer [[not] {inside | outside} filter_from_layer]  
[to_layer [[not] {inside | outside} filter_to_layer]]  
{external | internal | enclosure}  
distance_constraint  
[runlength constraint] [max_runlength rl_max [exception rl_ex] [ignore rl_ignore]]  
[measure_all | {separation sep}]  
[max_edge max] [angle_tolerance angle]  
[output_expand expval [to_edges_also] [manhattan | nonmanhattan]]  
# error-centric section  
[property '{'  
[min]  
[runlength_max]  
[runlength_min]'}' [classify, limit, or histogram block]  
[add_properties block]  
[pinpoint_output block]]
```

Description

This operation measures the distance from edges on the specified *from_layer*, finding the minimum distance from each edge in the *from_layer* to the nearest edge (either on the same layer, or edges on the optionally-specified *to_layer*) meeting the following criteria:

- inside the square mini-search region defined on the from-edge by *distance_constraint*
- meets the layer specification criteria
- meets the inside/outside/enclosure criteria
- meets the optional separation criteria
- meets the angle_tolerance criteria

The command outputs the bounding box of the region between the *from_layer* edges and the closest destination edges meeting the distance constraint. It tests the nearest edge (edges further out are ignored) found on the *to_layer*. It is typically used to check for soft pinching and soft bridging errors.

Concurrency Support

Concurrency is supported for multiple `measure_distance` calls (up to 32) that adhere to the following restrictions.

- The input layers (`from_layer`, `to_layer`, and `filter_from_layer`) must be the same.
- The measurement type (internal | external | enclosure) must be the same.
- The separation, angle_tolerance, and max_edge arguments must be the same.

Arguments

- **`from_layer`**

A required argument specifying the measurement-from layer. If no measure-to layer (`to_layer` argument) is specified, this layer is also used as the measure-to layer.

- [not] {inside | outside} `filter_from_layer`

An optional argument that specifies a filter layer; only `from_layer` edges (inside | outside) the filter layer are considered in the measurement function. Specifying the additional “not” option considers edges not (inside | outside) the filter.

- `to_layer`

An optional argument that specifies a second layer to be used as the measure-to layer. If not specified, the `from_layer` argument is copied and used as the measure-to layer.

- [not] {inside | outside} `filter_to_layer`

An optional argument that specifies a filter layer; only `to_layer` edges that completely match the filter layer (for example, fully inside the filter layer for “inside”) are considered in the measurement-to function. Specifying the additional “not” option considers edges not (inside | outside) the filter.

- **external | internal | enclosure**

A required argument specifying a check mode to perform:

- **external and internal**

Checks the external and internal distances.

- **enclosure**

Checks the enclosure of the measure `from_layer` by the measure `to_layer` and requires the `to_layer` argument.

- **`distance_constraint`**

A required argument. Specifies a bounded `distance_constraint` range from the `from_layer` to the `to_layer`, which must follow the standard SVRF constraint syntax rules.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

The minimum distance between the two edges must be less than the specified **distance_constraint** to generate output (the from edge has an output box written for it). This may cause some confusion when the distance constraint is of the type “ $a < x \leq b$ ” instead of the type “ $x \leq a$ ”.

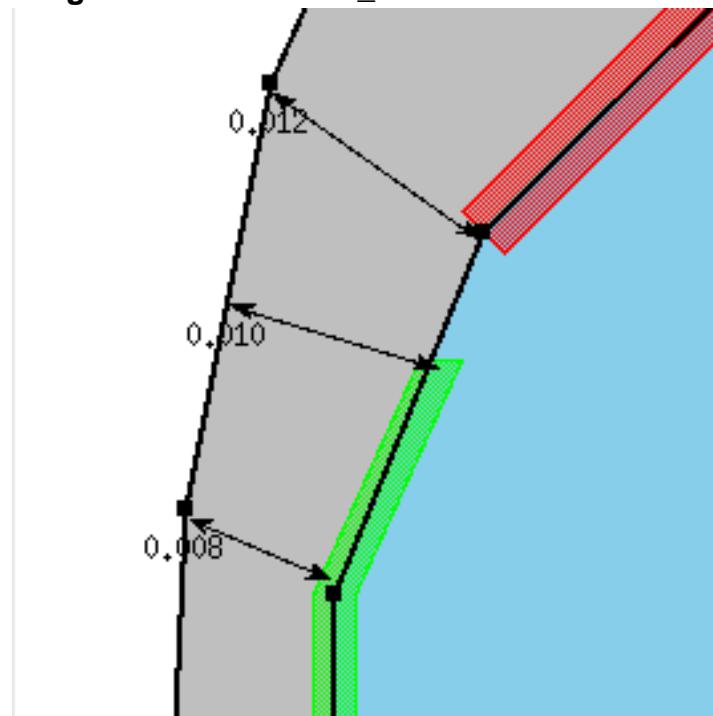
Given the following code, intended to bin all segments:

```
setlayer en10 = measure_distance contact contour enclosure <=0.01

setlayer en20 = measure_distance contact contour enclosure > \
    0.01 <=0.02
```

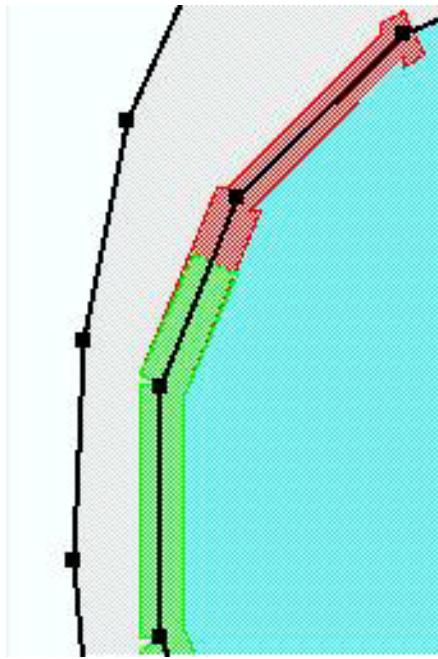
Running the code results in the following situation:

Figure 4-38. measure_distance Clarification



While you might expect the en10 (green) and en20 (red) markers to touch, they do not because the segment has a minimum distance for the segment of 0.008, so it is not picked up by the en20 check. The correct way to represent the bins is with an overlapping set of checks:

```
setlayer en10 = measure_distance contact contour enclosure <=0.01
setlayer en20 = measure_distance contact enclosure <=0.02
```

Figure 4-39. measure_distance Check With Overlap

- **runlength constraint**

An optional argument that instructs Calibre OPCVerify to filter out (ignore) any error with runlengths (defined as the total length of contiguous edges and edge fragments comprising the error, measured along the polygon boundary) that do not satisfy the specified *constraint*. When this option is used, the parameter `max_runlength` must also be specified.

- **max_runlength *rl_max* [exception *rl_ex*] [ignore *rl_ignore*]**

An optional argument that is required for use when the runlength constraint clause or any runlength property is specified. It instructs Calibre OPCVerify to compute the runlength for each error.

- An error with a runlength greater than *rl_max* in um is considered an exception, and the specified *rl_ex* value is used for this error instead of the real runlength value. (Default: 100 um.)
- If the optional parameter ignore *rl_ignore* is specified, interruptions in error markers shorter than the specified value in um are considered part of the error and included in its runlength. This feature allows Calibre OPCVerify to ignore small kinks that would otherwise cause the error to be split in several parts. (Default: 0.)

Note

 For consistent results on tile boundaries, the interaction distance has to be larger than *rl_max*. Using too large a *rl_max* value can impact Calibre OPCVerify performance.

- **measure_all**

An optional argument that when specified causes `measure_distance` to ignore polygon edges when performing measurements. This allows the command to “see through” polygons and measure the distance between edges that would otherwise be shielded by other edges. It works identically to the MEASURE ALL option for the INTERNAL/EXTERNAL/ENCLOSURE SVRF operations, and is only allowed when both `from_layer` and `to_layer` are specified. The `measure_all` option cannot be used with the separation argument.

- **separation *sep***

An optional parameter that sets a minimum separation distance along polygon perimeter(s) before edges on the polygon can be considered for measurement. As of the 2007.4 release, this can be applied as a one-layer or two-layer check. The separation option can not be used with `measure_all`.

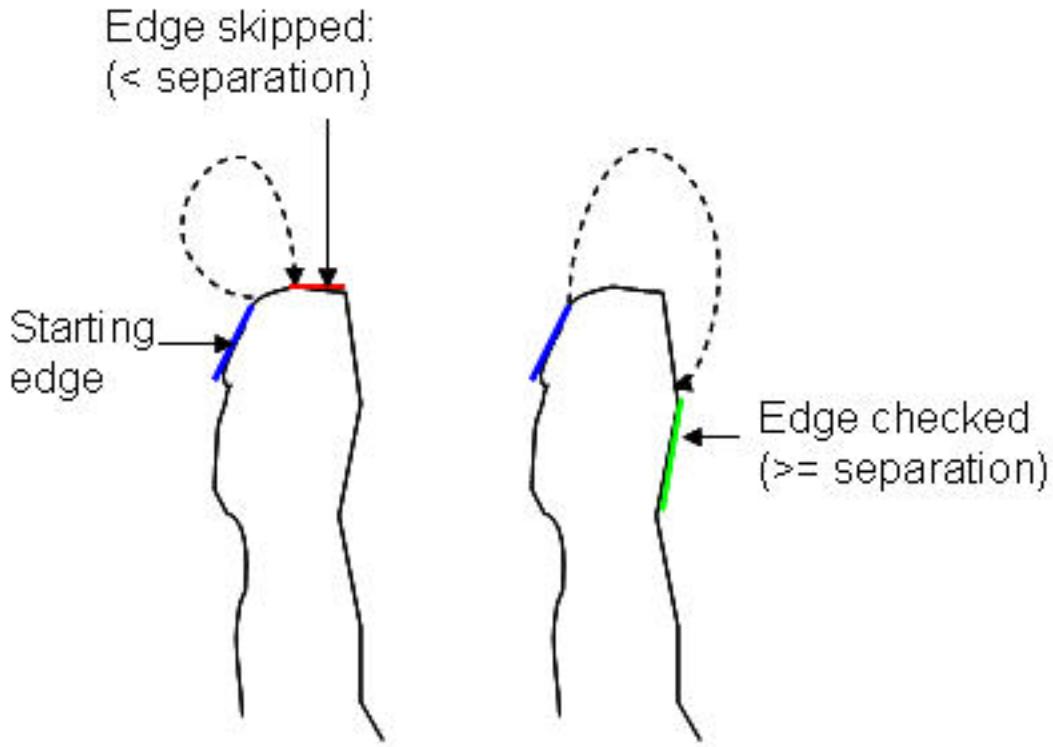
- **one-layer check**

Used as a one-layer internal and external check (a one-layer check has no `to_layer` argument), this argument is used to avoid creating checks that compare line ends to themselves (see the following figure), which could otherwise be returned as false errors. Defaults to 0 microns.

Tip

 Siemens EDA recommends that you set the separation value for one-layer checks to twice the minimum CD to best ensure that no false errors are caught. The `critical_dimension` setup command can be used in place of this argument. The `sep` value used for separation is $3 * \text{critical_dimension_value}$.

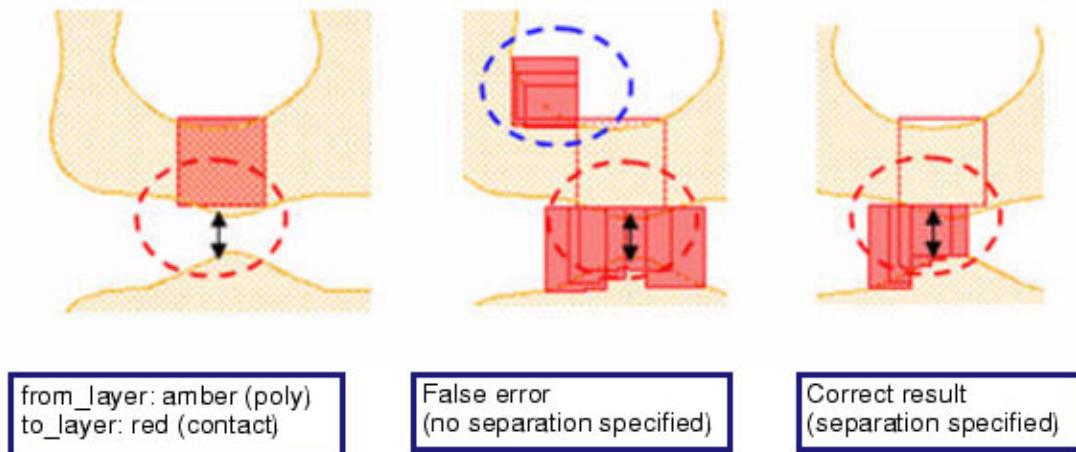
Figure 4-40. One-layer Separation Explained



- o two-layer check

Used as a two-layer check (a two-layer check has both a *from_layer* and a *to_layer* argument), this argument is used to avoid false errors that occur when parts of one layer are too close to a curve in the other layer (such as when a contact is exposed near an angle turn, as shown in the following figure).

Figure 4-41. Two-layer Separation Explained



Since the top left corner of the contact is exposed, the blue circle represents a false positive along with the correct errors (red circle). With a separation value, only the correct errors are found.

- `max_edge max`

An optional argument. Any filtered `from_layer` edge or `to_layer` edge longer than `max` distance will be fragmented into edges of a length smaller than `max`. This action may result in slight flat-versus-hierarchical differences.

Smaller values can result in better flat-versus-hierarchical consistency, but may impact performance. The default value is 0.1 micron.

Tip

The `critical_dimension` setup command can be used in place of this argument. The `max` value used for `max_edge` is $1 * \text{critical_dimension_value}$.

- `angle_tolerance angle`

An optional argument that specifies the maximum angle in degrees (0 to 90) between “from” and “to” edges in order to be appropriate for measurement. The default is 45 degrees.

- `output_expand expval [to_edges_also] [manhattan] nonmanhattan`

An optional argument which changes the output style. If `output_expand` is not specified, the command outputs the bounding box of the to- and from- edges that match the constraint.

- `output_expand expval`

An optional additional argument, which changes the command output to be only the bounding box of the `from_edge` expanded by `expval`. Larger values create fewer, larger error markers.

- `to_edges_also`

An optional additional argument that causes the matching to-edge to be output.

- `manhattan`

The default option. The command outputs the bounding boxes for the expanded edges.

- `nonmanhattan`

An optional argument specifying the output consists of just the expanded edges and not their bounding boxes. Requires the `output_expand` option to also be specified.

- `property '{`

`[min]`

`[runlength_max]`

[runlength_min]
'}'

An optional argument that specifies a property operation for the returned output of this command. One or more of the following operations can be requested:

- **min** — Returns the min(imum) distance measured.
- **runlength_max** — Returns the error runlength. If several error markers overlap, the property of the merged marker is the maximum of the runlength values for the overlapping errors.
- **runlength_min** — Returns the error runlength. For a single error, this is the same value as runlength_max. For merged error markers, the property is the minimum value of the runlength values for the overlapping errors.

Note



The runlength_max and runlength_min properties also require specifying the max_runlength argument.

The braces ({{}}) around the property keyword(s) are required syntax.

Note

Using the property option can cause a noticeable runtime impact, especially when there are large amounts of result markers generated through the use of this function. The SVRF DRC CHECK MAP rule file command will not retrieve properties from a layer output by this command; you must add an additional DFM RDB check as detailed in the section “Add Database and Output Reporting” in the “[Creating the SVRF Rule File](#)” topic.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Examples

The following example code creates a soft bridging check that detects shapes (on the same layer) that are closer than the specified constraint upper bound (.3) from each other.

```
setlayer softbridge = measure_distance CONTOUR external < .3 \
separation .6
```

The following example code similarly creates a soft pinching check and applies the property label ‘min’ to the results.

```
setlayer softpinch = measure_distance CONTOUR internal < .1 \
separation .3 property { min }
```

The separation parameter is effective in filtering out false errors.

measure_epe

Verification control

Measures the EPE (edge placement error) between target and contour edges.

Usage

```
measure_epe layer_contour layer_target
[[not] {inside | outside} layer_filter]
{epe_spacing spacing_microns | gauges_per_fragment number }
function {average | min | max | exception} [magnitude]
[exception also]
min_featsize value_microns
max_edgelen edge_value [fragment]
[angle_tolerance degrees_float]
[only [not] constraint]
[output_expanded_edges halfwidth_microns]
[trim_ends trim_microns]
# error-centric section
[property '{'keywords '}' [classify, limit, or histogram block]
[add_properties block]
[pinpoint_output block]]
```

Description

Measures EPE from *layer_target* edges to *layer_contour* edges. EPE is a signed quantity with positive values indicating that the contour is outside the target polygon, and negative values indicating that the contour is inside the target polygon.

The composite EPE is computed in various possible ways (min, max, average, and so on) by combining individual measurements taken from measurement gauges placed *spacing_microns* apart along *layer_target* edges. The EPE can be checked against an optional *constraint*.

By default, the output shapes are rectangles representing EPE with one edge being the measured *layer_target* edge, and the opposing edge being at a distance equal to $\text{abs}(\text{EPE})$ from it, either towards the polygon interior (negative EPE, as seen in pullback errors) or towards the polygon exterior (positive EPE).

Note

 A value of 0.00 EPE does not create an output shape. If the result of a function min, max, or average for a contour area under consideration is zero, no results will appear for that part of the contour, even if other parts of the contour area show a nonzero EPE in the layout. This is the expected behavior, and is based on the grid size setting in the setup file.

If the *output_expanded_edges* option is specified, the output instead consists of rectangles with one edge at a distance equal to *halfwidth_microns* from the reference edge towards the polygon interior, and the other edge at a distance equal to *halfwidth_microns* from the reference edge,

towards the polygon exterior. Measure_epe can perform line-end pullback checks or detect gross EPE errors.

Concurrency Support

measure_epe commands differing only in “function,” “magnitude,” “only,” and “output_expanded_edges” arguments are executed concurrently.

Usage Modes

measure_epe is typically used in one of the following modes:

- With the “only” and “output_expanded_edges” arguments

Use this mode to bin measured quantities into multiple bins. “only” specifies that the constraint must be met. “output_expanded_edges” displays boxes along the length of a contour.

- Without “only” and “output_expanded_edges” arguments

Use this mode to see how large measured errors are; the width of an output rectangle is equal to the size of the measured quantity.

Troubleshooting: Reasons for No Output

The following conditions will cause no output to appear for contour sections in the results of a **measure_epe** run:

- EPE is a zero result for function max / min / avg
- Constraint exceeded in “only” mode by one or more measurements in the contour area
- Angle tolerance is exceeded by a contour edge (the edge is discarded)
- min_featsize is set too small to detect edges, which must be within min_featsize/2 of the drawn feature edge to be found.

Arguments

- layer_contour***

A required argument, this is the layer to measure to, from ***layer_target***. Usually, this is the simulated contour layer.

- layer_target***

A required argument specifying the reference layer. Usually, this is the layer containing the target edges.

- [not] {inside | outside} ***layer_filter***

An optional argument limiting the search such that only reference layer edges meeting the selection type will be considered, effectively making ***layer_filter*** a filter layer:

- outside — Edges outside of ***layer_filter*** are selected.

- inside — Edges inside or coincident to *layer_filter* are selected.
- not — Reverses the selection criteria.

If this argument is not specified, Calibre OPCverify checks all edges.

- **epe_spacing spacing_microns**

A required argument (and alternative to **gauges_per_fragment**) specifying the spacing of gauge measurements (in microns) taken on a fragment. Each gauge is placed perpendicular to the *layer_target* edge or edge fragment.

- For edges of a length smaller than the *max_edgelen* parameter, the total number of edges will be odd.
- For edges that have an even-dbu length, a gauge will be placed at the center of the gauge; it will be placed within 1dbu of the center of the edge in all other cases.

- **gauges_per_fragment number**

A required argument (and alternative to **epe_spacing**) that specifies the number of measurement gauges to be placed on each *layer_target* edge or edge fragment.

If the [not] {inside | outside} *layer_filter* argument is also used, the **number** applies to the *layer_target* edges remaining after the filter application.

- **function {average | min | max | exception} [magnitude] [exception also]**

A required argument that specifies how multiple gauge data is collapsed into a single composite EPE value.

- average averages the EPEs measured along a fragment.
- min returns the minimum of all the measured EPEs along a fragment.
- max returns the maximum of all the measured EPEs along a fragment.
- exception produces output when no measurement gauges on an edge intersect a contour edge. This option must be specified with *output_expanded_edges*. It cannot be specified along with the “only” constraint.

Specifying the optional “magnitude” argument with **function** combines the absolute values of the gauge measurements.

Tip

 Using magnitude may result in only negative values being reported as properties unless a matching magnitude property is requested in the properties block. This can occur when the absolute values of the negative values are all larger than the positive values.

Specifying the optional argument “exception also” with average, min, or max outputs both exception markers (requiring *output_expanded_edges* to also be specified) and constraint-selected markers. Exception markers created using “exception also” are set to a property

value of -100 for min and min_magnitude, and to a value of 100 for max, average, max_magnitude, and average_magnitude.

- **min_featsize *value_microns***

A required argument specifying the minimum feature size in microns for the technology node (for example, 65nm, 90nm, and so on). Filtered layer target edges are orthogonally expanded by *value_microns*/2, in order to determine what contour edges are relevant. Contour edges outside the **min_featsize** setting are ignored.

Tip

 The **critical_dimension** setup command can be used in place of this required argument. The *value_microns* value used for **min_featsize** is $1 * \text{critical_dimension_value}$.

- **max_edgelen *edge_value* [fragment]**

A required argument that instructs Calibre OPCVerify to ignore all edges larger than *edge_value*. The optional argument “fragment” instructs Calibre OPCVerify to fragment edges longer than *edge_value* into edges that are less than *edge_value*, and EPE is calculated for each fragment. This allows you to perform a quick check for gross errors.

Note

 Using the fragment option may result in fragments that will not be consistent from cell to cell.

- **angle_tolerance *degrees_float***

An optional argument that specifies the aerial image contour slope tolerance, as a floating point angle value. If the angle between the contour and the fragment, measured at the cutline, is larger than *degrees_float* or smaller than -(*degrees_float*), the EPE at that cutline is discarded.

Specifying a negative value for *degrees_float* disables this check; the default value is **-1**.

Note

 Use of **angle_tolerance** is not recommended for this command; Siemens EDA advises the use of the **filter_generate** instead.

- **only [not] *constraint***

An optional argument, that instructs Calibre OPCVerify to only output EPEs that are within the given constraint. Specifying ‘only not’ returns the EPEs that do not meet the given constraint instead.

‘only’ cannot be specified with the ‘function exception’ option.

Tip

 See the section “[Constraints](#)” for more information on constraint syntax.

- **output_expanded_edges halfwidth**

An optional argument that changes the type of output to be fixed width boxes (instead of graphically showing the EPE), with edges equal to twice the specified halfwidth in microns. This argument is required if the property argument is also specified.

- **trim_ends trim_microns**

An optional argument that changes the size of the output boxes. Calibre OPCverify trims the output boxes around the edge of the target edge depending on the edge or fragment length:

- If the edge or fragment is longer than $2 * trim_microns + 3$ dbu: trim by $trim_microns$
- If the target edge/fragment is shorter than or equal to $2 * trim_microns + 3$ dbu: the trimming length is set so the output box length is 3 dbu

Note

 Before the Calibre 2017.1 release, the edge trimming length was $2 * trim_microns + 2$ dbu, and no trimming was performed if the fragment was shorter than that length.

(default: 0 microns; requires a non-zero value larger than **output_expanded_edges** if the average or average_magnitude property labels are requested.)

- **property '{'keywords
'}'**

An optional argument that specifies a property operation for the returned output of this command (see [Example 2 \(Properties Explained\)](#)). The braces ({})) around the property keyword(s) are required syntax. Multiple keywords can be specified, one per line from the list shown in the following table.

Table 4-21. measure_epe Property Keywords

min	max	average
min_magnitude	max_magnitude	average_magnitude
magnitude_min	magnitude_max	

Any property *keyword* can be used, regardless of what was specified in the function argument. Furthermore, all property *keyword* calculations are independent of the function calculation.

- min – The smallest EPE reading inside the error marker.
- max – The largest EPE reading inside the error marker.
- average – The mean EPE reading inside the error marker.
- min_magnitude – The smallest magnitude of EPE readings inside the error marker.
- max_magnitude – The largest magnitude of EPE readings inside the error marker.

- average_mag – The mean magnitude of EPE readings inside the error marker.
- magnitude_min – The absolute value of the smallest EPE reading inside the error marker.
- magnitude_max – The absolute value of the largest EPE reading inside the error marker.

Note

 Using the property option can cause a noticeable runtime impact, especially when there are large amounts of result markers generated through the use of this function. The SVRF DRC CHECK MAP rule file command will not retrieve properties from a layer output by this command; you must add an additional DFM RDB check as detailed in the section “Add Database and Output Reporting” in the “[Creating the SVRF Rule File](#)” topic.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Examples

Example 1

The following example code measures the EPE for line ends, by first setting up a filter just to find the line ends:

```
setlayer contour = image optical f0 resist_model vt5
setlayer ide = identify_edge TARGET length > .09 < .12 length1 > .250 \
length2 > .250 corner1 convex corner2 convex expand .002
```

The code measures the supplied contour versus the target layer, with a gauge measurement taken every .01 um, returning only EPEs in the range of -.1 um to .005 um.

```
setlayer epeout = measure_epe contour TARGET inside ide epe_spacing .01 \
function min min_featsize .01 max_edgelen .11 only > -0.1 < .005
```

Example 2 (Properties Explained)

Given a polygon configuration with measurements as shown in [Figure 4-42](#), the various property commands return the results shown in [Table 4-22](#), assuming the epe_spacing value is small enough.

Figure 4-42. measure_epe properties Example

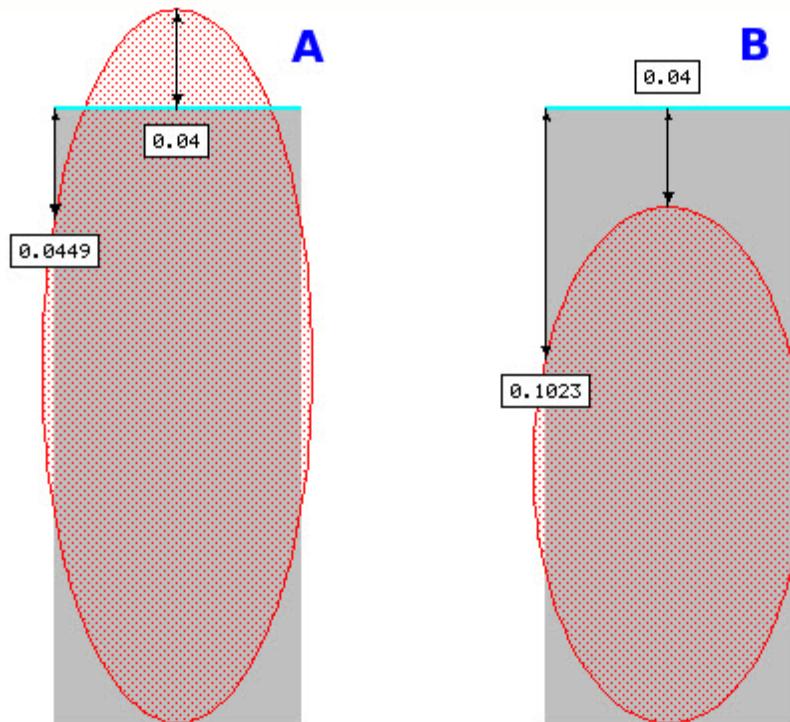


Table 4-22. measure_epe properties Results

Property	Value for A	Value for B
min	-0.045	-0.102
max	0.040	-0.040
average	0.016	-0.057
min_magnitude	0.000	0.040
max_magnitude	0.045	0.102
average_magnitude	0.026	0.057
magnitude_min	0.045	0.102
magnitude_max	0.040	0.040

meefcheck

Verification check

Checks the target's MEEF values versus a specified constraint.

Usage

meefcheck *constr layer_target*

```
{ {layer_im1 layer_im2} | {image_set image_set_name} }
[[not] {inside | outside} layer_target_filter]
[jog_filter jog_length_microns jog_extend_microns]
[exception_filter layer_exception_filter]
[trim_ends trim_microns] [expand expand_microns]
[delta_mask dm]
max_search min
[meef_spacing ms]
[contour_layer_spacing {layer_im1_im2_spacing_microns}]
[exception [also [property_value prop_value]]]
# error-centric section
[property '{'max'}' [classify, limit, or histogram block]
[add_properties block]
[pinpoint_output block]]
```

Description

Checks the target shape's MEEF (Mask Error Enhancement Factor). It uses the following flow:

1. The target shape edges (inside an optional filter if one is specified) are automatically subdivided into smaller edge fragments.
2. Each of those fragments is fitted with several measurement locations, spaced about meef_spacing apart.
3. The meef definition is computed at each measurement location as follows:
 - a. Let A be a measurement point on *layer_target*.
 - b. Find point B on *layer_im1* with the smallest Euclidean distance to point A such that the *layer_target* edge containing A and the *layer_im1* edge containing B have a non-negative inner product.
 - c. Find point C on *layer_im2* with the smallest Euclidean distance to point B, such that the *layer_target* edge containing A and the *layer_im2* edge containing C have a non-negative inner product.

$$\text{meef}(A) = \frac{\text{Euclidean_Distance}(B, C)}{\text{delta_mask}}$$

4. If the meef value for a given gauge satisfies the constraint, then an output shape representing the gauge will be written into the output layer.

Concurrency Support

Two **meefcheck** commands will run concurrently if they differ only in *constr*, exception, and property arguments.

Arguments

- ***constr***

An argument that specifies the range of meef values for which output shapes will be generated. Required except when the exception argument is specified, which changes the output of the command.

- ***layer_target***

A required argument that specifies the drawn target shapes layer.

- ***layer_im1 layer_im2***

A required argument that specifies two contours corresponding to two different sizings of the mask. *layer_im1* should correspond to the original mask; *layer_im2* should correspond to the sized up mask. The mask is assumed to be sized by “delta_mask.”

- ***image_set image_set_name***

A required argument that can be used in place of the *layer_im* arguments. It uses the named *image_set*’s contours for the mask sizing contours.

For this command, the image set must have only two image contours, which must be identical except for their mask bias. The image with the lowest absolute bias is intended to correspond with the original mask, and the delta mask value is calculated from the images.

- [not] {inside | outside} *layer_target_filter*

An optional argument specifying a filter. Only *layer_target* edges inside or outside *layer_target_filter* will be considered if the filter is specified. Otherwise, all *layer_target* edges will be checked.

- ***jog_filter jog_length_microns jog_extend_microns***

An optional argument specifying a jog filter. When specified, no measurements are performed around jogs on *layer_target*. Jogs are defined using the following arguments:

- ***jog_length_microns***

Specifies the maximum length of a jog, in microns.

- ***jog_extend_microns***

Specifies how far the jog-adjacent edges are filtered.

- exception_filter *layer_exception_filter*
An optional argument specifying that any measurement location inside or touching the filter is marked as an exception.
- trim_ends *trim_microns*
An optional argument that trims output boxes around each gauge. The boxes are trimmed by *trim_microns* along the target edge if the output box is longer than $2 * trim_microns + 2dbu$ along the target edge. This implies that $2 * trim_microns$ must be smaller than *meef_spacing* - 2dbu.
- expand *expand_microns*
An optional argument that specifies the halfwidth of the output markers; markers are expanded the specified distance in each direction from the edge. Defaults to 10 dbu when the trim_end argument is not also specified. Otherwise, it defaults to max(1dbu, min(10 dbu, *trim_microns* - 1dbu)).
- delta_mask *dm*
An argument that specifies the sizing on the mask in microns of the second image relative to the mask used on the first image. If the *layer_im1* and *layer_im2* arguments are supplied, this argument is required.
The delta_mask argument is optional if the **image_set** option is specified. The value is calculated between the bias settings of the first and second contours in the image set.
- **max_search *max***
A required argument that specifies the maximum Euclidean search distance from each measurement point on *layer_target* in order to find closest points on *layer_im1* and *layer_im2*.

Tip

 The **critical_dimension** setup command can be used in place of this required argument. The **max** value used for **max_search** is $0.5 * critical_dimension_value$.

- meef_spacing *ms*
An optional argument that specifies maximum distance between measurement locations. Default is 0.01 microns.

Tip

 The **critical_dimension** setup command can be used in place of this argument. The *ms* value used for *meef_spacing* is $0.2 * critical_dimension_value$.

- contour_layer_spacing {*layer_im1_im2_spacing_microns*}
- An optional argument that specifies the maximum distance between measurement locations on contours *layer_im1* and *layer_im2* during scanning (default: meefspacing*1.5).

- exception [also [property_value *prop_value*]]

An optional argument that outputs exception markers.

- When exception is specified, output is generated only for those measurement points where a difference could not be computed, due to point B or point C not being found within **max_search** of the measurement point. The exception option used alone is mutually exclusive with the **constr** and property arguments.
- When exception also is specified, **constr** is required and property specification is allowed. In this mode, **meefcheck** outputs both exception shapes and constraint-related results. If a property block is also specified, exception polygons have the user-specified *prop_value* attached. The default value is -0.0001. This is useful for limiting and histogram generation.

- property ‘{’max‘}’

An optional argument that specifies a property operation for the returned output of this command. Only the max(imum) meef can be computed for this command. The braces ({{}}) around the property keyword(s) are required syntax.

- *classify, limit, or histogram block*

An optional argument that specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see “[Pinpoint Output Block](#)” on page 102.

Examples

```
meefcheck > 3 poly im0 im1 inside active delta_mask 0.015 max_search 0.05
```

nilscheck

Verification check

Checks the NILS or ILS value versus a specified constraint.

Usage

nilscheck

```
constr layer_target max_search max
{{images i1 i2 [i3 ... in] levels L1 L2 [L3...Ln]} | {image_set image_set_name}}
[nils_spacing micron_spacing]
[[not] {inside | outside} layer_target_filter]
[width | space | both | disregard_cd]
[trim_ends trim_microns] [expand halfwidth_microns]
[exception [also [property_value prop_value]]]
# error-centric section
[property '{'min'}' [classify, limit, or histogram block]]
```

Description

Calculates the normalized image log-slope (NILS) or image log-slope (ILS) at target edges that form a CD. If the NILS or ILS value for a given gauge satisfies the constraint, then an output shape representing the gauge will be written into the output layer.

The edge selection algorithm works as follows:

1. The target shape edges are automatically subdivided into smaller edge fragments.
2. Each fragment is fitted with measurement gauges spaced about nils_spacing apart, of length 2*max_search centered at the edge fragment and placed perpendicular to the target layer edge.
3. Each gauge is tested to see if it is a part of an external (space) and internal (width) CD space or width smaller than 2*max_search. The gauges that are a part of an external or internal CD will have the NILS definition below computed. An exception is output for a given gauge if the gauge is a part of a CD, but does not intersect at least two appropriately oriented image contours.

To compute the NILS at a given gauge, **nilscheck** looks for an intersection of the gauge with two contours that have negative and non-negative EPEs of the smallest magnitude. If two such intersections are not found, the command then looks for two intersections having the smallest magnitude EPEs of the same sign. If two intersections are not found, an exception is generated for the gauge.

Let i and j be the indices of the two selected contours. “ln” denotes natural logarithm.

