

SIEMENS EDA

Calibre® Litho-Friendly Design User's Manual

Software Version 2021.2
Document Revision 14

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Unpublished work.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

Revision History

Revision	Changes	Status/ Date
14	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released April 2021
13	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released January 2021
12	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released October 2020
11	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released July 2020

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Siemens EDA documentation source. For specific topic authors, contact the Siemens Digital Industries Software documentation department.

Revision History: Released documents maintain a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation which are available on <https://support.sw.siemens.com/>.

Table of Contents

Revision History

Chapter 1		
Introduction to Calibre LFD		19
Calibre LFD Overview		19
Calibre LFD Workflow		20
Calibre LFD Requirements		20
Modes of Operation		22
Invoking Calibre LFD From Calibre Interactive		22
Command Line Examples to Invoke Calibre LFD		23
calibre -lfd		25
calibre -lfd -mbh		27
Advanced Vector Extensions (AVX) and Calibre LFD		27
Encrypted Rule Files		28
Syntax Conventions		29
Chapter 2		
Calibre LFD Basic Concepts		31
Sources and Impact of Process Variability		31
How Calibre LFD Works		34
Process Variation		34
Calibre LFD Checks		37
Calibre LFD Scores and Other Properties		38
Designing “Litho-Friendly”		39
Comparing Calibre LFD to DRC		42
Evolving with Your Process		43
Sharing Information Across the Flow		43
Chapter 3		
For Lithographers — Developing Calibre LFD Rules		45
Requirements for Writing Calibre LFD Rules		45
Background Information for Writing Calibre LFD Rules		47
What are PV-Bands?		47
Types of PV-Bands		49
The Overall Calibre LFD Flow		51
How PV-Bands Predict Variability		52
Generating Process-Specific Calibre LFD Kits		54
Defining Process Variation Experiments		55
Creating Optical Models		57
Visualizing PV-Band Data Using Calibre RVE		58
Using PV-Bands to Identify Failure Spots		60
Using Built-in Check Functions		61

Designing Custom Checks	62
Metrics to Score a Process or Design	63
Understanding Standard Metrics	63
Designing Custom Metrics	64
Ranking Checks and Errors	66
Built-in Ranking	66
Custom Ranking	67
Error Marker Properties for Ranking	67
Chapter 4	
For CAD Groups — Writing Calibre LFD Rule Files	69
What is a Process-Specific Calibre LFD Kit?	69
Writing Rules for Performing Calibre LFD	70
Writing Rules Using TVF	72
SVRF Pre-Processor Directives Compared to TVF Variables and Tcl Constructs	72
Avoiding Name Collisions	73
Avoiding Namespace Issues	73
Calibre LFD Function Calls	74
Accessing Variables in TVF Rule Files for Calibre LFD	74
Debugging with TVF for the Calibre LFD Rule File	74
Accessing the Calibre LFD Library	75
Defining the Calibre LFD Block	75
Calibre LFD Fast Mode	77
Calibre LFD and Pattern Matching	77
Functions Limiting Calibre LFD Cell Checks	78
Limiting Calibre LFD Simulation and Checks to Specific Regions	79
Customizable PV-Bands	81
Creating and Using Contours with LFD::Contour	81
Creating and Using PV-Bands with LFD::Band	82
Creating and Using Overlay Bands with LFD::OverlayBand	83
Calibre LFD Check Options for Customizable PV-Bands and Contours	83
Customizable PV-Bands Using a PDK Flow	84
Generating PV-Bands	86
Generating PV-Bands Without a PDK	86
Generating PV-Bands by Reusing a PV-Band	86
Generating PV-Bands with a PDK	87
PV-Band and Contour Processing Commands	87
Writing the Rule File for Generating PV-Bands	88
Requirements for the PVband Command	88
Setting Up the Rule File	88
Specifying Correct Layer Data to Generate PV-Bands	90
Describing Process Variation Experiments	91
Translating Failures into Rules	94
Built-in Check Functions	94
Writing Custom Checks	95
Coding Metrics to Score a Process or Design	97
Using Standard Metrics	97
Custom Metrics	98

Table of Contents

Accessing PV-Band Data	99
Accessing Contour Data	100
About Calibre LFD Databases.....	102
The Indexing Database	102
The Check Database	102
The Bands Database	103
Saving Calibre LFD Data to a Layout Database	103
Calibre LFD Flow with a PDK	105
About the PDK	106
About the PDK Specification	107
PDK File Commands.....	107
Writing TVF for use with a PDK.....	108
SVRF Rule File Macros.....	111
Macro Definition.....	111
DMACRO Arguments	112
LOCAL() Modifier	113

Chapter 5

For Designers — Incorporating Calibre LFD Into Your Flow 115

Calibre LFD Database Types.....	116
Indexing Database.....	116
Check Database.....	117
Bands Database.....	118
Running Calibre LFD Checks	119
Reviewing Calibre LFD Violations.....	121
Viewing Calibre LFD Check Results	121
Viewing Design Variability Index (DVI) Data.....	123
Making Sense of Variability Data	124
Viewing PV-Band Data	124
HTML Reporting for Calibre LFD Results	125
About Model-Based Hints (MBH)	127
MBH Operation and Flow	127
Running Calibre LFD with MBH.....	128
Viewing MBH Results	129
MBH Rule, Map, and Transcript Files	132
Sample MBH TVF Rule File	132
Sample MBH Map File.....	135
MBH Map File Format	139
MBH Transcript File Information.....	149
Modifying the Design	150
Re-Running Calibre LFD Checks on a Modified Design	151
rdb_flattener.....	154

Chapter 6

For Designers — Incorporating Calibre LFD Into Your Timing Simulations 157

About Model Based Extraction	157
The Importance of MBE	158
MBE and Your Timing Analysis Flow	159

About the CSG Function	161
CSG Data and SPICE Model Formulas	161
Files Generated by the CSG Function	162
About the CSI Function	163
Performing Model Based Extraction	166
Generating CSI Layers and CSG Files	167
Sample MBE Calibre LFD Rule File	169
Chapter 7	
Calibre LFD Syntax Descriptions	175
Calibre LFD Command Summary Tables	175
Calibre LFD Check Properties	179
Alternate Syntax Options for PDK Usage	179
Calibre LFD Checks	181
AddCustomCheck	183
AddCustomOPCVCheck	186
AreaCheck	194
AreaOverlayCheck	202
EndCapCheck	210
InterLayerSpaceCheck	217
LineEndCheck	227
MaxAreaVariabilityCheck	235
MaxCDVariabilityCheck	242
MinAreaOverlapCheck	250
MinOverlayCheck	259
MinSpaceCheck	269
MinWidthCheck	280
NonPrintingCheck	289
PrintableCheck	294
ViaWidthCheck	298
Calibre LFD General-Purpose Commands	304
Anchor	307
AnchorRule	314
Band	320
Begin	322
CaptureContour	323
ClassifyConfig	328
Contour	334
CSG	354
CSI	364
Drawn2Contour	367
End	371
FrameCellOptimizer	372
GenerateHints	374
GetMacroLayers	380
GetOPCInputLayers	382
GetOPCLayers	383
GetRetargetLayers	385

Table of Contents

GetVisibleLayers	386
IncrementalSelect	387
LayoutOptimizer	390
LFDregion	392
LoadPDK	394
Macro	395
MLDataGen	397
MLOptimizer	401
OutputBands	404
OverlayBand	408
PrintableBand	412
ProcessCorrection	414
PVband	415
ReadECO	428
ReadHIF	429
ReadiLPC	431
ReadRDB	433
RegisterBands	435
RegisterContour	437
RegisterLayer	439
RemoveErrors	441
SetSTOMode	449
SimConfig	451
StructureOptimizer	453
StructureOutput	466
TopCellOptimizer	469
VariabilityIndices	471
WriteHIF	474
WriteICC	476
PDK Commands	477
addPDKCheck	478
addPDKLayer	480
addPDKSTO	491
createPDK	493
System-Generated Names, Priorities, and Scores	495

Chapter 8

Encrypted Calibre LFD Flow	499
For the Litho Team	500
Creating Binary Optical Models	500
Embedding the Resist Model in the Setup File	501
For the CAD Group	503
Embedding the Setup File in the Rule File	503
Embedding the Resist Models in the Calibre LFD Rule File	505
Encrypting the Calibre LFD Rules	510
Encrypting the Calibre LFD Rule File Without a PDK	510
Encrypting LFD Commands Without a PDK	510
Encrypting Calibre LFD Rules with a PDK	511

Encryption Debugging	512
Packaging the Calibre LFD Kit	512
Appendix A	
Running Calibre LFD with Calibre Interactive.....	515
Basic Calibre LFD Flow with Calibre Interactive.....	515
Creating and Using a Calibre Interactive Runset for Calibre LFD	517
Appendix B	
Sample Calibre LFD Rule File and PDK	519
Sample Rule File	519
Sample PDK	526
Appendix C	
Guidelines for Calibre LFD.....	531
Specifications Required From the Foundry for Calibre LFD	531
Constraints on RET Recipe Decks Used for Calibre LFD	531
Basic Calibre LFD Guidelines.....	532
Appendix D	
Defining Highlight Colors to Match Histograms.....	535
Matching Histogram Colors to Highlight Colors	536
Calibre DESIGNrev, Calibre WORKbench, and Some Other Viewers.....	536
Cadence Virtuoso	536
Defining Colors for Calibre DESIGNrev and Calibre WORKbench	536
Appendix E	
Calibre LFD Error Messages.....	539
Common Errors to All Checks.....	539
Check and Command Specific Errors	539
Appendix F	
Using Calibre LFD in the Cadence Environment	547
Data Flow Between Calibre LFD and the Cadence Environment.....	547
Running Calibre LFD After Cadence Routing	548
Passing Calibre LFD Results to Cadence Tools	549
Calibre LFD errmap and Command Reference for Cadence Tools.....	551
errmap File Format	552
rdb2hif Command	553
Appendix G	
Using Calibre LFD in the Synopsys Environment.....	555
Data Flow Between Calibre LFD and the Synopsys Environment	555
Running Lithographic Hotspot Corrections in the Synopsys Environment	556

Table of Contents

Appendix H		
Drawn-To-Contour Flow (D2C)		559
D2C Functionality		559
dtcoptimize Command		560
dtcoptimize		561
Appendix I		
Optimizing Printability		569
Printability Commands		570
printableoptimize		571
Appendix J		
Using Pattern Matching Libraries		577
Pattern Library Generation Flow		577
Pattern Capture and Output		578
Command Line Invocations for Pattern Matching Flow		580
pdl_lib_mgr		581
How to Use LFD::RemoveErrors		582
Appendix K		
Calibre LFD Deep Neural Network (DNN) Flow.....		585
Calibre LFD DNN Machine Learning Model Preparation and Usage		586
Preprocessing and Data Generation		587
Model Training		589
lfd_dnn_train		590
Training Transcript Information.....		594
Prediction		597
Trials to Enhance Accuracy		598
Index		
Third-Party Information		

List of Figures

Figure 1-1. LFD Workflow	20
Figure 2-1. Increasing Variability in Delay as Process Nodes Shrink.....	32
Figure 2-2. Impact of Process Variation on Leakage	32
Figure 2-3. Performance Window	34
Figure 2-4. Process Window	35
Figure 2-5. How Process Variation Can Impact Printability.....	35
Figure 2-6. Single Feature and Associated PV-Band	36
Figure 2-7. Original Cell	36
Figure 2-8. OPC Corrected Cell	37
Figure 2-9. Calibre LFD Corrected Cell	37
Figure 2-10. PV-Band Representing Pinching	37
Figure 2-11. Viewing Calibre LFD Check Errors in the Layout Editor	38
Figure 2-12. The Calibre LFD Flow	40
Figure 2-13. Calibre LFD in Action	41
Figure 2-14. Calibre LFD in Your Design Flow	41
Figure 2-15. Mapping DRC Rules into Calibre LFD Checks	42
Figure 2-16. Calibre LFD Rules Evolve with Your Process	43
Figure 2-17. Sharing Information	44
Figure 3-1. PV-Bands Representing a Set of Simulated Print Images	48
Figure 3-2. Three Areas of a PV-Band Layer	49
Figure 3-3. Relating PV-Bands to Transistor CD Variation	49
Figure 3-4. Regular PV-Band	50
Figure 3-5. Absolute PV-Band	50
Figure 3-6. From Design to PV-Bands	51
Figure 3-7. Three Process Experiments Providing Full Coverage	53
Figure 3-8. Developing Process-Specific Calibre LFD Kits	54
Figure 3-9. Sampling Grid Points, not Entire Space	56
Figure 3-10. Planning Multiple Experiments	57
Figure 3-11. Failure Spots Identified by PV-Bands	61
Figure 3-12. Errors Ranked According to Criticality	66
Figure 4-1. Basic Flow for Running Calibre LFD with a PDK.....	106
Figure 5-1. Calibre LFD Loop.....	115
Figure 5-2. Viewing the Indexing Database	117
Figure 5-3. Viewing the Check Database	118
Figure 5-4. Viewing the Bands Database	119
Figure 5-5. The MBH Flow	128
Figure 5-6. MBH Hints	131
Figure 5-7. MBH Metal1 Hints	131
Figure 6-1. Typical Timing Analysis Flow	158
Figure 6-2. How Printing Affects Drawn Shapes	158

Figure 6-3. Calibre LFD to Calibre nmLVS Flow.....	159
Figure 6-4. CSG / CSI Inputs and Outputs	160
Figure 6-5. Understanding Ladj and Wadj	161
Figure 6-6. Contour Simplification	164
Figure 6-7. Using Simplified Contours for RC Extraction	164
Figure 6-8. Layers Generated by CSI, Shown over Associated Contours.....	165
Figure 6-9. Timing Analysis Flow Using MBE	166
Figure 6-10. Calibre LFD Rule File for Running CSI and CSG	167
Figure 7-1. AreaCheck.....	195
Figure 7-2. How AreaCheck Works	196
Figure 7-3. AreaOverlayCheck	203
Figure 7-4. EndCapCheck	211
Figure 7-5. Expected Output	211
Figure 7-6. InterLayerSpaceCheck	218
Figure 7-7. Checking for Inter-Layer Spacing Problems.....	219
Figure 7-8. How LineEndCheck Works	229
Figure 7-9. Checking PV-Band Variability.....	236
Figure 7-10. How MaxAreaVariabilityCheck Works	236
Figure 7-11. Checking CD Variability	243
Figure 7-12. Exceptions-Only Violation	244
Figure 7-13. Checking PV-Band Overlap	251
Figure 7-14. How MinAreaOverlapCheck Works.....	252
Figure 7-15. How MinOverlayCheck Works	261
Figure 7-16. Checking PV-Band Spacing	270
Figure 7-17. How MinSpaceCheck Works	271
Figure 7-18. Checking PV-Band Width	281
Figure 7-19. How MinWidthCheck Works	282
Figure 7-20. NonPrintingCheck.....	290
Figure 7-21. PrintableCheck	295
Figure 7-22. ViaWidthCheck.....	299
Figure 7-23. Anchors Generated	308
Figure 7-24. Anchors Generated Using width and space Values	308
Figure 7-25. Anchors Generated from Parallel Lines	309
Figure 7-26. Anchors Before and After Merge	309
Figure 7-27. LFD::Anchor Length LE.....	310
Figure 7-28. LFD::Anchor Space LE_LE	310
Figure 7-29. LFD::Anchor Space LE_E and E_E	311
Figure 7-30. LFD::AnchorRule Space	315
Figure 7-31. LFD::AnchorRule Length LE.....	315
Figure 7-32. LFD::AnchorRule Space LE_LE	316
Figure 7-33. LFD::AnchorRule Space LE_E and E_E	316
Figure 7-34. LFD::AnchorRule Widths.....	317
Figure 7-35. LFD::AnchorRule Stitch and Via Anchor Points	317
Figure 7-36. LFD::AnchorRule Parallel Lines Length and Space.....	318
Figure 7-37. Classification Duplicates.....	331

List of Figures

Figure 7-38. IncrementalSelect	388
Figure 7-39. Space Check Output Error Markers Without Calling RemoveErrors	442
Figure 7-40. Capturing Input Layer Polygons	442
Figure 7-41. Stored Pattern	443
Figure 7-42. Space Check Output After Waiving Error Marker	443
Figure 8-1. Encrypted Calibre LFD Flow	499
Figure A-1. Using Calibre Interactive with Calibre LFD	516
Figure F-1. Running Calibre LFD in the Cadence Environment	547
Figure G-1. Running Calibre LFD in the Synopsys Environment	556
Figure J-1. Pattern Library Creation	578
Figure J-2. RemoveErrors Flow	583
Figure K-1. Machine Learning Model Preparation and Usage	586
Figure K-2. Calibre LFD DNN Model Training	589

List of Tables

Table 1-1. Syntax Conventions	29
Table 2-1. Comparing Calibre LFD Checks to Design Rule Checks	42
Table 3-1. Allowed Expression Functions	65
Table 3-2. Default Ranking By Check	67
Table 4-1. Contents of a Typical Calibre LFD Kit	70
Table 4-2. Using Handles with Calibre LFD Checks	83
Table 4-3. Accessing PV-Band Data	95
Table 4-4. Expression Functions	98
Table 5-1. Manufacturability Based on Variability Data	124
Table 6-1. The Timing Analysis Flow Using MBE	166
Table 7-1. Calibre LFD Check Commands	175
Table 7-2. Calibre LFD General Purpose Commands	176
Table 7-3. Calibre LFD Commands for use with PDKs	178
Table 7-4. Set of Properties	179
Table 7-5. Calibre LFD Check Commands	181
Table 7-6. Calibre LFD General Purpose Commands	304
Table 7-7. CSG Variability Output Properties	355
Table 7-8. CSG Look-Up Table	356
Table 7-9. Transmission Argument Settings	418
Table 7-10. Model List Positions	421
Table 7-11. Calibre LFD Commands for use with PDKs	477
Table 7-12. Security Settings within a Layer Process Definition	483
Table 7-13. Security Settings	494
Table 7-14. System-Generated Check Names	495
Table 7-15. Default Priorities	496
Table 7-16. LFD Scoring	497
Table E-1. Error Messages Common to All Calibre LFD Check	539
Table E-2. Check-Specific Error Messages	540

Chapter 1

Introduction to Calibre LFD

Calibre® Litho-Friendly Design (Calibre LFD™) is introduced with an overview of the concepts and workflow, followed by the modes of operation, and the rules and syntax conventions related to using this product.

Calibre LFD Overview	19
Calibre LFD Workflow	20
Calibre LFD Requirements.....	20
Modes of Operation	22
Encrypted Rule Files	28
Syntax Conventions	29

Calibre LFD Overview

Calibre LFD is a pre-tapeout verification engine designed to help characterize sensitivity of designs to process variability.

Based on printed images simulated for a set of expected process conditions, the tool generates some or all of the following:

- Error markers identifying probable failure spots.
- Indices reporting on the design robustness across the process window.
- Simulated image contours designed as input to traditional verification such as LVS and extraction.

Why Use Calibre Litho-Friendly Design?

Calibre LFD gives designers access to information previously available only to lithographers working in the semiconductor fabrication facility. Adding LFD to your flow and combining DRC with LFD helps manage the complexity, prioritize corrections, and simplify the tasks of both the rule writer and the designer. As a result, you can produce designs measurably less sensitive to anticipated process variations.

Who Uses Calibre Litho-Friendly Design?

There are two types of LFD users:

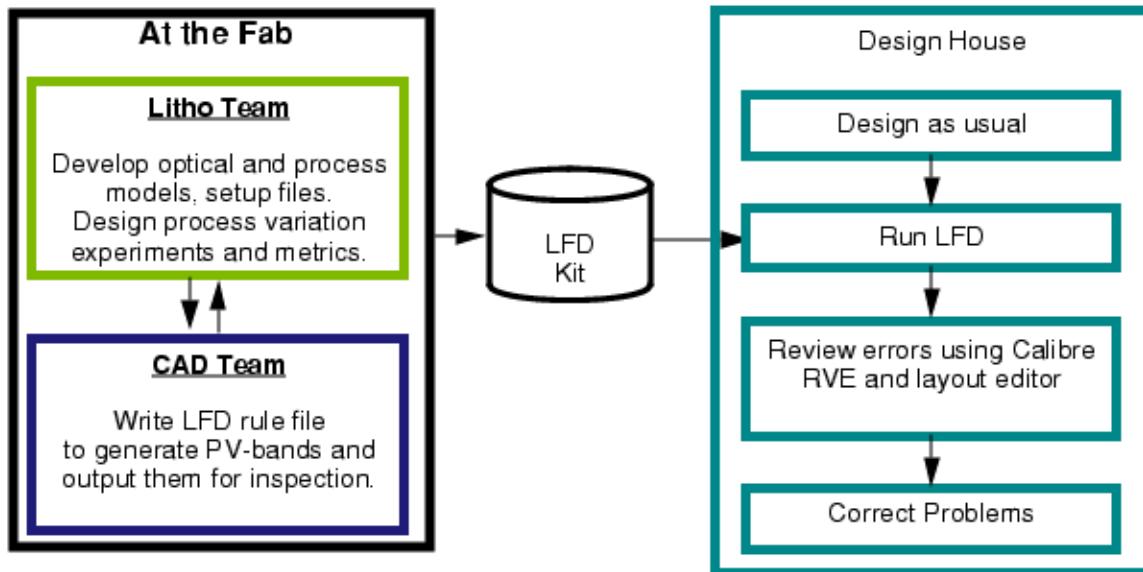
- The “end users” for Calibre LFD are the designers, who run LFD checks on their works-in-progress to identify and address problems while it is still relatively easy to correct them.
- Before that can happen, lithographers and rule writers work together to create the process-specific Calibre LFD design kit designers use for checking.

Calibre LFD Workflow

Lithographers and Computer Aided Designers (CAD) use Calibre LFD software to design process-specific Calibre LFD kits. Designers use Calibre LFD software with these LFD kits to check and improve their designs with respect to manufacturability. The process-specific kits serve as a bridge between lithographers and designers.

Figure 1-1 shows the basic workflow for Litho-Friendly Design.

Figure 1-1. LFD Workflow



Calibre LFD Requirements

Several requirements must be met in order to run Calibre LFD.

- A process-specific Calibre LFD kit, which can take the form of either a single file, called a Process Design Kit (PDK) or a set of individual files and directories. The Calibre LFD kit includes the following:
 - LFD rule file
 - Optical models
 - Resist models
 - OPC rule files
 - Additional rule files as needed for pre- and post-processing
- A design database in GDS or OASIS[®]¹ format.
- All necessary licenses. See “Calibre LFD” in the [Calibre Administrator’s Guide](#).
- Set the CALIBRE_HOME or MGC_HOME environment variable to the path of the Calibre software tree. Refer to the [Calibre Administrator’s Guide](#) for additional information.

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.

Modes of Operation

Litho-Friendly Design software is executed by the Calibre® hierarchical engine. You can invoke run Calibre LFD from Calibre Interactive DFM or from the command line.

- Calibre® Interactive™ — When checking a portion of a design cell, checking design modifications as you go, or to take advantage of the ease-of-use provided by the graphical user interface (GUI).
- Linux®² command line shell — When processing all or part of a design cell.

Invoking Calibre LFD From Calibre Interactive.....	22
Command Line Examples to Invoke Calibre LFD.....	23
calibre -lfd.....	25
calibre -lfd -mbh.....	27
Advanced Vector Extensions (AVX) and Calibre LFD	27

Invoking Calibre LFD From Calibre Interactive

When you run Calibre LFD from Calibre Interactive you can export the design from a connected layout design tool. You can also use the GUI to set run options and override settings in the rule file.

Prerequisites

- “[Calibre LFD Requirements](#)” on page 20
- A Calibre Interactive license.
- (Optional) A design open in a supported layout viewer.

Procedure

1. Invoke Calibre Interactive from a design tool or from the command line:
 - From a design tool, do one of the following to invoke Calibre Interactive:
 - From Calibre DESIGNrev or Calibre WORKbench, choose **Verification > Run DFM**.
 - From most other design tools, select **Calibre > Run DFM**.
 - From the command line:

```
% calibre -gui -dfm
```
2. (Optional) Select **File > Load Runset** to load a Calibre Interactive runset if you have one with the desired settings for your run.
2. Linux® is a registered trademark of Linus Torvalds in the U.S. and other countries.

3. Click the **Inputs** button on the left panel and do the following:
 - a. Select “Litho Friendly Design (LFD)” in the Run dropdown menu.
 - b. Click the **LFD** tab.
4. Click the buttons on the left panel to make required settings on the Rules, Inputs, Outputs, and Run Control panes.

Tip

The button text turns green when all necessary information is provided.

5. (Optional) Save a runset with **File > Save Runset**.
6. Click **Run LFD** to start the run.

Related Topics

[Running Calibre LFD Checks](#)

[Creating and Using a Calibre Interactive Runset for Calibre LFD](#)

[Calibre Interactive User’s Manual](#)

Command Line Examples to Invoke Calibre LFD

Command line options specify the processing mode.

Single CPU

```
calibre -lfd -hier ./myrulefile
```

Multi-CPU

For Calibre MT multi-threading, you specify the options `-turbo` and `-turbo_litho`.

```
calibre -lfd -hier -turbo -turbo_litho ./myrulefile
```

Multi-CPU with MTflex

Calibre LFD products follow the standard methodology for running in MTflex mode, which varies depending on whether running on one platform or many. For Calibre MTFlex distributed multi-threading, you specify any of the MT options, in addition to the options `-remote` and `-remotefile`.

The following invokes a homogeneous run in which remotes are on the same platform as the primary host:

```
calibre -lfd -hier -turbo -turbo_litho -remote m1,m2,m8 ./myrulefile
```

The following invokes a heterogeneous run in which remotes are on different platforms than the primary host:

```
calibre -lfd -hier -turbo -turbo_litho -remotefile ./mtflex.cfg ./rules
```

Related Topics

[Running Calibre LFD with Calibre Interactive](#)

[calibre -lfd](#)

calibre -lfd

You can run Calibre LFD from the command line.