```
(ln(Level_i) - ln(Level_j))  
NILS = absolute_value(-----) * CDtarget  
(EPE_i - EPE_j)
```

If (EPE_i - EPE_j) is equal to 0, NILS is set to 1000000.

If the “disregard_cd” option is specified, CDtarget is set to 1, and NILS is therefore calculated at every gauge unless it is an exception. This is also known as ILS, or (non-normalized) ‘image-log slope’ calculation.

Concurrency Support

Two **nilscheck** commands will run concurrently if they differ only in **constr**, exception, the CD type (width and space), trim_ends, expand, and property arguments.

Arguments

- **constr**

A required argument that specifies the range of NILS or ILS values for which output shapes will be generated. Must be the first argument, unless the exception argument is specified.

- **layer_target**

A required argument that specifies the drawn target layer. Must be the first argument if **constr** is not specified, or the second argument otherwise.

- **max_search max**

A required argument specifying the maximum search distance from target to contour to compute EPE for. Only target layer CDs of width or space $\leq 2*\text{max_search}$ are identified by this operation.

Tip

 The **critical_dimension** setup command can be used in place of this required argument. The **max** value used for **max_search** is $0.5*\text{critical_dimension_value}$.

- **images i1 i2 [i3 ... in]**

A required argument (an **image_set** argument can be used instead) specifying at least two input images in order to calculate the slope at the location of the drawn shape edges.

Note

 By convention, only CTR images are expected for this argument (generated by image commands using the aerial argument).

- **levels L1 L2 [L3...Ln]**

A required argument (an **image_set** argument can be used instead) specifying contour slice levels corresponding to and matching the aerial contour values supplied with the input images. This keyword must follow the images keyword immediately.

- **image_set *image_set_name***

An argument that can be used instead of the **images** and **levels** arguments. It uses the contours inside the specified **image_set** to test the NILS constraint. The **image_set** must have at least two input contours with different aerial contour levels. Other options inside the **image set** must be identical other than the aerial option.

- **nils_spacing *micron_spacing***

An optional argument that specifies a maximum gauge spacing distance in microns. Default is 0.01 microns. The smallest allowed spacing is 2dbu; if a smaller value is specified, it will be corrected up to the allowed value.

Tip

 The **critical_dimension** setup command can be used in place of this argument. The *micron_spacing* value used for **nils_spacing** is $0.2 * \text{critical_dimension_value}$. However, if **trim_ends** is specified, a different default of 5 dbu is used regardless of the **critical_dimension_value**.

- **[not] {inside | outside} *layer_target_filter***

An optional argument specifying a filter layer. Only reference layer edges inside or outside **layer_target_filter** will be considered if the filter is specified. Otherwise, all **layer_target** edges will be checked. It is recommended to filter out line ends to avoid a large number of exceptions.

- **{width | space | both | disregard_cd}**

An optional argument that specifies the type of CD (width or space) to compute NILS for.

- Use “both” when you have mixed line and space structures.
- If **disregard_cd** is specified, the normalization step is discarded and non-normalized image-log slope generation (ILS) is computed instead.

The default is **width**.

- **trim_ends *trim_microns***

An optional argument that specifies that the output boxes from this command are trimmed by the specified value along the target edge if the output box is longer than $2 * \text{trim_microns} + 2\text{dbu}$ along the target edge. This implies that **trim_microns** cannot be larger than the **nils_spacing** argument (**micron_spacing**)-2dbu.

- **expand *halfwidth_microns***

An optional argument that sets the halfwidth value of returned markers. If **trim_ends** is not also specified, the default value is 10dbu. If **trim_ends** is specified, the default value is set at a minimum of 1dbu and a maximum of 10 dbu or (**trim_microns** -1dbu), whichever is smaller.

- **exception [also [property_value *prop_value*]]**

An optional argument that outputs exception markers.

- When exception is specified, output is generated only around the gauges which were a part of a CD, but did not intersect at least two image contours. This option is mutually exclusive with the **constr** and property arguments.
 - When exception also is specified, a constraint is required and property specification is allowed. In this mode, **nilscheck** outputs both exception shapes and constraint-related results. If a property block is specified, exception polygons have a property value attached equal to *prop_value*. The default value is -0.0001. This is useful for limiting and histogram generation.
- **property {min}**
An optional argument that specifies a property operation for the returned output of this command. Only the minimum ILS or NILS value can be computed. The braces ({{}}) around the min keyword are required syntax.
 - **classify, limit, or histogram block**
An optional argument that specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

Examples

```
setlayer im1 = image optical opt aerial 0.3
setlayer im2 = image optical opt aerial 0.35
nilscheck <= 5 target max_search 0.050 images im1 im2 levels 0.3 0.35 \
nils_spacing 0.010 inside active width property { min }
```

-
- Tip** The levels chosen should be slightly above and below the nominal threshold value for the resist film. The threshold value can be found in the VT5 and the CM1 model file as the reference_threshold keyword.
-

not

DRC-type operation

Performs a multi-layer NOT by subtracting each input from *input1*, starting with *input2*.

Usage

not *input1* *input2* ... *inputN* [smooth max_error *error_value*]

Arguments

- *input1* *input2* ... *inputN*

A set of required arguments specifying the layers to operate on.

- smooth

An optional argument to correct small imperfections like slivers and kinks resulting from applying NOT operations to simulated contours. The smooth operation:

- Removes contour slivers if their size is less than *error_value* in microns
- Smooths kinks if it can be done without moving the contour by more than *error_value* in microns from its input position

The recommended value of max_error is 2dbu. For example, if the precision is 8000, the max_error setting should be between 0.00025 and 0.0004 microns.

Examples

If the layer *ti1* is the inner tolerance zone of a polygon shape, the following command returns the area of polygon that is not part of the inner tolerance zone:

```
setlayer n1 = not target ti1
```

not_printing

Verification control

Detects non-printing features on a layer.

Usage

```
not_printing target [[not] {inside | outside} filter_layer] contour [max_extent value]  
[slivers {off} | {size [ignore] | blots size1 [ignore] holes size2 [ignore]}] [manhattan | skew]]  
[small_holes extent [suppress | manhattan | {extents [centers size}]]]  
# error-centric section  
[property '{'  
min  
'}' [classify, limit, or histogram block]]
```

Description

This command detects non-printing features (a polygon that does not match any nearby contour points) on a layer. It checks the drawn (**target**) layer versus the specified **contour** layer. Any target shapes that do not interact with the contour layer are returned.

Arguments

- **target**
A required argument specifying the target layer.
- [not] {inside | outside} *filter_layer*
An optional argument that performs the specified check (inside, not inside, outside, or not outside) the filter. It marks only errors with at least one edge that satisfies the filter.
- **contour**
A required argument specifying a generated contour layer.
- **max_extent value**
An optional argument (default: 0.15) that Calibre OPCverify uses to distinguish between gross EPE errors at tile/hierarchical boundaries and non-printing defects.

Tip

 The **critical_dimension** setup command can be used in place of this argument. The *value* used for **max_extent** is *critical_dimension_value*.

- slivers {off} | {size [ignore] | blots *size1* [ignore] holes *size2* [ignore]}] [manhattan | skew]]
An optional argument that when specified filters out all narrow contour holes before checking for not_printing features. By default, sliver detection is off. The parameters determine how the slivers are handled.
 - *size* [ignore] — Calibre OPCverify builds a bounding box around every contour polygon and hole. If the smaller side of this bounding box is less than *size* microns,

the polygon or hole is considered to be a sliver and is removed from the contour layer.

When the ignore option is specified, the slivers are discarded. Otherwise, holes are marked by the not_printing command.

- blots *size1* [ignore] holes *size2* [ignore] — Different *size* criteria can be specified for blots (polygons) and holes.

In the default manhattan mode, the sliver bounding box is aligned with the X and Y axes. When the mode is skew, Calibre OPCVerify builds and tests two bounding boxes; one axes-aligned and one rotated by 45°. It returns the smaller bounding box.

- small_holes *max_extent* [suppress | manhattan | {extents [centers *size*]})]

An optional argument that sets how to mark not_printing contour holes with extents less than or equal to the *max_extent* value:

- suppress — Does not output anything for extra printing if the extent area is less than or equal to *extent*.
- manhattan — Convert to 0 or 90 degree lines and output the contour polygons.
- extents — Output a minimum bounding box for each polygon rather than the polygon itself.
- extents centers *size* — Output marker squares in the centers of the bounding boxes.

- property '{'min
'}'

A conditional argument that declares a property block. The “min” property block must be specified when you use the [pw_annotation](#) command with this check. However, note that the property “min” will always be zero.

The braces ({{}}) around the property keyword are required syntax. Property operations require a distance constraint except in the case of hard bridge-only checks.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

Examples

```
setlayer notp = not_printing poly contour1
```

Related Topics

[bridge](#)

[pinch](#)

[extra_printing](#)

notchfill

DRC-type operation

Attempts to fill in notches or nubs on the specified input layer. This command is designed to reduce irregularities in polygon edges.

Usage

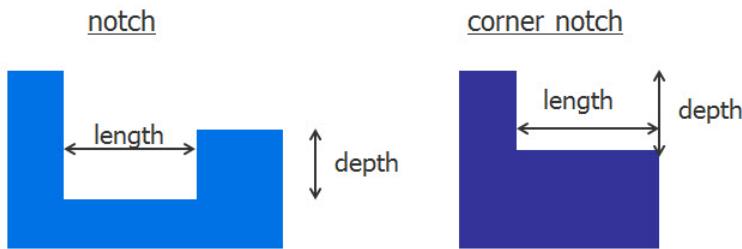
```
notchfill input {notch | nub} max_depth_microns [-length value_length] [cornernotch]  
[symmetry]
```

Description

Note

 The notchfill command can cause rare MRC violations when it is in a rule file with other setlayer commands, resulting in the generation of minor unintended holes. The workaround for this issue is to move the notchfill command to a different LITHO OPCVERIFY call.

Figure 4-43. notchfill Operation



Arguments

- ***input***

A required argument specifying a layer containing polygons to operate on.

- ***notch* | *nub***

A required argument. Either notch or nub must be specified. Selects the operation to be performed:

- notch removes notches (concave shapes inside a polygon)
- nub removes nubs (convex shapes outside of a polygon)

- ***max_depth_microns***

A required argument that specifies the largest size of notch or nub depth to fill in, in microns. If the -length argument is not specified, this amount is used for both length and depth of notches.

- -length *value_length*

An optional argument. By default, the **max_depth_microns** argument applies to both the length and depth of notches and nubs. Specifying -length uses the *value_length* argument for lengths and **max_depth_microns** for depths.

- cornernotch

An optional argument that specifies that notches on corners are removed as well (the default is to ignore notches on corners).

- symmetry

An optional argument that specifies that the command will generate symmetrical results on symmetrical input. This is achieved by running the filling algorithm twice (once counterclockwise and once clockwise) along each polygon.

Examples

```
setlayer n1 = notchfill target notch .01 cornernotch
```

or

DRC-type operation

Performs a multi-layer OR of all inputs.

Usage

or *input1* . . . *inputN*

Arguments

- *input1* . . . *inputN*

A set of required arguments specifying the layers to operate on.

Examples

```
setlayer or1 = or layer2 layer3
```

Figure 4-44. Two-Layer Boolean or



or layer2 layer3

pinch

Verification control

Checks for pinching errors that meet the specified constraint.

Usage

```
pinch target_layer [[not] {inside | outside} filter_layer [filter_contour_also]]  
  {contour_layer | {overlay | cutmask | keepmask}  
   max_error contour_layerA contour_layerB}  
  [{simple_constraint | double_constraint | annotated_constraint}  
   separation sep_microns  
   [runlength {> | >=} length max_runlength rl_max [exception rl_ex]]]  
  [projecting [projection_length]]  
  [max_tolerance tol_microns]  
  [max_edge edgelen_microns]  
  [output_expand width_microns [width_microns2]]]  
  [pinpoint [markers | gauges] [size]]  
  [slivers {off | {size [ignore] | blots size1 [ignore] holes size2 [ignore]} | {manhattan | skew}}]  
  [small_holes extent [extents | {extents centers size} | suppress | manhattan]]  
  [cd_search_dist cdist]  
  # error-centric section  
  [property '{'min  
  [target_cd]  
  [shift_dir]  
  [runlength]'}' [classify, limit, or histogram block]  
  [add_properties block]  
  [pinpoint_output block]]
```

Description

Checks for topological errors in printed contours that occur when two contour edges pinch (have gaps where they should not). This command can also check for near-pinching conditions (soft pinching). This command complements the [bridge](#) command.

The `pinch` command always catches [not_printing](#) errors.

Arguments

- ***target_layer***
A required target layer to check.
- [not] {inside | outside} *filter_layer* [filter_contour_also]
An optional argument that outputs results that are inside or outside the specified filtering layer. Specifying the “not” qualifier reverses the filter. “Extra hole” type errors are output only if all edges satisfy the filter condition.

Adding the filter _contour _also qualifier causes the command to exclude contour edges that are not assigned to any filtered target edges.

Note

 Exercise caution when using filter layers with the bridge/pinch commands. The performance of Calibre OPCverify may vary depending on the nature of the filter: Filters consisting of relatively large polygons over small parts of the design can improve performance over filterless bridge and pinch commands. Filters consisting of many small polygons over large parts of the design can slow down performance and are not recommended over filterless bridge and pinch commands. The best approach in this situation is to use bridge/pinch without filters, and then apply the Calibre OPCverify Boolean [and](#) operation to the results with your intended filter layer.

- **contour_layer**

A required printed contour layer to check. You can generate a contour layer using the [image](#) command.

- **overlay | cutmask | keepmask max_error contour_layerA contour_layerB**

An alternative to **contour_layer** (single contour) mode, overlay, cutmask, and keepmask are designed to enhance the basic pinching check to include a check between two contour layers (**contour_layerA** and **contour_layerB**) for use with double patterning designs.

- The **overlay** argument adds an additional check for pinching conditions in a contour defined as the union (A OR B) of the two contour layers. **overlay** is typically used for LELE designs.
- The **cutmask** argument adds an additional check for pinching conditions in a contour defined as the difference (A NOT B) between the two contour layers. **cutmask** is typically used for cutmask designs.
- The **keepmask** argument adds an additional check for pinching conditions in a contour defined as the intersection (A AND B) between the two contour layers. **keepmask** is used in keepmask designs.

The **overlay**, **cutmask**, and **keepmask** arguments require a **max_error** distance in microns, which is used as the maximum expected shift of an individual mask (and hence of an individual contour) from its nominal position, assuming that each mask can randomly shift by **max_error** / 2 in any direction. In effect, in addition to checking for pinch conditions on the nominal contour (**contour_layerA**), this command checks a second, shift-sensitive contour created for all possible unions (**overlay**), differences (**cutmask**), or intersections (**keepmask**), which is produced by shifting the masks from their nominal positions.

Tip

 Error markers resulting from an overlay, cutmask, or keepmask pinch command may be the result of either the normal pinch check, or the overlay, cutmask, or keepmask pinch check.

- $\{simple_constraint \mid annotated_constraint\}$ separation *sep_microns* [runlength { $>$ | \geq } *length* [max_runlength *rl_max* [exception *rl_ex*]]]

An optional keyword set that defines the soft bridging conditions and how to mark them.

- *simple_constraint*

Takes the form:

$<[=] distance_microns$

Checks for soft pinching conditions, defined as a contour that is less than (or equal to if \leq is specified) *distance_microns* from the nearest contour.

Note

 If a constraint is not specified, the pinch command only detects hard pinching.
Using ≤ 0 for *simple_constraint* will cause an error.

- *double_constraint*

Takes the form:

$>[=] lower_bound <[=] upper_bound$

Separates out soft pinching errors into bins. However, since this method will return false positives around line ends, a *double_constraint* must be accompanied by a *filter_layer*. The filter should exclude line ends according to the separation value.

- *annotated_constraint*

Note

 *annotated_constraint* arguments require the *target_layer* input argument to be the output from a previous [annotate](#) command.

Takes the form:

default $<[=] distance_microns annotation_type1 \{<[=] distance_microns2 \mid ignore\}$
[*annotation_type2* { $<[=] distance_microns3 \mid ignore\}$...]

When one or more *annotation_type* arguments are supplied (only “orientation” is used for pinching checks), it uses the argument as the selection constraint for qualifying shapes that have the matching annotation type.

Only annotation types supported by the specific annotation command that created the layer can be detected. In other words, “orientation” can have at most three (default, horiz_edge, and vert_edge).

The default constraint is used for any shapes that do not have a specified *annotation_type*, and is the equivalent of a *simple_constraint*.

If an *annotation_type* constraint is set to ignore, error shapes that contain an instance of the *annotation_type* are skipped.

The *sep_microns* argument is used as a minimum separation distance between measurement points to avoid detecting part of the same contour.

Tip

- i** The [critical_dimension](#) setup command can be used in place of the separation argument. The value used for *sep_microns* is $3 * \text{critical_dimension_value}$.
-

If runlength is also specified, the command ignores soft pinches that have a run length (the length along the contour border that violates the error constraint) smaller than the specified *length*. In other words, only errors that have a run length that is greater than (or equal to if \geq is specified) the specified *length* are reported.

runlength is not compatible with **overlay** or **cutmask** mode.

If the max_runlength argument is specified, errors with a run length larger than *rl_max* are considered exceptions, and *rl_ex* is used as the output value instead of their value. In order to report consistent results on tile boundaries, *rl_max* is added to the interaction distance, so using too large a value for *rl_max* can degrade performance.

max_runlength is only used if the runlength property is requested in the property block. The default value for *rl_ex* is 100 microns.

- projecting [*projection_length*]

An optional argument that limits the command to check only those contour edges with associated target edges that are parallel and project on each other. The target edges must project for a minimum *projection_length* in microns (default is [1.0 dbu](#)) for the command to check the distance between the contour edges. Otherwise, the contour edges are not considered in the check.

- max_tolerance *tol_microns*

An optional argument specifying the maximum search range that this command will attempt to find a matching contour in. If there are no contours within *tol_microns* of a target edge, the edge is marked as an error. Siemens EDA recommends setting the max_tolerance argument to be equal (or nearly equal) to the minimum feature size (default is [.15 microns](#)):

- Setting max_tolerance to a value smaller than the minimum feature size will start generating false errors; setting it to less than half of the CD is definitely too small.
- Setting max_tolerance to a value greater than the minimum feature size will increase the CPU computation time; setting max_tolerance larger than a feature extent is not recommended.
- The output shape parameters max_edge and output_expand take their default values from max_tolerance, and may give unusable results if you set max_tolerance too large.

Tip

- i** The [critical_dimension](#) setup command can be used in place of this argument. The *tol_microns* value used for max_tolerance is $1 * \text{critical_dimension_value}$.
-

- `max_edge edgelen_microns`

An optional argument that specifies the maximum length of a contour edge or target edge before it is subdivided. Smaller values of `max_edge` give a finer granularity at the cost of run time. Default is the value set for max_tolerance.

- `output_expand width_microns [width_microns2]`

An optional argument that specifies the maximum width an error marker can expand from the marked error contour. If only one `width_microns` value is specified, it sets how far the marker expands inside the target. If `width_microns2` is also specified, it sets how far the marker expands outside the target.

Values far smaller than the CD mark the error more precisely; larger values (slightly larger than the CD) produce larger markers that are easier to see. Default is the value set for max_tolerance.

For contact or via layers only, set the additional `width_microns2` argument to 0 to avoid merging together markers at the corner of contacts. Otherwise, leave the value at the default (1dbu).

- `pinpoint [markers | gauges] [size]`

An optional argument that replaces the normal output with markers showing the exact location of the minimum distance measured for the operation.

Note

This option has been superseded by the [pinpoint_output block](#). It is currently retained for backwards compatibility. It is not the same as the `pinpoint_output` block. If you use the `pinpoint` argument, it must appear before the error-centric section starts or it causes an error.

- `slivers {off | {size [ignore] | blots size1 [ignore] holes size2 [ignore]} [manhattan | skew]}`

An optional argument that filters out all narrow contour holes before checking for pinching. By default, sliver detection is off. The parameters specified determine how the slivers are handled.

- `size [ignore]` — Calibre OPCVerify builds a bounding box around every contour hole. If the smaller side of this bounding box is less than `size` microns, the hole is considered to be a sliver and is removed from the contour layer.

When the `ignore` option is specified, the slivers are discarded. Otherwise, sliver holes are marked by `pinch` commands.

- `blots size1 [ignore] holes size2 [ignore]` — Different `size` criteria can be specified for blots (polygons) and holes.

In the default manhattan mode, the sliver bounding box is aligned with the X and Y axes. When the mode is skew, Calibre OPCVerify builds and tests two bounding boxes; one axes-aligned and one rotated by 45°. It returns the smaller bounding box.

- `small_holes extent` [extents | {extents centers *size*} | suppress | manhattan]

An optional argument that sets how to mark contour holes with extents less than or equal to the *extent* value:

- extents — Output minimum bounding boxes of each polygon rather than the polygons themselves.
- extents centers *size* — Output marker squares in the centers of the bounding boxes;
- suppress — Does not output anything for holes with extents $\leq extent$.
- manhattan — Converts to Manhattan-type (45 and 90 degree shapes) and output the contour polygons.

Default is the value of max_tolerance.

- `cd_search_dist cdist`

An optional argument that specifies the maximum search distance in microns used when computing values for the “min” with “target_cd” property. If no nearby edges are detected within *cdist*, the target_cd value is set to 0.0.

Excessively large values for this argument can affect run-time performance.

Tip

The `critical_dimension` setup command can be used in place of this argument. The *cdist* value used for `cd_search_dist` is $2 * critical_dimension_value$.

- property ‘{’ min
[target_cd]
[shift_dir]
[runlength]
‘}’

An optional argument that specifies a property operation for the returned output of this command. The minimum value can be computed for this command with the min keyword.

If the target_cd qualifier is added after the min keyword (it must also be on its own line), the local target CD is recorded in target_cd as the closest distance from the target edge to any other target edge, measured normal to the error edge. The exceptions are as follows:

- For errors marked on image contours, target_cd is reported as 0.0.
- For error edges that are not 45 degree-skew lines, target_cd is reported as 0.0.

If the shift_dir qualifier is added after the min keyword (it must also be on its own line), shift_dir contains the direction in degrees that `contour_layer2` must be shifted to achieve the reported min property. The shift_dir qualifier can only be used when **overlay** or **cutmask** mode (see above) is active. The values range from -180 degrees to +180 degrees, starting at 0 degrees indicating a shift of `contour_layer2` rightward, +90 is upward, and -90 is

downward. If a shift in multiple directions can result in the min value, an arbitrary shift_dir value is returned. If the min value is possible in all directions, shift_dir contains a value of 360.

The size of the run length (the length of the contour that violates the distance constraint) for the error is returned when the optional “runlength” keyword is specified. A hard pinch returns a value of zero in the runlength property. Using the runlength property requires the use of the runlength keyword in the constraint.

If one error marker corresponds to several separate pinches, its runlength property is the maximum run length of the pinches inside the marker.

The braces ({{}) around the keyword(s) are required syntax.

Property operations require a distance constraint.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

- *add_properties block*

An optional argument that specifies an add_properties block. For more information, see “[add_properties Block](#)” on page 91.

- *pinpoint_output block*

An optional argument that specifies how pinpoint markers are drawn for this command. For more information, see the section “[Pinpoint Output Block](#)” on page 102.

Note

This command also has a pinpoint argument, only retained for backwards compatibility; it is a separate argument and is not the same as the pinpoint_output block.

Pinpoint output has special behavior for the pinch command. For a soft pinch, the min property is always a distance between two points, pt1 and pt2, located on edges of the contour layer.

- When the output mode is markers (the default), the marker is a tiny square, placed exactly midway between points pt1 and pt2.
- When the output mode is gauges, the marker is a rhombus, connecting pt1 and pt2. The gauges mode is recommended only for small layouts.

The *size* argument specifies the side of the squares (for markers) or the smaller diagonal of the rhombus (for gauges). It has a default of 2 dbu.

For a hard pinch, there is no obvious point to pinpoint. A square marker is placed in the center of the bounding box of the error shape in both markers and gauges modes. For L-shaped or n-shaped errors, the marker can easily end up outside the error shape.

Examples

A hard pinching check:

```
setlayer short = pinch m1 img_m1_ctr output_expand 0.005 max_tolerance  
0.05 property {min}
```

A soft pinching check:

```
setlayer soft_pinch = pinch m1 img_m1_ctr < 0.050 separation 0.240\  
max_tolerance 0.05 max_edge 0.050 output_expand 0.005 property {min}
```

Tip

 Remember to check your max_tolerance value. The value specified for max_tolerance is highly dependent on your minimum CD values; the examples above are specific to the test design and may not work as expected if your design CD is not 0.05 microns.

A double-sided constraint check is as follows:

```
setlayer le_fltr = filter_generate trgt expand 0.0015 line_end 0.02 0.012  
setlayer p1 = pinch tgt outside le_fltr cntr <= 0.04 separation 0.08  
setlayer p2 = pinch tgt outside le_fltr cntr >0.04 <= 0.042 \  
separation 0.08
```

In this example, p1 is a standard pinch check, and p2 is a double-sided constraint. It is important to understand that without the filter created with the [filter_generate](#) command, p2 returns a false positive result for the line end.

Related Topics

[bridge](#)

[extra_printing](#)

[not_printing](#)

pinch_tolerance

Verification control

Checks for inner tolerance band violations.

Tip

i The **pinch_tolerance** command is not the recommended method to find pinch errors. Use the **pinch** command instead.

See the section “[Replace Older OPCVerify Bridge and Pinch Detection Methods With Bridge and Pinch](#)” on page 498 for related best practice information.

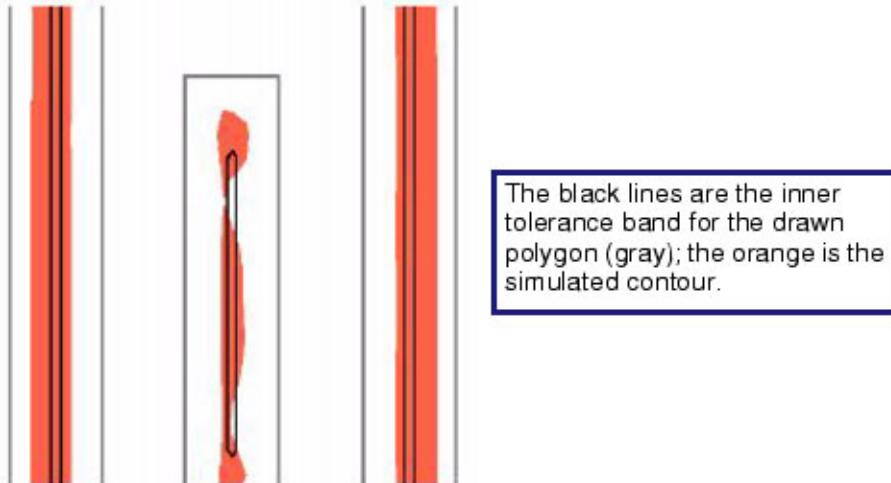
Usage

pinch_tolerance *contour tolerance_zone*

Description

Checks for inner tolerance band violations. This can be performed as an embedded operation (using setlayer). It requires a precomputed inner tolerance zone to be supplied.

Figure 4-45. Pinch Tolerance Example



Arguments

- ***contour***

A required argument specifying a simulation contour layer to check. You can generate a contour layer using [Image Operations](#).

- ***tolerance_zone***

A required argument specifying a layer with a generated inner tolerance zone to check for bridging. You can generate the inner tolerance zone on an input layer using a [build_tolerance](#) layer operation.

Examples

The following example code creates a pinch tolerance zone:

```
setlayer ti = build_tolerance inner TARGET max .04 stepsize .01
setlayer ti1 = cornerchop ti .02 .02
setlayer i1 = image optical o1 dose 1.04 aerial .03
setlayer pinch1 = pinch_tolerance i1 ti1
```

polygon_extent

Verification control

Returns polygons based on their extents.

Usage

polygon_extent
 layer constraint

Description

This command returns polygons on the supplied *layer* that meet the *constraint* for its extent (maximum distance in x or y dimensions).

Concurrency Support: Up to 16 **polygon_extent** commands run concurrently when used on the same input *layer*.

Arguments

- *layer*
A required argument specifying a layer with polygons to measure.
- *constraint*
A required argument specifying the constraint to test on the polygons.

Examples

Given the layer cntr, the following commands return shapes based on different extents:

```
setlayer ext_lt_0.05 = polygon_extent cntr <= 0.05
setlayer ext_0.05_to_0.1 = polygon_extent cntr > 0.05 <= 0.1
setlayer ext_0.1_to_0.2 = polygon_extent cntr > 0.1 <= 0.2
setlayer ext_0.2_to_0.5 = polygon_extent cntr > 0.2 <= 0.5
setlayer ext_0.5_to_2 = polygon_extent cntr > 0.5 <= 2
setlayer ext_gt_2 = polygon_extent cntr > 2
```

pvband

Verification control

Generates a process variation band from multiple contours. It forms the OR of the XOR of each unique pair of input image layers.

Usage

pvband *layer1* ... *layerN*

Arguments

- ***layer1* ... *layerN***

A set of required arguments for the simulation contour layers to check, separated by spaces. You must have previously generated the contour layers.

Examples

The following example code creates a group of contour layers, and then runs the pvband command on them:

```
for {set i 0} {$i <= 5} {incr i} {
    set j [expr .85+($*.05)]
    setlayer contour$j = image optical f0 dose $j resist_model vt5
}
setlayer pvout = pvband contour.85 contour.9 contour.95 contour1.0 \
contour1.05 contour1.1
```

pwcheck

Verification control

Checks if the calculated common process window fits inside the process window (PW) in the user-specified location.

Usage

```
pwcheck layer_target
  image_set image_set_name [focus_units {nm | um}]
    [rectangle | ellipse]
    [model_poly8 | model_poly13]
    {{min_dose min_dose max_dose max_dose} | {dose_latitude dose_latitude}}
    {{min_focus min_focus max_focus max_focus} | {dof dof}}
  max_search search_microns
    [tolerance [absolute] tol]
    [[not] {inside | outside} {layer_target_filter}| gauge gauge_layer]
    [target_cd {nominal_contour | drawn}]
    [dof_spacing spacing_microns]
    [width | space | auto_cd]
    [prob_fail_options]
    [trim_ends trim_microns]
    [separation sep]
    [exception [also [property_value prop_value]]]
    [common_ldof file_name [dose_tol dose_tol] [focus_tol focus_tol]]
    # error-centric section
    [property {'
      [focus_min]
      [focus_max]
      [dose_min]
      [dose_max]
      [focus_nom]
      [dose_nom]
      [dof]
      [dof_max]
      [cd_type]
      [prob_fail]
    ‘}’ [classify, limit, or histogram block]]
```

Description

This command finds contours that are not inside the given common process window. You define the common process window by using an [image_set](#) containing at least nine contours that have a varying dose and focus.

Concurrency

Two pwcheck commands will run concurrently if they differ only in their [image_set](#) arguments of dose or focus, or the rectangle, ellipse, width, space, exception, trim_ends or property arguments to the command.

Arguments

- ***layer_target***

A required argument specifying the drawn target layer.

- ***image_set image_set_name* [focus_units {nm | um}]**

A required argument that specifies a previously defined [image_set](#) for use with this command. The [image_set](#) must have at least 9 images with different dose and defocus conditions.

The focus_units option can be used to set the output for properties, common dof, and sgd output in the specified unit type (default is um). [image_set](#) is presumed to supply focus in nm.

- **rectangle | ellipse**

An optional argument that specifies the shape of the Common Process Window (CPW) to use. Ellipse is the default, starting with the 2019.2 release.

- **model_poly8 | model_poly13**

An optional argument specifying the polynomial model form used to fit the CD versus dose and focus data. [model_poly13](#) is the default, starting with the 2019.2 release.

[model_poly8](#) — The [model_poly8](#) option uses eight terms and is second order for dose and focus. It requires at least nine unique combinations of dose and focus contours, with at least three different dose conditions and three different focus conditions. The recommended use case has at least 25 contours (five dose x five focus).

[model_poly13](#) — The [model_poly13](#) option uses 13 terms, and is third order for dose, fourth order for focus. It requires at least 14 combinations of dose and focus contours, with at least four different dose conditions and at least five different focus conditions. The recommended use case has at least 49 contours (seven dose x seven focus).

Note



Due to the 2019.2 default change, pwcheck now requires a minimum of 14 images, not 9 images as it used to be before the 2019.2 release, unless the [model_poly8](#) option is specified explicitly.

- ***min_dose min_dose max_dose max_dose***

A required argument that specifies the extent of the CPW along the dose axis.

- **dose_latitude dose_latitude**

A required argument that specifies an alternate method for setting the required **min_dose** and **max_dose** arguments:

- $\text{min_dose} = 1 - \text{dose_latitude}/2$
- $\text{max_dose} = 1 + \text{dose_latitude}/2$

- **min_focus min_focus max_focus max_focus**

A required argument that specifies the extent of the CPW along the focus axis.

- **dof dof**

A required argument that specifies an alternate method for setting the required **min_focus** and **max_focus** arguments.

- $\text{min_focus} = -\text{dof}/2$
- $\text{max_focus} = +\text{dof}/2$

Note

 The dof property is computed using a fixed process window with dose centered at 1 and focus centered at 0. However, the reported dof property is computed using a floating process window, allowing the center dose and focus to vary. Thus, dof is equivalent to the maximum dof.

- **max_search search_microns**

A required argument that specifies the maximum search distance in microns from the target to contours.

- **tolerance [absolute] tol**

A optional argument that specifies the window of acceptable CD tolerance around the nominal CD. The default is 0.1, which means that a 10 percent CD variation from nominal is acceptable. The nominal CD is determined by the **target_cd** argument.

If ‘absolute’ is specified, the **tol** value is treated as the value of the CD tolerance from nominal in microns instead of a percentage.

- **[not] {inside | outside} {layer_target_filter | gauge gauge_layer}**

A optional argument that specifies a filter for **layer_target** edges. If gauge is specified, a **gauge_layer** filter must have been previously defined using the **RET INPUT** command.

- **[target_cd {nominal_contour | drawn}]**

An optional argument that specifies how to calculate the nominal CD value.

- By default (nominal_contour setting), the nominal CD is measured using the contour in the **image_set** with dose 1 and defocus 0 conditions. If the image set has no such image in its list, this command generates a parsing error.
- Specifying drawn calculates the nominal CD from the target layer.

- **dof_spacing *spacing_microns***
A optional argument that specifies the maximum gauge spacing distance in microns.
- **width | space | auto_cd**
An optional argument that sets whether the operation works with an internal (width) or external (space) CD. Width is the default. If auto_cd is specified, this command also requires a gauge file, because it uses the gauges in the gauge file to automatically detect CDs.
- ***prob_fail_options***
An optional arguments that define the normal distribution used to model the dose and focus for reporting with the prob_fail property. The arguments are:
 - **nomfocus** — The nominal focus value in microns or nanometers, based on the focus_units argument. (Default: 0.0)
 - **stdfocus** — The standard deviation of focus in microns or nanometers based on the focus_units argument. This argument is required when the prob_fail property is requested.
 - **nomdose** — The nominal dose. (Default: 1.0)
 - **stddose** — The standard deviation allowed for the dose. This argument is required when the prob_fail property is requested.
- **trim_ends *trim_microns***
An optional argument that specifies that the output boxes from this command are trimmed by the specified value along the target edge if the output box is longer than $2*trim_microns+2dbu$ along the target edge. This implies that trim_microns cannot be larger than the dof_spacing argument (*microns_spacing*) - 2dbu.
- **separation *sep***
An optional argument that specifies the minimum target separation required to measure a CD. Target separation is defined as the distance along the target polygon between the ends of the target CD. Locations with a target separation less than *sep* in microns are excluded from the measurement. This option allows you to exclude line and space ends from CD measurements without using a separate filter layer. Default is = 0 (no separation).
- **exception [also [*ex_focus ex_dose*]]**
An optional argument that specifies exception mode. Output is generated only around the gauges which where a part of a CD, but did not intersect enough image contours.

If exception also is specified, then when pwcheck detects an exception, the error marker is output with focus_min and focus_max set to the artificial value specified for *ex_focus*, and dose_min and dose_max are set to the artificial value specified for *ex_dose*. The default values for *ex_focus* and *ex_dose* are -1.

- common_ldof *output_file* [*dose_tol dose_tol*] [*focus_tol focus_tol*]

An optional argument that instructs the command to inscribe the configured figure (rectangle or ellipse) for process windows such that it maximally satisfies all output markers.

The result is written to the text file specified in *output_file*.

If the result is valid, the following format is used:

```
type {rectangle | ellipse}
dof {DOF_value}
[ WARNING: warning_report ]
dose_nominal {center_of_figure_in_dose_coordinates}
focus_nominal {center_of_figure_in_focus_coordinates}
```

Calibre OPCVerify may add one of the following *warning_report* messages:

- process window is out of the process condition area, results may be not accurate
- process window is greater than process condition pattern, results may be not accurate
- process window is not accurate

If the result is not valid (such as if there is no way to inscribe a process window figure), the following file format is written instead:

```
type {rectangle | ellipse}
exception: {exception_code}
```

The following exception codes are generated:

- No gauges with valid DOF measurements were found.
- No intersection between individual gauge PWs has been found.
- The gap in the common PW is not big enough to inscribe a figure with the given dose latitude.

An ellipse or rectangle can be inscribed with a given tolerance, defined with the optional *dose_tol* and *focus_tol* parameters. Tolerance is defined in dose versus focus units. The default value is automatically adjusted to keep the number of dose and focus steps below 250, and rounded to a multiple of 1, 2, or 5. For example, the tolerance for a defocus range of +/- 0.08µm will be (0.160/250)= 0.00064 which is rounded up to 0.001µm. Note that you should carefully choose tolerance values, because smaller tolerance values can affect memory consumption on the primary host as well as total processing time. Tolerance does not affect rectangle inscription; rectangles are inscribed with maximal accuracy with no additional resources.

- property ‘{’
[focus_min]
[focus_max]

```
[dose_min]  
[dose_max]  
[focus_nom]  
[dose_nom]  
[dof]  
[dof_max]  
[cd_type]  
[prob_fail]  
'}'
```

An optional argument set that specifies one or more property keywords. The focus_min and focus_max properties show the DOF the user would have to have in order to achieve the requested CD tolerance with the dose latitude specified in the command arguments.

The dose_min and dose_max properties show the dose latitude the user would have to have in order to achieve the requested CD tolerance with the DOF specified in the command arguments.

The dof property shows the DOF *centered at the nominal dose and focus* that the user would need in order to achieve the specified CD tolerance with the dose latitude specified by the command arguments.

The focus_nom, dose_nom, and dof_max properties show the center and width of a maximal figure (ellipse or rectangular) fitted between CD tolerance curves in focus versus dose. The dose latitude of the figure is specified in the command arguments.

The cd_type property shows the type of CD; 0 if width was specified or detected, 1 if space was specified or detected.

The prob_fail property shows the probability of failure as a value between 0 and 1, based on the simulated process window and distribution of focus and dose defined by the nomdose, stddose, nomfocus, and stdfocus parameters.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

shadow_bias

Verification control

Defines a shadow bias model options block.

Usage

shadow_bias layer MAP layer OPTIONS opts_name

This command also requires an options block immediately following the command.

Use the following syntax:

```
options opts_name '{  
    version 1  
    layer layer opc  
    shadow_bias_model {sb_model | litho_model_name}  
    [shadow_bias_direction {forward | reverse}]  
    [mrc_rule ...]  
    [step_size value]  
'}
```

Description

This command applies shadow biasing to the input layer, using the given model. It is primarily designed for use with EUV models. It empirically models thick mask effects.

Note the following caveats:

- Masks which include shadow bias correction cannot be used directly in simulation (for example in setlayer image). They will produce incorrect images. All simulations should use the OPC mask before shadow bias correction. This mask can be recovered from a shadow-bias-corrected mask by performing a preliminary run of setlayer shadow_bias before simulation, using “shadow_bias_direction forward” to reverse the correction.
- Do not use this command on contours.
- This command only works on databases for which all cells are instantiated at most once (quasi-flat). The run will abort at runtime if a non-flat database is supplied.

This command works inside the MDP EMBED command and with the EXPAND CELL “*” option in SVRF. The maximum bias limit (specified in **step_size**) is 50 nm. The fragment length is 1 micron.

For best practices, shadow bias effects have been moved to an automatic biasing function that is activated by putting the shadowbias file inside the lithomodel. This command is retained for backwards compatibility.

Arguments

- ***layer***
A required argument specifying a layer to perform shadow bias on. Use the same ***layer*** for **shadow_bias layer**, **MAP layer**, and the **layer layer** keywords inside the **options** block.
- ***options opts_name***
A required argument that defines an options block, which must be surrounded with braces ({}).
- ****version 1****
A required argument specifying the version of the shadow_bias algorithm. This argument must always be set to 1.
- ****shadow_bias_model {sb_model | litho_model_name}****
A required argument for shadow bias developers.
In non-litho model flows, use the ***sb_model*** argument to specify the name of a loaded shadow bias model. The model must have been previously loaded with a [shadow_bias_model_load](#) command.
In litho model flows, use the ***litho_model_name*** argument to specify the name or path of a litho model directory. This directory must be found in the modelpath or be a fully-specified path. The Litho Model must contain a shadow bias model.
- ***shadow_bias_direction {forward | reverse}***
An optional argument that specifies the bias direction.
If “forward” is specified, the biases specified in the model file are applied to corresponding edges. This should be used for verification.
If “reverse” is specified, the biases are multiplied by -1 before being applied. Use “reverse” on the OPC output post-OPC to perform shadow compensation.
- ***mrc_rule***
An optional argument that specifies rules that constrain mask movements. Refer to the [mrc_rule](#) Calibre nmOPC documentation for further descriptions of MRC rules.
- ***step_size value***
A required argument specifying the amount to attempt to bias the edges by in nm. The maximum value is 50 nm.

shift

Verification control

Shifts the shapes on the specified layer by the specified amounts. It is designed to be used on misaligned masks.

Usage

shift *layer* by *x-shift* *y-shift*

Arguments

- ***layer***
A required argument, specifying the layer to be shifted.
- ***x-shift* *y-shift***
A required pair of arguments, specifying the shift amount in the X and Y directions in microns.

Examples

```
setlayer phase_error = shift contour by .02 .02
```

show_annotation

Verification control

Shows markers on annotated layers.

Usage

show_annotation *annotated_layer* *annotation_class*

Arguments

- ***annotated_layer***

A required argument specifying the layer to scan, which must have previously been the output of an [annotate](#) statement.

- ***annotation_class***

A required argument specifying the type of annotation class to display. The appearance of the marker is set by the annotation type. Only one ***annotation_class*** can be supplied per **show_annotation** command.

- **double_via** — Connects the marked double-vias with a box.
- **horiz_edge** — Outputs 2-dbu wide rectangles marking each edge that was assigned the **horiz_edge orientation_type**.
- **vert_edge** — Outputs 2-dbu wide rectangles marking each edge that was assigned the **vert_edge orientation_type**.

Examples

```
// double_via annotation

setlayer show_via = show_annotation vialayer double_via
// orientation type annotation places two types of annotation
setlayer xytarget = annotate target orientation horiz_edge 0.10 \
    vert_edge 0.16
setlayer hzr_show = show_annotation xytarget horiz_edge
setlayer vrt_show = show_annotation xytarget vert_edge
```

size

DRC-type operation

Performs the sizing operation on the input layer, expanding or shrinking polygons on the input layer by the specified amount.

Usage

size *input* by *val*

Arguments

- *input*

A required argument, specifying the input layer to size.

- *val*

A required argument specifying the sizing value, as a floating point value in user units (default is microns). The input layer will be sized by *val**precision (rounded to the closest integer value) database units.

Examples

```
setlayer upsized = size target by .01
```

sraf_apa

Verification control

Computes the average printability analysis (APA) for SRAF regions.

Note

 This command is deprecated and is replaced with the [apa_check](#) command. It will be removed in an upcoming release.

Usage

```
sraf_apache image_layer constraint
  [[not] {inside | outside} filter_layer]
  sraf_layer sraf_layer [size_by val_um]
  stochastic_model modelName
  # error-centric section
  [property '{'
    apa [apa_actual] [apa_area] [apa_pixel_count]
  '}']
  [classify, limit, or histogram block]
```

Description

The `sraf_apache` command computes and checks the average printability analysis (APA) function value of the SRAF regions in the image layer. The value is between 0 and 1.0 and is normalized by the SRAF region area.

This command also requires a stochastic model to have previously been loaded with a [stochastic_model_load](#) setup command.

Arguments

- ***image_layer***
A required argument specifying a previously-generated [image](#) layer to be checked.
- ***constraints***
A required argument specifying a standard Calibre OPCVerify constraint to be applied to the computed APA value. Two-sided constraints are supported. A computed APA value satisfying the constraint is output as an error marker.
- ***sraf_layer sraf_layer [size_by val_um]***
A required argument specifying a layer with regions to perform the APA computations on. The layer can be pre-scaled using the optional `size_by` argument.
- ***stochastic_model modelName***
A required argument specifying the name of the stochastic model to use for computations. The model must have been previously loaded using the [stochastic_model_load](#) command.

- property '{' apa [apa_actual] [apa_area] [apa_pixel_count] '}'

An optional argument defining a property block, with property values enclosed in braces.

- apa — A required keyword when a property block is declared, the APA property value contains the result of the APA computation normalized over the area of the SRAF regions. Its value is dimensionless and is normalized by the SRAF region.
- apa_actual — An optional keyword that returns the raw computed APA value without any normalization.
- apa_area — An optional keyword that returns the area of the layout region from which the APA computation was done.
- apa_pixel_count — An optional keyword that returns the number of simulation grid points used in the APA computation.

- *classify, limit, or histogram block*

An optional argument that specifies an error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

sraf_print_flux

Verification control

Computes intensity flux for SRAFs on an image layer.

Usage

```
sraf_print_flux image_layer flux_constraint
  [[not] {inside | outside} filter_layer]
  {{sraf_layer | asraf_layer} sraf_layer [size_by val_um]
   [target_layer target_layer [size_by val_um]]]
  max_extent max_extent_um
  [grid_upsample upsample_fac]
  [flux_function path_to_object_file function_name]
  [constraint {flux | norm_flux}]
  [exceptions_also] [exception_flux flux_value]
  [output_type {deangle [max_dev dev_um] | polygons | extents}]
  # error-centric section
  [property '{'
   flux
   [norm_flux]
   [area]
   '}']
  [classify, limit, or histogram block]
```

Description

The sraf_print_flux command computes an integrated intensity flux function for extra printing SRAFs and checks it versus a constraint.

The command checks extra printing for either positive or negative SRAFs. Checking both kinds of SRAFs requires separate runs of the command.

Extra Printing Polygon Definitions

Extra printing polygons are defined depending on the mode selection of sraf_layer or asraf_layer:

- If sraf_layer is specified, extra printing of positive SRAFs will be checked.
 - Positive SRAF extra printing polygons are defined as those contour polygons that interact with the sraf layer.

Note

 This selection may include portions of the main features in the case where SRAF features have merged with the main features.

- If target_layer is also specified, positive SRAF extra printing polygons are also defined by selecting contour polygons that do not interact with the target layer. This may add some unwanted extra printing polygons not associated with SRAFs to the selection.
- If asraf_layer is specified, extra printing of negative ASRAFs will be checked.
 - ASRAF extra printing polygons are defined as all holes in the contour polygons for which the hole interacts with asraf_layer.
 - If target_layer is also specified, ASRAF extra printing polygons are also defined by selecting all holes in the contour that *do not* interact with holes in the target. This may add some general unwanted holes not associated with ASRAFs to the selection.

Flux Function Definition

The flux function is $F(I')$, where I' is $\text{abs}(\text{intensity}-\text{threshold})$.

The threshold value is taken from the image command that created the input image layer. It is the same threshold used in contour generation. It is either an aerial or a resist threshold depending on the image command.

Flux integration is performed by summing $F(I')$, evaluated at the center of the pixel over each grid pixel whose center is inside the extra printing polygon. The accuracy of integration can be improved by requesting upsampling of the grid before integration.

The default flux function is $F(I') = I'$, using a simple summation. User-defined functions are supported.

Extra printing polygons whose extent is larger than the max_extent argument are not checked, but may be reported and assigned an exception flux property.

Arguments

- ***image_layer***

A required argument specifying the image layer to be checked. This layer must be the output of a previously specified **image** command.

- ***flux_constraint***

A required argument specifying standard constraint to be applied to the flux properties. Two-sided constraints are supported.

- ***[[not] {inside | outside} filter_layer]***

An optional keyword specifying a filter. By default, only the extra printing polygons lying completely inside or completely outside the filter_layer are selected from the image layer. Specifying “not” inverts the selection.

- **{sraf_layer | asraf_layer} sraf_layer [size_by val_um]**

A required keyword setting the selection mode. If sraf_layer is specified, the *sraf_layer* argument is treated as a positive SRAF layer to be checked.

If asraf_layer is specified, the sraf_layer is treated as a negative SRAF (ASRAF) layer to be checked, and sraf_layer is used to select the polygons to be checked.

Optionally, sraf_layer may be sized by *val_um* before it is used for polygon selection.

- **target_layer target_layer [size_by val_um]**

An optional argument specifying a target layer. This layer is used to augment the selection of polygons to check.

Optionally, target_layer may be sized by val_um before it is used for polygon selection.

- **max_extent max_extent_um**

A required argument that specifies an exclusion extent limit. Extra printing polygons whose extent (in x or y) exceeds the specified value are not checked.

Note



max_extent should be set as small as possible. If the value exceeds 0.2um, performance may be adversely affected.

- **grid_upsample upsample_factor**

An optional argument specifying an integer grid upsample factor ≥ 1 . The default is 1. If specified > 1 , the intensity grid will be upsampled by this factor to provide more accurate integration.

The default size of the intensity grid pixels is listed in the SE: section of the transcript as “SE: CONTOUR GENERATION GRID.” It is derived from the Nyquist pixel size by applying the imagegrid aerial and final_upsample setup commands.

- **flux_function path_to_object_file function_name**

An optional argument specifying a user-defined flux function. *path_to_object_file* specifies the location and name of the source (.so) file, and *function_name* specifies a C++ function within the .so file.

Note



A single object file may contain several functions to be used for different usages of *sraf_print_flux*.

path_to_object_file may be an absolute path beginning with "/", or it may be an unqualified filename containing no "/" characters. In the latter case, it is found by looking in the search paths specified by the environment variable \$MGC_LIB_PATH.

For example, if the .so file is in the main working directory of the run, a bash run script might specify:

```
export MGC_LIB_PATH=${PWD}
```

If *path_to_object_file* is specified as a relative path such as *./my_lib.so*, it is searched for in the working directory of each Calibre instance. This search fails for MTFlex runs.

- constraint {flux | norm_flux}

An optional keyword specifying whether the constraint is applied to the flux property (default) or the norm_flux property in the property block.

- exceptions_also [exception_flux flux_value]

An optional keyword that changes the behavior of max_extent polygons.

By default, extra printing polygons that are excluded by the max_extent criterion are discarded and not reported. If exceptions_also is specified, these polygons are reported, and all of their properties are assigned a special exception value (9999, or a value specified by exception_flux).

- output_type {deangle [max_dev dev_um] | polygons | extents}

An optional argument that sets the appearance of output markers.

- The default is deangle, which outputs error markers as deangled versions of the extra printing polygons. The maximum deviation for deangle can be specified in microns using the optional max_dev argument. The default for max_dev is 0.010.
- If polygons is specified, the error markers are the polygons themselves. This may produce large output layers. Polygons can have many skew edges and require a lot more memory to store.
- If extents is specified, the error markers are the extents of the polygons. Extents require the smallest memory to store, but there is a possibility that error markers may merge together.

- property {'
flux
[area]
[norm_flux]
'}

An optional argument specifying a property block that associates property values with the output polygons.

The flux property is the result of the intensity integration described above. Formally it is:

$$\text{flux} = \sum(F(I')) dx dy$$

where dx and dy are the pixel dimensions and the sum is over intensity pixels whose center is inside the polygon.

If I' is regarded as dimensionless, the flux property is defined in units of square microns. This means that the property is dimensionally independent of the pixel size and upsampling factor.

The optional area property is the area in square microns of the extra printing polygon. Calibre OPCVerify computes the value before any deangling.

The optional norm_flux property is the flux divided by the area value. It is dimensionless.

- *classify, limit, or histogram block*

An optional argument that specifies an optional error-centric data property collection operation to be performed on the results of the command. For more information, see “[Error-Centric Section Blocks](#)” on page 69.

tilegen

Setup command

Creates an annotated layer to display tile and cell boundaries with properties.

Usage

```
setlayer out_layer = tilegen value [property '{'  
    keywords  
  '}']
```

Description

Calibre OPCVerify breaks input into smaller areas, referred to as tiles. The setlayer tilegen command creates a supplementary output layer that displays the tile and cell boundaries that were used in the run.

Arguments

- ***out_layer***

A required layer name for the new layer. This layer must be used in subsequent processing in the SVRF file for the operation to occur.

- ***value***

A required numeric argument that specifies the style of output. The ***value*** specified must be between -0.1 and 0.1.

-0.1 to <0— The tile is sized by ***value*** and written to ***out_layer*** as polygons.

0 to 0.1 — The boundaries between valid tile regions are sized, and the boundaries are written to ***out_layer***.

- property {
 keywords
 }

An optional property block specifying tile properties to write to an RDB database for viewing with Calibre RVE.

Note

 When using a property block, the ***value*** must be negative.

The following are valid values for keywords:

location — Outputs the tile location.

lvheap_current — Outputs the amount of memory used in processing the tile, in MB.

lvheap_max — Outputs the maximum amount of memory in MB.

remote — Outputs the cell name.

rss_max — Outputs the maximum amount of RAM used by Calibre in MB.

This argument follows the formatting rules of Calibre OPCverify property blocks, as described in “[Error-Centric Section Blocks](#).”

Examples

Example 1 — Grid Output

To display the tile boundaries as a grid:

```
setlayer t1 = tilegen 0
```

Example 2 — Polygon Output

To display the tiles with 0.1 microns between edges:

```
setlayer t1 = tilegen -0.05
```

Example 3 — Property Blocks

To output a tile layer with information to aid debugging:

```
setlayer t2 = tilegen -0.1 property {  
    location  
    remote  
}
```

Be sure to include it in a rule check and DRC CHECK MAP statement to write the data to the RDB; for example,

```
T2 = LITHO OPCVERIFY ... MAP t2  
T2_OAS { COPY T2 } DRC CHECK MAP T2 300  
T2_RDB { DFM RDB T2 tiles.rdb ALL CELLS CHECKNAME "tile_info" }
```

The RDB file will include data similar to this:

```
...  
p 1 4  
cell ghdac2b  
tile 190  
remote node1001  
-8272 -8272  
-400 -8272  
-400 -400  
-8272 -400
```

veb_simulate

Verification control

Performs etch bias simulation.

Usage

```
veb_simulate layer {etch_model veb_model | litho_model lm}  
[etch_underlying n underlayer_name]...
```

Description

Performs an etch bias simulation on an input contour layer. Used for model-based retargeting verification and VEB model verification purposes. The etch model being used should have been previously loaded with an [etch_model_load](#) command.

If a VEB model requires underlying layers, specify them using the optional etch_underlying argument.

Note

 The use of VEB models containing visibility kernels in the **image** command and in **veb_simulate** can increase runtime significantly. The larger the visibility kernel diffusion length, the longer the runtime. The recommended diffusion length of a visibility kernel is less than 0.8µm. Only one visibility kernel should be run at a time.

Arguments

- ***layer***
A required argument. This input layer should contain the resist contour before etch biasing.
- **etch_model *veb_model***
A required argument (exclusive with **litho_model**) specifying the name of the VEB etch model to use for the simulation.
- **litho_model *lm***
A required argument (exclusive with **etch_model**) that specifies a litho model that contains the VEB etch model to use for the simulation.
- **etch_underlying *n* *underlayer_name***
An optional argument that specifies an input layer to Calibre OPCverify that is associated with the selected kernel ID *n*. *n* must be 1 (corresponding to the SKERNEL in the VEB model containing the keyword “underlying=1”). *underlayer_name* specifies an input layer that must be set to type “hidden”. This layer is used as the underlying layer.

Examples

```
setlayer etch_contour = veb_simulate resist_contour etch_model etch_mod
```

width

DRC-type operation

Performs the equivalent of the WITH WIDTH operation in Calibre DRC, returning the polygons on the specified layer that meet the specified constraint.

Usage

width *layer* [not] *constraint*

Arguments

- *layer*

A required argument. This is the layer name to scan for shapes of the specified width constraint.

- [not] *constraint*

A required argument, specifying a width in user units (default is microns) and a conditional constraint. Calibre OPCVerify returns the polygons on layer that meet the constraint. If “not” is specified, this operation returns the set of polygons that fail this constraint.

Tip

 See the section “[Constraints](#)” on page 67 for more information on constraint syntax.

Examples

```
setlayer wid1 = width TARGET > .03
```

window

Verification control

Creates a window context region.

Usage

window size layer_to_window error_layer

Description

Creates a context layer (similar to [output_window](#)), using the error shapes specified as the selection criteria. The command de-angles the errors to 0, 45, or 90 degrees, then sizes the shapes up by the value specified in size, and ANDs the result with the *layer_to_window*.

window differs from [output_window](#) in the following ways:

- **window** only works on a single context layer. [output_window](#) can take multiple layers as context.
- **window** is used inside a [setlayer](#) command, which defines a result context layer that can be reused inside Calibre OPCVerify. [output_window](#) applies the window context to the whole layout. Use [output_window](#) to easily output windowed contours for all errors; use **window** for more flexibility.

Layers created with **window** cannot be used in other DRC operations, since it breaks polygons.

Arguments

- **size**
A required argument specifying an amount in microns to size up the error shapes by.
- **layer_to_window**
A required argument specifying the layer containing the layout context.
- **error_layer**
A required argument specifying the layer containing the error shapes to be used as a window criteria.

Examples

Given a target layer TARGET, an island layer ISLAND, and a generated layer pinch1 generated by a [pinch_tolerance](#) command, the following command reduces the context output to the area covered by the island layer:

```
setlayer ti = build_tolerance inner TARGET max .04 stepsize .01
setlayer ti1 = cornerchop ti .02 .02
setlayer i1 = image optical o1 dose 1.04 aerial .03
setlayer pinch1 = pinch_tolerance i1 ti1
setlayer winout = window .01 ISLAND pinch1
```

xor

DRC-type operation

Performs an exclusive OR of the inputs.

Usage

xor *input1 input2*

Arguments

- *input1 input2*

A required argument set specifying the layers to operate on.

Examples

```
setlayer xor1 = xor layer2 layer3
```

Appendix A

Calibre OPCverify Best Practices and Troubleshooting

This appendix is intended to be a rapid learning center for the Calibre OPCverify tool. It is separated into the following sections:

- [Essential Skillset](#) — Topics you should understand before attempting to create a Calibre OPCverify rule file.
- [Best Practices](#) — Advanced tactics to create the most effective Calibre OPCverify rule file.
- [Common Coding Errors](#) — Critical coding issues and known coding mistakes that should be checked for when problems arise.

Essential Skillset.....	490
Best Practices	494
Use Original Target Layers for Pinch and Bridge Checks	495
Use extra_printing Instead of interact for Extra Printing Errors.....	495
Use not_printing Instead of interact for Not Printing Errors	495
Choose Appropriate tilemicrons Settings	496
Choose Grid Settings Based On Your Configuration.....	497
Replace Older OPCverify Bridge and Pinch Detection Methods With Bridge and Pinch	498
Use an Error-centric Flow.....	499
Use critical_dimension	500
Use Only the Innermost and Outermost Process Window Variation Contours	501
Avoid Using output_window to Output In-Spec Points.....	502
Add a Classification and Limiting Block to Process Window Contours Used With output_window	504
Set “reflections_rotations_match” for Optical Models Below Version 10.....	505
Use Setlayer Concurrency	505
Use Tcl for Dynamic String Construction	507
Include These Checks for Poly, Metal, or Contact Layers.....	507
Use DFM RDB on Limited and Classified Errors Only.....	508
Use Filter Layer Markers to Check Square Contacts	508
Use jog_filter With contour_diff	509
When to Use direct_input	509
Using ULTRAFlex Versus TURBOflex	509
Common Coding Errors	510
Incorrectly Named Output Layers	511
ASCII Database Named the Same as Derived Layer	511

ASCII Database Files Do Not Take Multiple Input Types	512
Backslash Not at End of Line	512
Avoid Writing Out Full Chip Simulations	512
Do Not Use the size Command on Contour Layers	513
Do Not Use Large length Values for identify_edge	513
output_window is Not For Use With Classification keep duplicates	513
Post-OPC Layers are Not For Use in Classification	514
Classification max_size Set Too Small in Classification Blocks	514
Pinching Commands With max_tolerance Too Small	515
Multiple Property Layers Combined With OR	515
Usage of output_window on OR'd Output Layers	516
Improperly Used output_window filter_only Command	517
Avoid Using Large halo Settings	517
Soft Bridge and Soft Pinch Checks Have Strange Error Markers Near Line Ends	518
Use of DFM RDB NOPSEUDO Option	519
external, internal, and enclosure Inaccuracies When Used on Contours	520
Optical Models from 2005	520
Using the setlayer window Command to Return Classified Errors	520

Essential Skillset

It is important that you learn the following techniques before setting up an Calibre OPCVerify rule file.

Essentials to Remember

Calibre OPCVerify Code Is Not SVRF Code

Calibre OPCVerify commands are not interchangeable with SVRF code. They must be either in a separate file that is called by the SVRF rule file or as a bracket-enclosed inline rule file.

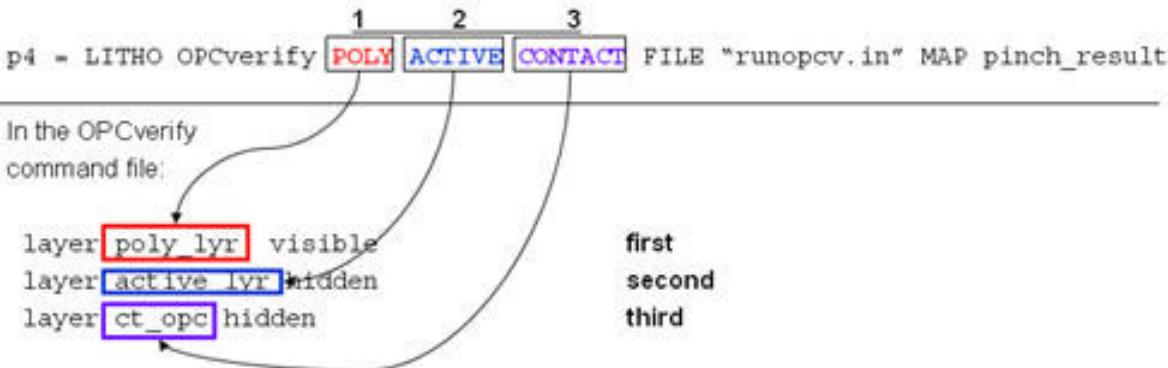
Layer Order is Important on Input to Calibre OPCVerify

Calibre OPCVerify command files are designed to be reusable code libraries callable by multiple SVRF rule files with a minimum of modification to the rule file. Calibre OPCVerify uses the following rules when called by an SVRF rule file.

- The layers input to Calibre OPCVerify are assigned to the layer statements in the order they are listed in the command file, regardless of their names in the SVRF rule file.
- An equal number of layers must appear in the LITHO OPCVERIFY call that are assigned in the Calibre OPCVerify command file.

Figure A-1. Layer Order in Calibre OPCVerify

SVRF:



Understanding the LITHO OPCVERIFY Call Syntax

The SVRF rule file calls the Calibre OPCVerify command file one or more times. Each call must be coded separately and extracts one of the derived layers from inside the Calibre OPCVerify command file. The syntax is as follows:

```
L1 = LITHO OPCVERIFY i_layer1 ... i_layerN FILE cmd_file MAP OPCv_layer1
```

Table A-1. LITHO OPCVERIFY Variable Items

Item	Description
<i>L1</i>	is an SVRF derived layer that stores the selected output results of the Calibre OPCVerify run. Copy this layer to a physical database (such as GDS or OASIS) or an RVE ASCII database file.
<i>i_layer1</i> through <i>i_layerN</i>	must be layers declared in the SVRF rule file. Only layers named in this list will be sent to the Calibre OPCVerify command file.
<i>cmd_file</i>	specifies the name of the Calibre OPCVerify command file to run.
<i>OPCv_layer1</i>	is an output layer generated in the Calibre OPCVerify command file that is copied into the <i>L1</i> derived layer.

Use the image Command to Create Contours

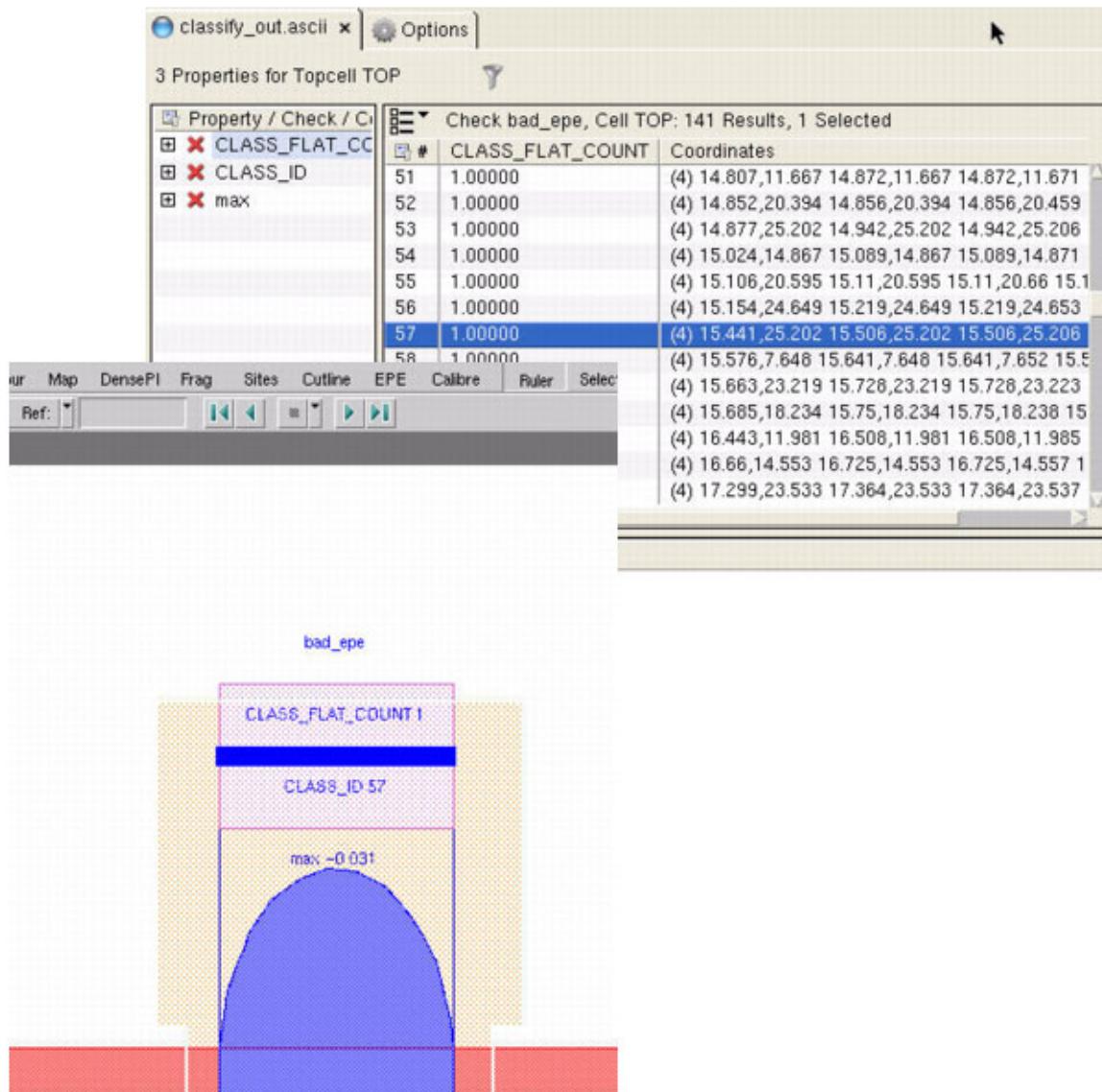
Many Calibre OPCVerify commands rely on having a simulated contour available. Two versions of the **image** command are available through Calibre OPCVerify: the legacy version and the explicit version. The default version simulates all layers that were set as visible. The explicit version allows you to specify exactly which layers to simulate.

Attach Property Values to Checks

Many Calibre OPCVerify commands have a property block listed as part of their command syntax.

When configured, these options add one or more properties that contain values to a marker that can be written to an ASCII database file readable by Calibre RVE (Figure A-2).

Figure A-2. Calibre WORKbench and Calibre RVE



You use these properties to collect and study the variance in measurements on aspects of interest. For example, recording the maximum value of all line end pullback instances can give you an idea of how sensitive the design is to such errors.

```
setlayer epeout = measure_epe contour TARGET inside ide epe_spacing .01
function min min_featsize .01 max_edgelen .11 only > -0.1 < .005 property
{max}
```

Use Classification and Limiting with Properties

When a large number of results (also known as error markers) are returned, using the additional classify and limit keywords can help isolate and identify errors more easily:

- Use classification blocks to discard errors that are duplicates of previously scanned errors. Duplicates are errors that have a similar surrounding context. Additional options can be specified to also classify reflections and rotations as duplicates, but these options should only be used if appropriate to your process.
- Use limit blocks to show only the markers with the worst calculated property values.

Related Topics

[Brief Calibre OPCVerify Command Setup File Syntax](#)

[Brief SVRF Rule File Syntax](#)

[Defining Layers in the Design](#)

[layer](#)

[LITHO OPCVERIFY](#)

[image](#)

[Error-Centric Section Blocks](#)

[Classification Block](#)

Best Practices

Items in this section are practices that generally improve performance and reduce issues. These suggestions were contributed by experienced Calibre OPCVerify users.

Table A-2. Best Practices for Calibre OPCVerify

Guideline	In Brief
Use Original Target Layers for Pinch and Bridge Checks	Improves performance.
Use extra_printing Instead of interact for Extra Printing Errors	
Use not_printing Instead of interact for Not Printing Errors	
Choose Appropriate tilemicrons Settings	
Choose Grid Settings Based On Your Configuration	
Replace Older OPCVerify Bridge and Pinch Detection Methods With Bridge and Pinch	Improves accuracy, performance, and ease of use.
Use an Error-centric Flow	
Use critical_dimension	
Use Only the Innermost and Outermost Process Window Variation Contours	Avoids unnecessary checks.
Avoid Using output_window to Output In-Spec Points	Returns only relevant data.
Add a Classification and Limiting Block to Process Window Contours Used With output_window	Avoids returning too many polygons.
Set “reflections_rotations_match” for Optical Models Below Version 10	
Use Setlayer Concurrency	Decreases processing costs.
Use Tcl for Dynamic String Construction	Avoids typographical errors.
Include These Checks for Poly, Metal, or Contact Layers	Consider using at least these checks for the layer type.
Use DFM RDB on Limited and Classified Errors Only	Reduces the database size.
Use Filter Layer Markers to Check Square Contacts	Improves accuracy.
Use jog_filter With contour_diff	
When to Use direct_input	Improves performance of litho-flat mode.
Using ULTRAflex Versus TURBOflex	

Use Original Target Layers for Pinch and Bridge Checks

The **pinch** and **bridge** commands are intended for use with original drawn (target) layers as input. While the commands can accept post-bias adjusted layers, the additional jogs that typically exist on bias-adjusted layers can create associated jogs in the error markers. This may degrade Calibre OPCVerify's classification capabilities on those markers, especially on input that use annotated constraints.

Related Topics

[pinch](#)
[bridge](#)

Use extra_printing Instead of interact for Extra Printing Errors

The **interact** command is designed to test interactions between polygons. Use the **extra_printing** command to test for contours that print without a corresponding drawn polygon, since it is a faster command due to having a smaller max_extent size and supports concurrency with similar commands such as **pinch**, **bridge**, and **not_printing**.

What to Avoid

```
setlayer extra_pi = interact contour target not > 0 max_extent 1
```

What is Recommended

```
setlayer extra_pi = extra_printing contour target max_extent 0.2
```

Related Topics

[extra_printing](#)
[interact](#)

Use not_printing Instead of interact for Not Printing Errors

The **interact** command is designed to test interactions between polygons. Use the **not_printing** command to test for polygons that do not have a corresponding printing contour, since it is a faster command due to having a smaller max_extent size and supports concurrency such as **pinch**, **bridge**, and **extra_printing**.

What to Avoid

```
setlayer missing_pi = interact target contour not > 0 max_extent 1
```

What is Recommended

```
setlayer missing_pi = not_printing target contour max_extent 0.2
```

Related Topics

[not_printing](#)

[interact](#)

Choose Appropriate tilemicrons Settings

Choosing the wrong tile size can lead to performance and memory issues.

Use the [tilemicrons](#) setup parameter from [Table A-3](#) to select the most appropriate value in microns.

Table A-3. Recommended tilemicrons Settings for Calibre OPCverify

Process Node	Poly (PC), Active (Rx), Contact or Via (CA) Layers	Metal1 (M1), Metal2 (M2), Other Metal (Mx) Layers
45/40 nm	80	70
32/28 nm	60	50
22/20 nm	50	40
14/10 nm	40	30

Note

 These recommendations are specific to Calibre OPCverify. Calibre nmOPC, Calibre pxOPC, and Calibre Local Printability Enhancement use different recommended tilemicrons settings.

For best results when changing tilemicrons settings:

- Use 3GB of memory per remote CPU core (double for Simultaneous Multi-Processors (SMT) hyperthreading).

Note

 Triple patterning layers or equivalent may need at least 4G remote memory.

- The settings above assume an interaction distance less than 3 microns. (This value can be found by searching the transcript for “INTERACTION DISTANCE”. Values higher than 4 microns can cause runtime and memory issues.)

- Using the recommended setting may not be the best tilemicrons value in all cases; you should always run a test on a smaller clip before running a full chip analysis.

Choose Grid Settings Based On Your Configuration

Choosing the best grid settings is a balance between simulation consistency and runtime performance.

The impact of grid settings can differ from layer to layer. Best practices can also differ according to the technology node.

The following best practices are general guidelines considering a balanced accuracy and performance requirement.

DUV Simulations

The following settings are the default best practices starting with the 2018.4 release for DUV technology nodes.

```
imagegrid aerial 1 2
final_upsample 1
optical_transform_size 768
contour_options interp_algo lagrange interp_degree 5 max_edge_merge_error
0.0001 max_segment_length 0.003
```

- If more accuracy is needed, use a value of imagegrid aerial 3 8, but this setting may increase runtime.
- Using a larger max_edge_merge_error argument provides faster performance at some accuracy cost.

EUV Simulations

Starting with the 2018.4 release, the best practice settings are as follows:

```
imagegrid aerial 1 1
final_upsample 1
contour_options interp_algo lagrange interp_degree 5 \
max_edge_merge_error 0.0001 max_segment_length 0.002
optical_transform_size 1024
```

If more contour accuracy is needed, use a setting of imagegrid 2 3, but using this setting may increase runtime.

Related Topics

[contour_options](#)
[final_upsample](#)
[imagegrid](#)
[optical_transform_size](#)

Replace Older OPCVerify Bridge and Pinch Detection Methods With Bridge and Pinch

In the original version of Calibre OPCVerify, detection of bridge and pinch errors was performed using a sequence of commands.

Specifically, the method used `build_tolerance` followed by `bridge_tolerance` or `pinch_tolerance` and the `measure_distance` and `interact` commands. While these methods were effective at finding errors of interest, Siemens EDA has developed commands to replace these old methods. Siemens suggests migrating to the newer commands as shown in [Table A-4](#) to take advantage of the performance and accuracy improvements.

Table A-4. Recommended Failure Detection Methods

Failure Type to Detect	Old Method	New Method	In Brief
hard bridging	<code>build_tolerance</code> followed by <code>bridge_tolerance</code>	<code>bridge</code>	Improves ease of use: <ul style="list-style-type: none">• New commands use less code. Provides more accuracy: <ul style="list-style-type: none">• Avoids <code>measure_distance</code> detection of false positives. Improved run time: <ul style="list-style-type: none">• Can use concurrency with many of these commands. Recommended setting: <ul style="list-style-type: none">• <code>max_tolerance</code> value near the minimum feature size. Do not use the default (.15 microns).
soft bridging	<code>measure_distance</code>		
hard pinching	<code>build_tolerance</code> followed by <code>pinch_tolerance</code>	<code>pinch</code>	
soft pinching	<code>measure_distance</code>		

Table A-4. Recommended Failure Detection Methods (cont.)

Failure Type to Detect	Old Method	New Method	In Brief
extra printing or not printing versus a drawn layer	interact	extra_printing or not_printing	<p>See the sections “Use extra_printing Instead of interact for Extra Printing Errors” and “Use not_printing Instead of interact for Not Printing Errors”.</p> <p>Recommended setting:</p> <ul style="list-style-type: none"> • max_extent value slightly larger than the worst line-end shortening result. Do not use the default (.15 microns).

Use an Error-centric Flow

For the most efficient runtime, error review cycle, and smallest output data size, Siemens EDA strongly recommends that you use the “error-centric flow” (defined as the use of properties, classification, scoring, limiting and output window in the table following).

Table A-5. Using the Error-Centric Flow

Error-Centric Flow Component	Notes
Properties	<ul style="list-style-type: none"> • see “Error-Centric Section Blocks” on page 69 <p>Defining a property block records the values used for scoring and limiting.</p>
Classification blocks (Integrated classification with scoring and limiting)	<ul style="list-style-type: none"> • see “Limit Block” on page 85 • see “Classification Block” on page 73 <p>Recommended settings:</p> <ul style="list-style-type: none"> • context — Use a target (preOPC) layer as the context. Do not use the a contour layer or post-OPC as the argument context, since this will create false unique errors from the jogs introduced in contours. • maxsize — Avoid using an overly small maxsize value, since errors larger than it will not be classified. <p>The default is 0.5 um.</p>

Table A-5. Using the Error-Centric Flow (cont.)

Error-Centric Flow Component	Notes
	<ul style="list-style-type: none"> • maximum_error_number — Siemens EDA strongly recommends against the use of a maximum_error_number that is too similar in size to the “worst count” argument, as under certain conditions classification can be turned off too early to capture significant errors. The suggested value for maximum_error_number is <u>at least</u> 10x relative to worst count. • pm_classify — The recommended parameter settings include max_search set to 1.0*halo and max_length set to 3-10*halo. • coarse_match — Use this argument to reduce unique errors by allowing for slight variations in the context layer. • suppress keep_duplicates — Use the default (suppress duplicates) to remove duplicate errors from the output, leaving unique and unclassified errors. CLASS_ID values are not attached to the output. • score ... worst x — Activating limiting mode returns exactly the number of unique errors specified in worst.  Note: Not using limiting can lead to high memory consumption and runtime impact.
output_window	<ul style="list-style-type: none"> • see “output_window” on page 203 <p>Outputs only contours and context layers around the worst errors.</p>

What is Recommended

```
setlayer pinch_inner = pinch target contour <= 0.020 max_tolerance 0.02
property { min } classify {
    context target
    halo 0.2
    maxsize 0.5
    coarse_match 0.002
    score bin_size 0.002 smallest
    worst 1000
    pm_classify max_search 0.2 max_length 1
}
output_window contour around pinch_inner
```

Use critical_dimension

critical_dimension is a setup command placed near the beginning of a Calibre OPCVerify command file.

Use the [critical_dimension](#) command to replace various default values for multiple Calibre OPCVerify command arguments. The [critical_dimension](#) reference page lists the affected commands.

You would use this command to change the CD size to match your node size design specifications. The Calibre OPCVerify commands that use [critical_dimension](#) use a multiple of the specified CD as appropriate instead of the set default values.

Use Only the Innermost and Outermost Process Window Variation Contours

While it is possible to generate and run pinching and bridging operations on multiple contours at different process conditions, the contours of interest are only at the worst pinching and bridging locations.

A coding best practice is to only implement pinch and bridge checks on those worst condition contours, since there will be less contours to check on error review and less output when the [output_window](#) command is used.

In the Calibre OPCVerify command file, four image contours are created:

```
setlayer image1 = image optical focus0 dose dose0
setlayer image2 = image optical focus0 dose dose_p
setlayer image3 = image optical focus0 dose dose_n
setlayer image4 = image optical focus_p dose dose0
```

Not Recommended

A less efficient usage example individually calculates pinch locations on all four layers:

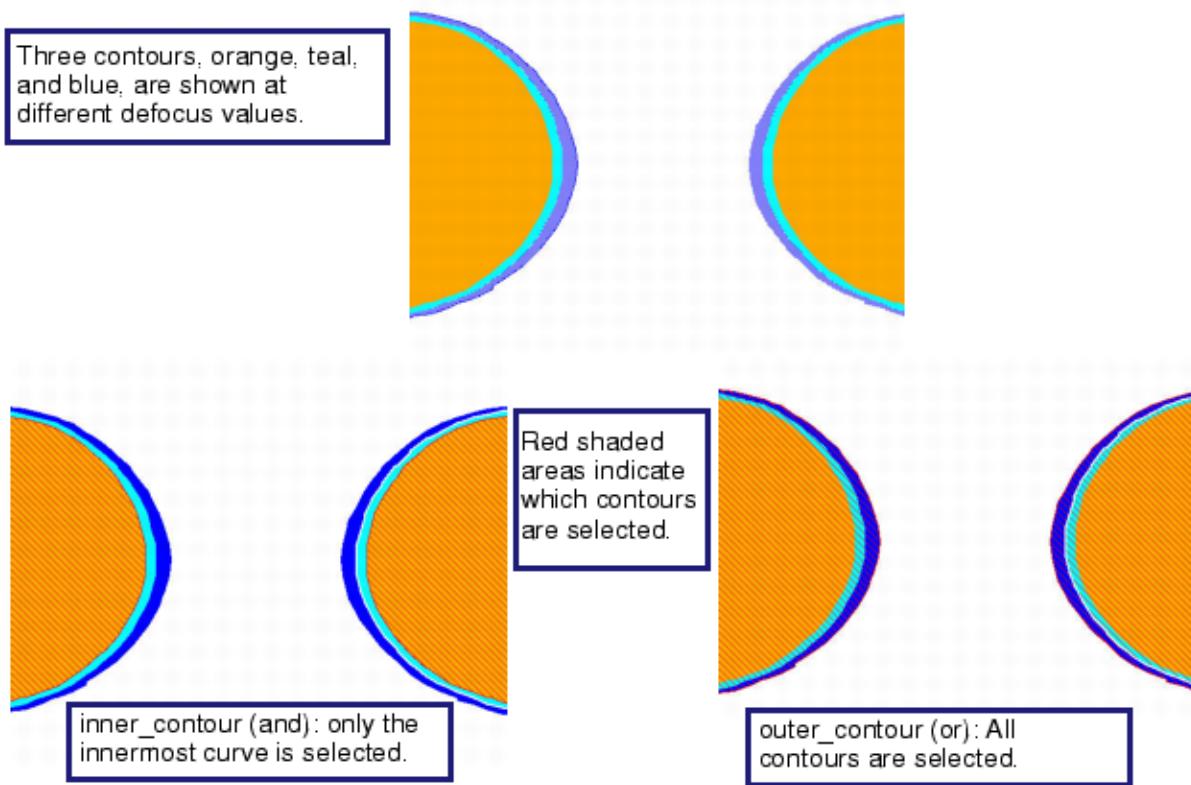
```
setlayer pinch_image1 = pinch target image1 <= 0.05 max_tolerance 0.05
.....
setlayer pinch_image4 = pinch target image4 <= 0.05 max_tolerance 0.05
```

What is Recommended

The recommended usage is to perform an “**and**” operation to get an inner contour, and an “**or**” operation to get an outer contour:

```
setlayer inner_contour = and image1 . . . image4
setlayer outer_contour = or image1 . . . image4
setlayer pinch_pw = pinch target inner_contour <= 0.05 max_tolerance 0.05
setlayer bridge_pw = bridge target outer_contour <= 0.05 \
    max_tolerance 0.05
```

Figure A-3. Inner Contours Versus Outer Contours



Related Topics

[bridge](#)

[pinch](#)

Avoid Using `output_window` to Output In-Spec Points

The `output_window` command is designed to report highly detailed information about errors.

For example, do not use the `output_window` command to report the entire result set of a `gate_stats` command, as you will end up with in-spec points that are not of interest.

- The best use of `output_window` is to closely inspect a small set of critical errors that have been limited and classified.
- (Optional) Use the `histogram` command to see the distribution of the check over the entire range. Histogram files convert the output to a simple ASCII data file that contains only the number of occurrences of a measured data value.

Figure A-4. Example Histogram File Output

0.047	0
0.048	1
0.049	7
0.05	6
0.051	31
0.052	28
0.053	41
0.054	12
0.055	17
0.056	48
0.057	121
0.058	41
0.059	123
0.06	166
0.061	166
0.062	149
0.063	158
0.064	63
0.065	90
0.066	60
0.067	83
0.068	49
0.069	14

Out of spec: too small. These errors should be inspected.

In spec. These error markers can be safely ignored.

Out of spec: too large. These errors should be inspected.

Example

Given a design with a minimum CD of 60nm and an in-spec simulated CD between 55 and 65 nm:

Not Recommended

The following command outputs all errors ≥ 0 and < 70 nm:

```
setlayer cdvar = gate_stats contour target inside active min >= 0 < 0.07
absolute cd_min 0.04 cd_max 0.07 no_split_marker property { min }

output_window contour around cdvar
```

What is Recommended

1. Set the constraint for an Calibre OPCVerify command to exclude the range that is in spec. Send these resulting out-of-spec results to **output_window**.

The following command outputs only errors not within the in-spec range by changing the constraint:

```
setlayer cdvar = gate_stats contour target inside active \
    min not >= 0.055 < 0.065 absolute cd_min 0.04 cd_max \
    0.06 no_split_marker property { min }

output_window contour around cdvar
```

2. If the whole range of errors need to be reported (in-spec and out-of-spec errors), add a histogram block to the command. A histogram block outputs a text file that includes all error bins and the corresponding error count for each bin.

The following command records all the errors in the zero to max CD range in a histogram file:

```
setlayer cdvar_hist = gate_stats contour target inside active \
    min >= 0 < 0.07 absolute cd_min 0.04 cd_max 0.07 \
    no_split_marker property
    { min }
        histogram {
            file cdvar_hist.txt
            bin_size 0.001
        }
```

Related Topics

[output_window](#)

[gate_stats](#)

[Add a Classification and Limiting Block to Process Window Contours Used With output_window](#)

Add a Classification and Limiting Block to Process Window Contours Used With output_window

Checks on non-nominal process window contours are expected to output a large number of errors. If these errors are sent to the output_window command, the contour layer generated around all the errors may increase runtime and memory usage. In these situations, use a classification and limit block in conjunction with the property block to reduce the number of errors.

If a setlayer check outputs duplicate errors and the setlayer check is used in the output_window statement, then the classify block for this check should contain the keep_no_context keyword. The keep_no_context keyword outputs duplicate errors in the RDB, but does not contours around each duplicate error in the output OASIS file. The contour is created only for the initial unique error. Failure to do use the keep_no_context keyword in this circumstance can cause a large impact in run time and performance by causing the primary host to go into swap.

What to Avoid

```
setlayer pinch_inner = pinch target contour <= 0.050 max_tolerance 0.05
property { min }
output_window contour around pinch_inner
```

What is Recommended

```
setlayer pinch_inner = pinch target contour <= 0.050 max_tolerance 0.05
property { min } classify {
context target
halo 0.2
score bin_size .01 smallest
worst 5000
}
output_window contour around pinch_inner
```

Related Topics

- [Error-Centric Section Blocks](#)
- [output_window](#)

Set “reflections_rotations_match” for Optical Models Below Version 10

Version 10 optical models include the symmetry information in the optical model.

For optical models below version 10 (2013.4 and earlier), [Classification Block](#) must have their “reflections_rotations_match” parameter explicitly specified to “yes”, “no”, or “reflections”. A known issue exists where the default value of “decide” incorrectly determines a pre-version 10 optical model to be asymmetric even if the optical model was symmetric. This caused symmetric errors to be incorrectly classified as unique.

Use Setlayer Concurrency

Use concurrency-based coding techniques to increase Calibre OPCVerify processing efficiency. Similar commands that have a Concurrency Support section are run in the same calculation cycle, as long as they follow these guidelines:

- They operate on the same target and contour layers.
- They follow any additional limitations listed in the command.

Some commands are related, and can be grouped together. For example, the following rules apply to pinch and bridge checks:

- For pinch and bridge commands, their max_tolerance and max_edge arguments are the same.
- If filter options are included, they must also be the same.
- If bridge and pinch operations include soft bridge and pinch checks, the separation arguments are the same.
- Their max_extent (for extra_printing and not_printing) arguments are the same.
- output_expand values are the same for all bridge checks.
- output_expand values are the same for all pinch checks.

Example

Set 1: soft bridge and pinch, separation and max_tolerance arguments match.

```
setlayer soft_bridge = bridge target contour < 0.059 separation 0.2
max_tolerance 0.06 property {min}

setlayer soft_pinch = pinch target contour < 0.059 separation 0.2
max_tolerance 0.06 property {min}
```

Set 2: hard bridge and hard pinch, max_tolerance arguments match.