Usage

calibre -lfd -hier [-turbo [n]] [-turbo_litho [N]] [-turbo_all] [-tvf_instrument] rules

Parameters

- **-lfd**
Required switch specifying LFD analysis.
- **-hier**
Required switch specifying hierarchical checking.
- **-turbo n**
Optional argument that selects multi-threaded parallel processing. The value of *n* specifies the number of CPUs to use for non-RET processes. If you do not specify *n*, the application uses the maximum number of CPUs on the machine. For additional information on multithreaded processing, refer to the *Calibre Verification User's Manual*.
- **-turbo_litho N**
Optional argument that selects multithreaded parallel processing for operations invoked by any of the RET operations, including the simulations performed to generate the PV-bands. The value of *N* specifies the number of CPUs to use for these processes. If you do not specify *N*, the application uses the maximum number of CPUs on the machine. For additional information on multithreaded processing, refer to the *Calibre Verification User's Manual*.

Note



The number specified for -turbo_litho does not need to agree with the number specified for -turbo.

- **-turbo_all**
Optional argument used in conjunction with -turbo or -turbo_litho. When the number of licenses specified in -turbo and -turbo_litho are not available, the Calibre hierarchical engine normally runs with fewer threads than requested. This command line option causes the hierarchical engine to exit if the tool cannot get the number of licenses specified.
- **-tvf_instrument**
Optional argument enabling compile-time Tcl Verification Format (TVF) syntax checking. When syntax checking is used, errors are reported with the file name and line number. If an error occurs in Standard Verification Rule Format (SVRF) code generated out of compile-time TVF, the original TVF location which produced the offending SVRF statement is reported.

- ***rules***

Required argument specifying the name of the rule file containing the Calibre LFD commands that perform Calibre LFD analysis. It should be either a TVF, SVRF, TVF encrypted, or SVRF encrypted rule file.

calibre -lfd -mbh

The **-mbh** switch runs Calibre LFD with model-based hints (MBH) functionality. This optional Calibre LFD feature generates a hints file for fixing lithographic hotspots. The designer can use the hint information as a repair reference.

Usage

calibre -lfd -mbh [-hier] *mbh_tvf_file_name*

Parameters

- **-lfd**
Required switch to run LFD analysis.
- **-mbh**
Required switch to run MBH analysis.
- **-hier**
Optional switch specifying hierarchical processing. Used when layer evaluation for MBH requires hierarchical processing rather than flat processing.
- ***mbh_tvf_file_name***
Required argument specifying the MBH TVF rule file that is used when running Calibre LFD with MBH generation. This TVF format file contains layer information, parameters, and arguments used by the function LFD::[GenerateHints](#).

Related Topics

[About Model-Based Hints \(MBH\)](#)

[Running Calibre LFD with MBH](#)

[Sample MBH TVF Rule File](#)

[GenerateHints](#)

Advanced Vector Extensions (AVX) and Calibre LFD

Calibre LFD supports the use of CPUs with AVX x86 instruction set architecture, which can improve the runtime performance of Calibre LFD RET applications.

Considerations for AVX

AVX is only used for AOI and IXL platforms and is recommended only for runs with same platform type (homogeneous) remote hosts and AVX compatible processors. The run results may vary slightly between AVX and non-AVX runs.

Note

-  Calibre nmSRAF gradient calculations are sensitive to the differences between AVX and non-AVX results. It is imperative for SRAFs that recipe development and production runs are done with identical AVX settings.
-

Environment Setting for AVX

AVX is not enabled by default. You must set an environment variable to enable AVX before running Calibre LFD. For example, in csh shell format, you can set (1/ON) the boolean AVX environment variable as follows:

```
% setenv CALIBRE_ENABLE_AVX ON
```

To unset the AVX environment variable, specify the following:

```
% unsetenv CALIBRE_ENABLE_AVX
```

See the *Calibre Post-Tapeout Flow User's Manual* for more information about using AVX and setting the CALIBRE_ENABLE_AVX environment variable.

Encrypted Rule Files

Encryption allows the foundry to give you access to sensitive data without disclosing any proprietary information. This is true for both standard design kits and process-specific Calibre LFD design kits. Running Calibre LFD with an encrypted rule file does not require any special licenses or commands. Encrypted rule files begin with header text containing a legal disclaimer and a reference to the version of Calibre used to verify the rule file. Below the header, you see both encrypted and unencrypted code. Encrypted code begins with #DECRYPT and ends with #ENDCRYPT.

```
LAYOUT SYSTEM GDSII
LAYOUT PRIMARY "TOP"
LAYOUT PATH "mydesign.gds"

PRECISION 1000
RESOLUTION 5
LAYOUT ERROR ON INPUT NO

FLAG SKEW YES
FLAG ACUTE YES
FLAG OFFGRID YES

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE result.gds GDSII PSEUDO
DRC SUMMARY REPORT result.rep
DRC MAXIMUM VERTEX 199

LAYER POLY 2

#DECRYPT "'?~;UP\YPN5^LX@@"TBBQ'`->Z7=A^(:W; (7;K) 0&HK6B-
!#'7J^&B [^QI.(EJMT219L/1!!
><I:;95RAN[!V#"9R%DIUQ!!"05)X$SNT&E13@&H2'EK/#Q!!"
?:IH%;1S<D()X\4M(59R`1!!"P,6H8)M'.\JXW&MR?#>'2$H<X_U5M"NA]TY$ZOQ/QC1!#
#ENDCRYPT
```

When customizing the Calibre LFD rule file with the layout name, layers, and so on, edit only unencrypted lines of code.

Syntax Conventions

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

Table 1-1 identifies the syntax conventions used for Calibre LFD.

Table 1-1. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	Underlining indicates either the default argument or the default value of an argument.
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.

Table 1-1. 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]
```

Chapter 2

Calibre LFD Basic Concepts

Calibre LFD is a pre-tapeout verification engine designed to help design teams cope with the increasing sensitivity of designs to process variability. Several key concepts are important to understand when adding Calibre LFD to your design flow.

Sources and Impact of Process Variability	31
How Calibre LFD Works	34
Designing “Litho-Friendly”	39
Comparing Calibre LFD to DRC.....	42
Evolving with Your Process	43
Sharing Information Across the Flow	43

Sources and Impact of Process Variability

As process nodes shrink, the sensitivity to variability increases. Slight changes to dose, focus, or mask bias lead to fluctuations in properties such as delay and leakage.

Dose changes are due to variations across the wafer in the intensity of the light used in the manufacturing process.

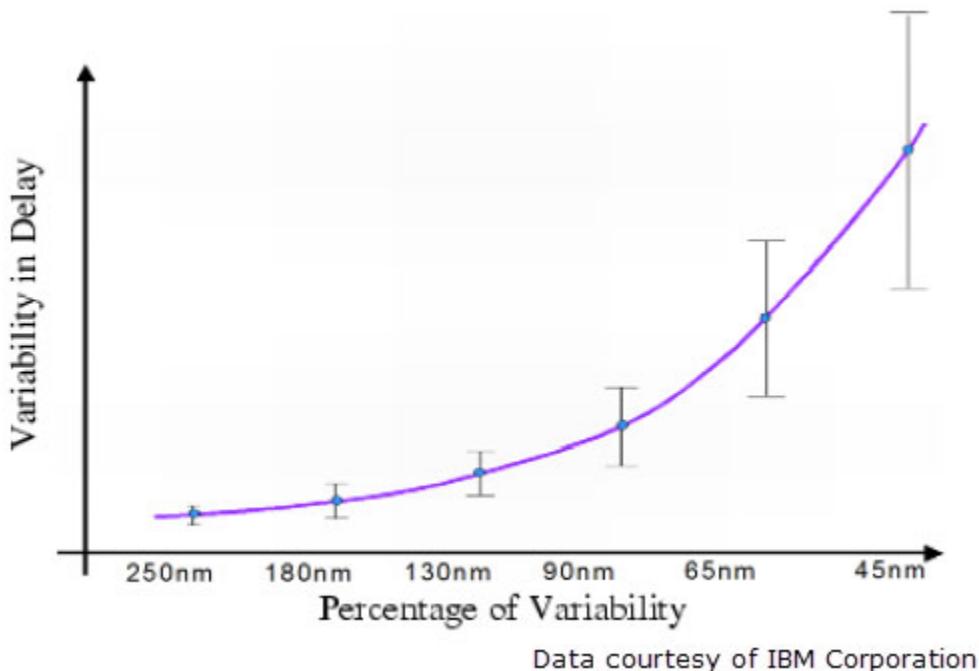
Focus changes are caused by the variation in the alignment of the wafer in the Z-axis during the manufacturing process, as well as different film stack thickness variations.

Mask bias variations are mainly introduced during the manufacturing of the mask itself.

While there are other sources of variations, these three variations can have a large impact in the devices, but more importantly they can often be minimized or eliminated by reorganizing the layout configuration.

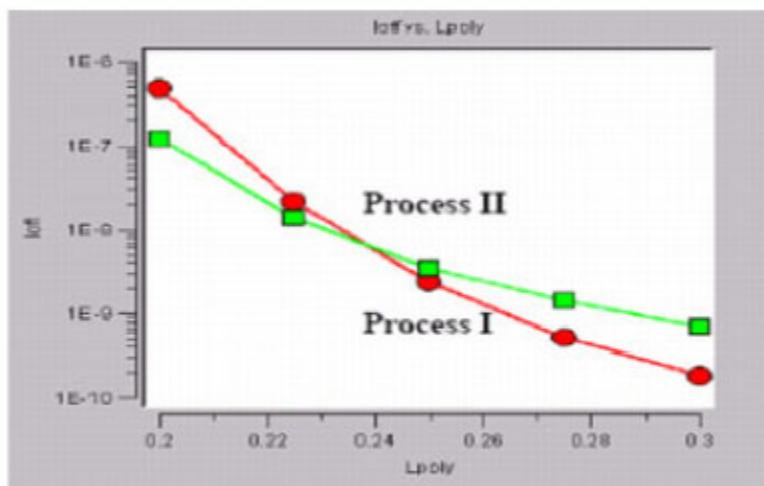
Figure 2-1 and Figure 2-2 on page 32 show the severity of the problem.

Figure 2-1. Increasing Variability in Delay as Process Nodes Shrink



Data courtesy of IBM Corporation

Figure 2-2. Impact of Process Variation on Leakage



Data courtesy of IBM Corporation

Though not as critical as dose, focus, and mask bias, *resist* and *etch* variation can also lead to fluctuations in delay and leakage. Resist variation is the inconsistency in the removal of the resist during manufacture. Etch variation is the inconsistent etching during manufacture.

The DRC-based approach to modeling process requirements involves the translation of complex behavior into simple geometry constraints. Sometimes the processing effects cannot be translated into simple geometric rules, or very restrictive rules limit the ability of designing cost-effective devices.

Adding Calibre LFD to a design flow, and combining it with traditional DRC checks, enables a more effective management of the complexity and priority of modifications in the layout and simplifies the tasks for both the manufacturing and design teams. As a result, it is possible to produce designs measurably less sensitive to process variations.

How Calibre LFD Works

At each stage in the design process, metrics provide insight into how process variation impacts design manufacturability.

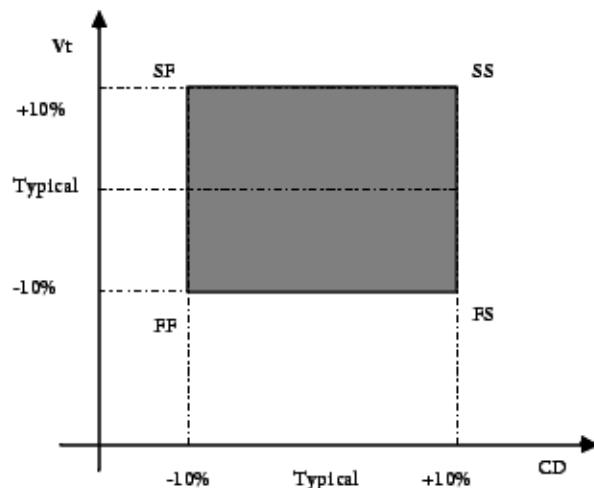
Process Variation	34
Calibre LFD Checks	37
Calibre LFD Scores and Other Properties	38

Process Variation

Calibre LFD characterizes design specific process variation and provides information that is context and performance relevant.

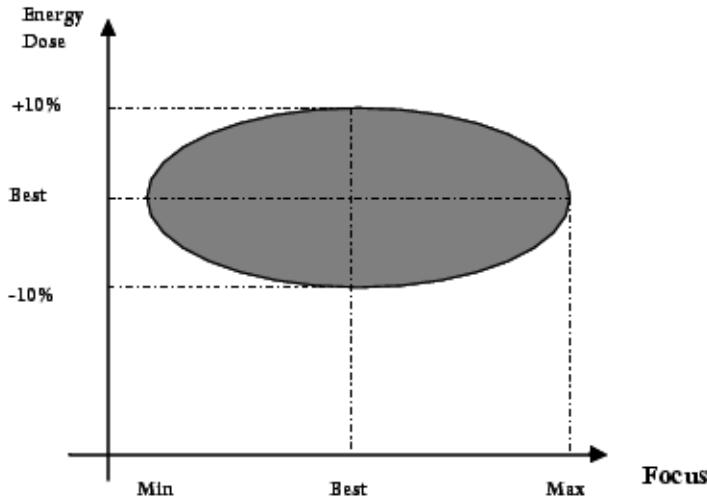
For example, a designer might look at a basic SPICE performance window, where the transistor performance (FF, SS, FS, SF) is plotted out against threshold voltage (V_t) and critical dimension (CD):

Figure 2-3. Performance Window



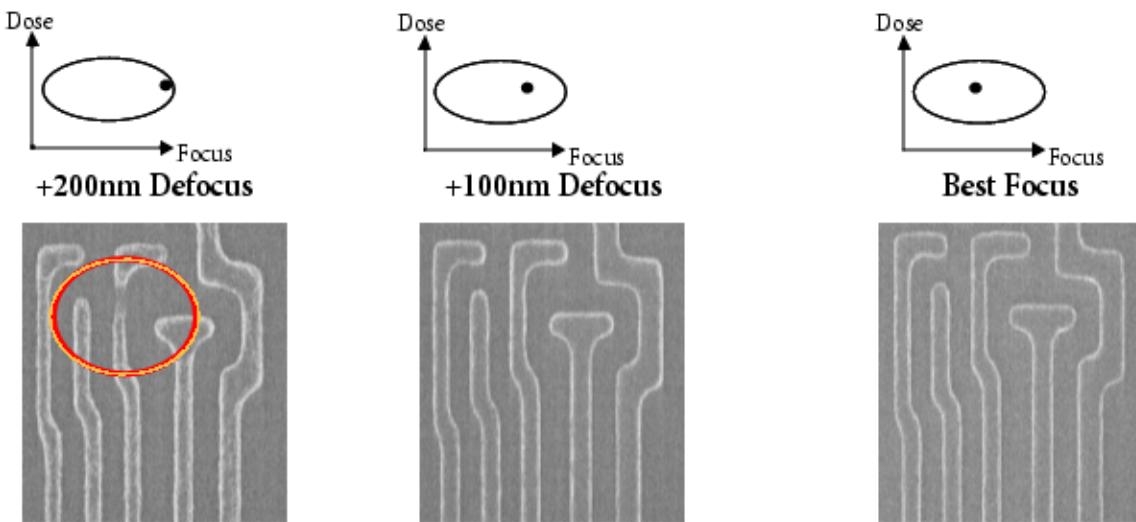
A lithographer, on the other hand, looks at a process window to see the range of normalized exposure dose and focus within which a mask prints:

Figure 2-4. Process Window



The two metrics are not completely unrelated in that the process-voltage-temperature (PVT) window takes into account the baseline lithographic process window. However, the PVT window provides no insight into the fact that the lithographic process window is not static, or that the process window for one feature may not be the same as the process window for an adjacent feature:

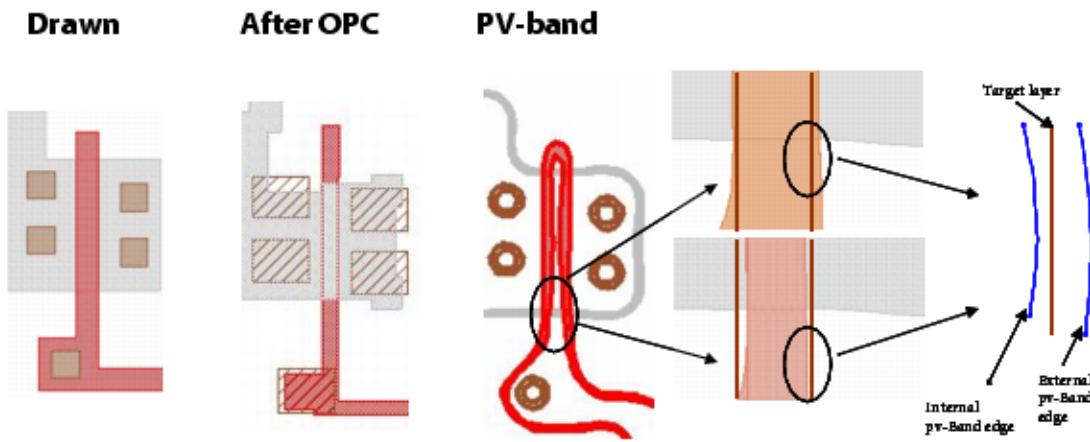
Figure 2-5. How Process Variation Can Impact Printability



Calibre LFD characterizes process variation with respect to a specific design. Using model-based simulations, the Calibre LFD tool creates new objects called process variability bands (PV-bands). PV-bands are created by running RET and OPC tools through a sequence of

process conditions. PV-bands show how much and where a design varies in response to process variations:

Figure 2-6. Single Feature and Associated PV-Band



PV-bands are context-sensitive. The PV-band for a feature in one location is different from the PV-band for the same feature somewhere else in the design with different neighboring features.

[Calibre LFD Checks](#) as well as [Calibre LFD Scores and Other Properties](#) generated from the PV-band provide designers with valuable information they can use to modify their designs to make the layout more robust to the anticipated process window variations. [Figure 2-7](#) through [Figure 2-9](#) on page 37 show how to improve the timing behavior through the process window by using the Calibre LFD application.

Figure 2-7. Original Cell

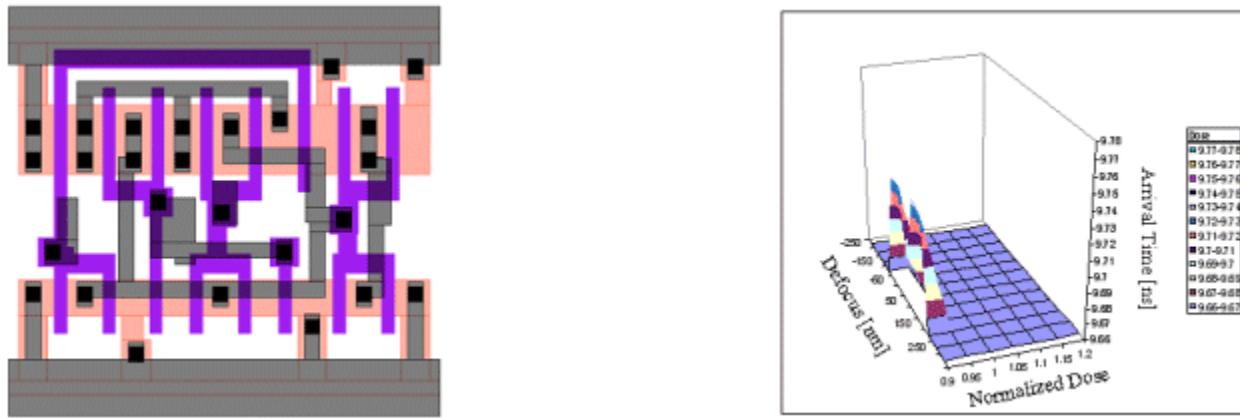


Figure 2-8. OPC Corrected Cell

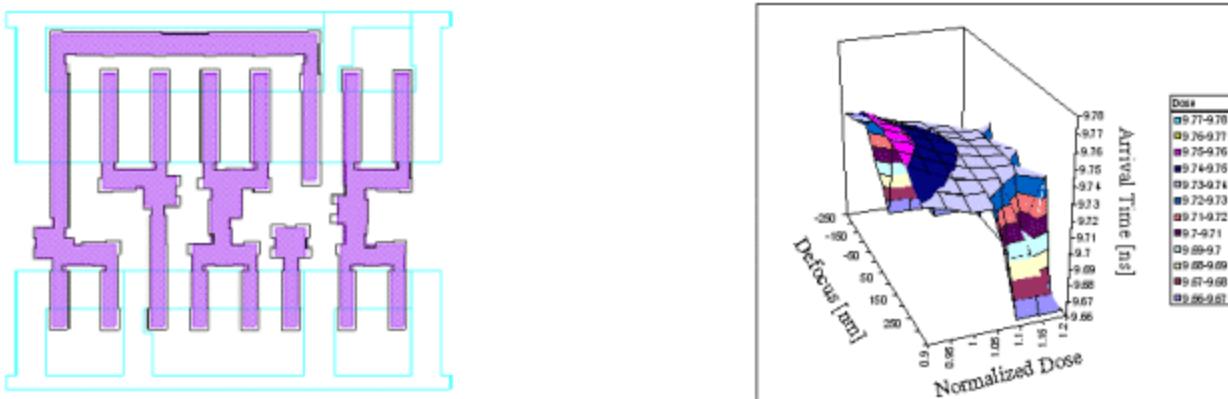
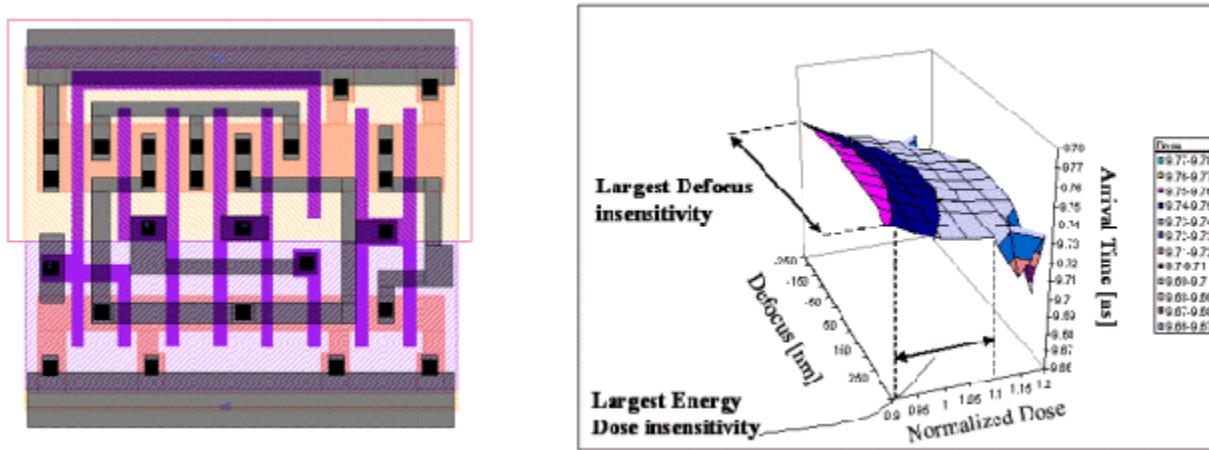


Figure 2-9. Calibre LFD Corrected Cell

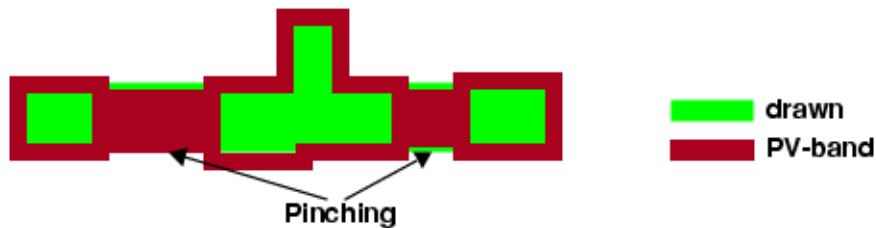


Calibre LFD Checks

Calibre LFD checks look at the interrelationships between PV-bands or between PV-bands and target features.

Take, for example, a potential pinching problem. This problem is represented by a PV-band with extra thick areas created by two portions of the PV-band overlapping:

Figure 2-10. PV-Band Representing Pinching

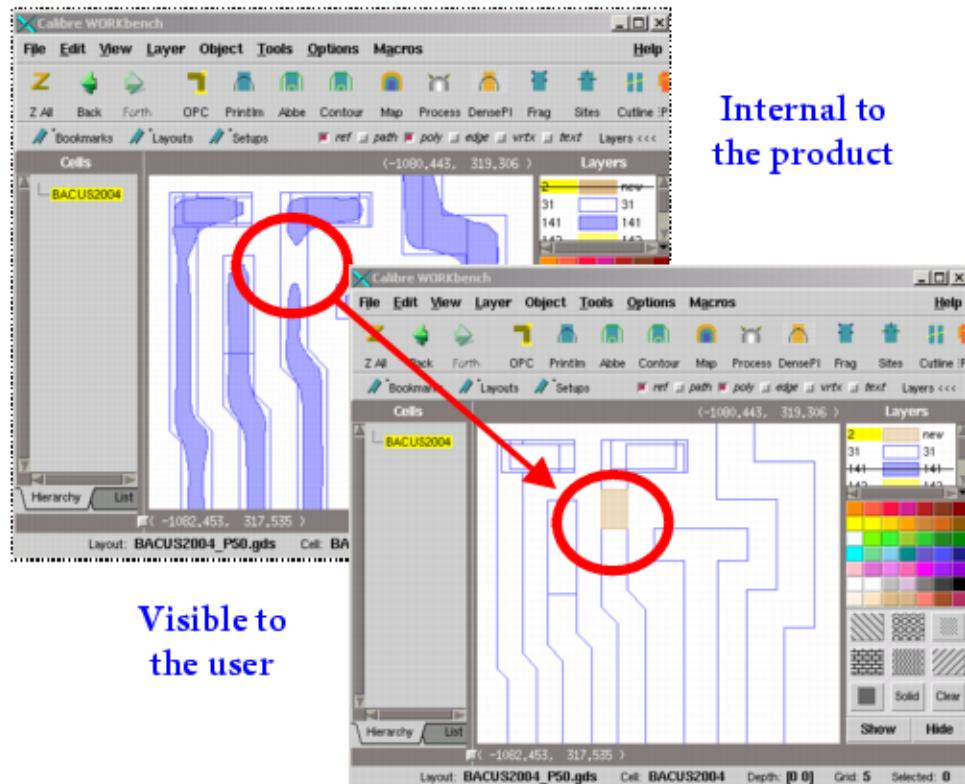


The Calibre LFD check to identify pinching might be written as:

```
LFD::MinWidthCheck -layer metal1 -subwindow 1 -minDRCwidth 0.2
                     -minLFDwidth 0.04 -checkName metal1_pinch_critical
                     -database lfd_errors.rdb
```

Creating the PV-bands and evaluating them with regards to the Calibre LFD checks is performed internally. The tool displays the results for you as DRC-like markers in the layout.

Figure 2-11. Viewing Calibre LFD Check Errors in the Layout Editor



Calibre LFD Scores and Other Properties

Calibre LFD metrics based on PV-band data return scores and other properties for the design, process, and type of error. Layout designers can use these metrics to guide them as they choose which problem areas to address first and how to make trade-off decisions.

Several metrics can be defined and accessed from Calibre LFD:

- **Process Variability Index (PVI™)** — This metric returns a score for the process, indicating the degree to which printing is impacted by changes in the process. This metric is most useful to the fabrication facility because it helps determine the best process conditions.

- **Design Variability Index (DVI™)** — This metric returns a score for the design, indicating how likely it is the variations in printing negatively impacts yield. This metric is most useful to design teams because it helps to identify sensitive and critical topologies.
- **Check Specific Properties** — These metrics are related to each type of check. For example, LFD::[MinSpaceCheck](#) provides a property called “Space” which for each check stores the minimum spacing found in the violation. Other properties such as MinCD, area, can be accessed depending on the Calibre LFD check of interest.

After modifying a layout, the designer can use Calibre LFD to analyze the drawn layout and verify it prints correctly with the appropriate process margin.

Designing “Litho-Friendly”

Calibre LFD can range from a simple yes/no design rule in which an error must be fixed, to a more gradual analysis of the printability of a given layout. For this reason, Calibre LFD provides several methods to define the priorities of checks, establish rankings, and manage trade-offs to adapt to the desired mode of operation. Modes of operation can be required rules, recommended rules, or analysis.

In any of these modes of operation, Calibre LFD predicts the effects of process variation on the printability of a specific layout design, and allows the design team to achieve more robust layouts.

Regardless of the mode of operation, deploying Calibre LFD requires the execution of three main steps:

1. Characterize process variations. For this phase in the product development, Calibre LFD does this by capturing lithography effects in the form of exposure dose, focus, and mask bias.
2. Through interactive and batch analysis, modify the layout to find a more robust layout pattern.
3. Stop when your design is “LFD Clean¹” (similar to “DRC clean”)

1. The term “LFD clean” refers to a drawn layout that prints correctly with the appropriate process margin. As with “DRC clean,” you must work with the semiconductor fabrication facility to characterize what is “LFD clean” for a particular process.

Figure 2-12. The Calibre LFD Flow

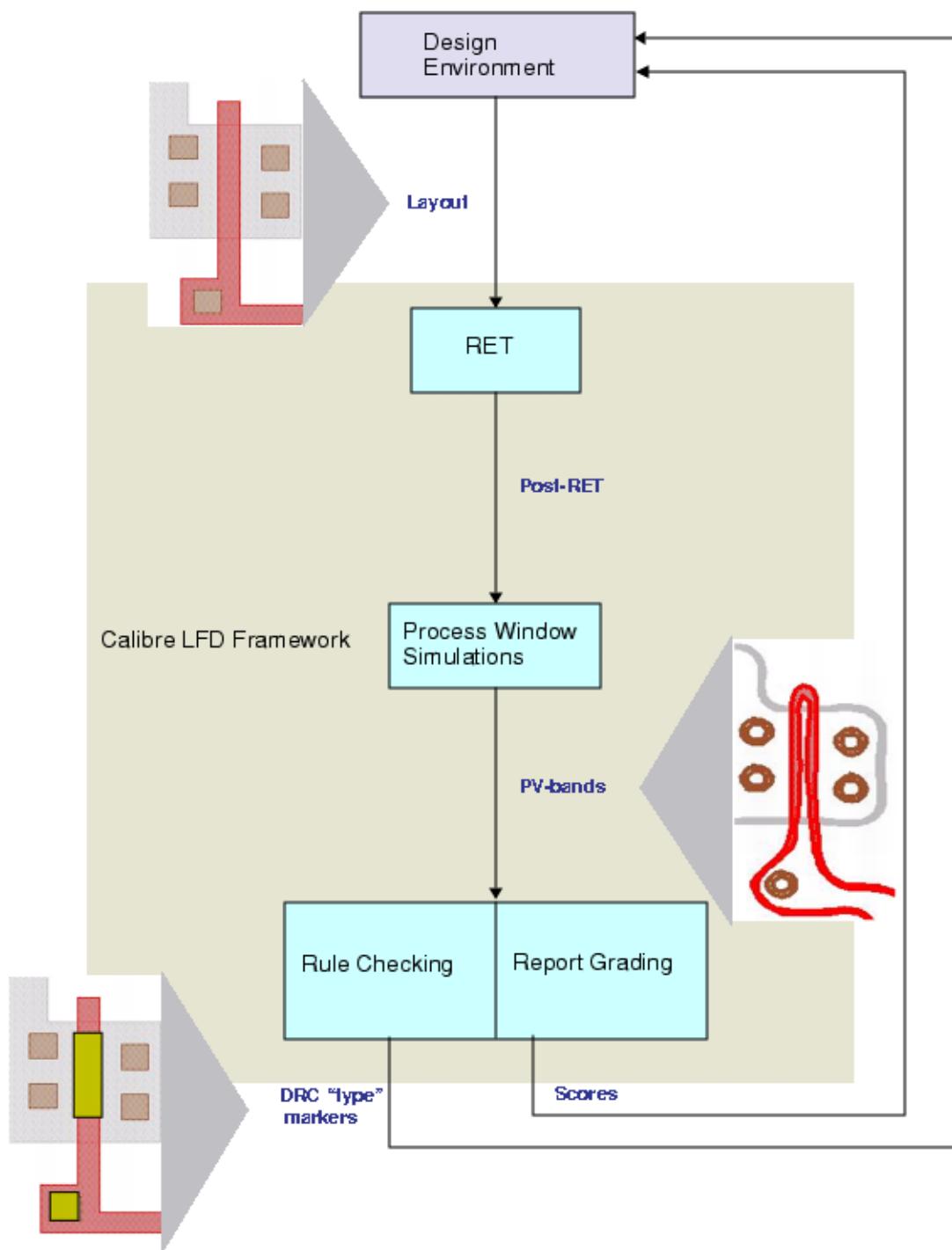
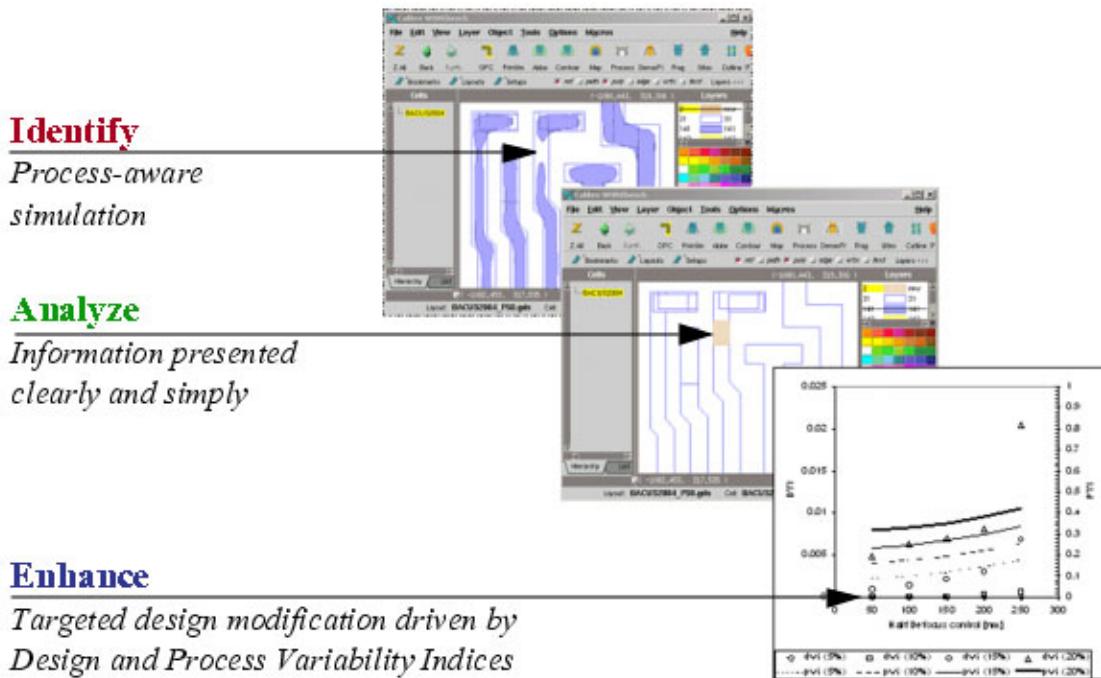
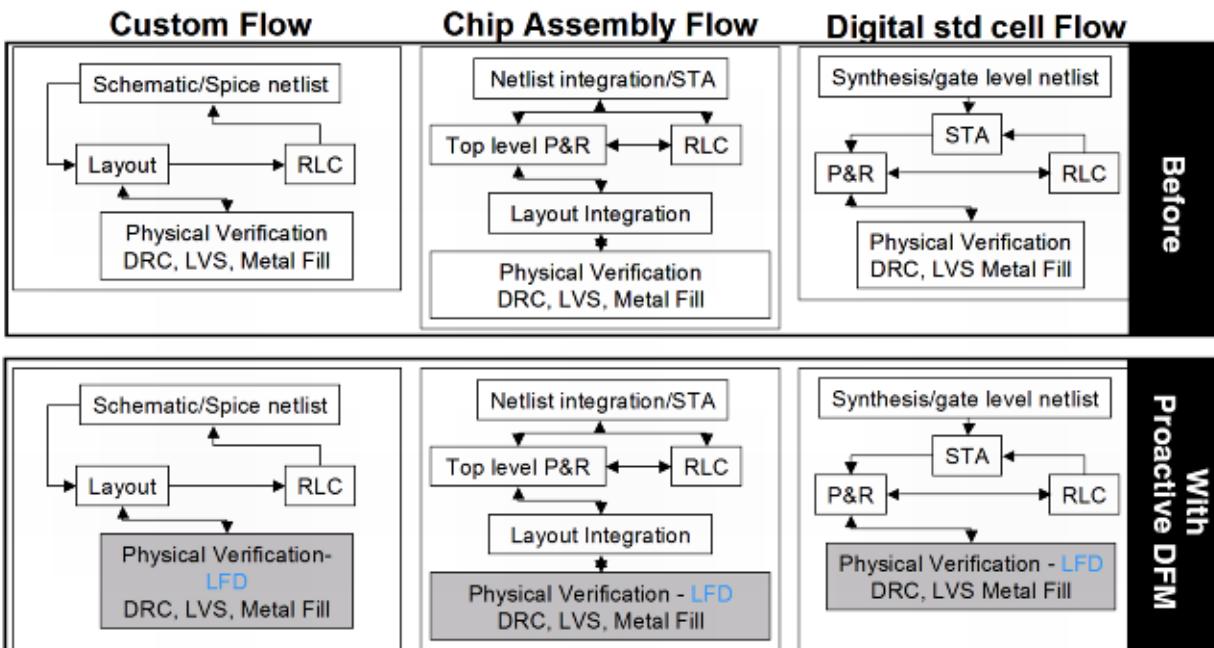


Figure 2-13. Calibre LFD in Action



Because yield-aware design using Calibre LFD can be performed iteratively as well as interactively, it fits well into any existing design flow. [Figure 2-14](#) shows how Calibre LFD can fit into any of the basic flows: Custom, Chip Assembly, and Digital Standard Cell.

Figure 2-14. Calibre LFD in Your Design Flow



Comparing Calibre LFD to DRC

The DRC-based approach to modeling process requirements is not sufficient to account for the effects of process window and device variations. This is due to design rules not modeling the sensitivity of a lithographic process window to the actual features being printed, or to the effect of the sensitivity on semiconductor chip yield.

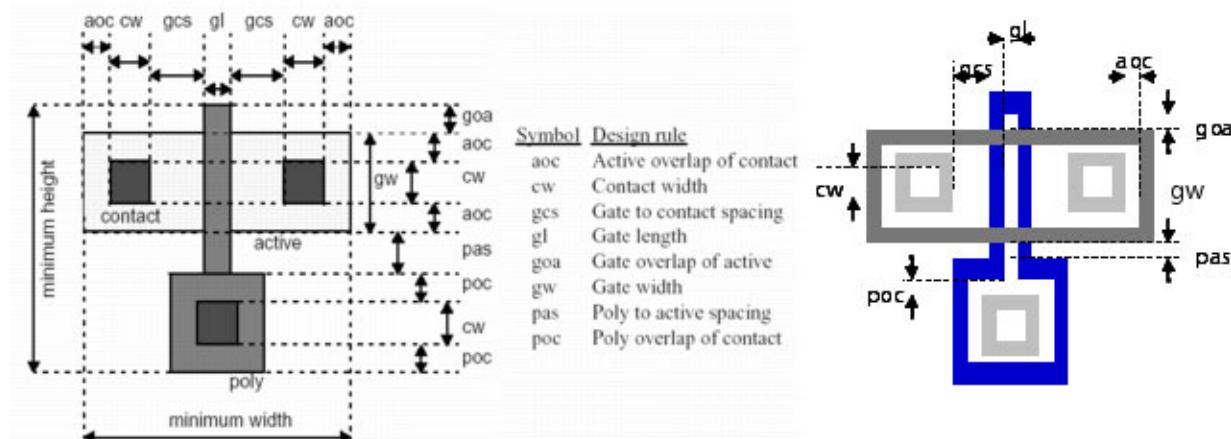
The key differences between Calibre LFD and DRC are shown in [Table 2-1](#).

Table 2-1. Comparing Calibre LFD Checks to Design Rule Checks

	DRC	Calibre LFD
Checks applied to...	Drawn layer	PV-bands (internally created by the product)
RET factored into checks?	No	Yes
Checks are...	Binary (pass fail)	Continuous (results differ depending on the set of tolerable process variations being investigated)
Provide insight into which errors are most critical?	No	Yes, errors can be prioritized based on the size of the process variation experiment.

While Calibre LFD checks are different from design rule checks, many of the properties investigated using DRC can also be investigated using Calibre LFD checks. Because the Calibre LFD checks are written with respect to PV-bands rather than drawn data, results of the checks offer a different perspective on the manufacturability of your design.

Figure 2-15. Mapping DRC Rules into Calibre LFD Checks

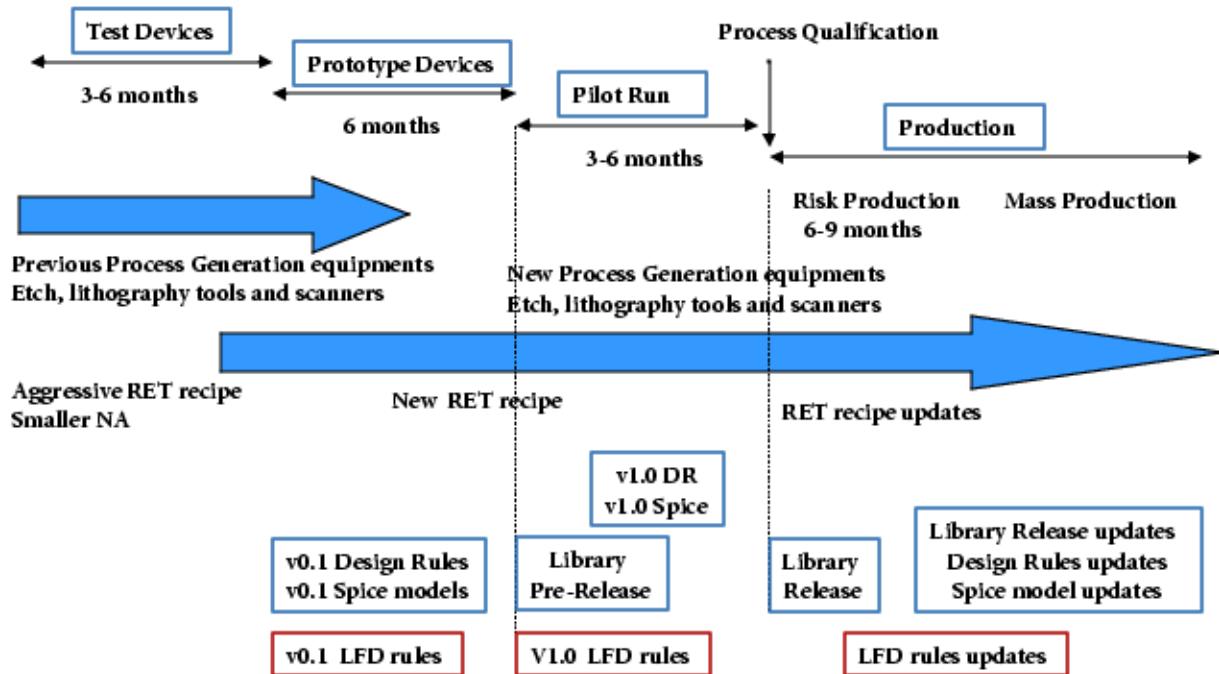


Evolving with Your Process

Calibre LFD rules and models evolve with your process. You create your preliminary process-specific Calibre LFD design kit (rules, models, and RET recipe) based on knowledge of similar processes and existing equipment. As the process matures, you update the Calibre LFD rules to reflect the current level of understanding.

Once the Calibre LFD checks are deemed complete, slight changes in the process can be accounted for immediately by updating the models used by Calibre LFD. This permits a more efficient connection between the process state and the verification tools.

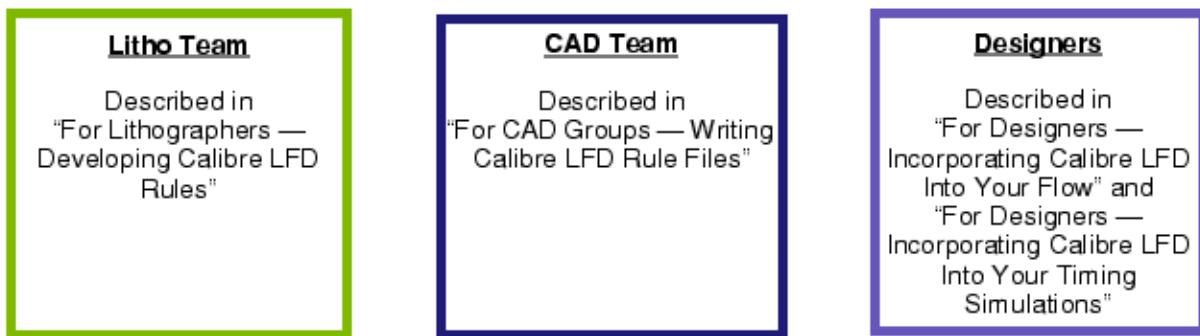
Figure 2-16. Calibre LFD Rules Evolve with Your Process



Sharing Information Across the Flow

Calibre LFD works by giving designers access to information previously available only to lithographers working in the fabrication facility. Due to this, setting up a Calibre LFD process at your site involves the Litho, CAD, and Design teams.

Figure 2-17. Sharing Information



Where relevant, chapter titles indicate the team for which the information is most relevant. It is possible you may fill more than one of these roles, in which case you should read all chapters that apply.

Chapters:

- “[For Lithographers — Developing Calibre LFD Rules](#),” — Written for the lithography team, which provides the process-specific models and RET recipes, and develops Calibre LFD rules based on process variation data.
- “[For CAD Groups — Writing Calibre LFD Rule Files](#),” — Written for the CAD team, which translates the Calibre LFD rules into valid TVF or SVRF code.
- “[For Designers — Incorporating Calibre LFD Into Your Flow](#),” — Written for the designers who run the Calibre LFD checks and use the results to create more robust designs.
- “[For Designers — Incorporating Calibre LFD Into Your Timing Simulations](#),” — Written for designers who need to evaluate design performance across process variation or predict timing and performance based on simulated shapes.

Chapter 3

For Lithographers — Developing Calibre LFD Rules

The lithographer's role in implementing the Calibre Litho-Friendly Design (Calibre LFD) flow is to design process variation experiments and develop the rules identifying potential failure spots. The following topics describe what is needed from the lithography team and in what format.

The information for lithographers is used along with the information “[For CAD Groups — Writing Calibre LFD Rule Files](#)” on page 69.

Requirements for Writing Calibre LFD Rules	45
Background Information for Writing Calibre LFD Rules	47
Generating Process-Specific Calibre LFD Kits.....	54

Requirements for Writing Calibre LFD Rules

You must have a layout database, a basic rule file, an optical model, and a resist model in order to write Calibre LFD rules.

- A layout database containing the types of topographies you anticipate finding on a production chip.
- For each layer to be evaluated using Calibre LFD:
 - A TCCcalc vector optical model represents the illumination system to use in the target process. This model should reflect the optimal dose and focus, as well as optionally the mask bias, resist, and etch variation settings.
 - A CM1/VT5 resist model represents the resist behavior for the target process. If you are designing Calibre LFD rules to be used for a process that is not yet completely known, you should use a resist model that captures as much information as is known.
 - A basic setup file designed for use with the target process.
 - A basic TVF rule file used for implementing the RET recipe for the target process.

The Tcl Verification Format (TVF) is a programmable extension to the Standard Verification Rule Format (SVRF) language for Calibre. The rule file used to run Calibre LFD must be written using TVF.

- Optionally, an etch model.

Related Topics

[Tcl Verification Format \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Background Information for Writing Calibre LFD Rules

Some background information about PV-bands, process variability, and the Calibre LFD flow is useful before starting to write Calibre LFD rules.

What are PV-Bands?	47
Types of PV-Bands	49
The Overall Calibre LFD Flow	51
How PV-Bands Predict Variability	52

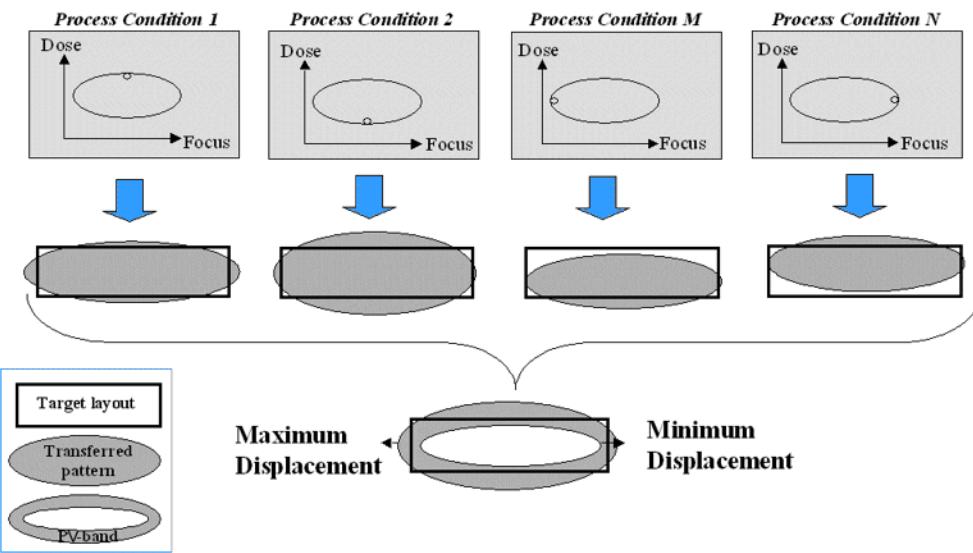
What are PV-Bands?

A *PV-band* is a geometry that shows how edge locations respond to process variations. It represents the area within which a feature prints as the process conditions vary. Generating PV-bands involves simulating the printed image at the various process conditions and combining the resulting printed images. PV-bands have widths due to process variation.

Calibre LFD runs simulations across all of the intended layers through multiple points in its lithographic process window to generate PV-bands. It is on these PV-bands (or the underlying inner and outer contour layers) that Calibre LFD runs its LFD checks. PV-bands are context-sensitive. The PV-band for a feature in one location is different from the PV-band for the same feature somewhere else in the design with different neighboring features.

All PV-band data are present as polygon layers. Inner contour layers represent locations in the layout that always print, across all lithographic process window conditions. Outer contour layers represent locations in the layout that print at some point within the lithographic process window conditions.

Figure 3-1. PV-Bands Representing a Set of Simulated Print Images



A *contour* is a single simulation exposure condition, represented by a polygon layer. Several contours are used to make a PV-band. The maximum and minimum displacement lines in a PV-band are each represented by polygon layers.

Examining a PV-band provides the following information:

- **Variability Region** — The PV-band geometry itself, represents the region within which printing varies in response to process variations.
- **Printability Region** — The area inside of the PV-band's donut-like shape, represents the printing region independent of process variations.
- **Non-Printability Region** — The area outside of the PV-band's donut-like shape, represents the non-printing region independent of process variations.

Figure 3-2. Three Areas of a PV-Band Layer

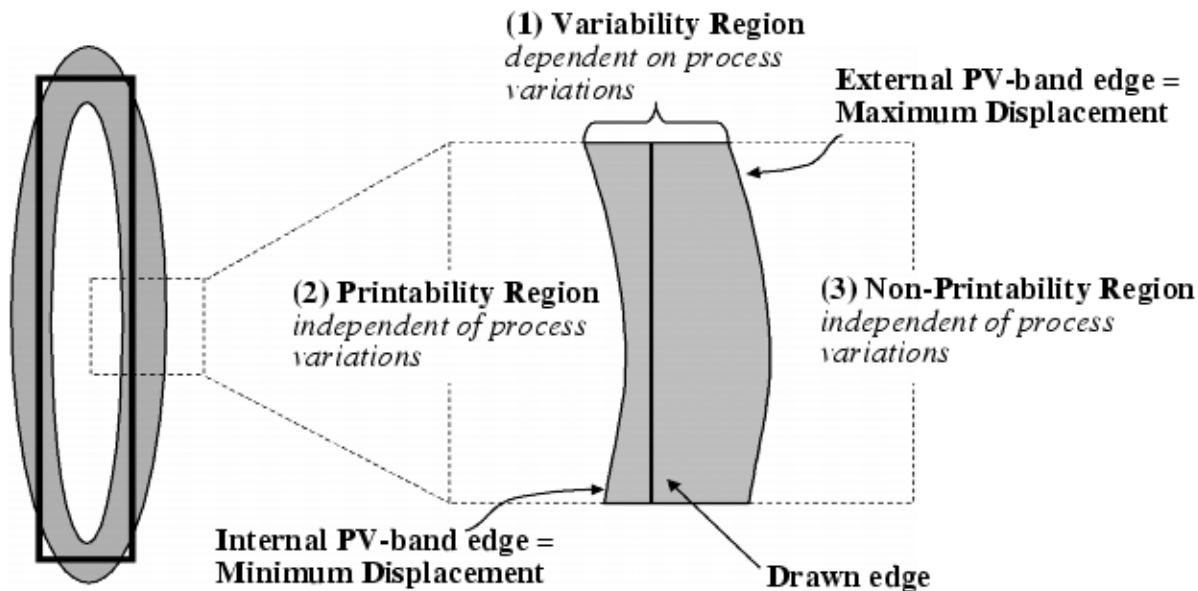
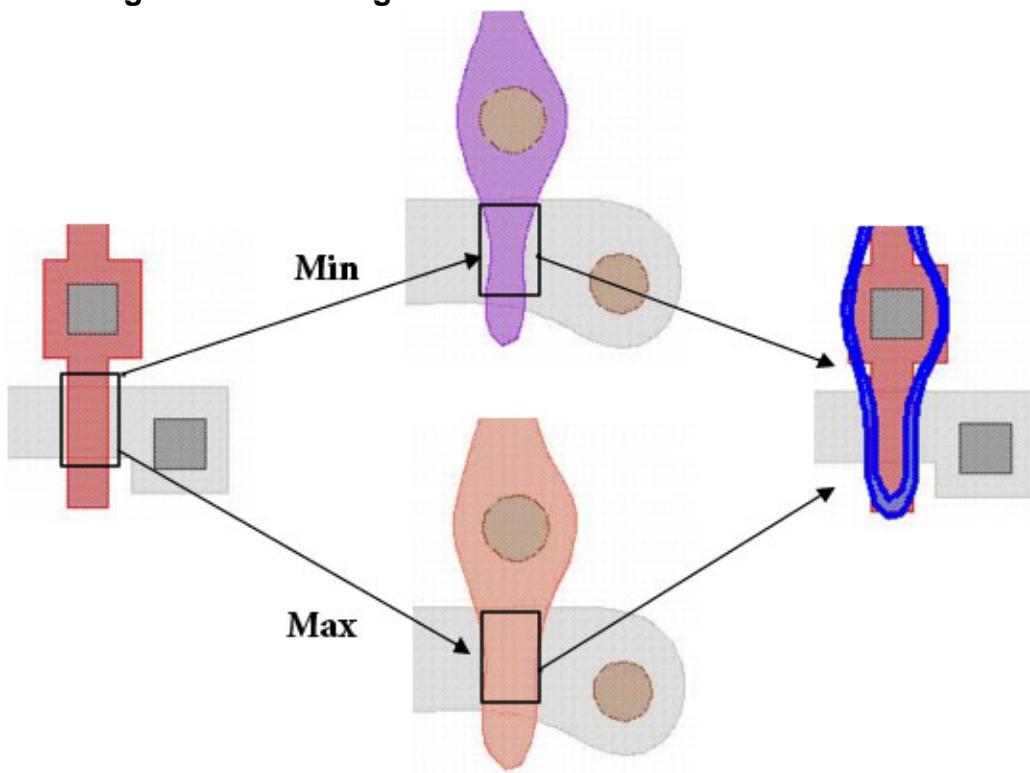


Figure 3-3. Relating PV-Bands to Transistor CD Variation



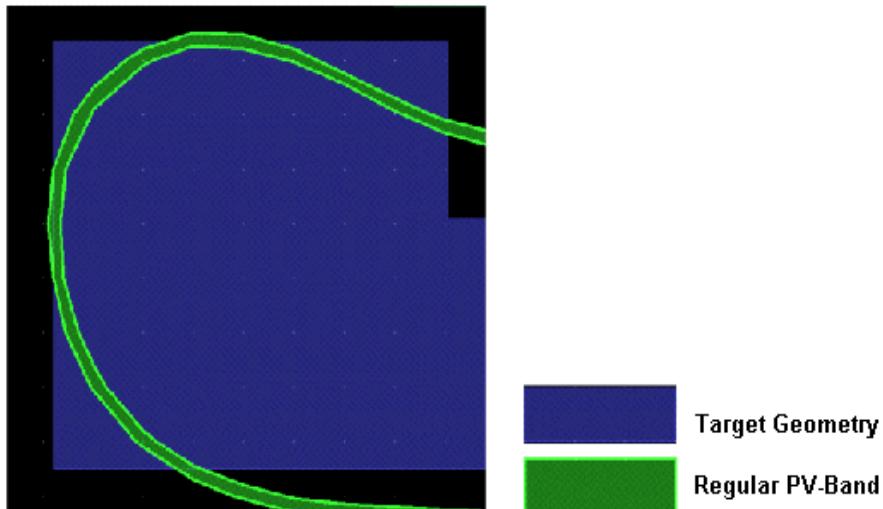
Types of PV-Bands

The Calibre LFD tool generates regular and absolute PV-bands.

Regular PV-bands

Regular PV-bands represent the maximum and minimum edge displacement.

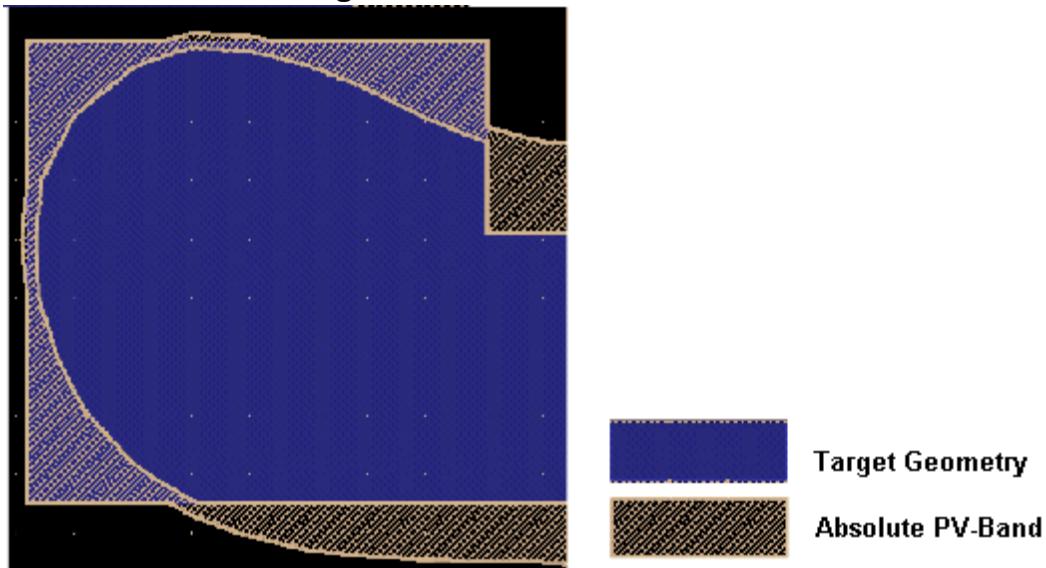
Figure 3-4. Regular PV-Band



Absolute PV-bands

Absolute PV-bands represent the maximum and minimum deviation from the target layout. Whereas regular PV-bands let you see how much feature shape varies from one process condition to another, absolute PV-bands let you see how much the feature shape deviates from the drawn shape. If the outer PV-band edge is within the target layer, the outer PV-band snaps to the design edge. Similarly, if the inner PV-band edge is outside the layout target, the inner PV-band edge snaps to the design edge. Regular PV-bands are always within the absolute PV-bands.

Figure 3-5. Absolute PV-Band



Notice the regular PV-band is a subset of the absolute PV-band.

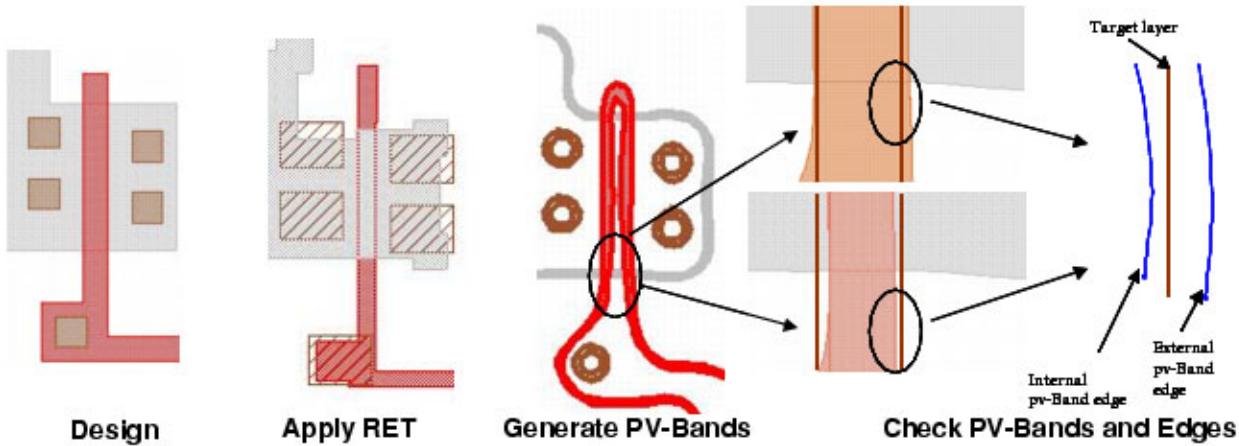
The Overall Calibre LFD Flow

Using PV-bands to identify potential problems and predict how well a design prints involves several steps.

1. Apply RET to predict the actual mask geometries.
2. Generate PV-bands based on the mask geometries.
3. Perform checks by examining relationships between pairs of PV-bands or between PV-bands and the original design layer.
4. Calculate variability scores based on the PV-bands and on any errors found during the check process.

Figure 3-6 shows the first three steps in the flow.

Figure 3-6. From Design to PV-Bands



Expanded Flow

An expanded version of the flow follows. The overall Calibre LFD flow consists of the following steps, with the following requirements needed to proceed:

1. Receive GDS or OASIS file containing design layers — This starting point assumes a DRC-checked design.
2. Perform retargeting and add scattering bar (SBAR) polygons — Retargeting involves modifying dimensions of polygons slightly to overcome dimension-changing effects of manufacturing. This is also known as biasing, however it should not be confused with mask biasing, which is variation in the manufactured mask.

3. Perform OPC — Optimize the mask pattern shape for printability. This step and the prior SBAR step require the RET recipe (SBAR and OPC). In the previous diagram, steps 2 and 3 are referred to as the “Apply RET” phase.
4. Contour generation — This is the simulation process resulting in separate output of experiments saved as polygon layers. At this point we have simulated wafer information.
5. PV-band generation — The simulated wafer information along with process variation (PV) information is contained in PV-bands. Simulation exposure requires models.
6. Checks — Rule checking requires thresholds.
7. Score — Scoring formula also requires thresholds. In the previous diagram, steps 6 and 7 were labelled “Check PV-Bands and Edges”.

How PV-Bands Predict Variability

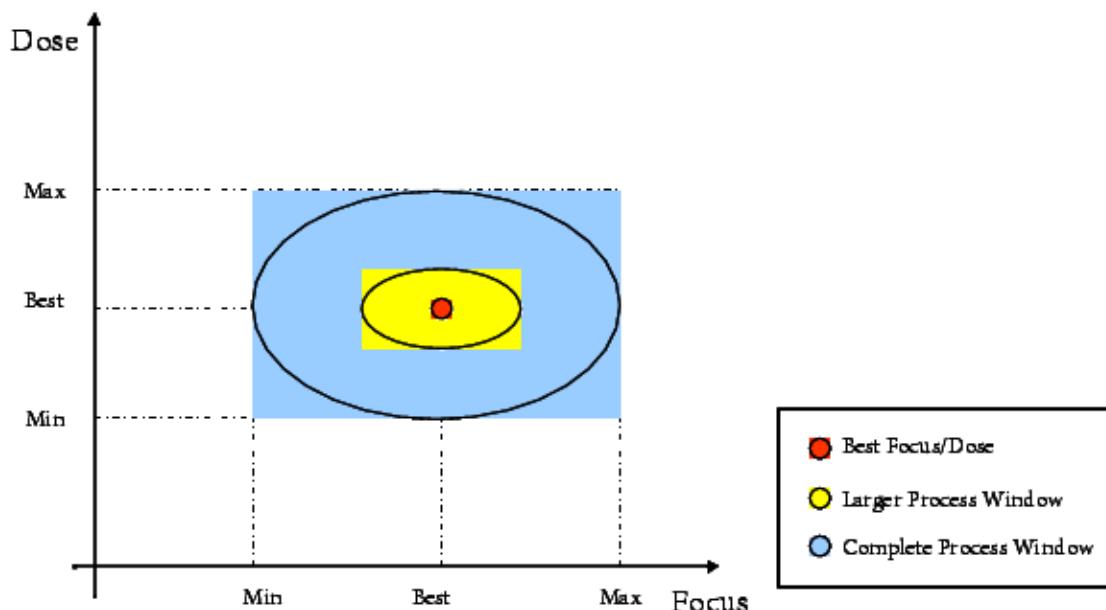
Each PV-band represents all possible printing locations resulting from one set of dose, focus, mask bias, resist, and etch variation conditions. That set of exposure conditions is referred to as *a process variation experiment*.

When you run Calibre LFD checks using one process variation experiment (one set of focus and dose conditions), the PV-bands indicate whether or not the features prints at all process conditions. When the PV-bands indicate a failure, it is not clear whether the feature fails to print under a single condition or under many.

By nature, no process can be completely free of variations, however, under normal circumstance, the process should spend most of the time close to nominal conditions and less time farther away from that point. Multiple experiments, each testing a different set of dose, focus, and other variability conditions, provide insight into over-all robustness as well as the criticality of a specific failure. Together these multiple experiments make it possible to predict and calculate a “probability” of feature variability in multiple process-condition environments.

The three process experiments shown in [Figure 3-7](#) explore ever-increasing process areas, ranging from best case (best dose and focus where the inner and outer PV-bands overlap completely) to complete coverage of the process area anticipated during the production run. Variation is what creates the window in the larger two process areas.

Figure 3-7. Three Process Experiments Providing Full Coverage



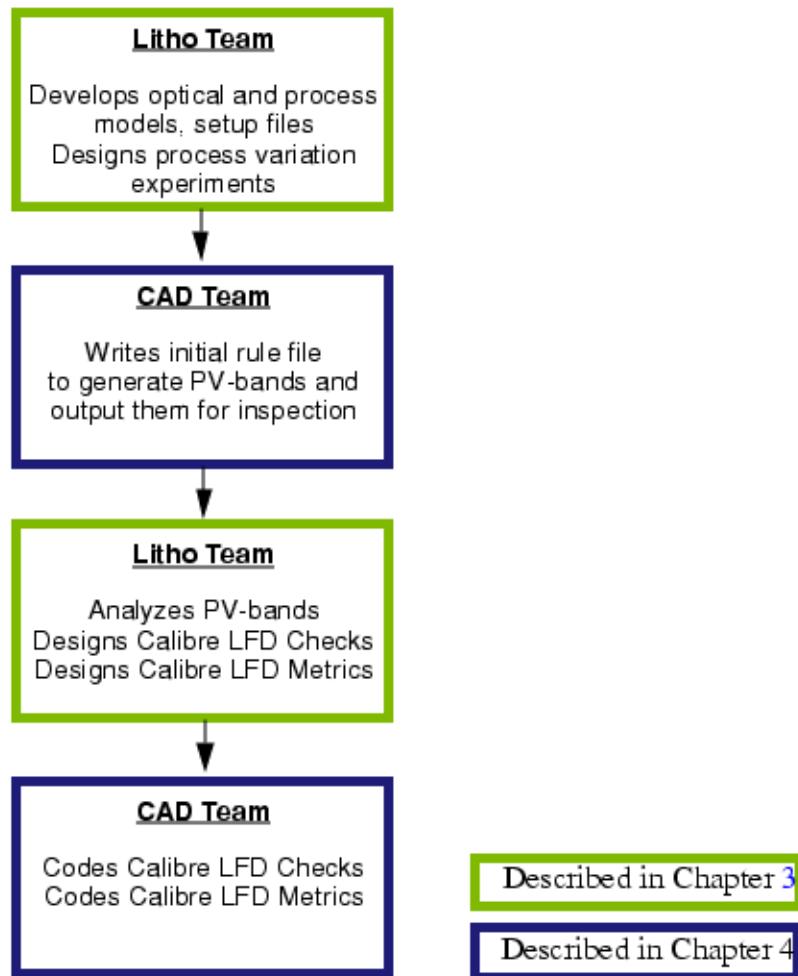
The goal of Calibre LFD is to optimize the design so that:

1. For experiment 1: Best focus and dose
 - o The Design Variability Index (DVI) is 0.
 - o The design is LFD clean.
2. For experiment 2: Larger process window
 - o DVI is as low as possible.
 - o The design is as LFD clean as possible.
3. For experiment 3: Complete process window
 - o You have selected the lowest DVI possible among different layout options.
 - o You have done the best job possible given the yield versus schedule trade-off.

Generating Process-Specific Calibre LFD Kits

Generating process-specific Calibre LFD kits for design verification is a cooperative effort involving lithographers and rule writers. Depending on the size of your organization, this will be one team or two, one person or many. For simplicity, the assumption is that there are two separate teams: the litho team and the CAD team.

Figure 3-8. Developing Process-Specific Calibre LFD Kits



Because it is likely there are two different teams, the litho team and the CAD team, each with their own area of expertise, these tasks are divided between two chapters:

- “[For Lithographers — Developing Calibre LFD Rules](#)” on page 45
- “[For CAD Groups — Writing Calibre LFD Rule Files](#)” on page 69

In the event your organization has a single team composed of lithographers and rule writers, the links below are organized according to the sequence of tasks that your team is most likely to follow.

1. Defining Process Variation Experiments
2. Creating Optical Models
3. Writing the Rule File for Generating PV-Bands
4. Visualizing PV-Band Data Using Calibre RVE
5. Using PV-Bands to Identify Failure Spots
6. Metrics to Score a Process or Design
7. Translating Failures into Rules
8. Coding Metrics to Score a Process or Design

The following information is primarily for the litho team, but it is also useful for the CAD team.

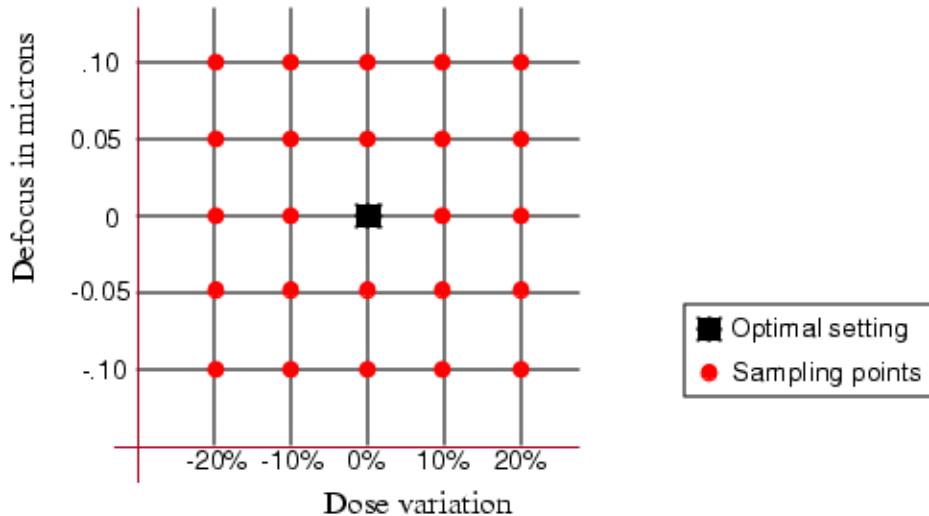
Defining Process Variation Experiments	55
Creating Optical Models	57
Visualizing PV-Band Data Using Calibre RVE	58
Using PV-Bands to Identify Failure Spots	60
Using Built-in Check Functions	61
Designing Custom Checks	62
Metrics to Score a Process or Design	63
Ranking Checks and Errors	66

Defining Process Variation Experiments

In generating PV-band data, you define the process variations to explore by specifying a set of dose, focus, mask bias, resist, and etch variation settings relative to optimal values.

The LFD::PVband function evaluates printing only at the specified grid points.

Figure 3-9. Sampling Grid Points, not Entire Space



Procedure

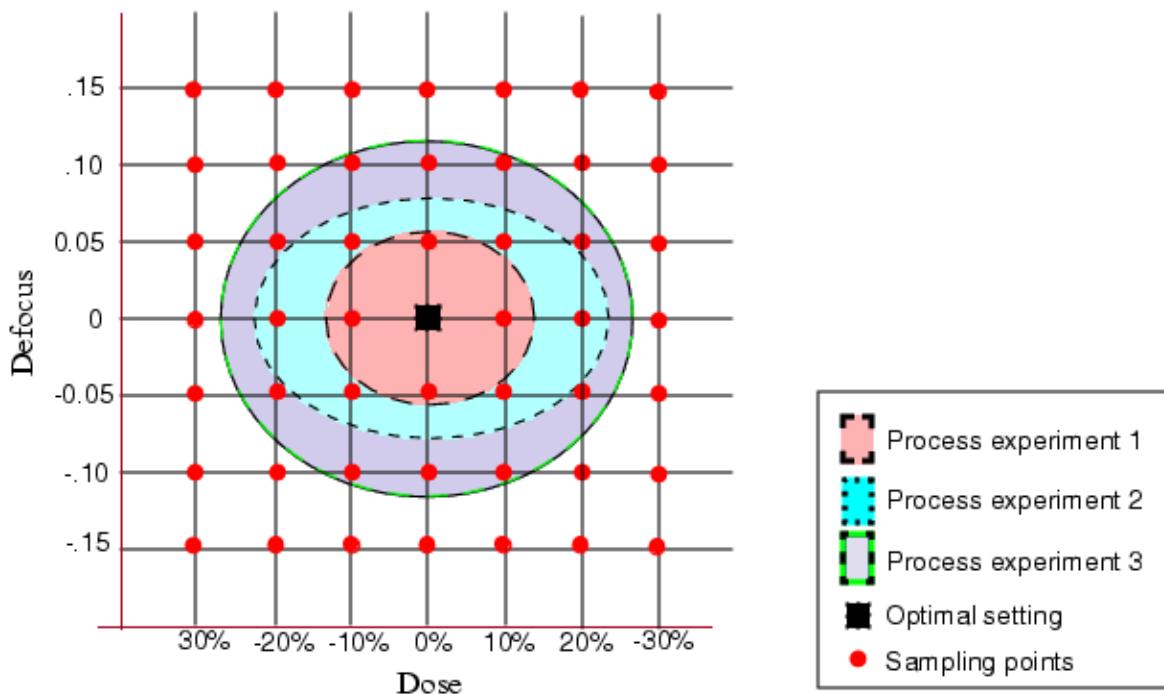
1. Identify the optimal process variability settings (best focus, dose, mask bias, resist, and etch settings).
2. Identify the focus, dose and other variations anticipated during the production run.
3. Decide whether to design one experiment or many:
 - For multiple experiments (recommended), select two or more sets of focus and dose conditions with one set to represent a small process space containing the optimal setting and another set to represent a larger space that captures the anticipated corner cases.
 - For a single experiment, select a set of focus and dose conditions that capture the range of process variability settings you anticipate during the majority of the run. This is a subset of the variations from step 2.

Note

 Multiple experiments can be structured similarly to those shown in [Figure 3-7 “Three Process Experiments Providing Full Coverage”](#) and in [Figure 3-10 “Planning Multiple Experiments,”](#) or partition the parameter space differently to suit your needs.

4. For each design experiment, select a set of discrete dose and focus values to be evaluated. Each of the sampling points within a shaded area represents a contour to be created by the Calibre LFD tool.

Figure 3-10. Planning Multiple Experiments



5. Use this information for [Creating Optical Models](#), creating resist models, and creating etch models to use in generating PV-bands.

Creating Optical Models

An optical model is a set of functions for creating a numerical approximation of the image behavior. Optical modeling requires a parameter file containing all parameters needed for creating an optical model.

You must create a different optical model for each defocus setting to be evaluated. The different dose settings do not require separate models.

For example, the process variation experiments represented by [Figure 3-10](#) on page 57, requires creating five optical models:

- D0 — The base optical model.
- D50 — The model for simulating a defocus off by 0.05 microns.
- DM50 — The model for simulating a defocus off by -0.05 microns.
- D100 — The model for simulating a defocus off by 0.10 microns.
- DM100 — The model for simulating a defocus off by -0.10 microns

Procedure

1. Open the Optical Model Tool in Calibre WORKbench — **Tools > Optical Model**.
2. Load the base model.
3. In the **Film Stack** tab, modify the **Beamfocus**, adding the value in microns (positive or negative) by which defocus varies.
4. In the Main tab, modify the **Model Name** to reflect the defocus variation.
5. Click **Generate** to save the optical model file and generate the model.
6. Repeat for each of the other defocus variations.
7. Provide the process variation experiment data plus the paths to these models to the CAD team, who writes the rule file for generating PV-band data.

Visualizing PV-Band Data Using Calibre RVE

Calibre RVE is a graphical debug program that interfaces with most IC layout tools. You can view the PV-band data using Calibre RVE in conjunction with the layout viewer.

Prerequisites

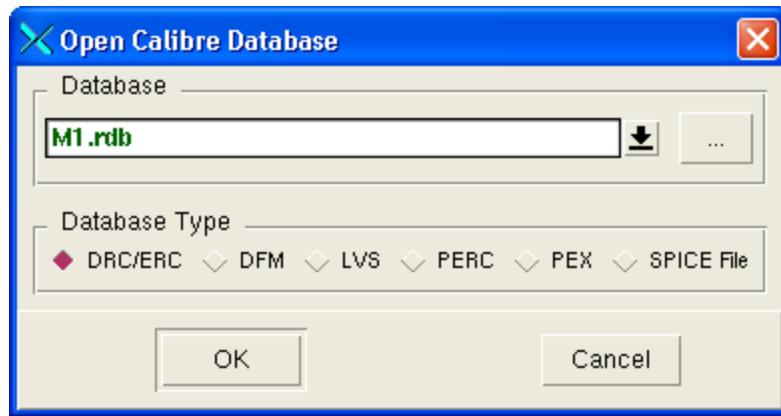
- A set of PV-bands for your test data. The assumption is that someone has written the Calibre LFD rule file as described in “[Writing the Rule File for Generating PV-Bands](#)” on page 88.

PV-band data is saved in the form of GDS, OASIS, or results database (RDB). If saving to GDS or OASIS, view the PV-bands with the layout viewer. It is best to avoid viewing an RDB due to the large number of sample points.

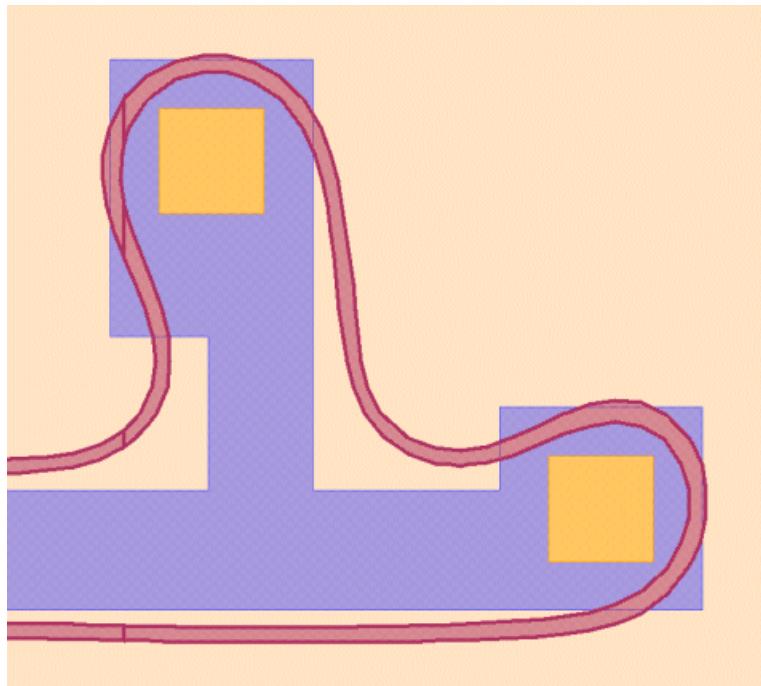
Procedure

1. Open the design database in Calibre DESIGNrev or Calibre WORKbench.
2. Choose **Verification > Start RVE**.

3. In the Calibre RVE dialog box, make sure to select the “DRC/ERC” database type, supply the path to the Bands database¹, and then click **OK**.

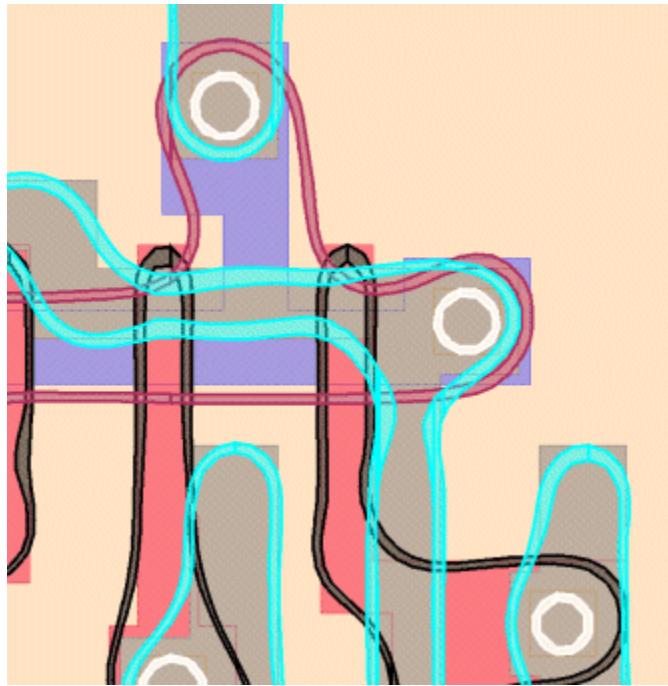


4. In the RVE window, select the PV-band data for a single experiment on a single layer.
5. Click the **Highlight Current** button, or right-click and select **Highlight**.



1. This is the pathname you supplied through the -database keyword to the OutputBands function.

6. Repeat for other layer/experiment/band types as desired.



Note

 While you could display all PV-band data at once by selecting **Highlight > Highlight all**, the steps described here are preferred because each set of PV-band data is displayed on a different layer. This allows you to work with each set of PV-bands individually.

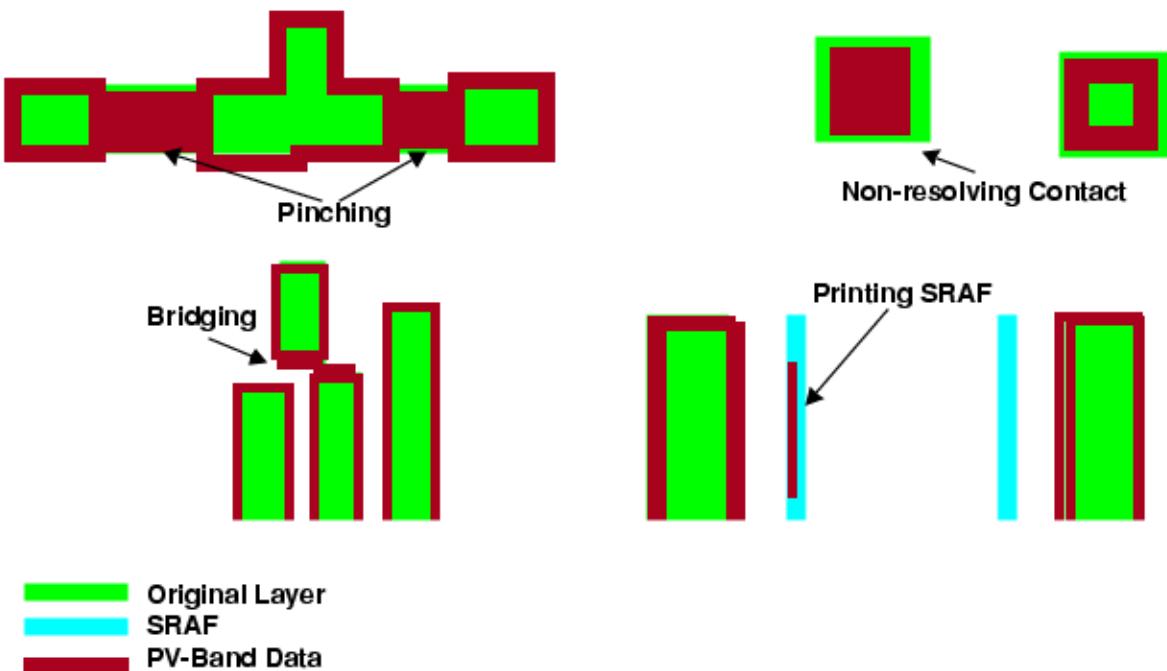
Using PV-Bands to Identify Failure Spots

The topology of a PV-band can predict the type of failure.

Four of the most obvious failure configurations are these:

- **Pinching** — The PV-band for a feature spans the entire feature in at least one spot.
- **Bridging** — The PV-band for one feature overlaps or comes very near the PV-band of another feature.
- **Non-resolving contacts** — The PV-band for a contact covers all or most of the contact.
- **SRAFS that print** — PV-band geometry where there is no geometry on the target layer.

Figure 3-11. Failure Spots Identified by PV-Bands



The process-specific Calibre LFD design kit provides two methods for designing checks to catch these problem topologies:

- “[Using Built-in Check Functions](#)” on page 61
- “[Designing Custom Checks](#)” on page 62

Using Built-in Check Functions

The built-in check functions provide a quick way to check for the PV-band configurations most commonly considered to be problematic.

The built-in check functions are listed in the section “[Calibre LFD Checks](#)” on page 181.

For each of the checks, you must specify:

- The target layer under investigation.

Note

 Two layer checks examine interactions between two layers, so they require you specify both layers.

- The process variation experiment these checks relate to. You reference individual process variation experiments by their positions in the ordered lists (the subwindows) of lists supplied for the -opticalSpanList and -doseSpanList arguments to the [PVband](#)

function. For more information, refer to the section on [Describing Process Variation Experiments](#).

- The database to which you want check scores written.
- The values to check for. These vary according to the check.

Designing Custom Checks

Designing custom checks involves describing problem situations in terms of properties or relationships between original or derived layers. For best results, provide the rule writer you are working with verbal, mathematical, and graphical descriptions of the problem and the solution.

The [PVband](#) function generates six derived layers *for each experiment*, all of which are available to you for designing custom checks.

- [Regular PV-bands](#)
- Inner edge of the regular PV-band
- Outer edge of the regular PV-band
- [Absolute PV-bands](#)
- Inner edge of the absolute PV-band
- Outer edge of the absolute PV-band

Inner edge layers are presented as polygon layers. They represent locations in the layout that always print across all process window conditions.

Outer edge layers are presented as polygon layers. They represent locations in the layout that print at some point within the process window conditions.

If you want to visualize the inner and outer edges, work with your CAD team to write the code necessary to output these layers to the RDB. Once they are saved to the RDB, view them as described in “[Visualizing PV-Band Data Using Calibre RVE](#)” on page 58.

Related Topics

[AddCustomCheck](#)

[AddCustomOPCVCheck](#)

Metrics to Score a Process or Design

Metrics to help the designer prioritize and understand the need for corrections.

Depending on how you design them, metrics can be used for any of the following:

- Scoring designs to compare against other designs

Compare multiple designs, all simulated using the same optical and process models.
Generate a single score for each design.

- Scoring processes to compare against other processes

Run Calibre LFD many times on the same design, each run simulating with different process conditions, including different RET options. Generate a single score for each Calibre LFD run.

Scoring individual areas of a design for prioritizing layout modifications

- Run Calibre LFD once, partitioning the design into manageable sized areas and generating one score for each area.

Scores can be based on numbers of errors, sizes of errors, sizes of PV-bands or other properties of PV-band layers, error layers, and target layers. You can even write your own metrics to base scores on other layers you supply.

Understanding Standard Metrics.....	63
Designing Custom Metrics	64

Understanding Standard Metrics

The Calibre LFD software package provides two metrics, Design Variability Index (DVI) and Process Variability Index (PVI).

- **Design Variability Index (DVI)**

The DVI score indicates how likely it is the variations in printing negatively impact yield. It represents the proportion of the area under investigation that is problematic. The total area on the error layers divided by the total area. This index is best used by design teams to locate and assign priority to errors for correction.

$$DVI = \sum \frac{AREA(LFDerrors)}{AREA(window)}$$

- **Process Variability Index (PVI)**

The PVI score indicates the degree to which printing is impacted by changes in the process. It represents the ratio of the area of PV-band layer data to the area of target layer data.

$$PVI = \sum \frac{AREA(pvband(layer))}{AREA(layer)}$$

When you perform multiple process variation experiments on the same layer, all the experiments are factored into the scores. Refer to “[Ranking Checks and Errors](#)” on page 66 for complete description of how the individual experiments are factored into the final scores.

Provide the rule writer with the following information:

- The name of the target layer to which the score applies.
- The size and shape of the windows into which the design is partitioned for scoring. Your choices are the entire design under investigation or a regular rectangular window. When you specify a rectangular window, Calibre LFD divides the design into windows of this size, with the lower left corner of the first window located at the lower left corner of the design.
- The pathname for the database to which scores are written.

Note

 Both metrics are calculated using the specified window shape and size. You can generate multiple sets of scores for same layer, each examining a different window size or shape, as long as each set of scores is written to a different database.

Related Topics

[Using PV-Bands to Identify Failure Spots](#)

Designing Custom Metrics

Designing custom metrics involves calculating a score based on statistics for one or more layers. Using a pre-defined set of statistics and operations, you define the equation to use to calculate the score.

Procedure

1. Define the scope of the metric: the cell, an area within the design, or the design itself.

2. Decide which layer(s) are used to calculate the metric:

Regular PV-band	Absolute PV-band
Inside edge of regular PV-band	Inside edge of absolute PV-band
Outside edge of regular PV-band	Outside edge of absolute PV-band
Target layer	Other layers created outside of Calibre LFD
Error layers	Data capture window

3. Check error marker properties.

Note

 The data capture window is a temporary layer containing a single polygon enclosing the area to which the score applies.

4. Write an expression to calculate the score. Expressions must be written using the operators *, /, +, - and the following functions:

Table 3-1. Allowed Expression Functions

Function	Description	Comment
AREA(<i>layer</i>)	Calculates the area of <i>layer</i> polygons inside the data capture window.	Use AREA() to represent area of the data-capture window.
PERIMETER(<i>layer</i>)	Calculates the perimeter of <i>layer</i> polygons in the data capture window.	Use PERIMETER() to represent the perimeter of the data capture window.
COUNT(<i>layer</i>)	Calculates the number of <i>layer</i> objects in the data capture window.	

5. Pass all this information along to the rule writer.

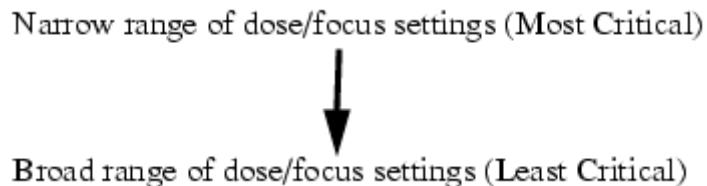
Ranking Checks and Errors

Ranking errors allows you to sort through checks and errors to simplify studying the design and planning modifications.

Built-in Ranking	66
Custom Ranking	67
Error Marker Properties for Ranking	67

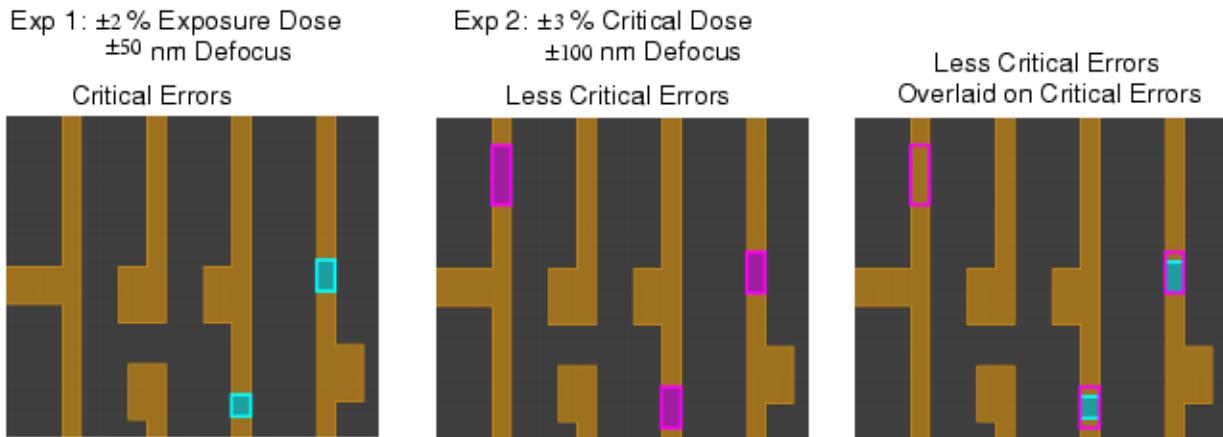
Built-in Ranking

When you run Calibre LFD checks using a single process variation experiment, all you can really tell from the checks results is whether or not a feature is susceptible to printing problems at one or more process conditions tested in the experiment. If you run multiple experiments, each testing a different set of dose, focus and other variability conditions, you can rank the Calibre LFD results according how likely it is for an error to occur.



Ranking of errors occurs automatically, based on the positions of the experiments in the **-opticalSpanList** and **-doseSpanList**. The first experiment in the lists is assigned the highest priority (most critical) and the last experiment in the lists is assigned the lowest priority (least critical).

Figure 3-12. Errors Ranked According to Criticality



In addition to ranking based on position in the experiment list, priority calculations take into account the type of check being performed. By default, the system uses a three-tier ranking system based on check type.

Table 3-2. Default Ranking By Check

Priority	Rank	Checks
High	1	MinSpaceCheck MinWidthCheck NonPrintingCheck Custom Checks
Medium	2	MinAreaOverlapCheck MinOverlayCheck InterLayerSpaceCheck
Low	3	AreaCheck MaxAreaVariabilityCheck

The final priority value is calculated based on the following equation:

$$1000 - (100 * \text{subwindow}) + (11 - \text{rank})$$

Thus, an LFD::MinSpaceCheck call on subwindow 2 has a default priority of 810.

Custom Ranking

For a more granular ranking than provided by the built-in ranking, you can define your own user-defined ranking system and assign each check a different priority using the `-priority` keyword. When you specify a custom ranking, the system writes that value directly to the results database. Experiment and check ranking is ignored.

To use custom ranking, specify an integer value for the `-priority` keyword for each check to be performed.

Error Marker Properties for Ranking

Error markers contain properties, and you can export these markers.

Use the SVRF DFM PROPERTY command to bin the markers as you wish. Then these binned layers can be output using DFM RDB. The following example outputs error markers into categories that “must be fixed”, “are fixed”, and “can be ignored”:

```
LFD::MinWidthCheck -layer poly .... -minLFDwidth 0.06 \
                     -layerOut poly_mwc_all -property MinCD

tvf::VERBATIM "
poly_mwc_mustFix = DFM PROPERTY poly_mwc_all \
                     [MinCD=PROPERTY(poly_mwc_all,MinCD)] <= 0.01
poly_mwc_shouldFix = DFM PROPERTY poly_mwc_all \
                     [MinCD=PROPERTY(poly_mwc_all,MinCD)] > 0.01 <= 0.03
poly_mwc_canbeignored = DFM PROPERTY poly_mwc_all \
                     [MinCD=PROPERTY(poly_mwc_all,MinCD)] > 0.03
"
```

Related Topics

[DFM Property \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DFM RDB \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Chapter 4

For CAD Groups — Writing Calibre LFD Rule Files

The Computer Aided Designers (CAD) group's role in implementing the Litho-Friendly Design flow is to write the process-specific Calibre LFD kits designers use to analyze a design using Calibre LFD rules. The following topics describe how to create process-specific Calibre LFD kits:

What is a Process-Specific Calibre LFD Kit?	69
Writing Rules for Performing Calibre LFD	70
Writing Rules Using TVF	72
Defining the Calibre LFD Block.	75
Calibre LFD Fast Mode	77
Customizable PV-Bands	81
Generating PV-Bands	86
PV-Band and Contour Processing Commands.....	87
Writing the Rule File for Generating PV-Bands	88
Translating Failures into Rules	94
Coding Metrics to Score a Process or Design	97
Accessing PV-Band Data.....	99
Accessing Contour Data	100
About Calibre LFD Databases	102
Saving Calibre LFD Data to a Layout Database	103
Calibre LFD Flow with a PDK.....	105
PDK File Commands.....	107
Writing TVF for use with a PDK.....	108
SVRF Rule File Macros.....	111

What is a Process-Specific Calibre LFD Kit?

A *process-specific Calibre LFD kit* is the set of rule files and models required for applying Calibre LFD practices to a specific design.

The Calibre LFD kit can take either of two forms:

- It can be a stand-alone file, called a Calibre LFD Process Definition Kit, or PDK. Refer to [PDK File Commands](#) for the content of this file.
- It can be a set of files or directories, provided to you as a package.

When the collection of the *process-specific* rules and models required for applying litho-friendly design practices to a specific design is shipped to you as a PDK. PDKs provide many advantages over shipping the rules and models as separate files. From the PDK creator's perspective, the primary advantages are portability and security. From the layout designer's perspective, the primary advantage is ease of use.

No matter which form you use to deliver the Calibre LFD kit, it must contain the following:

Table 4-1. Contents of a Typical Calibre LFD Kit

Kit Contents	Used for...
Calibre LFD Rules	Performing Calibre LFD processing
LITHO setup file data	Generating the PV-bands
Optical models	Generating the PV-bands
Resist models	Generating the PV-bands
OPC rules	Generating the PV-bands
Additional rules	Pre or Post processing

Writing Rules for Performing Calibre LFD

Rule files for performing Calibre LFD work must accomplish several tasks.

1. Generate the OPC-corrected layer data.
2. Use the [PVband](#) function to generate PV-band data.
3. Check PV-band and original design data to identify problem areas on the design (Calibre LFD rules).
4. Analyze the results of Calibre LFD rule checking and write the results to the appropriate databases.

The litho team should provide you with the following:

- All necessary setup files.
- Descriptions of the Calibre LFD rules to write.
- Descriptions of any metrics to write.

- Any rule files used to generate the initial PV-bands they examined when designing the checks and the metrics.

Note

 This chapter describes the basic process of writing a Calibre LFD rule file for a single layer, however rule files for performing Calibre LFD analysis can operate on many layers.

Writing Rules Using TVF

The Tcl Verification Format (TVF) is a programmable expansion to the Standard Verification Rule Format (SVRF) language for Calibre. The rule files used to run Calibre LFD must be written using TVF.

After writing these rule files you have the option to convert them from TVF into encrypted SVRF, as described in “[Encrypting the Calibre LFD Rules](#)” on page 510.

The Calibre LFD functions special TVF commands used only for Calibre LFD. In addition to these functions, you must understand how to use the `tvf::VERBATIM` command, which allows you to include SVRF code in a TVF file. For complete information on TVF refer to “[Tcl Verification Format](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

Note

 Because the Calibre LFD rule file typically incorporates the RET recipes for multiple layers, all layer names and variable names used within the LITHO setup files or rule files must be unique.

SVRF Pre-Processor Directives Compared to TVF Variables and Tcl Constructs ...	72
Avoiding Name Collisions	73
Avoiding Namespace Issues	73
Calibre LFD Function Calls	74
Accessing Variables in TVF Rule Files for Calibre LFD.....	74
Debugging with TVF for the Calibre LFD Rule File	74
Accessing the Calibre LFD Library.....	75

SVRF Pre-Processor Directives Compared to TVF Variables and Tcl Constructs

A rule file written using SVRF can use special pre-processor directives such as `#DEFINE` and `#IFDEF` to control processing based on user-defined variables. You can use Tcl variables and constructs to perform the same functions in a TVF rule file.

There are many restrictions on the use of SVRF pre-processor directives, and they can complicate the task of debugging your rule file. TVF is more flexible because it allows you to use variables as you build the actual SVRF calls to be evaluated. It is recommended that you take advantage of Tcl controller conventions such as if-then-else constructs and avoid SVRF pre-processor directives.

Example of SVRF containing pre-processor directives:

```
...
LAYER myLayer 100
...
#define sizeLayer
...
#ifndef sizeLayer
    myNewLayer = SIZE myLayer BY 1.0
#else
    myNewLayer = COPY myLayer
#endif

myNewLayer {COPY myNewLayer}
```

Example of TVF using TCL controller constructs:

```
...
tvf:::LAYER myLayer 100
...
set sizeLayer 1
...
if {$sizeLayer==1} {
    tvf:::SETLAYER myNewLayer = SIZE myLayer BY 1.0
} else {
    tvf:::SETLAYER myNewLayer = COPY myLayer
}

tvf:::RULECHECK myNewLayer {COPY myNewLayer}
```

Avoiding Name Collisions

Rule files used for performing Calibre LFD incorporate the RET recipes for one or more layers. Each recipe requires its own models, setup file, and SVRF rules. Before including or embedding these files, make sure all layer names and variable names are unique.

Avoiding Namespace Issues

It is important to avoid namespace collisions in Tcl programming.

Namespaces are an important concept in Tcl programming. Essentially, a namespace is a context for commands and variables. Using multiple namespaces lets you re-use names. For example, the name “foo” in namespace *A* is something completely different from the name “foo” in namespace *B*. If you are unfamiliar with namespaces, consider researching the topic within Tcl books or at Tcl websites.

When writing TVF rule files for use in Calibre LFD, be aware that your code is processed by multiple TVF interpreters, and involve multiple namespaces. It is good practice to add the namespace prefix before every command.

Calibre LFD Function Calls

To ensure the Calibre LFD functions are processed correctly, add the prefix “LFD::” to every function call.

For example, to call the function [AreaCheck](#), invoke the function as LFD::AreaCheck:

```
LFD::AreaCheck -layer contact -subwindow 1 -minLFDarea 0.003 \
    -database areaCheck.rdb
```

Accessing Variables in TVF Rule Files for Calibre LFD

There are several TVF commands for accessing variables in the rule file and command line arguments.

The follow commands are among the most useful to the Calibre LFD rule file developer:

- **tvf::get_tvf_arg** — This command lets you access arguments passed from the Calibre command line with the -tvfarg switch. If the command line does not contain the -tvfarg switch, the command returns nil.
- **Global variables** — You can access global variables; see “[Global Variables in Compile-Time TVF](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

Related Topics

[Coding Guidelines for Compile-Time TVF \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[TVF Command Reference \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Debugging with TVF for the Calibre LFD Rule File

You can echo the SVRF commands created from translating the TVF rule file.

tvf::echo_svrf {0 | 1} — Use this command to control reporting of SVRF layer operation calls created from translating TVF code:

1 — Echo calls to stdout.

0 — Disable echoing (default).

Related Topics

[tvf::echo_svrf \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[Error Handling in Compile-Time TVF \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Accessing the Calibre LFD Library

You gain access to Calibre LFD functionality by loading the Calibre LFD package.

To do this, include the following code in your TVF rule file:

```
package require CalibreDFM_LFD
```

Defining the Calibre LFD Block

The Calibre LFD block is the portion of the rule file that calls the Calibre LFD package, and it must be set off from the rest of the rule file. The LFD::Begin function sets off the Calibre LFD processing block, and the LFD::End function signals the end of the Calibre LFD block.

For example:

```
// Begin LFD Block
LFD::Begin
LFD::PVband ...
LFD::MinWidthCheck ...
LFD::CaptureContour ...
LFD::End
// End LFD Block
```

Between the LFD::Begin and LFD::End function calls, the command flow is as follows:

1. Optional configuration functions

LFD::SimConfig — Configures the simulations to be performed as part of Calibre LFD.

2. Optional cell-limiting functions

LFD::StructureOptimizer — Reduces the layout data set being passed into Calibre LFD simulation, improving overall runtime performance.

LFD::LayoutOptimizer — Defines problem areas based on geometric rules.

LFD::FrameCellOptimizer — Performs pre-processing that reduces the area to be analyzed. Removes areas that do not require processing by Calibre LFD as they have already been checked or because they are IP the designer cannot modify.

LFD::TopCellOptimizer — Aids in checking geometries at the top level of the design hierarchy.

LFD::ReadHIF — Reads a HIF file and maps all the polygons from the checks for a given layer onto a single output layer.

LFD::IncrementalSelect — Performs intelligent clipping of one or more input layers based on the marker layers.

LFD::[LFDregion](#) — Defines regions within the layout, which can then be used to limit checking to those areas.

3. Use one of the PV-band generation functions

LFD::[RegisterLayer](#) — Registers a design layer with the Calibre LFD processing engine.

LFD::[RegisterBands](#) — Makes PV-band data generated in a previous run available for analysis in the current run.

LFD::[PVband](#) — Generates PV-band data for the specified layer.

LFD::[Drawn2Contour](#) — Generate empirically calibrated PV-bands using Square Directional Kernels (SDK) when access to the exact OPC recipe and models are not available.

4. Checks

See the section “[Calibre LFD Checks](#)” on page 181 for information on the Calibre LFD check functions.

5. Scoring functions

LFD::[VariabilityIndices](#) — Generates variability indexes (scores) for the specified layer.

6. Functions to output contours

LFD::[OutputBands](#) — Writes PV-band data to the specified database.

LFD::[CaptureContour](#) — Stores a previously generated contour as a derived layer.

Calibre LFD Fast Mode

For rapid execution, Calibre LFD enables you to restrict checking to specific layout areas. Creating OPC layers is a time-consuming process, however, with LFD Fast Mode kits only regions of interest are subjected to OPC operations. Regions of interest can be defined by pattern matching, some SVRF statements, or by marking layers.

To improve execution time using pattern matching, Calibre LFD fast mode uses the Calibre Pattern Matching software to identify only those layout areas that require detailed lithographic simulation. These areas are sent through a full LFD run, and remaining areas are not simulated. Calibre pattern matching is the process of locating specific polygons in a layout from a pattern you specify. This approach of reducing the area of the design that is verified, by translating hotspots into patterns, significantly speeds up the LFD tool, as simulation occurs on a subset of the original design.

Calibre LFD and Pattern Matching.....	77
Functions Limiting Calibre LFD Cell Checks	78
Limiting Calibre LFD Simulation and Checks to Specific Regions	79

Calibre LFD and Pattern Matching

You use the pattern matching Pattern Library Manager GUI and the pattern matching batch utility, `pdl_lib_mgr`, to create and edit pattern libraries. You use the pattern matching Tcl-based API, `pdl_lib_mgr` prompt, to edit and traverse pattern libraries and the pattern objects within them.

The pattern matching GUI and the pattern matching `pdl_lib_mgr` batch utility are the subject of the *Calibre Pattern Matching User's Manual*. The pattern matching API interface is the subject of the *Calibre Pattern Matching Reference Manual*.

Use of the pattern matching GUI to create and edit pattern libraries can be time-consuming for large numbers of patterns, however the pattern matching API can be used to edit or search thousands of patterns automatically. The pattern matching API provides a group of commands that can be combined together in a Tcl script to retrieve information about the content of patterns. The output of the pattern matching API commands is used to create a smaller subset of the pattern library that the LFD::[StructureOptimizer](#) command uses for running pattern matching. This is followed by simulation of the design on the patterns for hotspot verification.

Collections of patterns are stored as pattern libraries in a Siemens EDA-specific format known as Pattern Matching Database (PMDB). The PMDB file is compiled into an SVRF file, referred to in LFD::[StructureOptimizer](#) command calls. Once a foundry identifies potential problematic structures as possible lithographic hotspots, a hotspot library can be developed to capture these areas of interest.

Note

 Use of the pattern matching software to create and edit pattern libraries, as well as use of the pattern matching API to edit and traverse pattern libraries are separate flow steps that are independent of Calibre LFD. They are used for managing the pattern library and are not part of any LFD runs. You run pattern matching before you simulate the design.

Functions Limiting Calibre LFD Cell Checks

Optional Calibre LFD functions can limit the cells that are checked by the tool. By pre-filtering the cells that are simulated, performance can be improved dramatically.

These optional cell-limiting functions can reduce the data set prior to simulation:

- LFD::[StructureOptimizer](#) — Runs pattern matching in the design to find matches for patterns containing possible lithographic hotspots. The function enables you to reduce the layout data set being passed into Calibre LFD simulation if the problematic areas are known. The function provides a rapid method for evaluating a hotspot library in new databases under test, thus significantly reducing the runtime of Calibre LFD checks.
- LFD::[LayoutOptimizer](#) — Searches through designs for potential problematic areas based on geometrical rules defined in an LFD::[Macro](#). The function allows the designer to select hotspot candidate structures within a design that correspond to a pre-defined structure within a library. When layout optimization is complete, the selected edge collections a reduced subset of edges derived from the full chip) are passed into Calibre LFD for simulation.
- LFD::[FrameCellOptimizer](#) — Removes user-specified cells from the layout, maintaining context data and cell intruding shapes for correct simulation. The function performs pre-processing that reduces the area to be analyzed using Calibre LFD by removing areas that do not require processing by Calibre LFD because they have already been checked or because they are IP the designer cannot modify.
- LFD::[TopCellOptimizer](#) — Looks at all metal outside of the specified cells and applies the appropriate simulation to the surrounding areas. The function aids in checking geometries at the top level of the design hierarchy by limiting Calibre LFD execution to layout features at the top cell level. Typically, designers at the place and route stage in the design flow cannot control edit IP design block and standard cell geometries. However, at this point designers do have control over how the router places connecting shapes. Therefore, using the LFD::TopCellOptimizer function allows designers to run Calibre LFD on what the router has already laid down.
- LFD::[ReadHIF](#) — Reads HIF-formatted check files generated by one of the Cadence routing tools. The check file describes regions modified by the router. The function reads the check file, creates marker layers from the areas specified in the file, and then produce layers from the original layers that include only the design data inside the marker layers regions.

- LFD::[IncrementalSelect](#) — Performs intelligent clipping of one or more input layers based on user-supplied marker layers. The function is used to ensure accurate processing of selected portions of the design by evaluating neighboring features to see whether or not they have lithographic impact on features within the areas of interest. Neighboring features that are lithographically significant are included in the output layers, along with the features in the areas of interest.
- LFD::[LFDregion](#) — Defines regions within the layout, which can then be used to limit checks to those areas. These regions apply to all PV-bands created, regardless of how many times the LFD::[PVband](#) function is issued. See the following section for more information.

Limiting Calibre LFD Simulation and Checks to Specific Regions

You can use the LFD::LFDRegion command to restrict LFD checking to specific areas in the layout.

By default, Calibre LFD operates on the entire layer you pass to the LFD::[PVband](#), LFD::[RegisterLayer](#), or LFD::[RegisterBands](#) functions. The LFD::[LFDregion](#) function restricts checking to specific areas in the layout.

This command uses a region layer to identify the areas to be simulated and checked. The region layer contains one or more polygons that enclose the portions of the layout to be evaluated with Calibre LFD. Since the rule file you are writing must work in both batch and interactive modes (invoked from the command line, or through Calibre Interactive) you should define a standard layer number and layer name for this layer and document it for use by the designer.

When generating PV-bands, the area that is actually checked is the specified area plus a halo region around the area. The halo ensures that any nearby geometries that affect printing are factored into the calculations. You specify the halo size as part of the LFD::LFDregion command. This is typically 1/2 the optical diameter specified in the model.

Restrictions and Limitations

- You can issue the LFDregion command at most one time per layer.
- When issued, the command limits all checking performed during a specific run.
- This command must be placed above any calls to the LFD::PVband function.

Procedure

1. Read in the region layer.

```
layer check_region 117
```
2. Calculate the appropriate halo size.

3. Add the LFD::LFDregion command, referencing this layer:

```
LFD::LFDregion -region check_region -halo halo_size
```

Customizable PV-Bands

The Calibre LFD Customizable PV-bands interface provides you with flexibility in generating simulation contours and PV-bands, and the ability to use your custom-generated contours with the Calibre LFD Checks.

This flexibility is useful at and below the 22 nm process node, where OPC and RET techniques differ from layer to layer. You can use this interface to:

- Create custom contours.
- Apply logical operations on simulated contours.
- Perform LFD checks on these contours.
- Create a PV-band or overlay on these contours.

This customizable PV-bands interface uses the following commands:

- LFD::[Contour](#)
- LFD::[Band](#)
- LFD::[OverlayBand](#)

Creating and Using Contours with LFD::Contour	81
Creating and Using PV-Bands with LFD::Band	82
Creating and Using Overlay Bands with LFD::OverlayBand.....	83
Calibre LFD Check Options for Customizable PV-Bands and Contours	83
Customizable PV-Bands Using a PDK Flow.....	84

Creating and Using Contours with LFD::Contour

You can create contours using the LFD::Contour command. The interface also allows you to use the created contours in creating other contours, PV-bands, or in applying checks to them.

Each contour represents a single Calibre® OPCverify™ operation, such as:

- Logical operations
- The shift operation
- The size operation
- The simulation operation

You specify a handle for each defined contour. This handle must be unique per layer. For example, the layer poly can have only one contour handle c1, but both layer poly and layer active may have a contour handle c1.

You define the required input layers for each operation. The input layers for each operation must be one of the following:

- Original GDS layers
- Derived layers
- Previously-defined contour handles

For operations that require arguments, you must specify the needed input arguments of the operation. For example, the xshift and yshift value in the shift operation, and the sizeValue in the case of the size operation.

For operations that require input models, you must specify the full path to those models. Models supported by the interface are these:

- Optical models
- Resist models
- Etch models
- DDM models

For simulation operation and user-defined operation, you must specify a definition of those operations. This definition is treated as a template with a placeholder for layers and models. The placeholders are later replaced with the corresponding model and layer names.

Related Topics

[Contour](#)

Creating and Using PV-Bands with LFD::Band

You can create any number of PV-bands using the LFD::Band command. A PV-band is defined with two input contours, its minimum PV-band edge, and its maximum PV-band edge. These contours are either original or derived layers, or contour handles generated using the LFD::Contour command.

Given two input contours, Calibre LFD generates the PV-band and absolute PV-band layers as follows:

- BandLayer = NOT max_band_edge min_band_edge
- AbsBand layer = and layer BandLayer

You can use the PV-bands generated with this command with LFD checks.

Each PV-band generated with the LFD::Band command has a unique handle per layer. If this handle is given a value of “default”, this PV-band is treated as if it was created using the LFD::PVband command.

Creating and Using Overlay Bands with LFD::OverlayBand

You can create any number of overlay PV-bands using the LFD::OverlayBand command. Overlay PV-bands represent a shifted version of an original PV-band. For example: if *band1* consists of the contours {c1 c2 c3}, a shifted version of this PV-band in the north direction consists of {c1_N c2_N c3_N} where c1_N, c2_N and c3_N are the shifted versions of the contour in the north direction.

The overlay PV-band is attached to an original PV-band. If you defined the 8 shifted PV-bands for an original PV-band, the check performs the same way it performs when the -overlay option is defined using the LFD::PVband command. In other words, the check is performed on the eight shifted PV-bands and the result is the OR of the results of the eight checks.

The overlay PV-band is the maximum and the minimum edges of the PV-band. The PV-band layer and the absolute PV-band layer are generated by Calibre LFD package:

- OverlayBandLayer = NOT max_band_edge min_band_edge
- OverlayAbsBand layer = and layer BandLayer

The input “maximum” and “minimum” contours can be either contour handles defined earlier in the Calibre LFD kit, or an original layer name.

Related Topics

[OverlayBand](#)

Calibre LFD Check Options for Customizable PV-Bands and Contours

Calibre LFD check options can also be used to perform Calibre LFD checks on specific PV-band or contour handles.

Table 4-2. Using Handles with Calibre LFD Checks

Check	Options
LFD::AddCustomOPCVCheck	-layerContoursHandle -layer1ContoursHandle -layer2ContoursHandle

Table 4-2. Using Handles with Calibre LFD Checks (cont.)

Check	Options
LFD::AreaCheck	-contourHandle
LFD::AreaOverlayCheck	-contour1Handle -contour2Handle
LFD::EndCapCheck	-contour1Handle -contour2Handle
LFD::InterLayerSpaceCheck	-contour1Handle -contour2Handle
LFD::LineEndCheck	-contourHandle
LFD::MinAreaOverlapCheck	-contour1Handle -contour2Handle
LFD::MinOverlayCheck	-contour1Handle -contour2Handle
LFD::MinSpaceCheck	-contourHandle
LFD::MinWidthCheck	-contourHandle
LFD::NonPrintingCheck	-contourHandle
LFD::ViaWidthCheck	-contourHandle
LFD::CaptureContour	-contourHandle
LFD::OutputBands	-bandHandle
LFD::MaxAreaVariabilityCheck	-maxContourHandle -minContourHandle

Customizable PV-Bands Using a PDK Flow

The customizable PV-bands functionality is supported in the Calibre LFD Process Design Kit (PDK) flow. Each addPDKLayer call in the PDK defines a layer template, and the information in that template is used in generating OPC output, simulated contours, and checks. You can define contours, PV-bands and overlay PV-bands for each subwindow in the addPDKLayer PvBands section.

The LFD::Contour, LFD::Band, and LFD::OverlayBand commands can be used in the addPDKLayer as follows:

```
addPDKLayer {  
    ...  
    PvBand {  
        subwindow1 {  
            contour { ... }  
            contour { ... }  
            contour { ... }  
            band { ... }  
        }  
    }  
}
```

Input models passed to simulation contours or user-defined contours use a model handle defined earlier in the PDK.

Input Layers to contours is either:

- Contour handle previously defined in the PDK.
- Placeholder of a layer defined in the PDK.

Generating PV-Bands

PV-bands can be obtained in several ways, depending on whether you have a PDK or have an existing PV-band.

Generating PV-Bands Without a PDK	86
Generating PV-Bands by Reusing a PV-Band	86
Generating PV-Bands with a PDK.....	87

Generating PV-Bands Without a PDK

Certain commands are used when you generate a PV-band without a PDK.

The LFD::[PVband](#) or LFD::[Drawn2Contour](#) command generates PV-band data for a specified layer. In addition, you can generate custom PV-bands using the LFD::[Band](#) command.

For example:

```
LFD::Begin
source layersfile.lyr
LFD::PVband ...
LFD::MinSpaceCheck -pdkCheckName Metall1_MSC_p1 -layer metall1 \
    -database rdb/lfd_checks.rdb
LFD::End
```

Generating PV-Bands by Reusing a PV-Band

You use the LFD::RegisterBands command when you want to reuse a PV-band.

The LFD::[RegisterBands](#) command makes PV-band data generated in a prior run available for analysis in the current run.

For example:

```
LFD::Begin
source layersfile.lyr
LFD::LoadPDK lfd/pdkfile.pdk
LFD::RegisterBands -layer metall1 -subwindow 1 -maxBandEdge out_m1 \
    -minBandEdge inner_m1
LFD::MinSpaceCheck -pdk lfd_pdk -pdkCheckName Metall1_MSC_p1 \
    -layer metall1 -database rdb/lfd_checks.rdb
LFD::End
```

Related Topics

[RegisterContour](#)

Generating PV-Bands with a PDK

Use the LFD::LoadPDK and LFD::RegisterLayer commands to generate PV-bands when you have a PDK.

The LFD::LoadPDK command loads the specified PDK and makes it available to Calibre LFD command calls. The LFD::RegisterLayer command registers a design layer with the Calibre LFD processing engine.

If you use a PDK, you do not need to call the LFD::PVband function directly.

For example:

```
LFD::Begin
source layersfile.lyr
LFD::LoadPDK lfd/pdkfile.pdk
LFD::RegisterLayer -pdk lfd_pdk -Designlayer metal1 \
    -Processlayer pdklyr_metal1
LFD::MinSpaceCheck -pdk lfd_pdk -pdkCheckName Metal1_MSC_p1 \
    -layer metal1 -database rdb/lfd_checks.rdb
LFD::End
```

PV-Band and Contour Processing Commands

After generating PV-bands, the contours and PV-bands are processed. This can include hotspot checks, scoring, contour output, and model-based extraction.

The contour and PV-band processing functions are placed before the LFD::End function.

- **Hotspot checks**

See “[Calibre LFD Syntax Descriptions](#)” on page 175 for information on checks.

- **Scoring commands**

LFD::[VariabilityIndices](#)

- **Commands to output contours**

LFD::[OutputBands](#)

LFD::[CaptureContour](#)

- **Model-based extraction commands**

LFD:[CSG](#)

LFD:[CSI](#)

Writing the Rule File for Generating PV-Bands

Several steps are required for writing the rule file to generate PV-Bands.

Requirements for the PVband Command	88
Setting Up the Rule File	88
Specifying Correct Layer Data to Generate PV-Bands	90
Describing Process Variation Experiments	91

Requirements for the PVband Command

You generate PV-band data using the LFD::PVband function.

The LFD::PVband function requires the following input:

- Two input layers, the original layer, and the OPC-corrected layer. The rule file for generating the PV-bands must generate the OPC-corrected layer on the fly. Typically, running OPCpro to generate the OPC-corrected layer for Calibre LFD is identical to a standard OPCpro run.
- The path to the models directory, which contains the optical model and process models used for simulation.
- Descriptions of the process variation experiments to perform.

Setting Up the Rule File

Some basic commands are required for all rule files in Calibre LFD.

Prerequisites

- A basic understanding of TVF and SVRF rule writing.
- Access to the rule file used for implementing the RET recipe for the target process.

Procedure

1. Create the basic TVF rule file with all the required standard code for TVF. This should define the inputs (such as LAYOUT SYSTEM and LAYOUT PATH) and outputs (such as DRC RESULTS DATABASE and DRC SUMMARY REPORT)

2. Copy the *namespace import* and *Load Calibre LFD package* code from the sample Calibre LFD rule file provided in “[Sample Calibre LFD Rule File and PDK](#)” on page 519. Paste this code after the opening line (#! tvf) of the new TVF file:

```
#=====
# Load LFD package
    package require CalibreDFM_LFD*
#=====
```

3. Add the LFD::Begin and LFD::End statements to define the Calibre LFD block within the rule file.
4. Copy the example [PVband](#) function from the sample Calibre LFD file provided in “[Sample Calibre LFD Rule File and PDK](#)” on page 519 and paste it into the new file, between the LFD::Begin and LFD::End statements. Repeat for each layer to be investigated.

```
//=====
// Begin LFD Block
//=====
LFD::Begin
LFD::PVband  "-layer metall1_target \
               -layerRET lfd_metal1_mopc \
               -foreground clear \
               -background {attenuated 0.06} \
               -pixel 0.02 \
               -modelDir {./metall1/models} \
               -resistFile {metall1.mod} \
               -opticalSpanList {{P1W_D0 P1W_D0 P1W_D0 \
                                 P1W_D50 P1W_D50 P1W_D50} \
                                 {P1W_D0 P1W_D0 P1W_D0 P1W_D100 P1W_D100 P1W_D100}} \
               -doseSpanList {{0.9500 1.0000 1.0500 \
                               0.9500 1.0000 1.0500} \
                               {0.9500 1.0000 1.0500 0.9500 1.0000 1.0500}}}"
LFD::End
//=====
// End LFD Block
//=====
```

5. Modify the command arguments as shown below. Consult the [PVband](#) function syntax description, if needed.

-pixel — Set to approximately 1/3 the line width.

-modelDir — The parent directory for the models created in “[Creating Optical Models](#)” on page 57.

-resistFile — VT5 resist model represents the resist behavior for the target process.

6. Set all other arguments as discussed in these topics:

- “[Specifying Correct Layer Data to Generate PV-Bands](#)” on page 90

- “[Describing Process Variation Experiments](#)” on page 91

Specifying Correct Layer Data to Generate PV-Bands

To generate PV-band data, you need a minimum of two layers.

- The target layer, containing the shapes you want to print on the wafer.
- The OPC-corrected layer.

You may also supply additional layers containing scattering bars or other features required for your RET recipe.

By default, the application operates on the entire layer, as passed to the [PVband](#) function. For information on restricting Calibre LFD checking to one or more regions within the layer, refer to “[Limiting Calibre LFD Simulation and Checks to Specific Regions](#)” on page 79.

Procedure

1. Before the Calibre LFD block, add the code for generating the OPC-corrected layer data. If the rule file used for implementing the RET recipe for the target process is written in TVF, you can incorporate it into the Calibre LFD rule file directly. If it is written in SVRF, consider using an INCLUDE statement in the TVF file or enclose it within the VERBATIM command.
2. Save the results of LITHO processing as a derived layer. For example:

```
mopcPoly = LITHO OPC FILE "./setup/poly_opc.in" poly
```

Typically, you do not need to write this layer to the output layout database (GDS or OASIS). If you need to write this layer to a file, you must COPY this derived layer to a Calibre rule check.

3. Modify the layer related arguments to the LFD::[PVband](#) functions:
 - Supply the original design layer using the **-layer** keyword. This layer served as the target layer when generating the opc-corrected layer and serves as the target layer when evaluating Calibre LFD errors.
 - Supply this layer using the **-layerRET** keyword.
 - Define the optical transmission values for the features on this layer using the **-foreground** keyword.
 - Define the optical transmission value for the background this layer using the **-background** keyword.

4. If you need to supply additional external layers for scattering bars or other required features:
 - Add the layer names to the argument for the **-layerRET** keyword. This means you supply a Tcl list of layers that includes the OPC-corrected layer as one layer plus all other required layers. Enclose this list in a set of braces ({}).
 - Add the transmission values to the argument the **-foreground** keyword. This also becomes a Tcl list. Be sure to arrange the values such that the first transmission value represents the first layer in the list, the second represents the second, and so on. Enclose this list in braces ({}).
 - Make sure the optical transmission value specified using the **-background** keyword works for all the layers in the list of layers.

Describing Process Variation Experiments

You issue the LFD::PVband function once for each layer. This one function instructs Calibre LFD to perform all the process variation experiments designed for that layer.

To use a single invocation to instruct Calibre LFD to process multiple experiments, you must pass in descriptions of the process variation experiments in the form of opticalSpanLists and doseSpanLists, which are arguments to the LFD::PVband function. When you need to apply a bias to the original geometry during the PV-band generation step, you must also supply the optional sizeSpanLists.

These lists contain one value for each focus and dose pair:

- The opticalSpanList contains the model created for the specified focus.
- The doseSpanList contains the dose.
- The sizeSpanList contains the size of the bias.
- The resistSpanList contains the resist model.
- The etchSpanList contains the etch model.

Note

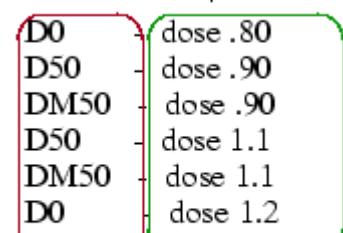
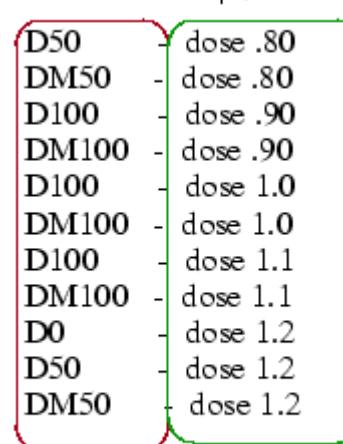
 Experiments involving double exposures require two opticalSpanLists, two doseSpanLists, and two sizeSpanLists.

Procedure

1. For each process variation experiment, create a list of focus and dose combinations to be investigated.

Experiment 1 or Subwindow 1	Experiment 2 or Subwindow 2	Experiment 3 or Subwindow 3
defocus 0 - dose 1.0	defocus 0 - dose .80	defocus 50 - dose .80
defocus 50 - dose 1.0	defocus 50 - dose .90	defocus -50 - dose .80
defocus -50 - dose 1.0	defocus -50 - dose .90	defocus 100 - dose .90
	defocus 50 - dose 1.1	defocus -100 - dose .90
	defocus -50 - dose 1.1	defocus 100 - dose 1.0
	defocus 0 - dose 1.2	defocus -100 - dose 1.0

2. In your lists, replace each defocus setting with the name of the optical model created using that setting. This is shown in the table in Step 3.
3. Create one opticalSpanList and one doseSpanList for each experiment. Be sure to keep the orders the same in both lists.

Experiment 1 or Subwindow 1	Experiment 2 or Subwindow 2	Experiment 3 or Subwindow 3
<p>doseSpanList1</p>  <p>D0 dose 1.0 D50 dose 1.0 DM50 dose 1.0 D0 dose .90 D0 dose 1.1</p> <p>opticalSpanList1</p>	<p>doseSpanList2</p>  <p>D0 dose .80 D50 dose .90 DM50 dose .90 D50 dose 1.1 DM50 dose 1.1 D0 dose 1.2</p> <p>opticalSpanList2</p>	<p>doseSpanList3</p>  <p>D50 dose .80 DM50 dose .80 D100 dose .90 DM100 dose .90 D100 dose 1.0 DM100 dose 1.0 D100 dose 1.1 DM100 dose 1.1 D0 dose 1.2 D50 dose 1.2 DM50 dose 1.2</p> <p>opticalSpanList3</p>

You only need to define new experiments in experiment 2 and 3 span lists, unless using the LFD::PVband -independentWindows option.

- opticalSpanList1 — {D0 D50 DM50 D0 D0}
- doseSpanList1 — {1.0 1.0 1.0 .90 1.1}

- opticalSpanList2 — {D0 D0 D50 DM50 D0 D50 DM50 D0 D50 DM50 D0}
doseSpanList2 — {.80 .90 .90 .90 1.0 1.0 1.0 1.1 1.1 1.1 1.2}
 - opticalSpanList3 — {D0 D50 50 D0 D50 DM50 D100 DM100 D0 D50 DM50
D100 DM100 D0 D50 DM50 D100 DM100 D0 D50 DM50}
doseSpanList3 — {.80 .80 .80 .90 .90 .90 .90 1.0 1.0 1.0 1.0 1.0 1.1 1.1
1.1 1.1 1.2 1.2 1.2}
4. Combine the opticalSpanLists to create the Tcl list of lists that is the argument for the -opticalSpanList keyword. Be sure to enclose the list of lists in braces ({}):
- ```
-opticalSpanList {{D0 D50 DM50 D0 D0} {D0 D0 D50 DM50 D0 D50 DM50 D0
D50 DM50 D0} {D0 D50 50 D0 D50 DM50 D100 DM100 D0 D50 DM50 D100 DM100
D0 D50 DM50 D100 DM100 D0 D50 DM50}}
```
5. Combine the doseSpanLists to create the list of lists that is the argument for the -doseSpanList keyword. Be sure to enclose the list of lists in braces ({}):
- ```
-doseSpanList {{1.0 1.0 1.0 .90 1.1} {.80 .90 .90 .90 1.0 1.0 1.0  
1.1 1.1 1.1 1.2} {.80 .80 .80 .90 .90 .90 1.0 1.0 1.0 1.0 1.0 1.0  
1.0 1.1 1.1 1.1 1.1 1.2 1.2 1.2}}
```
6. If sizeSpanLists are required:
- Return to step 1, and define the bias to be applied for each focus and dose setting in each experiment.
 - Create one sizeSpanList for each experiment.
 - Combine the sizeSpanLists to create the list of lists that is the argument for the -sizeSpanList keyword. Be sure to enclose the list of lists in braces ({}):