```
setlayer hard_bridge = bridge target contour max_tolerance 0.06

setlayer hard_pinch = pinch target contour max_tolerance 0.06
```

Set 3: extra_printing and not_printing, max_extent arguments match.

```
setlayer extra_pi = extra_printing target contour max_extent 0.06

setlayer missing_pi = not_printing target contour max_extent 0.06
```

Since all three sets operate on the same contour and target layers, they will be treated as a single concurrent operation.

Related Topics

[Concurrency Execution of Multiple Calibre OPCverify Statements in Calibre](#)

Use Tcl for Dynamic String Construction

Calibre OPCVerify command files are passed through the Tcl parser. It allows for looping and dynamic string construction.

Calibre OPCVerify accepts variable substitution in its commands.

```
setlayer pinch = ....  
setlayer bridge = ....  
set all_checks "pinch bridge"  
eval output_window contour around $all_checks
```

Related Topics

[Best Practice: Using Tcl with Calibre OPCVerify](#)

Include These Checks for Poly, Metal, or Contact Layers

Calibre OPCVerify users who are creating layer-designated checks (poly, metal, or contact) should probably include the following kinds of verification checks depending on the layer type.

Table A-6. Suggested Checks By Layer Type

Check	Poly	Metal	Contact
bridge and pinch	Yes	Yes	No
extra_printing and not_printing	Yes	Yes	Yes
measure_epe (for line end pullback)	Yes	Yes	No
measure_epe (for asymmetrical contacts)	—	—	Yes
measure_cd and measure_epe with a filter layer (for square contacts)	—	—	Yes
area_ratio and contour_diff (for contact coverage)	Yes	Yes	Yes
area_overlay (for misalignment checks)	Yes	Yes	No
area_compute (for contact area)	No	No	Yes
end_cap (measures poly outside active)	Yes	No	No

Table A-6. Suggested Checks By Layer Type (cont.)

Check	Poly	Metal	Contact
gate_stats (checks for gate errors)	Yes	No	No

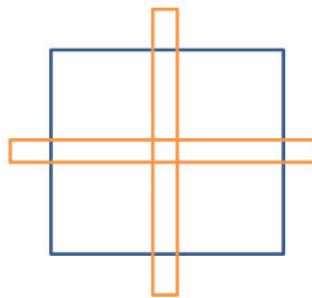
Use DFM RDB on Limited and Classified Errors Only

Where possible, you should apply a limit block, a classification block, or both on errors. This will reduce the size of the database that would otherwise be created with unclassified, non-limited properties.

Use Filter Layer Markers to Check Square Contacts

When checking square contacts, create a separate filter layer containing cross shapes that cover the contacts such that each end of the cross extends across the middle of the edge of the square:

Figure A-5. Filter Layer Suggestion for Square Contacts



You use this filter layer as the “inside” argument to the `measure_cd` and `measure_epe` commands to ensure accurate output properties for errors, since the errors will be focused only at the design measurement locations.

In the following example, you have a square contact, 0.08 x 0.08 um in dimension. The following SVRF code creates a cross-hair filter layer.

```
cross_hair_filter = EXPAND EDGE target OUTSIDE BY 0.005 INSIDE BY 0.04
EXTEND BY FACTOR -0.45
...
measure_cd ca_img inside cross_hair_filter ...
measure_epe ca_img inside cross_hair_filter ...
...
```

In this example using the `Expand Edge` SVRF command, `OUTSIDE BY` determines how far outside the target you want to extend the filter. The `INSIDE BY` command should be set to greater than or equal to $0.5 \times CD$. The `EXTEND BY FACTOR` command value determines the

width of the filter. A value between -0.4 to -0.49 is normally sufficient. A value of -0.5 or greater causes the filter to have zero width and, consequently, no filter is generated.

Use jog_filter With contour_diff

Whenever possible, use drawn target layers with the **contour_diff** command.

If you must use contour layers with **contour_diff**, use the **jog_filter** argument. Failure to include this option may cause Calibre OPCVerify to detect false errors with unusually large error values reported compared to actual measurements, especially around target kinks.

The recommended value for **jog_filter** is 0.05 0.05 for most current designs, and should eliminate this issue around jog corners.

When to Use direct_input

Siemens EDA recommends the use of the **direct_input** tasks with litho-flat in situations where the design size is larger than the memory available on the primary host.

Using ULTRAFlex Versus TURBOflex

The use of ULTRAFlex versus TURBOflex is dependent on the situation and the software you are using.

Consult the section “Which Construction Method to Use” in the *Calibre Post-Tapeout Flow User’s Manual* for the most current best practices.

Common Coding Errors

The following table summarizes common code problems seen in rule files submitted to Customer Support.

Table A-7. Coding Error Checklist

Check For...	Priority	In Brief
Incorrectly Named Output Layers	High	Ends with “Could not find output layer”.
ASCII Database Named the Same as Derived Layer	High	Error due to duplicate rule check name.
ASCII Database Files Do Not Take Multiple Input Types	High	Error due to attempt to write two different data types.
Backslash Not at End of Line	High	Ends with syntax error.
Avoid Writing Out Full Chip Simulations	High	May crash from lack of memory. Runs take much longer.
Do Not Use the size Command on Contour Layers	High	Causes significant runtime increase.
Do Not Use Large length Values for identify_edge	High	
output_window is Not For Use With Classification keep duplicates	High	Outputs far more error markers than necessary.
Post-OPC Layers are Not For Use in Classification	High	Outputs far more error markers than necessary.
Classification max_size Set Too Small in Classification Blocks	High	Results can contain multiple unclassified errors.
Pinching Commands With max_tolerance Too Small	High	Soft pinch and bridge errors will be flagged as hard pinch or bridge errors.
Multiple Property Layers Combined With OR	Medium	Missing property values on output layers.
Usage of output_window on OR'd Output Layers	Medium	Losing classified results (unclassified results are output).
Improperly Used output_window filter_only Command	Medium	<ul style="list-style-type: none">empty_layer declaration must be associated with output_window layer or errors occur.OR'd filter layers should be performed in SVRF instead of Calibre OPCVerify.

Table A-7. Coding Error Checklist (cont.)

Check For...	Priority	In Brief
Avoid Using Large halo Settings	Medium	Large halo values increase simulation times.
Soft Bridge and Soft Pinch Checks Have Strange Error Markers Near Line Ends	Medium	Setting output_expand width_microns width_microns2 values affects the output.
Use of DFM RDB NOPSEUDO Option	Medium	“Unique” errors appear multiple times.
external, internal, and enclosure Inaccuracies When Used on Contours	Medium	Less accurate results. Edges may be skewed.
Optical Models from 2005	Medium	Missing information.
Using the setlayer window Command to Return Classified Errors	Low	Classified errors were expected.

Incorrectly Named Output Layers

Problem: Run time error, “Could not find output layer.” Sometimes results are written to an unintended layer.

Solution: Check your OPCverify command file to see that all output layers are named correctly in both the command file and the corresponding LITHO OPCVERIFY call. Output is case sensitive. Layer names may not contain spaces or punctuation.

ASCII Database Named the Same as Derived Layer

Problem: Run time error. The DFM RDB command that writes to the ASCII RVE database file must be named differently than the derived layer from the LITHO OPCVERIFY call, since SVRF does not allow duplicate rule names within the same rule file.

Incorrect code:

```
bad_gate = LITHO OPCVERIFY ... MAP bad_gate
bad_gate { COPY bad_gate } DRC CHECK MAP bad_gate 100
bad_gate { DFM RDB bad_gate "opcv_out.ascii" ...}
```

Correct code:

```
bad_gate = LITHO OPCVERIFY ... MAP bad_gate
bad_gate { COPY bad_gate } DRC CHECK MAP bad_gate 100
rbdb_bad_gate { DFM RDB bad_gate "opcv_out.ascii" ...}
```

ASCII Database Files Do Not Take Multiple Input Types

Problem: The DRC PROPERTY and DRC CHECK MAP commands cannot write to the same output ASCII file.

```
bad_gate = LITHO OPCVERIFY ... MAP bad_gate
bad_gate { COPY bad_gate } DRC CHECK MAP bad_gate 100
rdb_bad_gate { DFM RDB bad_gate "opcv_out.ascii" ... }
marker { COPY bad_gate } DRC CHECK MAP marker ASCII "opcv_out.ascii"
```

Solution: Use different names for the input files.

Backslash Not at End of Line

Problem: Syntax error, typically one of “something to do with unrecognized keyword” or “unexpected end of line”.

Solution: The backslash (\) continuation character must always be the last character on a line when it is used in an OPCverify setup command file. If your run exits with syntax errors that do not appear to apply, check that there are no spaces, tabs, or other characters after the backslash on the line.

Avoid Writing Out Full Chip Simulations

Problem: Performance slowdowns or out-of-memory crashes due to multi-gigabyte output files.

OPCverify uses a dense printing and scanning algorithm that generates a large amount of data for accuracy purposes. Outputting OPCverify contours for a full chip run can create a multiple gigabyte GDS or OASIS file. If Calibre does not run out of memory trying to process the file, performance is still degraded by the amount of time it takes to write out such a large file.

For similar reasons, you should also not use the DRC Map command to map the full chip simulation back to the Calibre database or pass contours to SVRF for further checking.

Solution:

- Only output complete contour windows when absolutely necessary.
- Use the [output_window](#) command (which supports per-layer output around markers on a filter layer) to select parts of the design to output only simulation contours for found or suspected errors. These can be identified using the embedded checks within the OPCverify setup file.

Do Not Use the size Command on Contour Layers

Problem: Performance slowdowns due to calculations required to size multiple polygon edges.

The OPCVerify `size` command resizes polygons based on the length of the sides. Since contour layers are made up of a very large number of tiny edges, sizing a contour layer represents a significant runtime impact.

Solution: You should only use the `size` command on drawn layers or error layers with simple polygon shapes.

Do Not Use Large length Values for identify_edge

Problem: Performance slowdowns.

Solution: When specifying the `identify_edge` length parameter, it is recommended that you not use a very large value (>3 um), since constraints of that size will result in increased interaction distance, which results in flattening of data and slower run times.

output_window is Not For Use With Classification keep duplicates

Problem: Missing expected data. The `output_window` command is designed to print detailed context clips to the output design file based around unique errors. Adding the classification block setting that keeps duplicates in addition to unique errors will result in the generation of many more errors than necessary to check errors.

What to Avoid

```
setlayer pinch = pinch target contour <= 0.05 max_tolerance 0.05 property
{ min } classify {
    halo 0.5
    context target
    keep duplicates # do not use this setting
    ...
}
output_window contour around pinch
```

What is Recommended

```
setlayer pinch = pinch target contour <= 0.05 max_tolerance 0.05 property
{ min } classify {
    halo 0.5
    context target
    suppress duplicates # use this setting
    ...
}
output_window contour around pinch
```

Post-OPC Layers are Not For Use in Classification

Problem: Classification relies on finding identical matches of error markers. Using a post-opc or contour layer as the context layer for classification will result in a large number of unique errors due to various variations in the context layer shapes.

In the following example, target is the target layer and postopc is the post-opc layer.

What to Avoid

```
setlayer pinch = pinch postopc contour <= 0.050 max_tolerance 0.05
property { min } classify {
    context postopc
    halo 0.5 }
```

What is Recommended

```
setlayer pinch = pinch target contour <= 0.050 max_tolerance 0.05 property
{ min } classify {
    context target
    halo 0.5 }
```

Classification max_size Set Too Small in Classification Blocks

Key Concept: Errors are sorted during classification into three categories:

- unique, which identifies one type of error configuration by shape.
- duplicate, which matches a previous unique error.
- unclassified, which is not checked for uniqueness and are automatically considered as unique.

Problem: Any error markers that are larger than max_size are set as unclassified, which subsequently sorts them as if they were unique errors. A large amount of error markers will be produced when the max_size keyword is set to a very small value inside a classification block.

What to Avoid

```
setlayer pinch = pinch target contour <= 0.050 max_tolerance 0.05 property
{ min } classify {
    context postopc
    maxsize 0.05
    halo 0.5 }
```

What is Recommended

```
setlayer pinch = pinch target contour <= 0.050 max_tolerance 0.05 property
{ min } classify {
    context target
    maxsize 0.5
    halo 0.5
    worst 25 # adding a limit is also useful
}
```

Pinching Commands With max_tolerance Too Small

Problem: Necking (soft pinch) errors are being detected as opens (hard pinch) errors when max_tolerance is set to a small value. This causes problems because max_tolerance is used as the search distance for nearby contours, and setting max_tolerance too small results in contours being missed.

This problem also applies to bridge checks where soft bridging errors are being detected as hard bridging errors.

Solution: Set max_tolerance equal to your minimum CD value.

Multiple Property Layers Combined With OR

Problem: A known SVRF practice is to combine multiple derived error layers together and then output the single combined error layer. However, this method does not work in Calibre OPCverify on layers that contain property values. While the shapes will be combined on the new layer, property values are not copied along with the shapes into the new output layer.

What to Avoid

In the OPCverify command file, this code attempts to combine multiple property values into a single output layer using the **or** statement:

```
setlayer pinch = pinch target contour <= 0.05 max_tolerance 0.05 \
    property {min}
setlayer bridge = bridge target contour <= 0.05 max_tolerance 0.05 \
    property { min }
setlayer all_failures = or pinch bridge
```

In SVRF, this would be called by the following command:

```
All_failures = LITHO OPCV .... MAP all_failures
```

What is Recommended

Any command that creates property values can only be used as an output layer to SVRF. The corrected version of the code would look like the following:

In the OPCverify command file, this code creates two separate layers without combining them:

```
setlayer pinch = pinch target contour <= 0.05 max_tolerance 0.05 \
    property { min }
setlayer bridge = bridge target contour <= 0.05 max_tolerance 0.05 \
    property { min }
```

In SVRF, each individual output layer is retrieved by a separate LITHO OPCVERIFY call:

```
pinch = LITHO OPCV .... MAP pinch
bridge = LITHO OPCV .... MAP bridge
```

Usage of output_window on OR'd Output Layers

Problem: Using **output_window** on a set of output layers that are **or'd** together (such as to create a filter layer) loses classification and limiting results. Additionally, some error markers may contain contour shapes, which are not recommended for use with SVRF operations.

What to Avoid

```
setlayer pinch = pinch target contour <= 0.050 max_tolerance 0.05 property
{ min } classify {
    context target
    halo 0.5 }

setlayer bridge = bridge target contour <= 0.050 max_tolerance 0.05
property { min } classify {
    context target
    halo 0.5 }
setlayer all_checks = or pinch bridge
output_window contour around all_checks
```

What is Recommended

You can create a summary of all error layers by replacing the last two lines of the above example with the following code.

```
set all_checks "pinch bridge"
eval output_window contour around $all_checks
```

Tip

 To create a filter layer around all errors, see the solution in the next section, “[Improperly Used output_window filter_only Command](#).”

Improperly Used output_window filter_only Command

Problem: An output_window statement that uses the filter_only argument can cause unexpected errors in output layers if an empty_layer is not declared for use with the output_window layer. Furthermore, the empty_layer declaration must appear before the output_window statement that uses it.

What to Avoid

```
layer PC_TARGET
layer PC_OP
...
output_window output_window_filter filter_only around error1 error2
setlayer output_window_filter = empty_layer # wrong order
```

What is Recommended

Ensure that the empty_layer statement appears before any output_window statement that uses filter_only.

```
layer PC_TARGET
layer PC_OP
...
output_window contour around error1 error2 # okay, no filter_only
setlayer output_window_filter = empty_layer # output_window must follow
                                                # this line to work properly
output_window output_window_filter halo 0.5 filter_only around \
error1 error2
```

Using output_window as a Filter

Notice the addition of the halo 0.5 parameter; this creates an area around the error clip that can be sized up in SVRF and then used as a filter on the original design to isolate error clips.

Avoid Using Large halo Settings

A large halo size (greater than 1.0 microns, for example) should be avoided, as this will create a large interaction distance (unless filter_only is also specified), which in turn increases simulation runtime. Large filter layers are best created in SVRF after using the default value for halo (0.5 um) for output_window and sizing up the results.

What to Avoid

```
output_window target halo 5 around error1 error2 error3
outout_window sraf halo 5 around error1 error2 error3
```

What is Recommended

```
setlayer filter = empty_layer  
output_window filter halo 0.5 around error1 error2 error3
```

Then, in the SVRF file, add the following commands:

```
filter = LITHO OPCVERIFY ... MAP filter  
final_filter = size filter by 10  
target_filtered = AND target final_filter  
sraf_filtered = AND sraf final_filtered
```

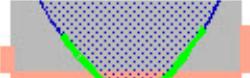
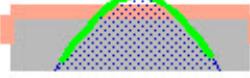
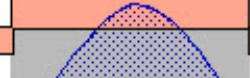
Soft Bridge and Soft Pinch Checks Have Strange Error Markers Near Line Ends

Problem: A single soft **bridge** or **pinch** error generates strange errors in the same location.

This phenomenon ([Figure A-6](#)) occurs for errors that continue around the corner of the target, and according to how the `output_expand width_microns width_microns2` arguments are set to different values as shown in the figure.

- For small values of `width_microns` (a, 0.002), the error markers are distinct to each side of the error contour (shown in green), and thusly more accurately describe the error.
- For large values of `width_microns` (b, 0.025), the error markers expand to create a single large error covering the gap.
- When the optional `width_microns2` argument is specified (c) as 0, the error marker is split up, with each ‘corner’ marker having its own error value applied.

Figure A-6. output_expand width_microns2 Argument

output_expand	Result
output_expand 0.002	
output_expand 0.025	 a
output_expand 0.025 0	  b  c

Solution: If you use the *width_microns2* argument, it should always be set to 0.001 for non-contact/via layers, and to 0 only for contact/via layers, as separate error markers is the desired behavior for contacts and vias that have [pinch](#) and [bridge](#) issues.

Use of DFM RDB NOPSEUDO Option

Problem: Additional unique errors appear in the database. This occurs in situations where the DFM RDB NOPSEUDO option was specified. The NOPSEUDO option flattens the hierarchy, creating new instances of geometries that may be classified as unique. It also increases runtime.

Solution: Do not use NOPSEUDO.

external, internal, and enclosure Inaccuracies When Used on Contours

Problem: Non-45 degree geometries and possible inaccuracies.

The [external](#), [internal](#), and [enclosure](#) operations are based off of DRC-style measurement commands and sometimes have problems when measuring contour shapes.

Solution: Use the [measure_cd](#), [measure_epe](#), [measure_distance](#), and the other tolerance band commands to measure the distance between shapes.

Optical Models from 2005

Problem: Optical models created in versions 2005.1, 2005.2, 2005.3, and 2005.4 that include a generated `.fsk` file in the optical directory are missing information on diffusion and non-TCCS kernels.

The resist models do not need to be regenerated if you were using a version 6 optical model; if you were using an earlier optical model (5 or earlier) and you regenerate your optical model, you will also need to regenerate your resist model.

Solution: Regenerate your optical models using the current Calibre version.

Using the setlayer window Command to Return Classified Errors

Problem: Using the `window` command on a check with a `classify` block does not return the expected output, because `window` is an older command that returns pre-classified errors rather than post-classified errors; in other words, it works on pre-classified, pre-limited results.

What to avoid:

```
setlayer pinch = pinch target contour <= 0.050 max_tolerance 0.05 property
{ min } classify {
    context target
    halo 0.5 }
setlayer bridge = bridge target contour <= 0.050 max_tolerance 0.05
property { min } classify {
    context target
    halo 0.5 }
setlayer pinch_win = window 0.5 contour pinch
setlayer bridge_win = window 0.5 contour bridge
```

Solution: Use the newer `output_window` command instead, which also has the added benefit of being able to support multiple layers on the same line.

Note

 output_window is a setup-style command and should appear at the end of your command file. The “window” command is a setlayer-style command.

Appendix B

Scoring Information

In order to select the most effective process settings, it is important to be able to “score” the overall performance of the layout through a process window.

Note

 The information in this appendix is provided for informational purposes, and is not required to use Calibre OPCVerify.

Scoring must take into account the fact that we want to be able to score on the basis of either a single process condition or on the basis of the full process window. There are two types:

- [Layout Scoring](#) — Integrated score for a layout
- [Error Scoring](#) — Degree of “badness” with respect to a given error mode

In general, we can think of scoring in terms of either:

- A weighted **count** of errors over a process window
 - or
- A weighted **area** of errors over a process window

Layout Scoring	523
Error Scoring	524
Scoring Functions	524

Layout Scoring

The layout score for an individual process condition can be computed as:

area_out_of_tolerance_band

The layout score for the entire process window could be either:

$$\sum_{i = \text{conditions}} \text{area_out_of_tolerance_band} \times W_i$$

or

area_of_pvband

Error Scoring

Scoring failures allows you to rank all failures by their score, and thus prioritize what needs to be fixed. There is a exponential behavior of bridging and pinching with respect to dose at a fixed focus condition. This property can be used to help define a failure score for bridging and for pinching.

If the bridging score is:

$$score_b = \sum_{i=conditions} area(\text{contour NOT outer_zone}) \times W$$

Then the pinching score is:

$$score_p = \sum_{i=conditions} area(\text{inner_zone NOT contour}) \times W$$

Scoring Functions

After coding the calls to LITHO OPCVERIFY and any other output or derived layer calls you want to use, you can add one or more scoring functions in order to consolidate the various process results.

Typically, you would use the **DFM ANALYZE** SVRF command, as in the following code:

```
score {  
    DFM ANALYZE e1 e2 e3 e4 WINDOW 100  
    [ w1*AREA(e1)+w2*AREA(e2)+w3*AREA(e3)+w4*AREA(e4) ] > 0  
    RDB ONLY score.ascii  
}
```

You can also perform count-based scoring:

```
score {  
    DFM ANALYZE e1 e2 e3 e4 WINDOW 100  
    [ w1*COUNT(e1)+w2*COUNT(e2)+w3*COUNT(e3)+w4*COUNT(e4) ] > 0  
    RDB ONLY score.ascii }
```

Appendix C

Results Viewing in Calibre RVE

When you generate an ASCII results file, you have the ability to use the more powerful Calibre RVE tool to inspect the results instead of using the RVE Viewing tool in Verification Center.

The process flow for using Calibre RVE with Calibre OPCverify results is as follows:

Table C-1. Calibre RVE Results Viewing Process

Stage	Task
1	Invoking and Configuring Calibre RVE
2	<ul style="list-style-type: none">• Selecting a Property to View• Error Properties in Tabular Format
3	Plotting Properties as a Histogram

Invoking and Configuring Calibre RVE

Follow this procedure to invoke and configure the Calibre RVE interface for use with Calibre OPCverify results.

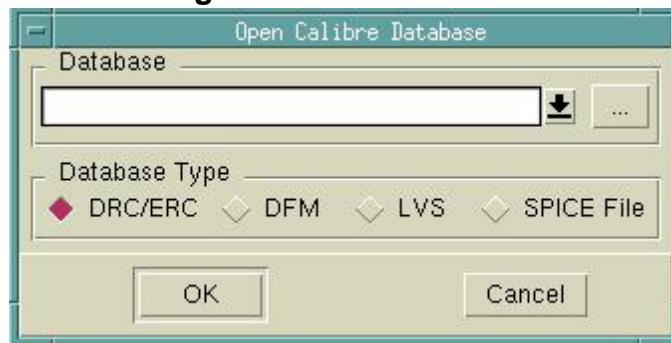
Prerequisites

- A Calibre OPCverify command file run with any command that included [Error-Centric Section Blocks](#).
- An ASCII database output produced by following the procedure “[Creating the SVRF Rule File](#)” with “Add Database and Output Reporting” (in the “Using Calibre OPCverify” chapter).
- The accompanying design file result from running the Calibre OPCverify command file.

Procedure

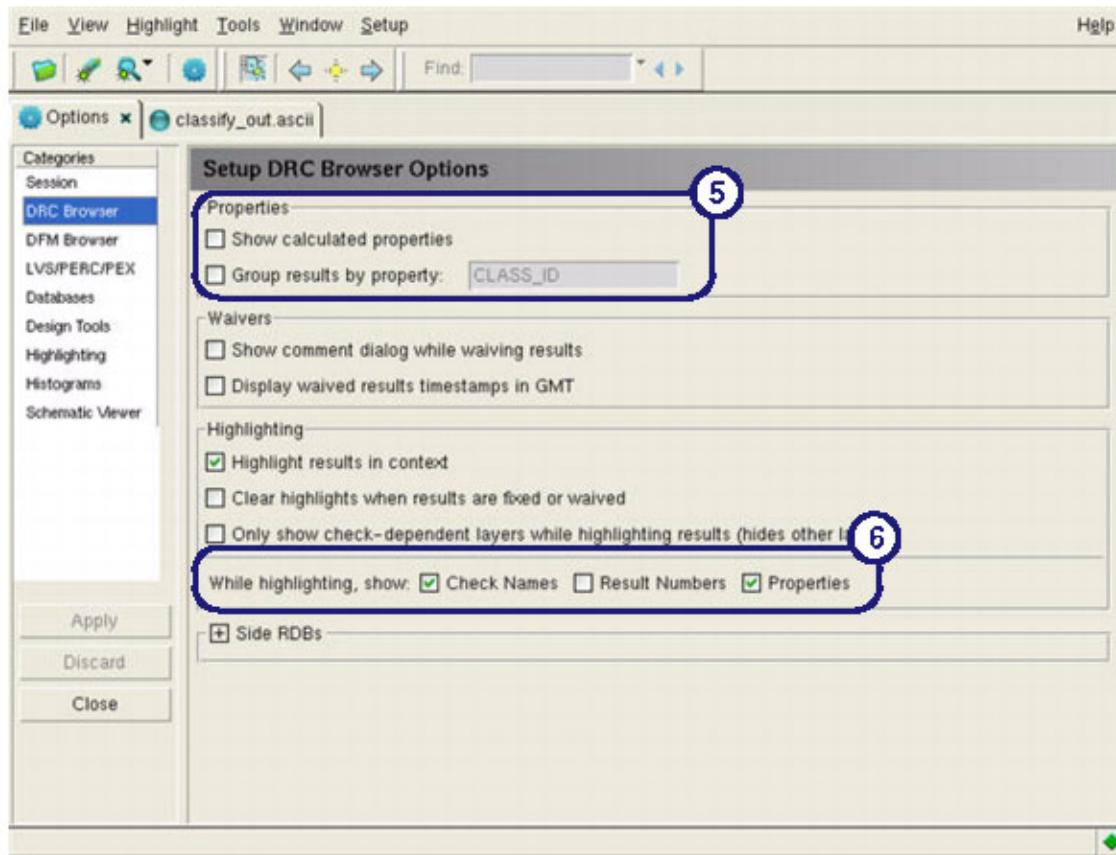
1. In Calibre WORKbench, load the design file with results on it.
2. In the **Verification** menu, select **Start RVE**.
3. In the Calibre RVE dialog box that appears, navigate to and load the ASCII database you created earlier, loading it as type DRC/ERC.

Figure C-1. Calibre RVE



- When the RVE window appears, first select the **Setup > Options** menu option, switching to the DRC Browser section (Figure C-2).

Figure C-2. Calibre RVE Options



- Turn **off** the following options:
 - Show calculated properties
 - Group results by property

6. Select one or both of the following display options:

- Display check names while highlighting
- Display result properties while highlighting

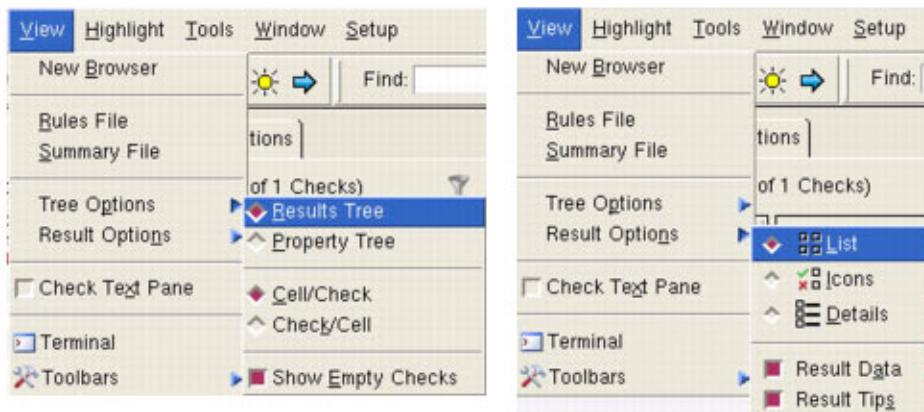
These two settings will cause Calibre RVE to mark a highlight in Calibre WORKbench with the selected information whenever highlighting commands are issued.

7. Click **Apply** to save the changes.

8. In the **View** menu, select the following menu items ([Figure C-3](#)):

- **Tree Options > Results Tree**
- **Tree Options > Cell/Check**
- **Result Options > List**

Figure C-3. Selecting Options from the RVE View Menu



Results

The window should show the rule checks as a subset of a cell and a numbered list of error markers.

Selecting a Property to View

Since the ASCII results file contains a record of the properties in the rule file, you have multiple options on how to view the properties.

Prerequisites

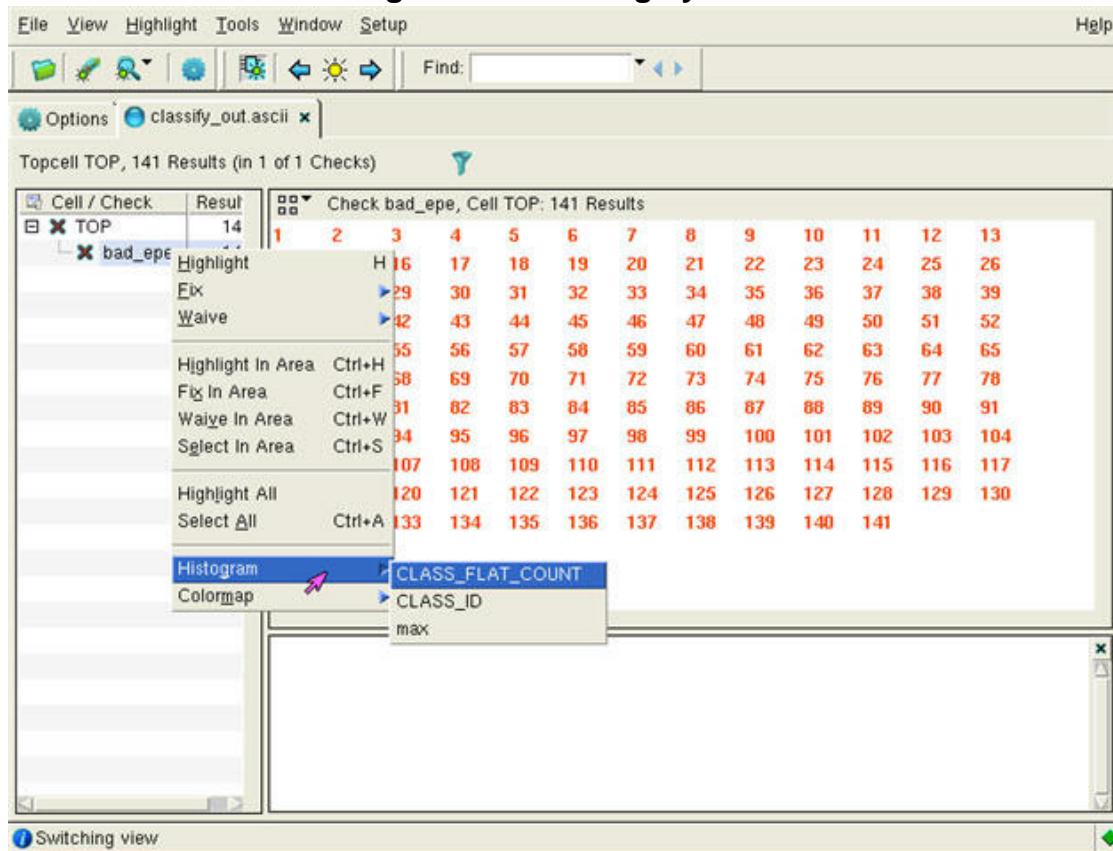
- You have completed the [Invoking and Configuring Calibre RVE](#) procedure.

Procedure

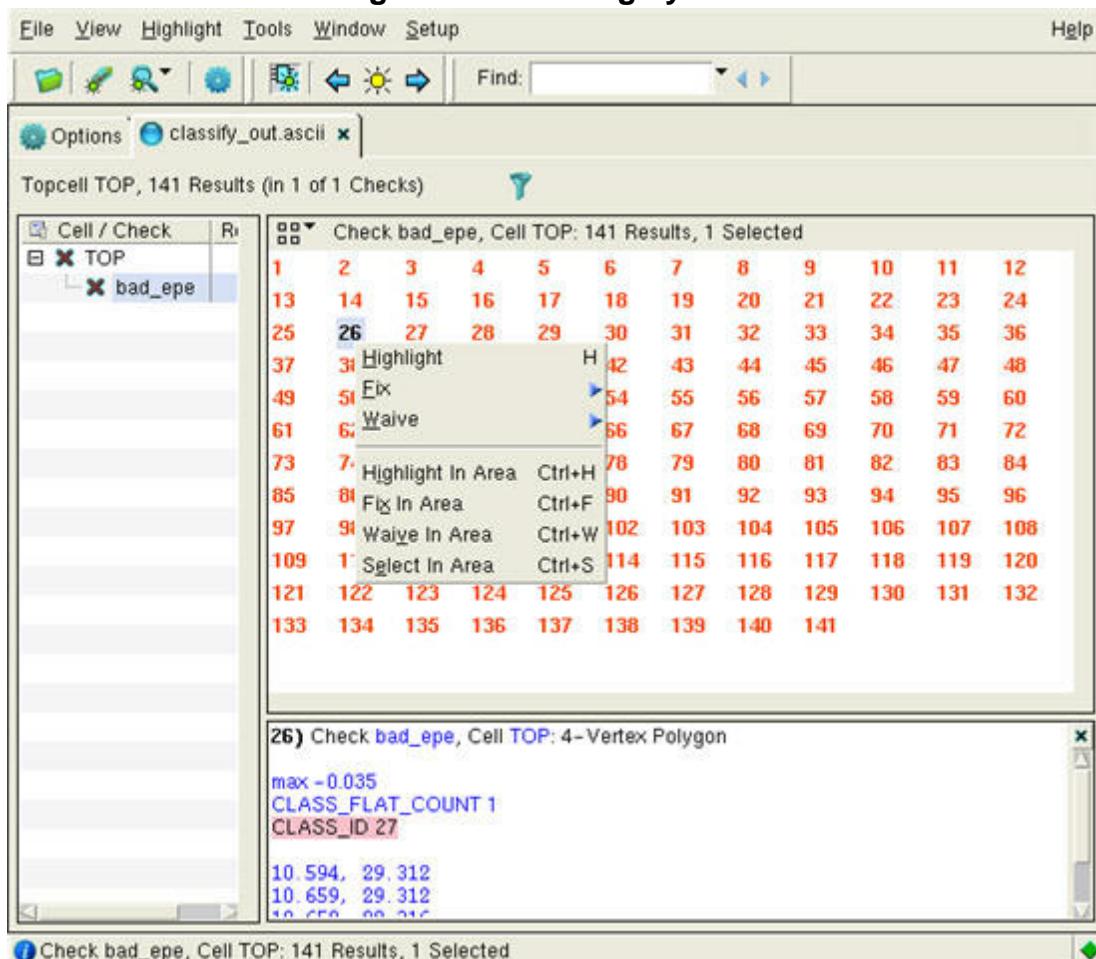
1. Click on a rule to show all the error markers available for that rule ([Figure C-4](#)).

- In the context menu for the rule (right-click the rule name) you can highlight all instances of the rule result in the Calibre WORKbench viewer by selecting **Highlight All**.
- If you right-click on a rule result from the information pane, you can display a histogram of all the values for that property type (see [Plotting Properties as a Histogram](#) for more information).

Figure C-4. Viewing by Rule



2. Click on an error to show the values for the properties for that specific error in the pane at the bottom of the window ([Figure C-5](#)).
- In the context menu for an error (right-click to bring up the menu) selecting **Highlight** shows that specific error in the Calibre WORKbench layout viewer.

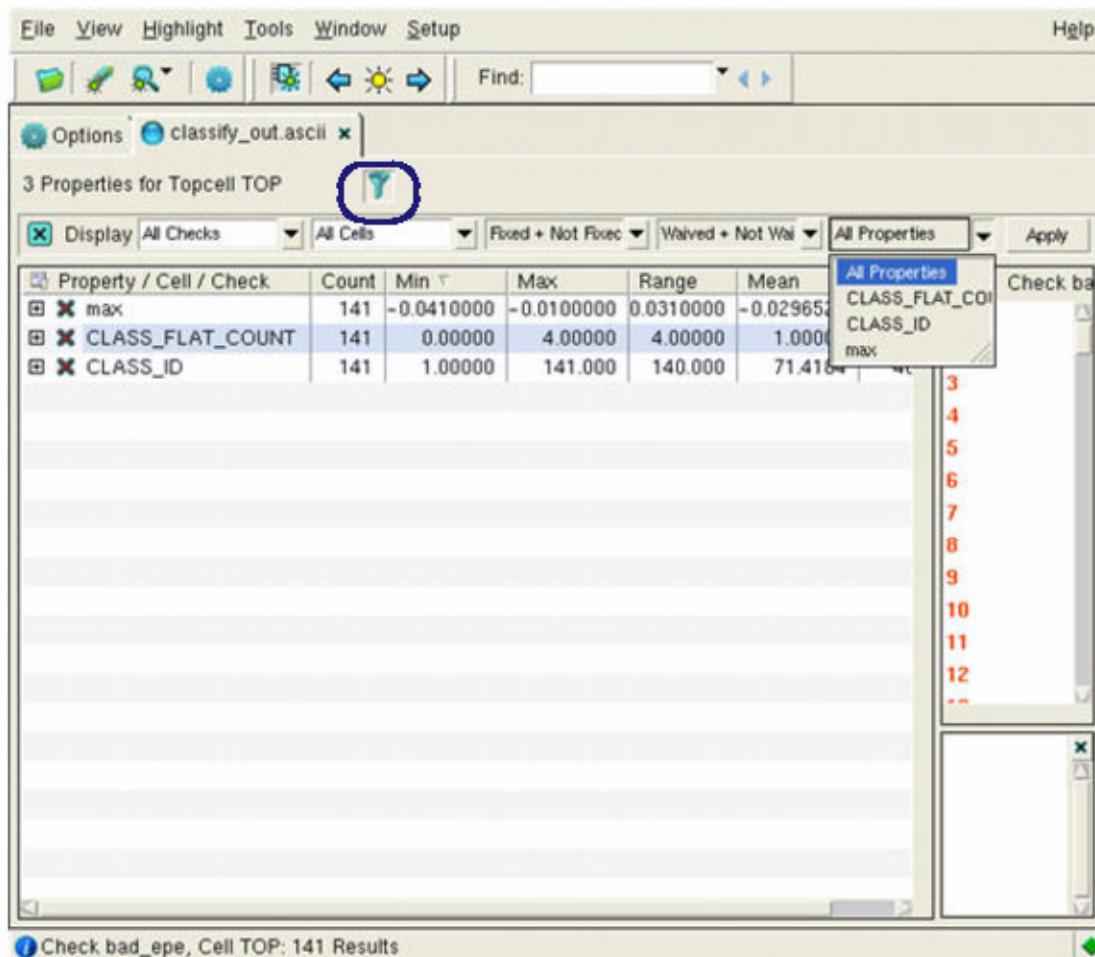
Figure C-5. Viewing by Error

Error Properties in Tabular Format

In addition to histogram display, properties can also be viewed and sorted in tabular format.

- Selecting the **View> Tree Options > Property Tree** option lists all properties associated with the rules in the database in the database tab (**Figure C-6**).

Figure C-6. Property Map



You can use filter controls by clicking the funnel button (circled) just above the table. This makes filter controls available that limit the list of properties displayed.

- All Checks - Filters by the Check you select from the dropdown.
- All Cells - Filters by the Cell you select from the dropdown.
- Fixed + Not Fixed - Not used in this task.
- Waived + Not Waived - Not used in this task.
- All Properties - Filters by the Property you select in the dropdown. Selecting a property also opens up additional fields to select a range or apply a constraint on the property values ([Figure C-7](#)).

Figure C-7. Filtering by Value

Waived + Not	max	>=	-0.041	Apply
I_epe, Cell TOP: 141 Results, FLAT_COUNT CLASS_ID				
	74	>=	00 (4) 18.615,27.744	
	75	>	00 (4) 18.696,11.981	
	76	<	00 (4) 18.913,29.312	
	77	<=	00 (4) 18.923,27.866	
	78	!=	00 (4) 19.002,16.915	

- Selecting the **View> Tree Options > Results** option again and the **View > Result Options > Details** lists all the properties by value (Figure C-8). Each property is given its own column that can be sorted by the values in that column.

Figure C-8. Results Table

FLAT_COUNT	CLASS_ID	max	Coordinates
1		-0.0340000	(4) 7.539,25.632 7.6
2		-0.0350000	(4) 7.864,10.22 7.92
3		-0.0350000	(4) 7.864,14.553 7.9
4	1	-0.0350000	(4) 7.864,16.314 7.9
5	1	-0.0350000	(4) 7.864,20.647 7.9
6	1	-0.0320000	(4) 8.069,12.203 8.1
7	1	-0.0340000	(4) 8.319,26.9 8.384
8	1	-0.0340000	(4) 8.346,12.203 8.4
9	1	-0.0360000	(4) 8.396,29.26 8.4,
10	1	-0.0140000	(4) 8.65,29.059 8.65
11	1	-0.0320000	(4) 8.958,28.998 9.0
12	1	-0.0340000	(4) 8.974,10.494 9.0
13	1	-0.0350000	(4) 9.164,18.886 9.2
14	0	-0.0310000	(4) 9.5,14.867 9.565
15	0	-0.0310000	(4) 9.5,16.0 9.565,1
16	0	-0.0310000	(4) 9.5,20.333 9.565
17	1	-0.0340000	(4) 9.619,8.3 9.684,i
18	1	-0.0140000	(4) 9.869,10.412 9.8
19	4	-0.0140000	(4) 9.869,14.745 9.8
20	n	0.0140000	(4) 9.869,16.061 9.8

Plotting Properties as a Histogram

Histograms graphically display the frequency of specific data values. In the context of a Calibre OPCverify results database, this is the frequency of properties on a derived layer resulting from a specific operation. If multiple properties were generated, each property is attached to each instance of the error on the derived layer.

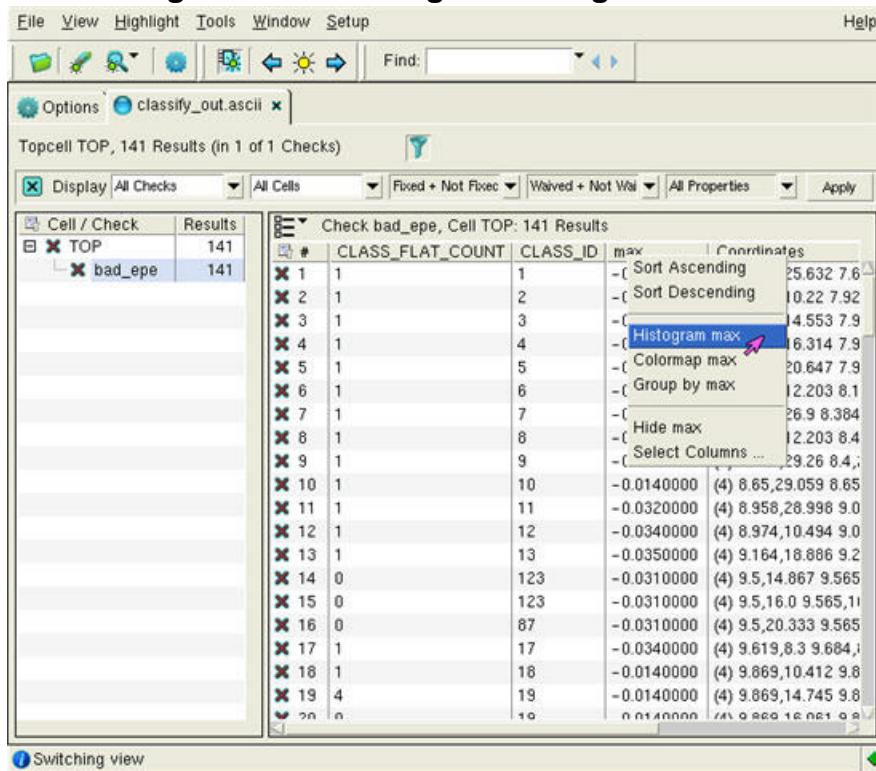
Prerequisites

- The Calibre RVE interface is opened and a database loaded (as described in [Invoking and Configuring Calibre RVE](#))

Procedure

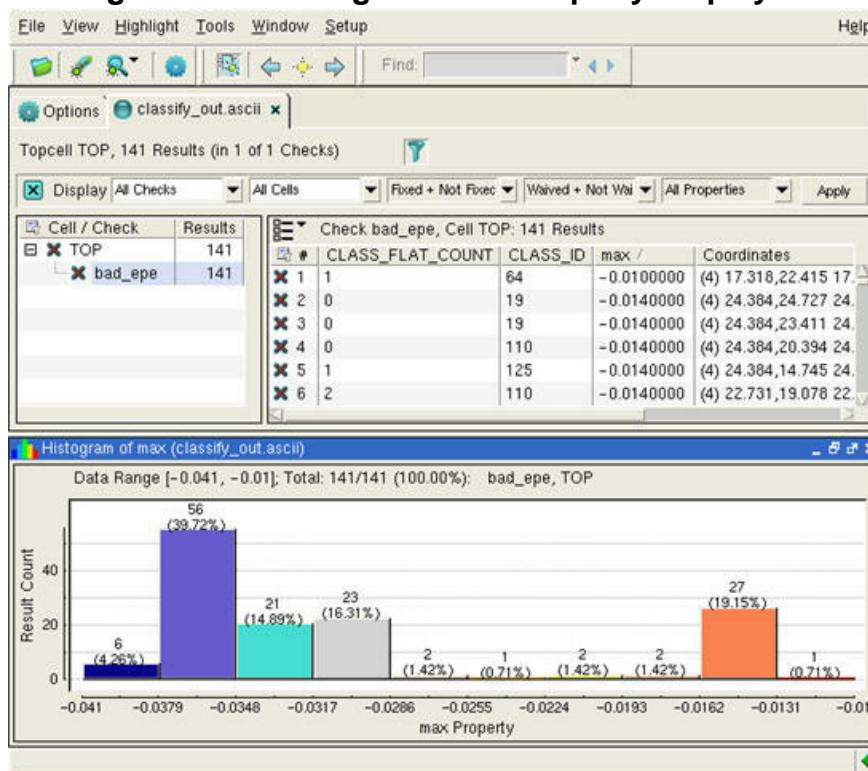
1. Select a rule by clicking on its name in the Calibre RVE viewer.
2. Right-click one of the properties at the top of its column, and select **Histogram** *propertyname* (as shown in [Figure C-9](#)).

Figure C-9. Invoking the Histogram Pane



3. When the Histograms pane appears at the bottom of the window, it shows the data ranges for the selected property ([Figure C-10](#)).

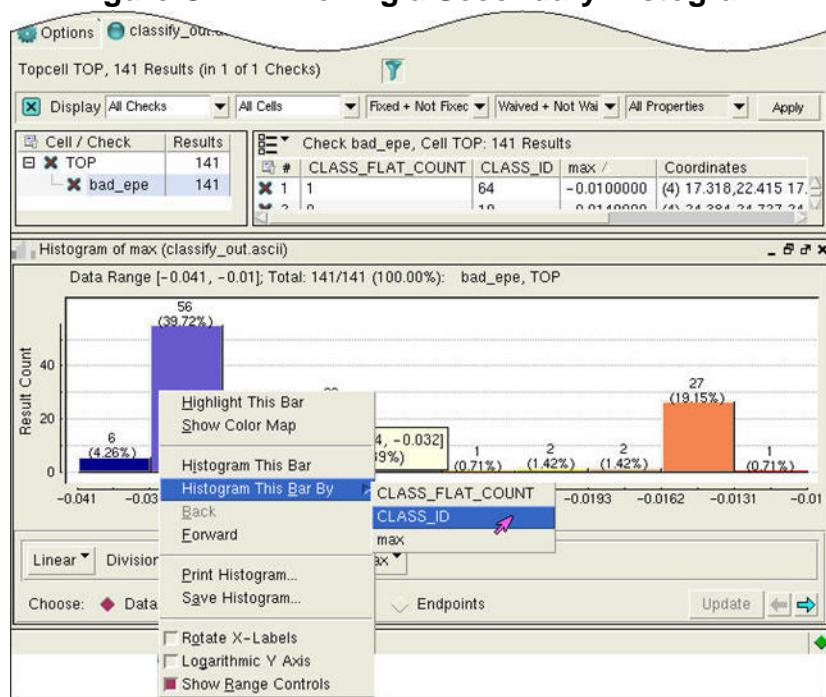
Figure C-10. Histogram for a Property Displayed



Right-click inside the Histogram pane and select **Show Range Controls** from the popup menu.

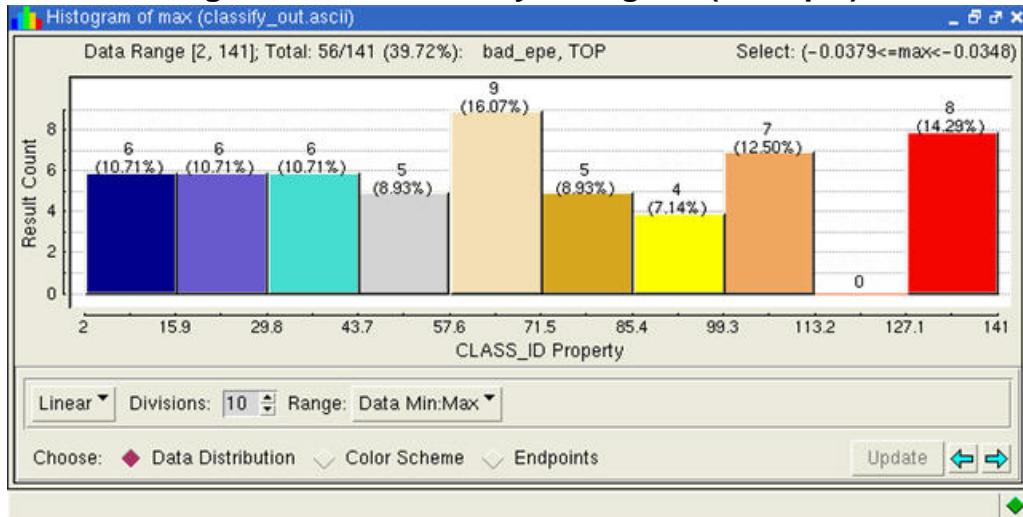
- Custom range sizes can be set by changing the Data:Min:Max dropdown to Custom Range and typing a new range in the field, or by switching the Choose radio button group from **Data Distribution** to **Endpoints** and using the slider arrows.
 - Custom bin sizes can be set to refilter the histogram as follows:
 - Switch the selector from Linear to Custom.
 - In the text field that appears, enter two or more new custom bin ranges.
 - Click **Update** to redraw the histogram with the new changes.
4. For rule results that have multiple properties, you can see a histogram of a single column for the values of a secondary property.
- Right-click one of the histogram bars ([Figure C-11](#)).
 - In the context menu that appears, select **Histogram This Bar By**.

Figure C-11. Invoking a Secondary Histogram



The histogram is replaced by the secondary histogram plot (Figure C-12). You can use the browse arrow buttons in the lower right hand corner of the window to switch back to an earlier histogram plot.

Figure C-12. Secondary Histogram (Example)



Tip

i You can further select any bar in this secondary histogram and view a histogram of another property for that results bar in a tertiary histogram plot, repeating step 4 for as long as you have additional properties of interest.

Appendix D

Calibre OPCverify Source Example: Metal1 Layer Checks

This appendix contains a heavily annotated source code listing of a typical Calibre OPCverify rule file.

Try It! 	Calibre OPCverify Tutorial and Example Kit (eKit) This eKit contains known working SVRF rule files, optical models, and a sample OASIS® design database. Accompanying documentation explains each annotated rule file and highlights expected output. Users of this eKit can learn from the example code and conduct modifications to see the effects of their changes. Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
---	---

About the Example File..... **535**

About the Example File

The combined SVRF rule file and Calibre OPCverify command file represent some suggested checks for a metal1 layer. It generates simulated image contours for the metal1 and contact layer, and then performs critical dimension checks based on those contours.

Section 1 (SVRF)

The SVRF sections of the rule file (Sections 1 through 3) follow the standard SVRF conventions; Section 1 is for setup and configuration. The layout loaded is an OASIS file located in a shared design directory that has poly, active, metal, and contact layers. OPC has already been performed on the design, and the resulting layers are also included in the design file.

```
//////////  
// 1. Setup //  
//////////  
LAYOUT SYSTEM OASIS  
LAYOUT PATH ".../shared/design/input.oas"  
LAYOUT PRIMARY **  
PRECISION 1000  
RESOLUTION 1  
UNIT LENGTH u
```

The output of the rule file will be placed in an output directory.

```
DRC RESULTS DATABASE "output/opcv_m1_out.oas" OASIS PSEUDO
DRC SUMMARY REPORT "reports/opcv_m1_out.rep"
DRC MAXIMUM RESULTS ALL
DRC MAXIMUM VERTEX ALL
```

The layout window is fixed at a certain part of the design; this is the SVRF LAYOUT WINDOW command, which should not be confused with the Calibre OPCVerify window or output_window commands.

```
LAYOUT WINDOW 5.986 5.043 27.237 32.055
LAYOUT WINDOW CLIP YES
```

Section 2 (SVRF)

This part of the SVRF section of the rule file specifies the layers that exist in the input OASIS design file, and copies them directly to the output file. Notice that the input layers are intentionally renumbered in the transfer to the output database.

```
///////////
// 2. Original Layers I/O //
///////////
LAYER POLY      1
LAYER POLY_OPCT 2
LAYER ACTIVE    4
LAYER CONTACT   5
LAYER CONTACT_OPCT 8
LAYER M1        6
LAYER M1_OPCT   7
POLY           { COPY POLY          } DRC CHECK MAP POLY      1
POLY_OPCT     { COPY POLY_OPCT    } DRC CHECK MAP POLY_OPCT 2
ACTIVE         { COPY ACTIVE       } DRC CHECK MAP ACTIVE   3
CONTACT        { COPY CONTACT      } DRC CHECK MAP CONTACT  4
CONTACT_OPCT  { COPY CONTACT_OPCT } DRC CHECK MAP CONTACT_OPCT 5
M1             { COPY M1          } DRC CHECK MAP M1        6
M1_OPCT       { COPY M1_OPCT     } DRC CHECK MAP M1_OPCT  7
```

Section 3 (SVRF)

In this section, the **LITHO OPCVERIFY** calls are defined. Each call runs the OPCVerify command file in order to retrieve one of the derived layers in the command file by name.

```
///////////
// 3. Verification Layer IO //
///////////
// Simulation Contours with different dose and focus values
img_ct_ctrl = LITHO OPCVERIFY CONTACT CONTACT_OPCT M1 M1_OPCT FILE
OPCV_M1_SETUP MAP img_ct_ctrl
```

In the code above, the CONTACT, CONTACT_OPCT, M1, and M1_OPCT layers are sent as input to the OPCVerify command file OPCV_M1_SETUP, and the derived layer img_ct_ctrl from the OPCVerify command file is saved to the SVRF layer with the same name.

You do not need to name the SVRF derived layer with the same name as the OPCVerify derived layer; this is done as a convenience.

```
img_ct_ctr      { COPY img_ct_ctr } DRC CHECK MAP img_ct_ctr 11
```

The derived layer containing the copy of the Calibre OPCVerify layer is copied to the output file as layer 11.

Similar pairs of LITHO OPCVERIFY and [COPY ... DRC CHECK MAP](#) statements are defined for each of the layers to be retrieved from the Calibre OPCVerify derived layers. You do not need to return all the layers in the Calibre OPCVerify command file; only define output layers for output you are interested in.

```
img_m1_ctr      = LITHO OPCVERIFY CONTACT CONTACT_OPCT M1 M1_OPCT FILE
OPCV_M1_SETUP   MAP img_m1_ctr
img_m1_ctr      { COPY img_m1_ctr } DRC CHECK MAP img_m1_ctr 12
// Short & open check
short           = LITHO OPCVERIFY CONTACT CONTACT_OPCT M1 M1_OPCT FILE
OPCV_M1_SETUP   MAP short
open            = LITHO OPCVERIFY CONTACT CONTACT_OPCT M1 M1_OPCT FILE
OPCV_M1_SETUP   MAP open
short           { COPY short } DRC CHECK MAP short 21
open            { COPY open } DRC CHECK MAP open 22
```

In addition to sending output to a design file viewable in Calibre WORKbench, some Calibre OPCVerify commands support calculation of value data that accompanies the command's shape output. These data items must be collected in a Calibre RVE database using the [DFM RDBSVRF](#) command, which requires the name of the relevant derived layer. In the code that follows, the short and open layers have their property values separately saved to an ASCII file.

```
rdb_short       { DFM RDB short          "./ascii/m1_out.ascii" MAXIMUM ALL
ALL CELLS NOEMPTY CHECKNAME short        }
rdb_open        { DFM RDB open           "./ascii/m1_out.ascii" MAXIMUM ALL
ALL CELLS NOEMPTY CHECKNAME open         }
```

Soft bridge and pinch calls to the Calibre OPCVerify command file are defined in the following code.

- A soft bridge is where simulated contour curves are close together, but not touching.
- A soft pinch is where simulated contour shapes are very thin, but not broken.

```
// Soft bridge and pinch check using bridge and pinch commands
bridge_v2      = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC FILE
OPCV_M1_SETUP  MAP   bridge_v2
pinch_v2       = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC FILE
OPCV_M1_SETUP  MAP   pinch_v2
bridge_v2      { COPY bridge_v2 } DRC CHECK MAP bridge_v2    33
pinch_v2       { COPY pinch_v2 } DRC CHECK MAP pinch_v2    34
rdb_bridge_v2 { DFM RDB bridge_v2      "./ascii/m1_out.ascii" MAXIMUM \
    ALL ALL CELLS NOEMPTY CHECKNAME bridge_v2 }
rdb_pinch_v2   { DFM RDB pinch_v2      "./ascii/m1_out.ascii" MAXIMUM \
    ALL ALL
CELLS NOEMPTY CHECKNAME pinch_v2 }
```

Contact coverage and contact alignment check calls are defined in the following code.

```
// Contact coverage checks
m1_ct          = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC \
FILE OPCV_M1_SETUP MAP   m1_ct
covered_ct     = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC \
FILE OPCV_M1_SETUP MAP   covered_ct
bad_ct         = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC \
FILE OPCV_M1_SETUP MAP   bad_ct
m1_ct          { COPY m1_ct } DRC CHECK MAP m1_ct      41
covered_ct     { COPY covered_ct } DRC CHECK MAP covered_ct 42
bad_ct         { COPY bad_ct } DRC CHECK MAP bad_ct      43
rdb_bad_ct     { DFM RDB bad_ct      "./ascii/m1_out.ascii" \
MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME bad_ct }
```

// Contact alignment check

```
bad_ct_al      = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC FILE \
OPCV_M1_SETUP MAP   bad_ct_al
bad_ct_al     { COPY bad_ct_al } DRC CHECK MAP bad_ct_al  51
rdb_bad_ct_al { DFM RDB bad_ct_al      "./ascii/m1_out.ascii" \
MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME bad_ct_al }
```

The Calibre OPCVerify command file also includes a bad critical dimension (CD) check; the following code defines a LITHO OPCVERIFY call to collect that output layer.

```
// CD uniformity check
bad_CD          = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC FILE \
OPCV_M1_SETUP MAP   bad_CD
filter_2D        = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC FILE \
OPCV_M1_SETUP MAP   filter_2D
filter_2D        { COPY filter_2D } DRC CHECK MAP filter_2D    61
bad_CD          { COPY bad_CD } DRC CHECK MAP bad_CD    62
rdb_bad_CD      { DFM RDB bad_CD      "./ascii/m1_out.ascii" \
MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME bad_CD }
```

Extra printing and fail to print calls will return a contour layer section that does not have a corresponding drawn shape (extra printing) and drawn shapes that do not have a corresponding contour (fail to print, also known as not printing).

```
// Extra printing and fail to print check
extra_p      = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC FILE \
OPCV_M1_SETUP MAP extra_p
fail_p       = LITHO OPCVERIFY CONTACT CONTACT_OPC M1 M1_OPC FILE \
OPCV_M1_SETUP MAP fail_p
extra_p      { COPY extra_p           } DRC CHECK MAP extra_p      91
fail_p       { COPY fail_p          } DRC CHECK MAP fail_p      92
```

Calibre OPCVerify Command File, Inline

Including the file inline is one of two possible methods to access a Calibre OPCVerify command file (the other is to name an external file in the LITHO OPCVERIFY call above for the FILE argument). In this case, the **LITHO FILE** keyword defines the named inline file definition OPCV_M1_SETUP, and is commented starting with Section 4. The entire command file must be enclosed within bracket characters ([/* and */]) to prevent it from being run as an SVRF command.

```
///////////////////////////////
// Inline Setup File           //
// OPCverify commands appear below this line. //
///////////////////////////////
[/*
```

Note

 Inside a Calibre OPCVerify command file, comments are signified by a hash symbol (#) at the beginning of a line; this is a Tcl convention. Outside the command file, double forward slashes (//) are used for comments.

Section 4 (OPCVerify Command File)

The optical and resist models used by this command file are loaded and named in this section. The models you load must be the same models used in your Calibre OPCpro or Calibre nmOPC process.

```
##### 4. Load models#####
# 4a. Specify model path
modelpath ..../shared/models
# 4b. Loading optical models
# defined_name filename
# -----
optical_model_load opt_ct CT_opt
optical_model_load opt_m1 M1_opt
# 4c. Loading resist model
resist_model_load res_ct CT.mod
```

Subsection 5 (OPCVerify Command File)

Recall that this is an inline command file that is called from the SVRF section of this rule file. Section 5 defines the input layers to the Calibre OPCVerify command file, similar to a program function's arguments.

```
#####
# 5. Identify input layers from SVRF "LITHO OPCVERIFY" func call
#####
```

The background keyword configures the background properties of the design mask for simulation purposes.