```
-sizeSpanList {{0.001 0.001 0.001 -0.002 0.001} {-0.001 -0.002  
-0.002 -0.002 0.001 0.001 0.001 0.001 0.001 0.002} {-0.001  
-0.001 -0.001 0.001 0.001 -0.002 -0.001 0.001 0.001 0.001 0.001  
0.001 0.001 0.001 0.001 0.000 0.000 0.000 0.000 0.002 0.002  
0.002}}
```

Translating Failures into Rules

Calibre LFD failure rules should identify those areas where variation in printing due to process variation is likely to cause critical errors. The lithographer should provide you with the graphical, mathematical, or verbal descriptions of the rules (checks) you need to write.

You can write Calibre LFD rules using either the [Built-in Check Functions](#) or using standard Calibre nmDRC statements to begin [Writing Custom Checks](#).

The results from Calibre LFD checks can produce so many errors markers that they can be difficult to review and understand. Three optional keywords, available both for the built-in checks and custom checks, can help you to write checks that produce more readable results:

-checkName — Assigns a user-defined name to the rule check. For best results design names that are easily readable and fully describe the check that was performed.

-priority — Assigns a priority to the check. Assign a higher value to errors that catch the most critical checks. Designers can sort checks by this value and examine the most critical problems first.

-comment — Provides information about the check. Use whenever possible to describe the problem and suggest steps for correcting the problem.

When these keywords are not supplied, the application uses [Default Priorities](#) and comments, plus [System-Generated Check Names](#).

Built-in Check Functions	94
Writing Custom Checks	95

Built-in Check Functions

The built-in check functions provide a quick way to check for the PV-band configurations that are most commonly considered to be problematic.

The built-in check functions are listed in the section “[Calibre LFD Checks](#)” on page 181.

To use these checks, you must specify:

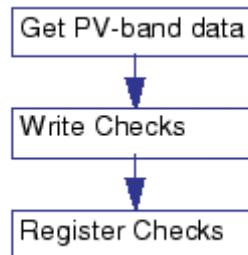
- The target layer under investigation. When checks examine interactions between multiple layers, you must specify all layers involved.
- The process variation experiment these checks relate to. You reference individual process variation experiments by their positions in the ordered lists of lists supplied for the **-opticalSpanList** and **-doseSpanList** arguments to the [PVband](#) function. For more information, refer to “[Describing Process Variation Experiments](#)” on page 91.
- [The Check Database](#) to which you want error markers and scores written.

- The values to check for. These vary according to the check.
- If you plan to write error data to a layout database or perform additional processing on the errors a check finds, you must save the data as a derived layer, using the **-layerOut** keyword.

Writing Custom Checks

Writing custom checks allows you to evaluate PV-band data for specific situations or characteristics beyond those provided through the built-in checks.

The basic process is as follows:



Procedure

1. Identify the PV-band data that is of interest.

The **PVband** function generates six derived layers, all of which are available to you for designing custom checks.

Table 4-3. Accessing PV-Band Data

Layer Content	Access Through	See...
Regular PV-band	OutputBands	Accessing PV-Band Data
Inner edge of the regular PV-band	CaptureContour	Accessing Contour Data
Outer edge of the regular PV-band	CaptureContour	Accessing Contour Data
Absolute PV-band	OutputBands	Accessing PV-Band Data
Inner edge of the absolute PV-band	CaptureContour	Accessing Contour Data
Outer edge of the absolute PV-band	CaptureContour	Accessing Contour Data

All PV-band data is present as polygon layers. Inner edge layers represent locations in the layout that always print, across all process window conditions. Outer edge layers represent locations in the layout that print at some point within the process window conditions.

2. Use the **-layerOut** keyword with [OutputBands](#) or [CaptureContour](#) to save the desired PV-band data as a derived layer.
3. Manipulate these PV-band layers as needed (just as you manipulate any other derived polygon layers). For example, you can combine them using Boolean operations, isolate specific portions using user-supplied island layers, and so on. Island layers define areas of special interest or requiring special treatment.
4. Write custom checks using standard Calibre nmDRC operations, such as EXTERNAL, INTERNAL, and ENCLOSURE. Use these operations to examine properties of these PV-band layers or explore relationships between sets of PV-band layers or between PV-band layers and other layers (original or derived). Output results in the form of error markers (polygons).

Note

 Within the TVF rule file, all code used for preparing data or performing the actual checks must be supplied in the form of a VERBATIM block.

5. Use [AddCustomCheck](#) to register the each check with the Calibre LFD engine so results are factored into variability indices and written to the Calibre LFD output database. You must supply the following information:
 - The name of the derived layer containing the error markers resulting from the custom check.
 - The name of the layer for which the PV-bands are generated.
 - The experiment (subwindow) for which the PV-bands are generated.

Examples

Generate error markers for those areas where the target and PV-bands do not overlap.

```
VERBATIM "
// Begin custom check
targetProcessBandMismatch_poly = lfd_poly_Band_p1 NOT targetBand_poly
Mismatch_poly_check = Extent targetProcessBandMismatch_poly

LFD::AddCustomCheck -layer poly
    -customCheck Mismatch_poly_check
    -subwindow 1
    -checkName Mismatch_poly_check
    -priority 900
    -comment "target and PV-bands do not overlap"
    -database $lfd_results"
// End custom check
"
```

Coding Metrics to Score a Process or Design

The lithographer should provide you with a complete description for each of the metrics to be created. You can use standard metrics or custom metrics.

Using Standard Metrics.....	97
Custom Metrics	98

Using Standard Metrics

The Calibre LFD software package provides two metrics, Design Variability Index (DVI) and Process Variability Index (PVI).

The [VariabilityIndices](#) command generates two types of scores:

$$PVI = \sum \frac{AREA(pvband(layer))}{AREA(window)}$$

$$DVI = \sum \frac{AREA(LFDerrors)}{AREA(window)}$$

Procedure

1. Include the [VariabilityIndices](#) command once for every set of scores to be generated.
2. Specify the size and shape of the windows into which the design is partitioned for scoring using the **-windowSize** keyword:
 - To calculate a single score for the entire area under investigation, set **-windowSize** to “extents”.
 - To calculate scores for regular square areas, set **-windowSize** to the length of the edge of the square.
 - To calculate scores for regular rectangular areas, set **-windowSize** to a pair of values, the first represents the width, the second represents the height. When specifying two values, you must enclose them in braces ({}).
3. Specify the pathname for [The Check Database](#) to which scores are written, using the **-database** keyword. Tcl variables are allowed.

Note

 You can supply multiple [VariabilityIndices](#) commands for a single layer, as long as each one writes the scores to a different database.

Custom Metrics

Custom metrics take the form of rulechecks that use the DFM ANALYZE and DFM PROPERTY operations to calculate a score based on statistics for one or more layers. Within the TVF rule file, all code used for preparing data or calculating the scores must be supplied in the form of a VERBATIM block.

- Make sure all the layers needed to calculate the score exist. If necessary, derive additional layers using Boolean operations, isolate specific portions using user-supplied island layers, and so on.
- Within the rulecheck, use DFM ANALYZE or DFM PROPERTY commands to calculate the scores. DFM ANALYZE is recommended when calculating properties in a shifting window, while DFM PROPERTY is recommended when calculating properties for individual error markers.
- Pass all layers used to calculate the metric to the DFM ANALYZE or DFM PROPERTY command.
- Expression used to calculate the score can be written using the operators *, /, +, - and the following functions:

Table 4-4. Expression Functions

Operation	Input Layer Type	Function	Specification <i>Without layer</i>
AREA(<i>layer</i>)	original or derived polygon	Calculates the area of <i>layer</i> polygons inside the data capture window.	AREA() returns area of the data-capture window.
PERIMETER(<i>layer</i>)	original or derived polygon	Calculates the perimeter of <i>layer</i> polygons in the data capture window.	PERIMETER() returns the perimeter of the data capture window.
COUNT(<i>layer</i>)	derived polygon, derived edge, original error	Calculates the number of <i>layer</i> objects in the data capture window.	Compiler error.

- Be sure to specify a results database (RDB) to which the scores are written, using the keyword RDB or the DFM RDB command. Technically, this is not required, however, if you omit the RDB, you are not able to inspect the scores in map or histogram form using the Calibre RVE graphical debug program.

Note

 Because each custom metric is created with a unique name (the name of the rulecheck) you can write all custom metrics to the same RDB.

Examples

Assume that there are areas that refer to critical regions (ProcessBandFailure_poly) and they should have a larger weight in the variability index.

```
customMetric_poly {DFM ANALYZE targetBand_poly
    targetProcessBandMismatch_poly ProcessBandFailure_poly
    [ (AREA(targetProcessBandMismatch_poly) +
        30*AREA(ProcessBandFailure_poly))/AREA(targetBand_poly)] >= 0
    RDB customCheck2.rdb}

// or

PolyErrors_w_customMetric = DFM PROPERTY targetBand_poly \
    targetProcessBandMismatch_poly ProcessBandFailure_poly \
    [ customeMetric = [(AREA(targetProcessBandMismatch_poly)+\
        30*AREA(ProcessBandFailure_poly))/AREA(targetBand_poly)] >= 0
```

The metric calculates a weighted average using a formula that multiplies the area of failures inside the critical regions by 30, and the area of normal failures by one.

Accessing PV-Band Data

You must use the LFD::OutputBands command in order to have access to PV-band data for later processing or saving to a database.

The [PVband](#) function generates two types of PV-bands:

- Regular PV-bands
- Absolute PV-bands

By default, this data is not made available to you for further processing or writing to an output database. To gain access to this data, you must use the [OutputBands](#) function.

Procedure

1. Issue the [OutputBands](#) function once for every PV-band to be saved.
2. Specify the name of the target layer used to generate the contour through the **-layer** keyword.
3. Specify the name of the derived layer through the **-layerOut** keyword.

4. Specify the experiment for which the PV-band was generated through the **-subwindow** keyword.
5. If you are only interested in the PV-bands for areas where Calibre LFD checks identified problems, also supply the **-errorFilterSize** keyword with the size_filter set to the amount by which you want to grow the existing error markers.

Note

 When you use **-errorFilterSize**, the operation returns PV-band data within all the error markers associated with the specified layer and experiment. This includes error markers generated by any built-in checks you used plus and custom checks that you registered with Calibre LFD using the [AddCustomCheck](#) function.

Accessing Contour Data

If you want to perform additional processing or checking that involves individual contours generated by Calibre LFD, you must first save those contours as derived layers. You do this using the `LFD::CaptureContour` function.

You issue this function once for every contour to be saved. You can save these contours:

- Inner edge of the regular PV-band (minimum contour).
- Outer edge of the regular PV-band (maximum contour).
- Printing contour for a specific focus and dose condition.
- Inner edge of the absolute PV-band (minimum contour).
- Outer edge of the absolute PV-band (maximum contour).

The contours you save exist in memory only, as derived layers. Refer to “[Saving Calibre LFD Data to a Layout Database](#)” on page 103 for instructions for saving these contours to an output database.

Procedure

1. Issue the [CaptureContour](#) function once for every contour to be saved.
2. Specify the name of target layer used to generate contour through the **-layer** keyword.
3. Specify the name of the derived layer through the **-layerOut** keyword.
4. Specify the contour to be saved as follows:
 - a. For a printing contour, define the focus and dose condition that of interest:
 - Use **-optical** to specify the optical model generated for the desired focus. This can be any of the optical models used to generate the PV-bands for the target

layer. (That is, one of the optical models specified through the -opticalSpanLists keyword for the **PVband** function).

- Use **-dose** to specify the desired dose. This can be any of the dose values investigated through a subwindow (experiment) that also investigated the focus you defined using -optical. (That is, one of the dose values specified through the -doseSpanList keyword for the **PVband** function).
- b. For a PV-band contour, indicate which contour is of interest:
 - Use **-displacement** to indicate whether you want the minimum displacement (inner edge) or maximum displacement (outer edge).
 - Use **-reference** to indicate the type of PV-band of interest: regular or absolute.
 - Use **-subwindow** to specify the experiment for which the PV-band was generated.

Results

The PV-band contour is saved as a derived layer.

About Calibre LFD Databases

When writing a Calibre LFD rule file, you can generate several types of databases.

- [The Indexing Database](#) — ASCII database output when using variability indices which contain priority scoring.
- [The Check Database](#) — ASCII database containing markers for each check indicating failure points within the design.
- [The Bands Database](#) — A database containing the PV-band information. It is recommended that you output PV-bands and contours to an OASIS file, as described in “[Saving Calibre LFD Data to a Layout Database](#)” on page 103, to save disk space and to reduce the time it takes to load PV-bands and contours into a layout viewer.

The Indexing Database	102
The Check Database	102
The Bands Database	103

The Indexing Database

The indexing database contain variability scores. You can create as many of these databases as needed. They contain priority rankings for the *whole layer*.

You control the name of this database through the **-database** option for the [VariabilityIndices](#) command.

The Check Database

The check database contain two types of data.

- Errors markers generated by each of the Calibre LFD checks.
- Priority rankings for *each check*.

You can sort results in these databases several ways:

- Individual errors are ranked according scores (severity).
- Calibre LFD checks can be sorted by name or priority.

You control the name of this database through the **-database** option for the check commands (see “[Calibre LFD Checks](#)” on page 181). You can create as many of these databases as needed.

The Bands Database

The bands database contains the actual PV-band structures, stored in ASCII format. You can create as many of these databases as needed, though at the beginning we recommend that you write all PV-bands to the same Bands Database.

You control the name of this database through **-database** for the [OutputBands](#) function.

The names of the PV-bands are system-generated based on:

- The target layer for which the PV-bands are generated.
- The process variation experiment they represent.
- The type of PV-band (regular or absolute, described in “[Types of PV-Bands](#)” on page 49).

Because PV-bands are memory-intensive objects, it is recommended that you use the **-errorFilterSize** option to prevent data explosion. Please refer to “[CaptureContour](#)” on page 323 for a discussion of this keyword.

Saving Calibre LFD Data to a Layout Database

Many Calibre LFD operations allow you to write Calibre LFD results to a database you specify using the **-database** keyword. This data can include error markers generated by checks as well as PV-band and contour data.

Procedure

1. Create a derived layer for each PV-band, contour, set of error markers, or set of scores. For details refer to the following topics:
 - “[Accessing PV-Band Data](#)” on page 99
 - “[Accessing Contour Data](#)” on page 100
 - “[Writing Custom Checks](#)” on page 95
 - “[Built-in Check Functions](#)” on page 94
2. Use COPY within a VERBATIM block to create a rulecheck for the derived layers you want to write to the layout database.
3. Write a DRC CHECK MAP statement to write the rulecheck to the desired output database. Be sure the DRC CHECK MAP statement includes:
 - The type of database — ASCII, GDS, or OASIS.
 - The layer number for the data — For GDS or OASIS only.
 - The path to the database.

Note

 You can only reference TVF variables within SVRF code that is in a VERBATIM block, and only if you use quotes (" ") to define the VERBATIM block. TVF treats the contents in between braces ({{}}) as absolute notation, so if you use braces to define the VERBATIM block, you are not able to reference any TVF variables.

Examples

```
LFD::MinWidthCheck -layer active -subwindow 1 \
    -minDRCwidth 0.150 -minLFDwidth 0.05 \
    -database $lfdErrorDatabase \
    -layerOut myCheckLayer

LFD::OutputBands {-layer active -bandType regular -subwindow 1 \
    -database $lfdBandDatabase \
    -layerOut active_band_p1}

VERBATIM {
    active_band_p1 {Copy active_band_p1}
    myCheckLayer {Copy myCheckLayer}
}
DRC CHECK MAP active_band_p1 GDS 100 "./layout_lfs.gds"
DRC CHECK MAP myCheckLayer GDS 101 "./layout_lfs.gds"
```

The intricacies of TVF are outside the scope of this manual. However, be aware that you can choose between including the DRC CHECK MAP statement inside the VERBATIM block, or using the tvf::DRC statement outside the block.

If you followed the advice in “[Avoiding Namespace Issues](#)” on page 73, you can use “DRC” in place of “tvf::DRC”.

The preceding example takes advantage of using the DRC statement outside the block because it made it possible to pass the pathname for the ASCII database as a variable.

Calibre LFD Flow with a PDK

The flow for running Calibre LFD with a PDK is essentially the same as running without a PDK: You must write or obtain a rule file for running Calibre LFD, then run Calibre LFD by invoking Calibre from either the command line or the Calibre Interactive GUI.

There are two primary differences between these two flows:

- **Rule file contents**
 - When you do not use a PDK, the rule file must contain complete definitions of checks and process experiments. The commands you must use to adhere to the syntax are documented in “[Calibre LFD Syntax Descriptions](#)”.
 - When you have a PDK, the checks and process experiments have already been designed for you. In this case, the rule file serves as a ***control file*** and contains calls to these checks and experiments, rather than definitions of checks and experiments. The calls to PDK-defined checks and experiments are documented in the section “[Calibre LFD Syntax Descriptions](#)” on page 175.
- **Layer mapping**
 - When you use a PDK, you must map layers in the TVF control file to layers in the PDK.

For example, in the control file:

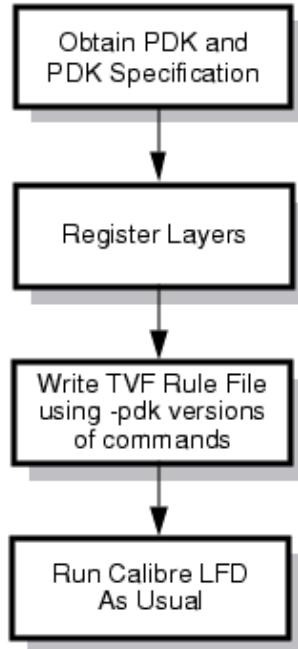
```
RegisterLayer -pdk lfd_pdk -Designlayer met1 \
    -Processlayer pdklyr_met1
```

And correspondingly in the PDK:

```
LFD:addPDKLayer { -name pdklyr_met1 ... }
```

[Figure 4-1](#) shows the flow for running LFD with a PDK:

Figure 4-1. Basic Flow for Running Calibre LFD with a PDK



[About the PDK](#) [106](#)

[About the PDK Specification](#) [107](#)

About the PDK

The PDK includes most of the information you need to run Calibre LFD.

Each PDK contains the following:

- Fully characterized process experiments for each layer process definition. A single layer process may be used for multiple layers in your design.
- A set of check templates appropriate for each layer process. Each check template defines the process conditions, error conditions, outputs and so on for a specific check and assigns that model a name.
- Complete RET recipes needed to generate accurate PV-bands.
- Pointers to optical models for performing lithographic simulations during PV-band generation.
- Resist models required to perform lithographic simulations during PV-band generation.

In almost all cases, the PDK is an encrypted file, and therefore not humanly readable. Security settings built into the PDK control read and write access to the data within the PDK:

- When data is read-protected, access to PV-band or contour data is limited and you do not see check details in the transcript file for the run.
- When data is write-protected, you must use the layer processes and checks as specified in the PDK. You cannot override any of the PDK settings.

About the PDK Specification

When you obtain a PDK from your foundry or design team, you should also receive a PDK Specification. The PDK Specification provides information you need for working with the PDK. This information is provided in the form of a text file called the PDK Specification.

The PDK Specification provides the following:

- A table defining the layers that must exist in the design database and the associated layer names used in the PDK process correction recipe. This is the information you must use to create the layermap file.
- Names and brief descriptions of each layer process definition. Refer to [RegisterLayer](#) for a discussion of how these are used.
- Security settings for each layer process definition, indicating whether or not you can use the optional arguments to the PV-band calls within TVF to override the settings defined in the PDK.
- Names and brief descriptions of each check template.
- Security settings for each check template, indicating whether you can use the optional arguments to the check calls within TVF to override the settings defined in the PDK.

PDK File Commands

A limited set of commands are allowed in the PDK.

The PDK file is comprised entirely of the following:

- Three PDK-specific commands:
 - [createPDK](#) — Can be issued one time only.
 - [addPDKLAYER](#) — Can be issued multiple times.
 - [addPDKCheck](#) — Can be issued multiple times.
- Macro definitions as needed to define macros referenced by the LFD::addPDKLAYER command.

See “[Sample PDK](#)” on page 526 for an example PDK.

Writing TVF for use with a PDK

This topic describes a simple process for writing a control file, which is the TVF rule file used with a PDK.

Prerequisites

- Knowledge of TVF programming. See “[Writing Rules Using TVF](#)” on page 72.
- A layout database.
- The PDK for the process you are using.
- The following information about the PDK:
 - Names and descriptions of the layer processes defined in the PDK.
 - Names and descriptions of the check models defined in the PDK.

Procedure

1. Create a new file in your ASCII text editor. This becomes the TVF rule file.
2. Begin the file with the following required first line and **package require** statement:

```
#! tvf
=====
# Load LFD-generic library
package require CalibreDFM_LFD
```

3. If you want to create variables storing the pathnames for output databases as variables, add **set** commands to create those variables:

```
set lfdErrorDatabase      "./rdb/lfd_New.rdb"
set lfdBandDatabase       "./rdb/bands_New.rdb"
set lfdTopClassification  "/dev/null"
```

4. Supply the SVRF statements needed to read in and evaluate your layout database:

- a. Begin with VERBATIM:

```
tvf::VERBATIM "
```

- b. Define the database and topcell:

```
LAYOUT SYSTEM GDSII
LAYOUT PATH \...\MODELSandLAYOUT/DIGBLK65.gds\
LAYOUT PRIMARY \dffrxl\
```

- c. Define the database precision:

```
PRECISION 1000
RESOLUTION 1
```

- d. Set the error controls:

```
FLAG SKEW YES
LAYOUT ERROR ON INPUT NO
```

- e. Load in the design layers:

```
LAYER active 11
LAYER poly 13
LAYER contact 15
LAYER metall 16
LAYER emptyLayer 1000
```

- f. End the VERBATIM command with a closing quote:

```
"
```

5. Supply the SVRF statements needed to write the Calibre LFD results to a RDB file:

- a. Begin with VERBATIM:

```
tvf::VERBATIM "
```

- b. Supply the SVRF code required

```
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "\"$lfdTopClassification\" ASCII
DRC SUMMARY REPORT "/dev/null\"\
DRC MAXIMUM VERTEX 199
DRC KEEP EMPTY YES
```

- c. End the VERBATIM command with a closing quote:

```
"
```

6. Add LFD::Begin to start the block used to perform Calibre LFD processing:

```
LFD::Begin
```

7. If you need to define any custom simulation settings (path to models, simulation size, or image grid) add the SimConfig command:

```
LFD::SimConfig -modelpath $env(MODEL_DIR)
```

8. Add the LoadPDK command to load the PDK.