```
background atten 0.06
```

Because four layers were sent as input from the LITHO OPCVERIFY call, four layers must be designated with layer statements here. The order of the layer arguments in the SVRF call is the order they will be defined inside the OPCverify command file.

```
#      layer_name    type   transmission
#
# -----
layer contact      hidden clear
layer contact_opc hidden clear
layer m1           hidden clear
layer m1_opc       hidden clear
```

A layer's type argument defines the default optical property of the layer. In this code, all four layers are marked as hidden by default, but in Sections 6 and 7 (OPCverify Command File) the code explicitly simulates them. Any layers marked as opc or visible will be simulated with standard image commands; any layers marked as hidden will not be simulated.

Sections 6 and 7 (OPCverify Command File)

Sections 6 and 7 create contour layers for use in other Calibre OPCverify commands using the image command. In both sections, an explicit image command is used that reconfigures the named layer to be visible for the duration of the single command.

```
#####
# 6. Generate CONTACT contour
#####
setlayer img_ct_ctrl = image optical opt_ct background atten 0.06 layer \
contact_opc clear resist_model res_ct
```

This image command turns the contact_opc layer visible with a type of clear and simulates it with the opt_ct optical model and the res_ct resist model. Recall that the layer name (img_ct_ctrl) is one of the layers that is returned to the LITHO OPCVERIFY call in Section 3.

```
#####
# 7. Generate M1 contour
#####
setlayer img_m1_ctrl = image optical opt_m1 background atten 0.06 layer \
m1_opc clear aerial_contour 0.0835
```

This image command turns the contact_opc layer visible with a type of clear and simulates it with the opt_m1 optical model and an aerial contour model with a threshold of 0.0835.

Section 8 (OPCVerify Command File)

With the simulated image contours generated in Sections 6 and 7, the command file now performs its measurement checks.

```
#####
# 8 . Pinch and bridge style checks
#####
#####
# 8a. OPEN & SHORT check
# Uses the bridge and pinch commands.
#####
setlayer short      = bridge m1 img_m1_ctr output_expand 0.005 \
    property { min }
setlayer open       = pinch   m1 img_m1_ctr output_expand 0.005 \
    property { min }
```

The bridge and pinch checks shown in this example are designed to detect hard bridging and pinching. Notice that in this case, omitting a constraint value specifically detects hard bridge and pinch errors.

Note

 Section 8b was intentionally omitted from this example. It described an alternative method for pinching and bridging that was provided for backwards compatibility.

Below, the bridge constraint has been changed to < 0.060 um for the soft bridge check, and to < 0.050 um for the soft pinching check.

```
#####
# 8c. Pinch and bridge check
# Uses the bridge and pinch commands.
#####
setlayer bridge_v2 = bridge m1 img_m1_ctr < 0.060      separation 0.2 \
    max_edge 0.050 output_expand 0.005 property { min }
setlayer pinch_v2  = pinch   m1 img_m1_ctr < 0.050      separation 0.2 \
    max_edge 0.050 output_expand 0.005 property { min }
```

Notice the property {min} argument. This is an example of a simple property block, part of the set of error-centric options. The minimum value at a sample point along a pinching or bridging curve along the targeted contour segment is the one that is stored with the polygon.

Section 9 (OPCVerify Command File)

An example of a sequential combination of Calibre OPCVerify commands that result in a contact coverage check.

```
#####
# 9. Contact coverage check using area_ratio
# Checks if the coverage of the contact shape by m1 contour
# meets a certain spec.
# First, two Boolean operations are used to select the m1 contact
# target and contacts covered by m1 contour.
# The area_ratio check then computes the ratio of covered area
# to total area and flags those contacts that had a coverage < 95%.
#####
setlayer m1_ct      = and m1   contact
setlayer covered_ct = and img_m1_ctr img_ct_ctr
setlayer bad_ct     = area_ratio covered_ct m1_ct < 0.90 max_extent 0.2 \
                      output_type reference property { area_ratio }
```

Section 10 (OPCVerify Command File)

Using the generated contact contour layer (m1_ct, created in Section 9), the area_overlay command displaces the generated contact shapes a short distance (the shift argument) in each of 9 directions in order to find how well the contact covers the metal layer when slight mask misalignments occur.

```
#####
# 10. Contact alignment coverage check using area_overlay
# This command is used to check for possible poor contact coverage
# if mask misalignments happen during the real mask process.
# The m1 contacts are temporarily shifted in 9 directions by the
# shift amount to simulate slight misalignment events during this check.
# If in any direction the coverage is less than the user specified spec,
# it will be highlighted as an error.
#####
setlayer bad_ct_al = area_overlay m1_ct img_m1_ctr < 0.90 shift 0.020 \
                                max_extent 0.2 property { min }
```

Section 11 (OPCVerify Command File)

The measure_cd command returns marker shapes that are outside a specified tolerance constraint for the CD value comparing the metal1 contour (img_m1_ctr, generated in Sections 6 and 7).

```
#####
# 11. CD uniformity check for critical line width using measure_cd
# First, filter_generate is used to generate a region that blocks
# out all corners with a 100nm extension.
# The purpose of this filter is to constrain the measure_cd check
# to only check within 1D regions.
#
# The goal of the measure_cd check is to highlight places on
# critical lines where the printed CD does not meet the spec.
# cd_max puts an upper limit on the target line width. Only those
# CD lines that have a width < cd_max will be checked.
# User can use the optional parameter cd_min to set the lower limit.
# This check uses a ratio function.
# "tol not > 0.95 < 1.05 ratio" means an error will be flagged when
# the measured contour CD is outside +-5% of the target CD.
#####
setlayer filter_2D = filter_generate m1 expand 0.002 convex 0.1 concave
0.1
setlayer bad_CD = measure_cd img_m1_ctr m1 internal outside \
    filter_2D cd_max 0.1
max_search 0.05 tol not > 0.95 < 1.05 ratio \
    property {
        min_ratio
        max_ratio
        min_target
        max_target
    }
```

Notice also that in this code, the property block contains four stored properties (min_ratio, max_ratio, min_target, and max_target) that can be viewed in Calibre RVE.

Section 12 (OPCverify Command File)

The extra_printing and not_printing commands are simple contour-to-target comparison commands.

```
#####
# 12. extra_printing and not_printing checks
#####
setlayer extra_p = extra_printing m1 img_m1_ctr
setlayer fail_p = not_printing m1 img_m1_ctr
```

Section 13 (OPCverify Command File)

The output_window command must be one of the last commands in the OPCverify command file, since it requires at least one derived layer created during the Calibre OPCverify run as an argument. output_window returns all specified layers in its argument list, but only in a small radius around markers on the named layers.

```
#####
# 13. Clip output contour using output_window
# This output_window option is used to only output a layer of interest
# within a user defined halo around the error markers on that layer.
# The example below defines that only the contour layer img_ctr
# within 0.5um of the pinch_v1 and bridge_v1 errors will be output
# to the results file for review.
# This feature is highly recommended for use, since full chip
# output can take a long time and a large amount of disk space.
#####
#remove to use# output_window img_ctr halo 0.5 around pinch_v1 bridge_v1
```

Note

 An inline Calibre OPCverify command file must be closed with a specialized bracket marker (*/]).

Appendix E

Calibre OPCverify Source Example: Process Windows and TVF Conversion

This appendix contains a heavily annotated source code listing of a typical Calibre OPCverify rule file.

Try It!	Calibre OPCverify Tutorial and Example Kit (eKit)
	<p>This eKit contains known working SVRF rule files, optical models, and a sample OASIS® design database. Accompanying documentation explains each annotated rule file and highlights expected output. Users of this eKit can learn from the example code and conduct modifications to see the effects of their changes.</p> <p>Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.</p>

About the Example File	545
Converting This Module to TVF	553
Adding a TVF Header and VERBATIM Blocks	553
Converting LITHO OPCVERIFY Calls to a TVF Loop	555
Converting the setlayer image Operations to a Tcl Loop	558

About the Example File

The combined SVRF rule file and Calibre OPCverify command file create multiple process window image contours by varying the dose and defocus values on the loaded optical models. A number of commands that are used with process window contours are also included in the code.

At the end of this appendix, the section “[Converting This Module to TVF](#)” on page 553 details the methods used to improve the code by converting it to a Tcl-based TVF file.

File Section Summary

Section 1 (SVRF)

The SVRF sections of the rule file (1 through 3) follow the standard SVRF conventions. Section 1 is for setup and configuration. The layout is an OASIS file located in a shared design directory that has poly, active, metal, and contact layers. The output layers from OPC run on this layout are also included in the design database.

```
//////////  
// 1. Setup      //  
//////////  
LAYOUT SYSTEM OASIS  
LAYOUT PATH "../shared/design/input.oas"  
LAYOUT PRIMARY "*"  
PRECISION 1000  
RESOLUTION 1  
UNIT LENGTH u
```

The output of the rule file will be placed in an output directory.

```
DRC RESULTS DATABASE "output/opcv_pw_out.oas" OASIS PSEUDO  
DRC SUMMARY REPORT "reports/opcv_pw_out.rep"  
DRC MAXIMUM RESULTS ALL
```

The rule file is run on a smaller window inside the larger design database; this region is specified using the SVRF **LAYOUT WINDOW** command. This statement should not be confused with the Calibre OPCverify **window** or **output_window** commands, which operate on layer shapes instead of within the coordinates specified.

```
LAYOUT WINDOW 0 0 27 30
```

Section 2 (SVRF)

This specifies the layers that exist in the input OASIS design file, and copies them directly to the output file. Notice that one of the input layers is intentionally renumbered in the output.

```
//////////  
// 2. Original Layers I/O ///  
//////////  
LAYER POLY      1  
LAYER POLY_OPCT 2  
LAYER ACTIVE    4  
POLY           { COPY POLY          } DRC CHECK MAP POLY      1  
POLY_OPCT      { COPY POLY_OPCT    } DRC CHECK MAP POLY_OPCT 2  
ACTIVE         { COPY ACTIVE       } DRC CHECK MAP ACTIVE   3
```

Section 3 (SVRF)

In this section, the **LITHO OPCVERIFY** calls are defined. Each call runs the OPCVerify command file in order to retrieve one of the derived layers in the command file by name.

This section implements calls for multiple image contours (img_dxxx_yyyy). Only non-commented contour layers are copied to the output with **DRC CHECK MAP** commands.

```
//////////  
// 3. Verification Layer IO ///  
//////////  
// Simulation poly contours  
// NOTE: Some of these contours were intentionally  
// commented out to reduce processing time, but are  
// provided for reference.
```

The following block of text defines multiple image contour requests to the Calibre OPCverify command file.

```



```

The poly_pvb layer assignment is used to generate the process variation band from the OPCverify command file in Sections 7 and 8 (OPCverify Command File).

```

poly_pvb = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP \
MAP poly_pvb
poly_pvb { COPY poly_pvb} DRC CHECK MAP poly_pvb          40

```

The pvb_width, pvb_inner, and pvb_outer calls implement a process variation band, which is calculated in Sections 7 and 8 (OPCVerify Command File).

```
pvb_width          = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE \
    OPCV_POLY_SETUP MAP pvb_width
pvb_inner          = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE \
    OPCV_POLY_SETUP MAP pvb_inner
pvb_outer          = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE \
    OPCV_POLY_SETUP MAP pvb_outer
    pvb_width      { COPY pvb_width } DRC CHECK MAP pvb_width 51
    pvb_inner      { COPY pvb_inner } DRC CHECK MAP pvb_inner 52
    pvb_outer      { COPY pvb_outer } DRC CHECK MAP pvb_outer 53
    pvb_width_rdb { DFM RDB pvb_width      "./ascii/pw_out.ascii"
                    MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME pvb_width }
```

The calls below are used in measuring mask error enhancement factor (MEEF) on the design and are calculated in Sections 9-11 (OPCVerify Command File).

```
img_d100_nom_sz     = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE \
    OPCV_POLY_SETUP MAP img_d100_nom_sz
lg_meef = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP \
    MAP lg_meef
img_d100_nom_sz { COPY img_d100_nom_sz } DRC CHECK MAP img_d100_nom_sz 51
lg_meef { COPY lg_meef } DRC CHECK MAP lg_meef 52
lg_meef_rdb{ DFM RDB lg_meef "./ascii/pw_out.ascii" \
    MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME lg_meef }
```

The calls below are used in measuring Normalized Image Log Slope (NILS) on the design and are calculated in Section 9.

```
img_ctr_0.135       = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE \
    OPCV_POLY_SETUP MAP img_ctr_0.135
img_ctr_0.130       = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE \
    OPCV_POLY_SETUP MAP img_ctr_0.130
low_contrast        = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE \
    OPCV_POLY_SETUP MAP low_contrast
    img_ctr_0.135 { COPY img_ctr_0.135 } DRC CHECK MAP img_ctr_0.135      71
    img_ctr_0.130 { COPY img_ctr_0.130 } DRC CHECK MAP img_ctr_0.130      72
    low_contrast { COPY low_contrast } DRC CHECK MAP low_contrast         73
    low_contrast_rdb { DFM RDB low_contrast "./ascii/pw_out.ascii" \
        MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME low_contrast }
```

The calls below are used in measuring DOF (depth of focus) on the design and are calculated in Section 11.

```
low_dof            = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE \
    OPCV_POLY_SETUP MAP low_dof
    low_dof { COPY low_dof } DRC CHECK MAP low_dof
    low_dof_rdb { DFM RDB low_dof "./ascii/pw_out.ascii" \
        MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME low_dof }
```

Calibre OPCVerify Command File, Inline

Including the file inline is one of two possible methods to access a Calibre OPCVerify command file. (The other is to name an external file in the LITHO OPCVERIFY call above for the FILE

argument.) In this case, the **LITHO FILE** keyword defines the named inline file definition **OPCV_POLY_SETUP**, and is commented starting with Section 4. The entire command file must be enclosed with bracket characters ([/* and */]) to prevent it from being run as an SVRF command.

```
///////////////////////////////
// Inline Setup File           //
// OPCVerify commands appear below this line. //
/////////////////////////////
LITHO FILE OPCV_POLY_SETUP /*
```

Note

 Inside a Calibre OPCVerify command file, comments are signified by a pound sign(#) at the beginning of a line; this is a Tcl convention.

Section 4 (OPCVerify Command File)

The optical and resist models used by this command file are loaded and named for use by Calibre OPCVerify in this section. The models you load must be the same models used in your OPCpro or nmOPC process.

```
#####
# 4 Load models#
#####
# 4a. Specify model path
modelpath ..../shared/models
# 4b. Loading optical models
optical_model_load opt_poly_nom      POLY_opt_nom
optical_model_load opt_poly_n050     POLY_opt_n050
optical_model_load opt_poly_p050     POLY_opt_p050
optical_model_load opt_poly_n100     POLY_opt_n100
optical_model_load opt_poly_p100     POLY_opt_p100
# 4c. Loading resist model
resist_model_load res_poly         POLY.mod
```

Command File, Section 5 (OPCVerify Command File)

Recall that this is a command file that is called from the SVRF section of this rule file. Section 5 defines the input layers to the Calibre OPCVerify command file. Input layers are handled similar to a program function's arguments.

```
#####
# 5. Identify input layers from SVRF "LITHO OPCVERIFY" function call
#####
```

The **background** keyword configures the background properties of the design mask for simulation purposes.

```
background clear
```

The layers (3) passed from the LITHO OPCVERIFY call are identified to OPCVerify here.

```
#      layer_name type      transmission
#      -----  ----  -----
layer poly      hidden -1.245000 0.000000
layer poly_opc  visible -1.245000 0.000000
layer active    hidden -1.245000 0.000000
```

Command File, Section 6 (OPCverify Command File)

This section uses multiple [image](#) commands to illustrate process window contour generation. The command file varies the dose from 0.95 to 1.05 and the defocus using the loaded optical models at nominal (nom), negative 0.5 and 1.0, and positive 0.5 and 1.0 for the variations.

```
#####
# 6. Generate poly contours
# Not all of the following contours will be used in the final output.
#####
setlayer img_d100_nom = image optical opt_poly_nom dose 1.00 \
  resist_model res_poly
setlayer img_d100_n050 = image optical opt_poly_n050 dose 1.00 \
  resist_model res_poly
setlayer img_d100_p050 = image optical opt_poly_p050 dose 1.00 \
  resist_model res_poly
setlayer img_d100_n100 = image optical opt_poly_n100 dose 1.00 \
  resist_model res_poly
setlayer img_d100_p100 = image optical opt_poly_p100 dose 1.00 \
  resist_model res_poly
setlayer img_d095_nom = image optical opt_poly_nom dose 0.95 \
  resist_model res_poly
setlayer img_d095_n050 = image optical opt_poly_n050 dose 0.95 \
  resist_model res_poly
setlayer img_d095_p050 = image optical opt_poly_p050 dose 0.95 \
  resist_model res_poly
setlayer img_d095_n100 = image optical opt_poly_n100 dose 0.95 \
  resist_model res_poly
setlayer img_d095_p100 = image optical opt_poly_p100 dose 0.95 \
  resist_model res_poly
setlayer img_d105_nom = image optical opt_poly_nom dose 1.05 \
  resist_model res_poly
setlayer img_d105_n050 = image optical opt_poly_n050 dose 1.05 \
  resist_model res_poly
setlayer img_d105_p050 = image optical opt_poly_p050 dose 1.05 \
  resist_model res_poly
setlayer img_d105_n100 = image optical opt_poly_n100 dose 1.05 \
  resist_model res_poly
setlayer img_d105_p100 = image optical opt_poly_p100 dose 1.05 \
  resist_model res_poly
```

Command File, Sections 7 and 8 (OPCverify Command File)

Two methods of working with process variation bands are shown in Sections 7 and 8.

- The [pvband](#) command draws a process variation band based on the input contours.
- The [contour_diff](#) command returns markers on parts of the inner and outer contours that have a variation difference of greater than 0.012 microns.

```
#####
# 7. Generate pvbnd
#####
setlayer poly_pvb = pvbnd img_d100_nom img_d100_n100 img_d100_p100
#####
# 8. Generate pvbnd inner outer contour & checking pvb width
#####
setlayer pvb_inner = and img_d100_nom img_d100_n100 img_d100_p100
setlayer pvb_outer = or img_d100_nom img_d100_n100 img_d100_p100
setlayer pvb_width = contour_diff > 0.012 poly pvb_inner
pvb_outer max_search 0.060 contour_diff_spacing 0.005
property { max }
```

Command File, Sections 9-11 (OPCverify Command File)

The following three sections calculate typical tests associated with process windows:

- MEEF, which tests the sensitivity of the mask to small sizing differences (the size argument of the image command scales the mask up by 0.002 microns)
- NILS, testing the viability of the edge definition for the resist for the aerial image contours at 0.135 and 0.130
- DOF, testing the tolerance of the image on five contours from negative 1.0 to positive 1.0

```
#####
# 9. Measure MEEF with 2nm of mask bias
#####
setlayer img_d100_nom_sz = image optical opt_poly_nom dose 1.00 \
    size 0.002 resist_model res_poly
setlayer lg_meef = meefcheck > 4 poly img_d100_nom_sz img_d100_nom \
    delta_mask 0.002 max_search 0.060 meef_spacing 0.005 property \
{ max }
#####
# 10. Measure NILS
#####
setlayer img_ctr_0.135 = image optical opt_poly_nom aerial_contour 0.135
setlayer img_ctr_0.130 = image optical opt_poly_nom aerial_contour 0.130
setlayer low_contrast = nilscheck < 0.5 poly max_search 0.060 images \
    img_ctr_0.135 img_ctr_0.130 levels 0.135 0.130 nils_spacing 0.020 \
    property { min }
#####
# 11. Measure DOF (50 nm in terms of focus)
#####
setlayer low_dof = dofcheck < 0.05 poly images \
    img_d100_nom img_d100_n100 img_d100_n050 img_d100_p050 img_d100_p100 \
    defocus 0 -0.1 -0.05 0.05 0.1 \
    max_search 0.060 tolerance 0.1 inside active property { min }
```

Command File, Section 12 (OPCverify Command File)

The output_window command must be one of the last commands in the OPCverify command file, because it requires at least one derived layer created during the Calibre OPCverify run as an argument. The output_window command returns all specified layers in its argument list, but only in a small radius around markers on the named layers.

```
#####
# 12. dynamic output
#####
setlayer c_poly      = copy poly
setlayer c_poly_opc = copy poly_opc
setlayer c_active    = copy active
output_window c_poly      halo 0.5 around low_contrast pvb_width
output_window c_poly_opc halo 0.5 around low_contrast pvb_width
output_window c_active    halo 0.5 around low_contrast pvb_width
output_window img_ctr_0.135 halo 0.5 around low_contrast
output_window img_ctr_0.130 halo 0.5 around low_contrast
output_window pvb_inner   halo 0.5 around pvb_width
output_window pvb_outer   halo 0.5 around pvb_width
```

This command file also features the use of a dynamic output block following the `output_window` command. It is used to write out snapshots of specific layers in the design that are timestamped. For more information, see the [dynamic_output](#) command.

```
dynamic_output {
    path ./test
    format oasis
    max_frequency 5
    max_file_queue 0
    max_dump_layer_edges 500
    layers low_contrast img_ctr_0.135 img_ctr_0.130 c_poly c_poly_opc \
        c_active {
            file_prefix low_contrast
        }
    layers pvb_width pvb_inner pvb_outer {
        file_prefix pvb_width
    }
}
```

Note

 An inline Calibre OPCVerify command file must be closed with a specialized bracket marker (*/]).

Converting This Module to TVF

Tcl Verification Format (TVF) is an enhancement to the Calibre Tools engine that allows Tcl programmers to implement Tcl as part of an SVRF rule file. Calibre OPCverify itself is a Tcl-based language, and therefore also supports Tcl intermixed with its commands.

Note

 A detailed list of TVF commands and their usage are described in the [Standard Verification Rule Format \(SVRF\) Manual](#).

This section converts the rule file and Calibre OPCverify command file documented earlier in this chapter into an equivalent file incorporating TVF and Tcl commands. Specifically, the following changes are performed:

1. Add the TVF header. Any SVRF commands that do not need to be modified are encapsulated in VERBATIM blocks.
2. Implement LITHO OPCVERIFY calls as a more compact looping TVF structure.
3. Add the beginning of a second VERBATIM block to block out any remaining calls until the beginning of the Calibre OPCverify command file (if the command file is inline, such as in this example).
4. Implement the corresponding Calibre OPCverify command file setlayer image generation commands as the equivalent Tcl loop.

Adding a TVF Header and VERBATIM Blocks [553](#)

Converting LITHO OPCVERIFY Calls to a TVF Loop [555](#)

Converting the setlayer image Operations to a Tcl Loop [558](#)

Adding a TVF Header and VERBATIM Blocks

Similar to designating a text file as a shell script or HTML file, a single header command at the top of an SVRF rule file indicates to the Calibre engine that it should treat the file as a TVF file.

In order to avoid introducing coding errors, converting non-essential parts of the code to be used “as is” is recommended. The tvf::VERBATIM command performs this function by instructing the SVRF interpreter to treat anything within a VERBATIM block as normal SVRF code.

For more information on VERBATIM blocks, see the section “[Compile-Time TVF](#)” in the [Standard Verification Rule Format \(SVRF\) Manual](#).

Prerequisites

- A known working rule file.

Procedure

1. Save a copy of your working text file, then edit the copy.
2. At the top of the file, add the following line:

```
#!tvf
```

3. Starting right after the TVF header block, add the VERBATIM encapsulation keyword:

```
tvf::VERBATIM {
```

4. Scroll down in the code to just above the text to be converted to Tcl (in this case, right after the comment block for Section 3) and add a closing brace. The example code closes the block right above the LITHO OPCVERIFY calls.

Results

Additions are highlighted in the code following.

```
#!tvf
tvf::VERBATIM {
///////////////
// Calibre(R) OPCverify(tm) Module 7 //
// Process Windows and Dynamic Output//
// Example Ruledeck //
// v2009.2 (6/01/09) //
///////////////
///////////////
/////
// This file includes SVRF/TVF Technology under license by Siemens EDA.
////
// Corporation. "SVRF/TVF Technology" shall mean Siemens EDA's Standard
// Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF")
// proprietary syntaxes for expressing process rules. You shall not use
// SVRF/TVF Technology unless you are a Siemens EDA customer as
// defined by having authorized access to Siemens EDA'
// password protected support site at http://support.sw.siemens.com.
// The exact terms of your obligations and rights are governed by your
// respective license. You shall not use SVRF/TVF Technology except:
// (a) for your internal business purposes and (b) for use with
// Siemens' Calibre(R) tools. All SVRF/TVF Technology
// constitutes or contains trade secrets and confidential information
// of Siemens EDA or its licensors. You shall not make SVRF/TVF
// Technology available in any form to any person other
// than your employees and on-site contractors, excluding Siemens EDA
// competitors, whose job performance requires access and who are under
// obligations of confidentiality.
///////////////
///
```

```
///////////
// 1. Setup //
///////////
LAYOUT SYSTEM OASIS
LAYOUT PATH "../shared/design/input.oas"
LAYOUT PRIMARY "*"
PRECISION 1000
RESOLUTION 1
UNIT LENGTH u
DRC RESULTS DATABASE "output/opcv_pw_out.oas" OASIS PSEUDO
DRC SUMMARY REPORT "reports/opcv_pw_out.rep"
DRC MAXIMUM RESULTS ALL
LAYOUT WINDOW 0 0 27 30
///////////
// 2. Original Layers I/O /**
/////////
LAYER POLY 1
LAYER POLY_OPCT 2
LAYER ACTIVE 4
POLY { COPY POLY } DRC CHECK MAP POLY 1
POLY_OPCT { COPY POLY_OPCT } DRC CHECK MAP POLY_OPCT 2
ACTIVE { COPY ACTIVE } DRC CHECK MAP ACTIVE 3
/////////
// 3. Verification Layer IO /**
/////////
}
```

Converting LITHO OPCVERIFY Calls to a TVF Loop

Since LITHO OPCVERIFY calls have the same basic structure, you can use a TVF loop to consolidate multiple calls into a single loop. This method can be used to easily expand the number of derived layers retrieved from a Calibre OPCVerify file.

Prerequisites

- Completed the task “[Adding a TVF Header and VERBATIM Blocks](#)” on page 553.
- For this example, an understanding of the following code from Section 3. There are two distinct blocks in the sample code: the LITHO OPCVERIFY call and the DRC CHECK MAP calls.

```
/////////
// 3. Verification Layer IO /**
/////////
// Simulation poly contours
// NOTE: Some of these contours were intentionally
// commented out to reduce processing time, but are
// provided for reference.
```

```
img_d100_nom = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d100_nom
img_d100_n050 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d100_n050
img_d100_p050 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d100_p050
img_d100_n100 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d100_n100
img_d100_p100 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d100_p100

img_d095_nom = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d095_nom
img_d095_n050 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d095_n050
img_d095_p050 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d095_p050
img_d095_n100 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d095_n100
img_d095_p100 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d095_p100

img_d105_nom = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d105_nom
img_d105_n050 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d105_n050
img_d105_p050 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d105_p050
img_d105_n100 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d105_n100
img_d105_p100 = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP MAP img_d105_p100

img_d100_nom { COPY img_d100_nom } DRC CHECK MAP img_d100_nom      11
img_d100_n050 { COPY img_d100_n050 } DRC CHECK MAP img_d100_n050    12
img_d100_p050 { COPY img_d100_p050 } DRC CHECK MAP img_d100_p050    13
img_d100_n100 { COPY img_d100_n100 } DRC CHECK MAP img_d100_n100    14
img_d100_p100 { COPY img_d100_p100 } DRC CHECK MAP img_d100_p100    15

img_d095_nom { COPY img_d095_nom } DRC CHECK MAP img_d095_nom 21
// img_d095_n050 { COPY img_d095_n050 } DRC CHECK MAP img_d095_n050 22
// img_d095_p050 { COPY img_d095_p050 } DRC CHECK MAP img_d095_p050 23
img_d095_n100 { COPY img_d095_n100 } DRC CHECK MAP img_d095_n100 24
img_d095_p100 { COPY img_d095_p100 } DRC CHECK MAP img_d095_p100 25

img_d105_nom { COPY img_d105_nom } DRC CHECK MAP img_d105_nom 31
// img_d105_n050 { COPY img_d105_n050 } DRC CHECK MAP img_d105_n050 32
// img_d105_p050 { COPY img_d105_p050 } DRC CHECK MAP img_d105_p050 33
img_d105_n100 { COPY img_d105_n100 } DRC CHECK MAP img_d105_n100 34
img_d105_p100 { COPY img_d105_p100 } DRC CHECK MAP img_d105_p100 35
```

Procedure

1. If possible, set aliases for common strings, such as the list of input layers.

Since all the input layers sent to LITHO OPCVERIFY contain the same list, they can be assigned to a single variable:

```
set input_list { POLY POLY_OPC ACTIVE }
```

2. Analyze the code, then create lists containing the variations on the LITHO OPCVERIFY calls.

In the code above, there are three sets of five LITHO OPCVERIFY calls with the dose as the variation, found by looking at the DRC CHECK MAP calls:

- Layers 11, 12, 13, 14, and 15 have d100 in common, start with nom, then alternate n and p, and have two 050 layers followed by two 100 layers.
- Layers 21, 22, 23, 24, and 25 have d095 in common, start with nom, alternate n and p, and have two 050 layers followed by two 100 layers.

- Layers 31, 32, 33, 34, and 35 have d105 in common, start with nom, alternate n and p, and have two 050 layers followed by two 100 layers.

This repeating loop can be implemented in a number of ways, but the least complex solution is to create 1-dimensional lists for each variable element (layer number and composite name).

```
set lyrlist {11 12 13 14 15 21 22 23 24 25 31 32 33 34 35}
set lyrname {d100_nom d100_n050 d100_p050 d100_n100 d100_p100
d095_nom d095_n050 d095_p050 d095_n100 d095_p100 d105_nom
d105_n050 d105_p050 d105_n100 d105_p100}
```

3. Write a Tcl loop to replace the individual calls with the list substitution. Note that the LITHO OPCVERIFY call for TVF is slightly different. A simple count variable is used to keep track of which element of the list to retrieve.

```
set lnum 0
foreach layerloop $lyrlist {

    tvf::setlayer img_[lindex $lyrname $lnum] = "LITHO OPCVERIFY
        $input_list FILE OPCV_POLY_SETUP MAP img_[lindex $lyrname $lnum]"

    tvf::RULECHECK img_[lindex $lyrname $lnum]
        "tvf::COPY img_[lindex $lyrname $lnum]"

    tvf::DRC "CHECK MAP img_[lindex $lyrname $lnum] $layerloop"
        incr lnum

}
```

4. Add the start of a new VERBATIM block below the loop to encapsulate everything that was after the TVF loop into the block. In other words, this includes any additional inline calls to the Calibre OPCVerify command file from this example code that were not part of the TVF loop.

```
tvf::VERBATIM {

poly_pvb = LITHO OPCVERIFY POLY POLY_OPC ACTIVE FILE OPCV_POLY_SETUP
MAP poly_pvb

poly_pvb { COPY poly_pvb } DRC CHECK MAP poly_pvb
40

...
}
```

5. After the end of the example code (after the closing square bracket (*/])), close the VERBATIM block.

```
layers pvb_width pvb_inner pvb_outer {
    file_prefix pvb_width
}
*/
}
```

Results

The code should now have two distinct VERBATIM blocks, one above and one below the TVF loop you have implemented.

Converting the setlayer image Operations to a Tcl Loop

While you could run the code “as is” at this point, this task demonstrates how to implement the matching image commands.

These were the commands you set up calls for in “[Converting LITHO OPCVERIFY Calls to a TVF Loop](#)” on page 555 to a Tcl loop.

The following code will be converted:

```
#####
# 6. Generate poly contours
# Not all of the following contours will be used in the final output.
#####
setlayer img_d100_nom = image optical opt_poly_nom dose 1.00
resist_model res_poly
setlayer img_d100_n050 = image optical opt_poly_n050 dose 1.00
resist_model res_poly
setlayer img_d100_p050 = image optical opt_poly_p050 dose 1.00
resist_model res_poly
setlayer img_d100_n100 = image optical opt_poly_n100 dose 1.00
resist_model res_poly
setlayer img_d100_p100 = image optical opt_poly_p100 dose 1.00
resist_model res_poly
setlayer img_d095_nom = image optical opt_poly_nom dose 0.95
resist_model res_poly
setlayer img_d095_n050 = image optical opt_poly_n050 dose 0.95
resist_model res_poly
setlayer img_d095_p050 = image optical opt_poly_p050 dose 0.95
resist_model res_poly
setlayer img_d095_n100 = image optical opt_poly_n100 dose 0.95
resist_model res_poly
setlayer img_d095_p100 = image optical opt_poly_p100 dose 0.95
resist_model res_poly
setlayer img_d105_nom = image optical opt_poly_nom dose 1.05
resist_model res_poly
setlayer img_d105_n050 = image optical opt_poly_n050 dose 1.05
resist_model res_poly
setlayer img_d105_p050 = image optical opt_poly_p050 dose 1.05
resist_model res_poly
setlayer img_d105_n100 = image optical opt_poly_n100 dose 1.05
resist_model res_poly
setlayer img_d105_p100 = image optical opt_poly_p100 dose 1.05
resist_model res_poly
```

Prerequisites

- “[Adding a TVF Header and VERBATIM Blocks](#)” on page 553.
- Implementing the SVRF code as a TVF loop (as described in “[Converting LITHO OPCVERIFY Calls to a TVF Loop](#)” on page 555) is not required, but is recommended in similar situations.

Procedure

1. Analyze the code, then set up lists based on the repeating sequence. While a simple set of lists similar to the one described in “[Converting LITHO OPCVERIFY Calls to a TVF Loop](#)” on page 555 could be used here, this example demonstrates the efficiency of a nested loop solution.
 - Five optical models are in use in the sequence of code shown above: opt_poly_nom, opt_poly_n050, opt_poly_p050, opt_poly_n100, and opt_poly_p100. These models each represent a different defocus level.

```
set optmod {nom n050 p050 n100 p100}  
  
set optprefix opt_poly_
```

- For each set of five optical models, the dose is consistent: dose 1.00, dose 0.95, and dose 1.05.
- ```
set doselist {1.00 0.95 1.05}
```
- The derived layer name is based on the optical model and dose.

```
set dosename {d100 d095 d105}
```

A nested loop using the code above could use the doselist as the outer loop and the optical models as the inner loop.

2. Implement a loop based on the repeating sequence.

```
set optnum 0
set dnum 0
foreach dlist $dosename {
 set dval [lindex $doselist $dnum]
 foreach optlist $optmod {
 setlayer img_${dlist}_${optlist} = image optical $optprefix$optlist
 dose $dval resist_model res_poly
 }
 incr dnum
 incr optnum
}
```

3. Save the code and run the rule file.

## Results

You should get identical results as the file from the run that does not have converted Tcl.



# Appendix F

## Creating DDE Runs in OPCverify

---

Direct Data Entry (DDE) is run using various litho flat commands in Calibre OPCverify.

|                                                                       |            |
|-----------------------------------------------------------------------|------------|
| <b>DDE and Litho Flat Overview</b> .....                              | <b>561</b> |
| <b>DDE and Litho Flat Workflow</b> .....                              | <b>561</b> |
| <b>Key Concepts</b> .....                                             | <b>562</b> |
| <b>DDE and Litho Flat Product Requirements</b> .....                  | <b>562</b> |
| <b>List of Tasks</b> .....                                            | <b>563</b> |
| Finding Extents for Full Chip Runs .....                              | 563        |
| Creating a Litho Flat SVRF Rule File Header and OPCverify Calls ..... | 564        |
| Setting up direct_input in the OPCverify Command File. ....           | 566        |
| Finding Extents for Small Designs .....                               | 567        |
| <b>Lithoflat Sample Code</b> .....                                    | <b>569</b> |

## DDE and Litho Flat Overview

Using litho flat mode for Calibre OPCverify is an alternate method to quickly process flattened data with Calibre OPCverify commands.

Litho flat mode is activated either by running an SVRF command (RET GLOBAL LITHOFLAT), [processing\\_mode](#) flat, or adding the [direct\\_input](#) command block in an existing rule file. While the [direct\\_input](#) method is not required for litho flat processing, it can increase the benefits litho flat mode provides, and is the focus of this appendix.

- 
- Tip**  More information on litho flat operations and direct data access can be found in the “Litho Flat Processing Mode” section of the [Calibre Post-Tapeout Flow Users Manual](#).
- 

## DDE and Litho Flat Workflow

You will be bypassing the HDB engine with the use of a dummy file that is crafted to parallel the layers in the real target file. You may want to re-run a design more than once using different tile sizes until you find the best tilemicrons setting.

## Key Concepts

Certain DDE-specific key concepts may prove helpful in learning how to create litho flat rule files.

- **Direct Data Entry (DDE)** — The practice of reading in an input layout directly by Calibre OPCverify, bypassing the HDB engine and skipping HDB construction time.
- **Litho Flat Mode** — The process described in this document. Calibre OPCverify reads in data, flattens it into tiles (rectangular subsections of the design), runs the instructed rulechecks, then pushes the data back into the hierarchy (into uniquely placed ICV cells).
- **direct\_input** — The primary Calibre OPCverify command used in this process. Direct input is an implementation of DDE, which loads a mostly-empty dummy file initially in the SVRF rule file but then swaps the actual file in when the Calibre OPCverify command file. A key requirement for direct\_input is to define the extent for the layout using one of the following tasks:
  - For small designs and layout window style checks, use the **POLYGON** draw statement as described in the task “[Finding Extents for Small Designs](#)” on page 567.
  - For large designs, use the task “[Setting up direct\\_input in the OPCverify Command File](#)” on page 566.
- **Calibre OPCverify Command File** — Calibre OPCverify runs from a specialized command file that contains both setup information specific to Calibre OPCverify and commands to the Calibre OPCverify simulation engine. This command file is only called from a LITHO OPCVERIFY statement from within an SVRF rule file. It can be an inline command file defined by a LITHO FILE statement or a separate file.

## DDE and Litho Flat Product Requirements

The DDE and litho flat flow requires the following tools:

- Calibre WORKbench
- Calibre OPCverify

# List of Tasks

The following table shows the tasks to implement litho flat rule files:

**Table F-1. Litho Flat in OPCverify Using direct\_input Tasks**

| Task                                                                                                                                                        | Described In                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Add designated dummy layers to the SVRF rule file. Find the extent that will be used for direct input.                                                      | Use one of the following two tasks: <ul style="list-style-type: none"><li>• <a href="#">Finding Extents for Full Chip Runs</a></li><li>• (alternatively: <a href="#">Finding Extents for Small Designs</a>)</li></ul> |
| Create the SVRF header with “dummy” bypass setup statements.                                                                                                | <ul style="list-style-type: none"><li>• <a href="#">Creating a Litho Flat SVRF Rule File Header and OPCverify Calls</a></li></ul>                                                                                     |
| Add required layer calls using direct_input to the Calibre OPCverify command file, which can be a separate file or a LITHO FILE block inside the SVRF file. | <ul style="list-style-type: none"><li>• <a href="#">Setting up direct_input in the OPCverify Command File</a></li></ul>                                                                                               |

## Finding Extents for Full Chip Runs

For full chip, you find the extent by loading a dummy file to the Calibre engine, and then reading the extent information directly from the actual intended design file.

**Note**

 There is an alternative step for this flow. See also “[Finding Extents for Small Designs](#)” on page 567.

### Prerequisites

- A flat design file (OASIS format only)

### Procedure

1. Create a mostly-empty OASIS design file. It must have at least the following characteristics:
  - Its database precision must match the actual design file’s precision setting.
  - Its grid size must match the actual design file’s grid size setting.
  - It must have at least one drawn object on layer 1. The layer should be named “dummy”.

Save the file as *dummy.oas*.

2. In a text editor, create an SVRF rule file. It should use the following format (use the proper values where needed):

- *extract\_layer\_num* must match the layer you want to get the extent from in *design\_filename.oas*, which is the actual design file.
- *precision\_value* must be the correct value.

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "./input/dummy.oas"
LAYOUT PRIMARY "*"
PRECISION precision_value
LAYER dummy 1
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "./output/dummy_opcv.oas" OASIS
DRC SUMMARY REPORT "./reports/dummy_opcv.rep"
// derive a layer to the extent of input file,
//this will also be the input
// file to OPCV in the second Calibre job
input_extent = MDP OASIS_EXTENT MAP initial_extent FILE [
 vboasis_path "./input/design_filename.oas"
 vboasis_precision_multiplier auto
 layout_subset initial_extent extract_layer_num
]
input_extent { COPY input_extent } DRC CHECK MAP input_extent
OASIS 0
```

This rule file uses the Calibre MDPview command MDP OASIS\_EXTENT (described in the [Calibre Mask Data Preparation User's and Reference Manual](#)) to scan and extract the extent information from the named design file.

3. Save the rule file as *extract.svrf* and run it in Calibre:

```
calibre -drc -hier extract.svrf
```

## Creating a Litho Flat SVRF Rule File Header and OPCverify Calls

The second step in the litho flat OPCverify flow is to start an SVRF rule file. This is a separate SVRF rule file that runs on the flat design file and conditions the data to work with Calibre OPCverify.

---

### Tip

 Developing this file is separated into two sections. This task is section 1 of 2.

---

### Prerequisites

- The output OASIS design file, named ‘dummy\_opcv.oas’ resulting from “[Finding Extents for Full Chip Runs](#)” on page 563. This is used as the input file in this task.

## Procedure

1. In a text editor, open a new file, and enter the following SVRF commands as the header:

```
LAYOUT SYSTEM OASIS
LAYOUT PRIMARY "*"
LAYOUT PATH "./dummy_opcv.oas"
LAYOUT ULTRA FLEX YES
PRECISION precision_value
DRC MAXIMUM RESULTS ALL
DRC MAXIMUM VERTEX ALL
DRC SUMMARY REPORT "./reports/lithoflat.report"
DRC RESULTS DATABASE "./output/lithoflat.oas" OASIS PSUEDO
```

These commands perform the following functions:

- Sets up the input file (*dummy\_opcv.oas*).
- Sets the input and output precision. The precision of the dummy file must match the actual file.
- Enables Calibre ULTRA FLEX data processing. (See the *Calibre Post-Tapeout Flow User's Guide* for more information on the LAYOUT ULTRA FLEX command. Calibre TURBO Flex is not currently supported.)

2. For each layer of interest in the actual input file, create a dummy layer declaration:

```
layer dummy1 1
layer dummy2 2
...
```

3. For each layer you declared, add copy statements to copy the dummy layer (which will be replaced by an actual target layer) to the output:

```
dummy1 {copy dummy1} drc check map dummy1 1
dummy2 {copy dummy2} drc check map dummy2 2
...
```

4. For each intended output layer from the Calibre OPCverify command file, create an SVRF block. This block must contain Calibre OPCverify command file calls using the following template:

For each layer to be output to the design file:

```
OPCv_output_layer1=LITHO OPCVERIFY FILE "opcv_cmdfile" dummy1
[dummy2...] MAP output_layer1
OPCv_output_layer1 {COPY OPCv_output_layer1 } drc check map
OPCv_output_layer1 layer_number
```

For layer output to the RDB database, also add the following line:

```
OPCv_output_layer1_RDB {DFM RDB OPCv_output_layer1 "logfilename"
MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME OPCv_output_layer1 }
...
```

5. Save the file and continue to the task, “[Setting up direct\\_input in the OPCverify Command File](#)” on page 566.

## Results

This file is in progress and does not yet have any results.

# Setting up direct\_input in the OPCverify Command File

For processing purposes, the target file needs to be swapped with the dummy file.

---

### Tip

 Developing this file is separated into two sections. This task is section 2 of 2.

---

You use a [direct\\_input](#) block, which designates the OPCverify command file as a litho flat operation.

## Prerequisites

- Completed the task, “[Creating a Litho Flat SVRF Rule File Header and OPCverify Calls](#)” on page 564
- Understanding of the layer information of your target file

## Procedure

1. Either by creating a [LITHO FILE](#) block inside the SVRF file, or by creating a separate Calibre OPCverify setup command file, start by configuring your environment:
  - Set a modelpath to your optical and resist models.
  - Load required optical and resist models.
  - (Optional) Set a starting tilemicrons value (varying the tilemicrons setting may either improve or hamper performance, since it changes the tile size; use the log\_options parameter to see the performance of the rule file).
2. Enter the following Calibre OPCverify statements:
  - [direct\\_input](#) - Starts a direct input block.
  - [vboasis\\_path](#) - Part of the direct\_input syntax, needs to point to the location of the target input file.
  - One or more layer statements using the syntax:

```
layer layer_number [datatype]
```

The number of layers and layer\_numbers in the list must match the dummy layers you defined in the “[Finding Extents for Small Designs](#)” or “[Creating a Litho Flat](#)

“[SVRF Rule File Header and OPCverify Calls](#)” task, which also must correspond to relevant layer numbers in the target file.

- } - A closing brace for the block.
3. Beneath the direct\_input block, enter the background information.
  4. For each of the layers in the target design, add a layer statement with the corresponding visibility and transmission values for the layer using the following syntax:

```
layer layer_name visibility transmission_values
```

The layer\_names must correspond to the layers in the target design, and the order of layer statements must match the layer number order given in the direct\_input block. For example, in the code below, layer 246 is poly, and layer 51 is poly\_opc.

```
direct_input {
 vboasis_path ./target_input.oas
 layer 246 0
 layer 51
}
...
background clear
layer poly hidden -1.245000 0.000000
layer poly_opc visible -1.245000 0.000000
...
```

5. Define one or more setlayer rulechecks based on your needs. You must create as many setlayer statements as the LITHO OPCVERIFY statements you created in the “[Finding Extents for Small Designs](#)” or [Setting up direct\\_input in the OPCverify Command File](#) task.
6. Save the file.
7. You should now be able to run the SVRF rule file that contains the calls to the LITHO OPCVERIFY file.

```
calibre -drc -turbo litho.svrf
```

## Finding Extents for Small Designs

Once you have created the header for the litho flat SVRF file, you must add relevant dummy layers and create an output window for the HDB engine to use as a template. You will be replacing these later in the Calibre OPCverify command file.

---

### Tip

 This is an alternate to the tasks “[Finding Extents for Full Chip Runs](#)” on page 563 and “[Creating a Litho Flat SVRF Rule File Header and OPCverify Calls](#)” on page 564.

---

**Note**

This procedure is intended for layout window-type analysis. Larger design analysis (for designs greater than several GB in size) should use the MDP EMBED procedure starting with “[Finding Extents for Full Chip Runs](#)” on page 563 instead of this task.

---

## Prerequisites

- Understanding of the target design topology

## Procedure

1. In a text editor, open a new file, and enter the following SVRF commands as the header:

```
LAYOUT SYSTEM OASIS
LAYOUT PRIMARY "*"
LAYOUT PATH "./dummy_opcv.oas"
LAYOUT ULTRA FLEX YES
PRECISION precision_value
DRC MAXIMUM RESULTS ALL
DRC MAXIMUM VERTEX ALL
DRC SUMMARY REPORT "./reports/lithoflat.report"
DRC RESULTS DATABASE "./output/lithoflat.oas" OASIS PSUEDO
```

These commands perform the following functions:

- Sets up the input file (*dummy\_opcv.oas*).
- Sets the input and output precision. The precision of the dummy file must match the actual file.
- Enables Calibre ULTRA FLEX data processing. (See the [Calibre Post-Tapeout Flow User’s Guide](#) for more information on the LAYOUT ULTRA FLEX command. Calibre TURBO Flex is not currently supported.)

2. Add layers as follows:

- Designate an output window for the target design using the SVRF POLYGON statement to draw a quadrilateral on a dummy layer *in the dummy design file* (defining four coordinates, x1 x2 y1 y2). Only the *corresponding area in the target design covered by these coordinates* will be processed.

**Tip**

 Use the Calibre WORKbench GUI to inspect your target design for the best coordinates to use with the POLYGON statement.

---

- For each layer in the target design, add a corresponding dummy layer in the dummy design. The dummy layers can be empty.
- Dummy layer names can be changed based on your needs, but the name you use as a dummy layer cannot exist in the target design. (Since the POLYGON statement

creates an extent by drawing an actual polygon on the layer, the real layer in the target design will be overwritten.)

---

**Note**

---

 If a matching dummy layer does not exist in *dummy\_opcv.oas*, then the POLYGON statement needs to be used to define the extent. Otherwise the dummy layers must exist with the correct extent in *dummy\_opcv.oas*.

---

```
POLYGON -17.5595 -17.5595 21887.5595 21608.5595 dummy1
```

3. For each layer of interest in the actual input file, create a dummy layer declaration:

```
layer dummy1 1
layer dummy2 2
...
```

4. For each layer you declared, add **copy** statements to copy the dummy layer (which will be replaced by an actual target layer) to the output:

```
dummy1 {copy dummy1} drc check map dummy1 1
dummy2 {copy dummy2} drc check map dummy2 2
...
```

5. Create an SVRF block that contains Calibre OPCVerify command file calls using the following template:

For each layer to be output to the design file:

```
OPCv_output_layer1=LITHO OPCVERIFY FILE "opcv_cmdfile" dummy1
[dummy2...] MAP output_layer1

OPCv_output_layer1 {COPY OPCv_output_layer1 } drc check map
OPCv_output_layer1 layer_number
```

For layer output to the RDB database, also add the following line:

```
OPCv_output_layer1_RDB {DFM RDB OPCv_output_layer1 "logfilename"
MAXIMUM ALL ALL CELLS NOEMPTY CHECKNAME OPCv_output_layer1 }
...
```

6. Save the file and continue to the task, “[Setting up direct\\_input in the OPCVerify Command File](#)” on page 566.

## Lithoflat Sample Code

The following sections contain code showing a full sample application.

## **litho.svrf**

This code is the result of performing the procedure “[Creating a Litho Flat SVRF Rule File Header and OPCverify Calls](#)” on page 564.

```
LAYOUT SYSTEM OASIS
LAYOUT PRIMARY "*"
LAYOUT PATH "./dummy.oas"
//LAYOUT ULTRA FLEX YES
LAYOUT TURBO FLEX YES
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 0
PRECISION 2000
DRC MAXIMUM RESULTS ALL
DRC MAXIMUM VERTEX ALL
DRC RESULTS DATABASE result.oas OASIS PSEUDO
DRC SUMMARY REPORT result.rep
DRC RESULTS DATABASE PRECISION 2000
// a "full chip" flow would already have extent data included before this
// part.
// a "small clip" flow would have a defined POLYGON statement:
// POLYGON -17.5595 -17.5595 21887.5595 21608.5595 dummy1
layer dummy1 1
layer dummy2 2
post_opc= LITHO OPCVERIFY dummy1 dummy2 FILE "opcv_esvrf.in" MAP post_opc
target = LITHO OPCVERIFY dummy1 dummy2 FILE "opcv_esvrf.in" MAP target
poly_ctr = LITHO OPCVERIFY dummy1 dummy2 FILE "opcv_esvrf.in" MAP poly_ctr
short= LITHO OPCVERIFY dummy1 dummy2 FILE "opcv_esvrf.in" MAP short
open = LITHO OPCVERIFY dummy1 dummy2 FILE "opcv_esvrf.in" MAP open
bad_epe = LITHO OPCVERIFY dummy1 dummy2 FILE "opcv_esvrf.in" MAP bad_epe
post_opc { COPY post_opc } DRC CHECK MAP post_opc 1
target { COPY target } DRC CHECK MAP target 2
poly_ctr { COPY poly_ctr } DRC CHECK MAP poly_ctr 11
short{COPY short} DRC CHECK MAP short 501
short_rdb{ DFM RDB short "ascii/out.ascii" MAXIMUM ALL ALL CELLS
NOEMPTY CHECKNAME short}
open { COPY open }DRC CHECK MAP open 502
open_rdb{ DFM RDB short "ascii/out.ascii" MAXIMUM ALL ALL
CELLS NOEMPTY CHECKNAME open }
bad_epe { COPY bad_epe} DRC CHECK MAP bad_epe 504
bad_epe_rdb { DFM RDB bad_epe "ascii/out.ascii" MAXIMUM ALL ALL CELLS NOEMPTY
CHECKNAME bad_epe }
```

## **opcv\_esvrf.in**

This is a sample OPCverify command file containing direct input commands, and is the result of performing the task “[Setting up direct\\_input in the OPCverify Command File](#)” on page 566.

```

modelpath $env(DESIGN_DIR)/models
optical_model_load f0 optical_full_chip
resist_model_load r0 CM1_mf130.mod
processing_mode flat
direct_input {
 vboasis_path ./input_design.oas
 layer 246 0
 layer 51
}
background clear
layer poly hidden -1.245000 0.000000
layer poly_opc visible -1.245000 0.000000
tilemicrons 69.4 exact
setlayer post_opc = copy poly_opc
setlayer target = copy poly
setlayer poly_ctr = image optical f0 dose 1 resist_model r0
setlayer short = bridge poly poly_ctr <= 0.04 separation 0.2 \
 max_tolerance 0.08 max_edge 0.05 output_expand 0.06 property {min} \
 classify { context poly \
 halo 0.2 \
 anchor poly \
 anchor_max_snap 0.03 \
 coarse_match 0.02 \
 reflections_rotations_match yes \
 score bin_size 0.002 property min smallest \
 worst 4000 \
 }
setlayer open= pinch poly poly_ctr < 0.04 separation 0.2 max_tolerance \
 0.05 max_edge 0.05 output_expand 0.005 property {min}
classify {context poly
 halo 0.2 \
 anchor poly \
 anchor_max_snap 0.030 \
 coarse_match 0.02 \
 reflections_rotations_match yes \
 score bin_size 0.002 property min smallest \
 worst 4000 \
}
setlayer lineend = identify_edge poly length <=0.15 length1 >= 0.150 \
 length2 >= 0.150 corner1 convex corner2 convex expand 0.01
setlayer bad_epe = measure_epe poly_ctr poly inside lineend only \
 not >= -0.005 <= 0.005 epe_spacing 0.005 function max min_featsize 0.05
max_edgelen 0.2 output_expanded_edges 0.005 property {max} limit {
 score smallest
 worst 4000
}
output_window post_opc halo 0.3 around short open bad_epe
output_window target halo 0.3 around short open bad_epe
output_window poly_ctr halo 0.3 around short open bad_epe
log_options { tile_efficiency on }

```



# Index

---

## — Symbols —

[]<sup>15</sup>  
{}<sup>15</sup>  
|<sup>15</sup>

## — A —

anchoring mode, [75](#)  
and command, [260](#)  
area\_compute verification command, [267](#)  
area\_overlay verification command, [271](#)  
area\_ratio verification command, [276](#)  
Attenuated masks  
    background, [136](#)  
    transmission values, [190](#)

## — B —

background setup command, [17, 136](#)  
Bold words, [14](#)  
bridge verification command, [287](#)  
bridge\_tolerance verification command, [18, 27, 296](#)  
build\_tolerance verification command, [18, 27, 32, 298](#)

## — C —

Cells, [483](#)  
center\_shift command, [300](#)  
clone\_transformed\_cells setup command, [143](#)  
Command sequencing, [17](#)  
Concurrency  
    in Calibre, [52](#)  
    setlayer operations, [53](#)  
Configuration  
    background command, [17](#)  
    clone\_transformed\_cells command, [143](#)  
    contour\_options command, [150](#)  
    ddm\_model\_load command, [154](#)  
    dynamic\_output command, [159](#)  
    etch\_model\_load command, [163](#)  
    filter command, [166](#)

flare\_model\_load command, [170](#)  
imagegrid command, [185](#)  
layer command, [17, 188](#)  
log\_options command, [196](#)  
modelpath command, [199](#)  
optical\_model\_load command, [200](#)  
optical\_transform\_size command, [202](#)  
progress\_meter command, [209](#)  
promote\_subframe\_slivers command, [210](#)  
resist\_model\_load command, [217](#)  
simulation\_deangle command, [228](#)  
stairstep command, [229](#)  
summary\_report command, [231](#)  
svrf\_var\_import command, [249](#)

Constraints, [67](#)

contour\_diff verification command, [305](#)  
contour\_options setup command, [150](#)  
copy command, [309](#)  
cornerchop verification command, [18, 28, 33, 310](#)

Courier font, [14](#)

## — D —

Dark  
    background, [137](#)  
    features, [189](#)  
DDM argument to image command, [370](#)  
ddm\_model\_load setup command, [154](#)  
Debug, [483](#)  
direct\_input command, [155](#)  
dofcheck verification command, [316](#)  
Dose, [191](#)  
Double pipes, [15](#)  
DRC-like commands  
    and, [260](#)  
    copy, [309](#)  
    enclosure, [322](#)  
    external, [330](#)  
    internal, [35, 402](#)  
    list of commands, [253](#)

- 
- not, 447
  - or, 452
  - size, 475
  - width, 486
  - xor, 488
  - dynamic\_output setup command, 159
- E —**
- empty\_layer verification command, 321
  - enclosure command, 322
  - end\_cap verification command, 325
  - etch\_model\_load setup command, 163
  - Examples
    - bridging check application, 26
    - converting a file to TVF, 545
    - gate CD check application, 35
    - generating multiple output layers, 52
    - generating PVband application, 36
    - Lithoflat sample code, 569
    - metall layer checks, 536
    - misalignment check application, 38
    - multiple dose calculations, 54
    - pinching check application, 32
    - SVRF file, 20
  - external command, 330
  - extra\_printing verification command, 333
- F —**
- filter setup command, 166
  - filter\_generate verification command, 336
  - flare\_longrange command, 168
  - flare\_model\_load setup command, 170
  - Frequently-asked questions, 64
- G —**
- gate\_stats verification command, 341
  - gauge\_set command, 171
  - gauges verification command, 352
- H —**
- Heavy font, 14
  - Histogram blocks, 87
  - holes verification command, 362
- I —**
- identify\_corner verification command, 364
  - identify\_edge verification command, 366
- image command, 369
  - image\_options command, 174
  - imagegrid setup command, 185
  - imax\_check command, 381, 384
  - Input layers, 46
  - intensity\_meefcheck verification command, 390
  - interact verification command, 399
  - internal command, 35, 402
  - Italic font, 14
- L —**
- layer setup command, 17, 46, 188
  - LAYER SVRF command, 50
  - layer\_properties, 193
  - Layers
    - and LAYER SVRF command, 50
    - defining, 46
    - input, 46
    - names, 189
    - OPCverify, 48
    - output, 48
    - types, 189
  - Litho model command, 195
  - Litho model format, 111
  - LITHO OPCVERIFY command, 106
  - Litho-flat rule files, 561
  - log\_options setup command, 196
- M —**
- Mapping
    - design layers, 50
    - OPCverify layers to SVRF output layers, 51
  - Masks, background illumination, 136
  - measure\_cd verification command, 405
  - measure\_cdv verification command, 415
  - measure\_cross\_section verification command, 419
  - measure\_distance verification command, 423
  - measure\_epe verification command, 432
  - MEEF, 37, 354, 439
  - meefcheck verification command, 439
  - Minimum keyword, 15
  - modelpath setup command, 46, 199
  - Multiple masks

---

defining background, 136  
defining mask number and dose, 190

— N —

nilscheck verification command, 443  
not command, 447  
not\_printing verification command, 448

— O —

OPCverify  
chaining commands together, 17  
concurrency in Calibre, 52  
frequently asked questions, 64  
input layers, 46  
setup file, 44  
SVRF file example, 20  
usage model, 43  
using Tcl, 62

OPCverify commands  
image, 369

OPCverify examples  
bridging check, 26  
gate CD check, 35  
generating PVband, 36  
misalignment check, 38  
pinching check, 32

OPCverify layers  
mapping to SVRF output, 51  
OPCverify LITHO calls, defining, 50  
OPCverify mapping, design layers, 50  
Optical models, selecting, 45  
optical\_model\_load setup command, 45, 200  
optical\_transform\_size setup command, 202  
or command, 452  
Output layers, 48  
output\_window verification command, 203

— P —

Parentheses, 15  
Phase Shifting Transmissions, 190  
pinch verification command, 453  
pinch\_tolerance verification command, 32, 461  
Pipes, 15  
polygon\_extent verification command, 463  
processing\_mode command, 208  
progress\_meter setup command, 209

promote\_subframe\_slivers setup command, 210

punch\_tolerance verification command, 461  
pvband verification command, 37, 464  
pw\_annotate command, 212  
pwcheck command, 465

— Q —

Quotation marks, 15

— R —

Regions, 483  
Resist models, selecting, 46  
resist\_model\_load setup command, 46, 217  
RET INPUT command, 126

— S —

save\_error\_center points command, 218

Select

optical models, 45  
resist models, 46  
Setlayer commands, 483  
Setup file commands  
background, 136  
layer, 188  
litho\_model, 195  
shift verification command, 473  
show\_annotation verification command, 474  
simulation\_deangle setup command, 228  
size command, 475  
Slanted words, 14  
Square parentheses, 15  
stairstep setup command, 229  
summary\_report command, 231  
SVRF rule file, creating, 50  
svrf\_var\_import setup command, 249

— T —

Tcl and OPCverify, 62  
tilemicrons command, 250  
Tiles, 483  
Topographical models, 56  
Transmission Value  
related to background, 137  
related to layer, 189

---

**— U —**

Underlined words, [14](#)

**— V —**

Verification commands

area\_compute, [267](#)  
area\_overlay, [271](#)  
area\_ratio, [276](#)  
background, [136](#)  
bridge, [287](#)  
bridge\_tolerance, [18, 27, 296](#)  
build\_tolerance, [18, 27, 32, 298](#)  
center\_shift, [300](#)  
contour\_diff, [305](#)  
cornerchop, [18, 28, 33, 310](#)  
dofcheck, [316](#)  
empty\_layer, [321](#)  
end\_cap, [325](#)  
extra\_printing, [333](#)  
filter\_generate, [336](#)  
gate\_stats, [341](#)  
gauges, [352](#)  
holes, [362](#)  
identify\_corner, [364](#)  
identify\_edge, [366](#)  
interact, [399](#)  
list of commands, [254](#)  
measure\_cd, [405](#)  
measure\_cdv, [415](#)  
measure\_cross\_section, [419](#)  
measure\_distance, [423](#)  
measure\_epe, [432](#)  
meefcheck, [439](#)  
nilscheck, [443](#)  
not\_printing, [448](#)  
output\_window, [203](#)  
pinch, [453](#)  
pinch\_tolerance, [32, 461](#)  
polygon\_extent, [463](#)  
pvband, [37, 464](#)  
shift, [473](#)  
show\_annotation, [474](#)

**— X —**

xor command, [488](#)

**— W —**

width command, [486](#)

## **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.