```
LFD::LoadPDK ../PDK/generic.pdk
```

9. Use the RegisterLayer statement to map your design layers to the layer processes defined in the PDK:

```
LFD::RegisterLayer -Designlayer active -Processlayer diff -pdk
myPDK
    LFD::RegisterLayer -Designlayer contact -Processlayer holes -pdk
myPDK
```

10. If any layers require special handling when generating the PV-bands, optionally use the [PVband](#) command to define the treatment required:

```
LFD::PVband -layer active -pixel 0.01
```

11. For each check you want to perform, add a call to that check (see “[Calibre LFD Syntax Descriptions](#)” on page 175):

```
LFD::MinSpaceCheck -layer contact -pdkCheckName contactMSC_PDK \
-database $lfdErrorDatabase
LFD::MinSpaceCheck -layer active -pdkCheckName activeMSC_PDK \
-database $lfdErrorDatabase
```

12. If you believe you benefit from being able to visualize the actual PV-band data, include the -pdk version of the [OutputBands](#) command for each layer:

```
LFD::OutputBands -layer active -pdkCheckName diffusionBands \
-database $lfdBandDatabase
```

13. End the Calibre LFD processing block with the LFD::[End](#) statement:

```
LFD::End
```

Results

See “[Sample Rule File](#)” on page 519 for an example TVF rule file referencing a PDK.

SVRF Rule File Macros

Macros are functional templates that can be called multiple times in a rule file. They are very useful when writing Calibre LFD rule files.

Calibre rule files for Calibre LFD must be written using the Tcl Verification Format (TVF) programmable expansion to the Standard Verification Rule Format (SVRF) language for Calibre. You can use the VERBATIM command to include SVRF code in TVF files. For complete information on SVRF and TVF, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#). Within SVRF rule files, you can use Calibre macros which are similar to macros of the C and C++ languages.

Macro Definition	111
DMACRO Arguments.....	112
LOCAL() Modifier	113

Macro Definition

A macro definition consists of the keyword DMACRO (define macro), followed by a macro name, followed by a list of zero or more argument names, followed by “{”, followed by a sequence of zero or more SVRF statements or operations, followed by “}”.

Here is the syntax:

DMACRO macro_name [arguments] {SVRF_code }

For example:

```
DMACRO WIDTH_CHECK lay val {
  S1 = INT lay < val ABUT < 90 SINGULAR REGION
  S2 = INT lay < val ANGLED == 2 PARALLEL OPPOSITE REGION
  S3 = INT lay < val CORNER TO EDGE REGION
  ( S1 OR S2 ) OR S3
}
```

A macro is invoked by the keyword CMACRO (call macro), followed by a macro name and a list of zero or more arguments:

CMACRO macro_name [arguments]

The **macro_name** must match that of some DMACRO definition. Each argument may be either a name (generally a layer or variable name) or a numeric constant. A sufficient number of arguments must be present after the CMACRO **macro_name** to match what the DMACRO specifies as arguments.

Macros are discussed in more detail in the “[Key Concepts](#)” section of the [Standard Verification Rule Format \(SVRF\) Manual](#).

Calibre LFD uses a specialized feature of Calibre macros that allows you to create local names within Calibre DMACROs. The LOCAL() modifier is useful for creating a local name only valid in the scope of a CMACRO call.

Related Topics

[DMACRO Arguments](#)

[LOCAL\(\) Modifier](#)

DMACRO Arguments

Layer definition names in DMACRO (define macro) statements are locally-scoped by default when the DMACRO is instantiated in a CMACRO, and references to the layer definition name in the DMACRO are substituted appropriately. If an argument to a DMACRO statement matches the name of a derived layer within the DMACRO block definition, then the corresponding CMACRO argument becomes the name of a global layer.

A regular substitution of DMACRO input layers is applied to the contents of the portion of the body of the DMACRO that is not in a comment. Otherwise stated, all strings in the macro are substituted unless they are found inside of a comment block. Once regular substitution has occurred, a namespacing process is applied to permit the DMACRO to be invoked multiple times.

If one or more of the INPUT layers to the DMACRO are the same as one of the MAP layers in an SVRF LITHO call, the MACRO mechanism substitutes the MAP output to one of the input layers.

The following is not allowed:

```
DMACRO test input output {  
    ...  
    /*  
    setlayer output = COPY input  
    */  
    ...  
    output = LITHO .... MAP output  
}
```

However this macro is allowed:

```
DMACRO test input outputM {  
    ...  
    /*  
    setlayer output = COPY input  
    */  
    ...  
    outputM = LITHO .... MAP output  
}
```

Related Topics

[SVRF Rule File Macros](#)

[LOCAL\(\) Modifier](#)

LOCAL() Modifier

The LOCAL() modifier is used to create local names that are valid only in the scope of each CMACRO call. This modifier is useful for defining specification statements within DMACROS.

Consider the following example:

```
DMACRO COPY_C L1 L2 C {  
    LAYOUT CELL LIST LIST1 C  
    L2 = DFM COPY L1 CELL LIST LIST1  
}
```

Assume that you want to use this DMACRO as follows:

```
LAYER A 1  
LAYER B 2  
CMACRO COPY_C A A1 "A*"  
CMACRO COPY_C B B1 "B*"
```

This use is invalid because the Layout Cell List specification statement is duplicated with the name LIST1. To resolve this issue, a LOCAL() modifier can be added to the DMACRO:

```
DMACRO COPY_C L1 L2 C {  
    LAYOUT CELL LIST "LOCAL(LIST1)" C  
    L2 = DFM COPY L1 CELL LIST "LOCAL(LIST1)"  
}  
CMACRO COPY_C A A1 "A*"  
CMACRO COPY_C B B1 "B*"
```

The name “LOCAL(LIST1)” is replaced by a name that is unique to each CMACRO call. Within one CMACRO call, multiple instances of LOCAL(name) are always replaced by the same local name (such as “LOCAL(LIST1)” in the example above). Different names used with LOCAL() in the same macro remain distinct after the replacement.

Note

 The quotes around the string “LOCAL(LIST1)” are required.

Related Topics

[DMACRO Arguments](#)

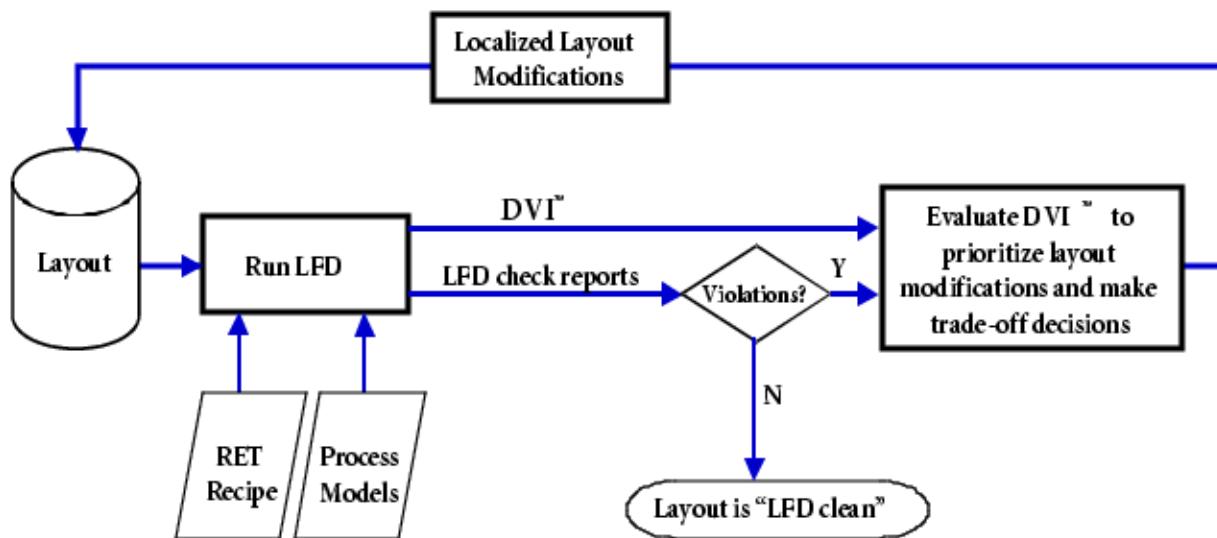
[SVRF Rule File Macros](#)

Chapter 5

For Designers — Incorporating Calibre LFD Into Your Flow

As a designer, you can incorporate Litho-Friendly Design into your existing flow by adding the Calibre LFD loop.

Figure 5-1. Calibre LFD Loop



The Calibre LFD flow procedures and generated databases are examined.

Calibre LFD Database Types	116
Running Calibre LFD Checks	119
Reviewing Calibre LFD Violations	121
About Model-Based Hints (MBH)	127
MBH Rule, Map, and Transcript Files	132
Modifying the Design.....	150
Re-Running Calibre LFD Checks on a Modified Design.....	151
rdb_flattener.....	154

Calibre LFD Database Types

When running Calibre LFD, databases are created that provide information and assist in the evaluation of Calibre LFD run results.

There are three types of databases generated by the Calibre LFD tool:

- [Indexing Database](#) — ASCII database that is output when using variability indices which contain priority scoring.
- [Check Database](#) — ASCII database that contains markers for each check. The markers indicate failure points within the design.
- [Bands Database](#) — A database that contains the PV-band information. It is recommended that you output bands and contours to an OASIS file to save disk space and to reduce the time it takes to load bands and contours into a layout viewer.

You can view these databases using Calibre RVE (Results Viewing Environment) in conjunction with your layout viewer.

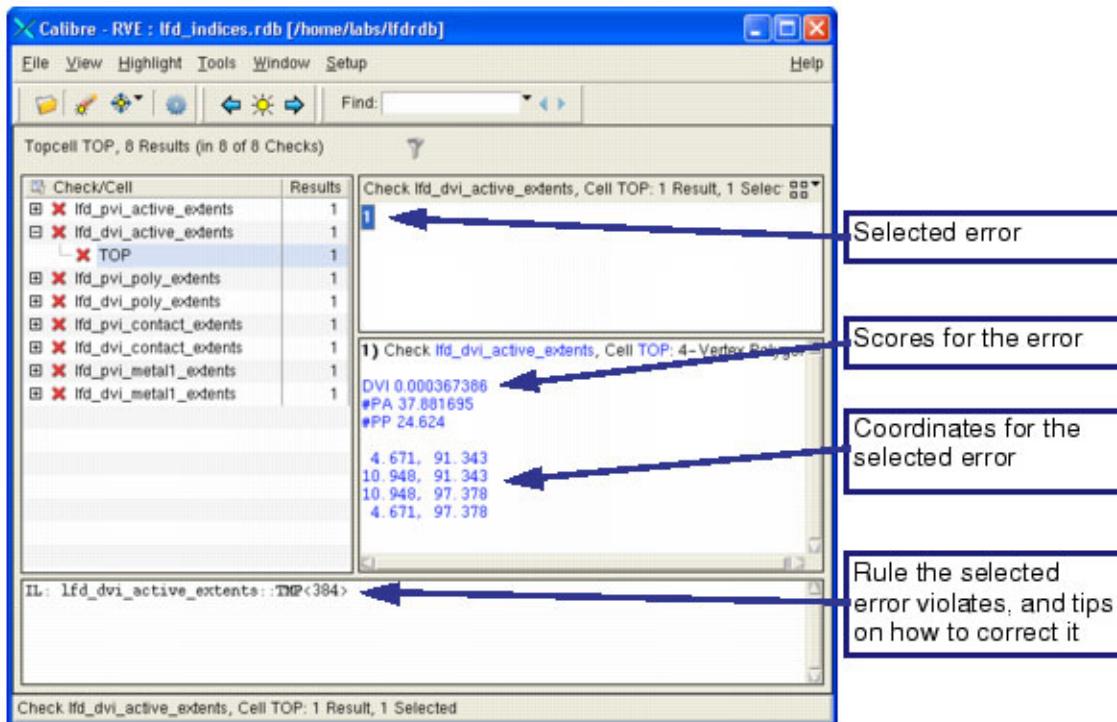
Indexing Database	116
Check Database	117
Bands Database	118

Indexing Database

The Indexing Database contains variability scores. There may be more than one Indexing Database. They contain priority rankings for the whole layer.

The name of this database is set through the **-database** option for the [VariabilityIndices](#) command. Use the Indexing Database to assess design layer robustness across the lithographic process window and to prioritize error fixing.

Figure 5-2. Viewing the Indexing Database



Check Database

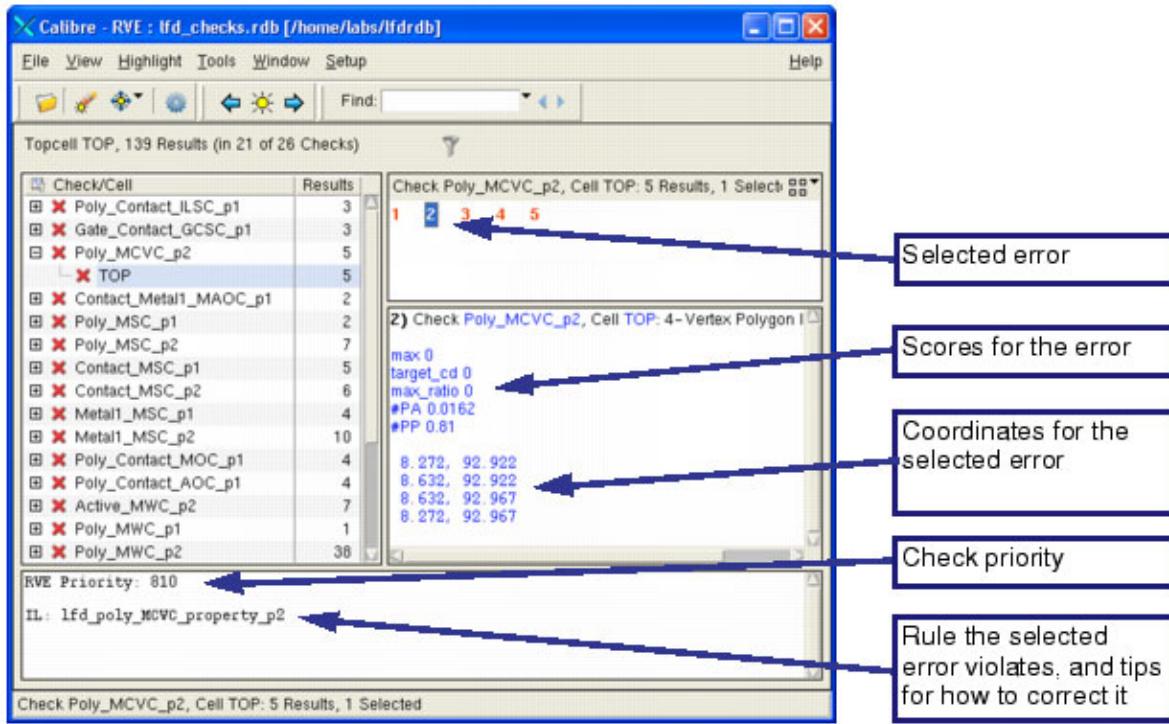
The Check Database contains error markers and priority rankings.

- Error markers generated by each of the Calibre LFD checks.
- Priority rankings for each check.

There may be more than one Check Database. Errors in the Check Database are organized according to the name of the check that encountered the error. Checks are organized according to priority.

You control the name of this database through the **-database** option for the check commands. You can create as many of these databases as needed.

Figure 5-3. Viewing the Check Database

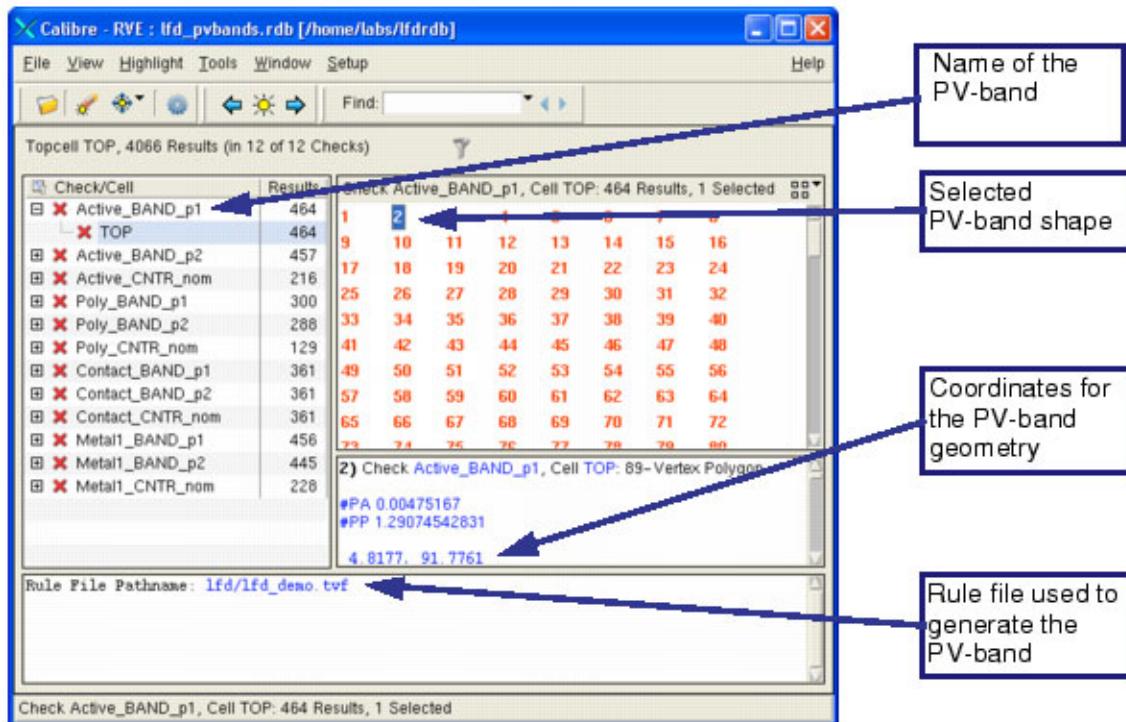


Bands Database

The Bands database contains the actual PV-band structures, stored in ASCII format. There may be more than one Bands Database.

You control the name of this database through **-database** for the [OutputBands](#) function.

Figure 5-4. Viewing the Bands Database



Running Calibre LFD Checks

You can run Calibre LFD checks using Calibre Interactive.

Prerequisites

- You have the layout open in a supported layout viewer.
- You have a Calibre Interactive runset.
- You have all required input files for the run, such as the rule file and required models. See “[Calibre LFD Requirements](#)” on page 20.

Procedure

1. Invoke Calibre Interactive from the layout viewer.
 - If you are using Calibre DESIGNrev or Calibre WORKbench, choose **Verification > Run DFM**.
 - For most other design tools, select **Calibre > Run DFM**.
2. Select **File > Load Runset** to load the Calibre LFD runset into Calibre Interactive.
3. Click the **Inputs** button on the left panel, then select the **LFD** tab.

4. Click the **Outputs** button on the left panel and enable “Show results in RVE” to open Calibre RVE automatically when the run finishes.
5. Click **Run LFD** to start the run.

Results

When the Calibre LFD run completes, Calibre RVE opens automatically and loads the results database on the Outputs pane. The database that Calibre RVE loads automatically is not the one you use for viewing Calibre LFD results.

For viewing results, see “[Reviewing Calibre LFD Violations](#)” on page 121.

Related Topics

[Running Calibre LFD with Calibre Interactive](#)

[Command Line Examples to Invoke Calibre LFD](#)

Reviewing Calibre LFD Violations

You can use Calibre RVE to view each of the results databases from a Calibre LFD run and highlight the results. You can also use DRC HTML Reporting to create a report you can easily share with others.

The best way to view the results is to display the design database in a layout viewer or editor and use Calibre RVE to view the check results and variability index databases. Calibre RVE is a graphical debug program that interfaces with most IC layout tools. When both tools are open at the same time and socket communication is established between the tools, you can select an error in Calibre RVE and see that portion of the design highlighted in the layout viewer/editor.

Viewing Calibre LFD Check Results	121
Viewing Design Variability Index (DVI) Data	123
Making Sense of Variability Data	124
Viewing PV-Band Data	124
HTML Reporting for Calibre LFD Results	125

Viewing Calibre LFD Check Results

You can use Calibre RVE to view the Calibre LFD check results.

One of the primary outputs of Calibre LFD checking is a [Check Database](#). This database references areas of the design that are most likely to cause problems through one or more process window condition settings.

Prerequisites

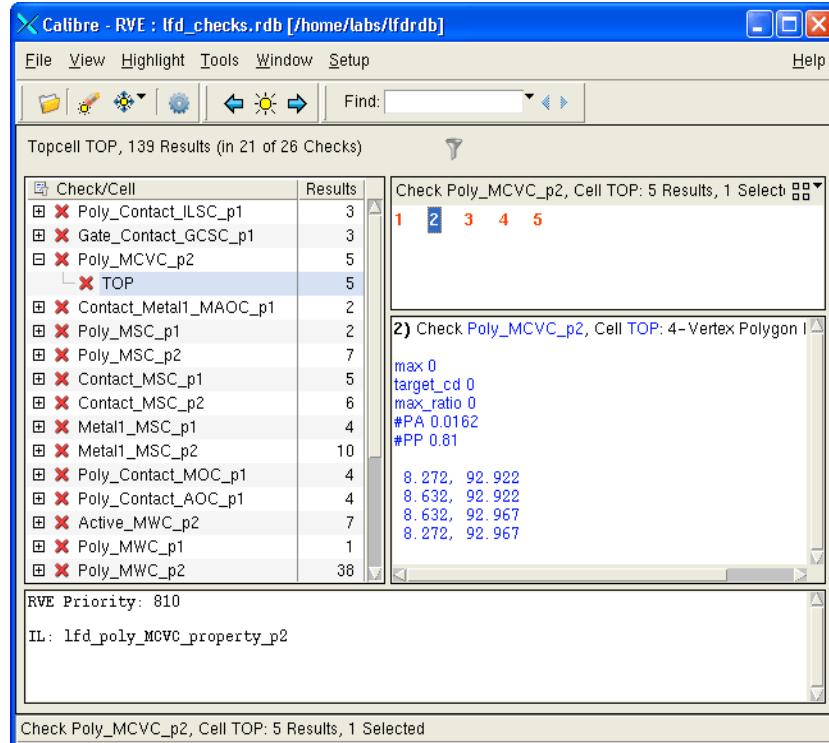
- You are using either the Calibre DESIGNrev or Calibre WORKbench layout viewer.

Procedure

1. If you invoked Calibre RVE automatically when the Calibre LFD run completed, then socket communication is already established and you can skip to step 4.
2. Open the design database in your layout viewer/editor.
3. In Calibre DESIGNrev or Calibre WORKbench, choose **Verification > Start RVE**.
4. In the Calibre RVE dialog box, select the **DRC/ERC** database type, supply the path to the Check Database¹, and then click **Open**.

1. Your CAD team should supply you with the name of this database.

5. View the numbered results on the upper right side of the Calibre RVE window to see how many problem spots need to be fixed.



6. Select one of the errors, then click the **Highlight Selected Results** icon in the toolbar to highlight it in the layout viewer.
7. Use any of the highlight options available through Calibre RVE to inspect the rest of the errors:
 - Next error — Click > in the Calibre RVE toolbar.
 - Previous error — Click < in the Calibre RVE toolbar.
 - Selected error — Click **Highlight Selected Results** icon in the Calibre RVE toolbar.
 - All errors — Select **Highlight > Highlight All**.
 - Clear highlights — Select **Highlight > Clear Highlights**.

Related Topics

[Running Calibre LFD with Calibre Interactive](#)

[Calibre RVE User's Manual](#)

Viewing Design Variability Index (DVI) Data

You can use Calibre RVE to view the Design Variability Index data (DVI) results from a Calibre LFD run.

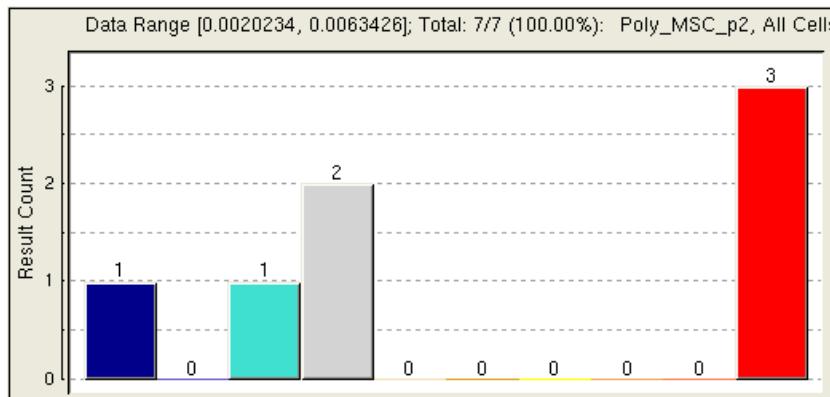
The [Indexing Database](#) contains Design Variability Index data (DVI) for each layer investigated. To generate variability indexes, the tool partitions the layout either by cell or into regular rectangles, as defined by the lithography team. For each area, it computes the DVI as the ratio of failure area to design area.

Prerequisites

- The layout is open in a layout viewer that is supported with Calibre RVE.
- The Indexing Database is open in Calibre RVE.

Procedure

1. In Calibre RVE, select **View > Tree Options > Group By > Check/Cell** to sort the tree view by check names, then cells.
2. Right-click a DVI check in the tree view and select **Histogram DVI** to display a histogram showing the DVI distribution for the cell.



The histogram shows the “scores” ordered from lowest to highest.

3. Right-click in the histogram area and select **Show Color Map** in the pop-up menu.

The DVI data is displayed as a color map in the layout viewer.

Tip

To simplify comparing histogram bins to the color map display, be sure the layer properties are configured as described in “[Defining Highlight Colors to Match Histograms](#)” on page 535.

Related Topics

[Making Sense of Variability Data](#)

[Running Calibre LFD with Calibre Interactive](#)

[Calibre RVE User's Manual](#)

Making Sense of Variability Data

Once you understand the variability indexes, you can use them to prioritize modifications to the design. As a rule of thumb, the higher the index, the higher the priority.

While the DVI can be zero, indicating that a section of the design has no failure areas, the PVI is never zero. This is reflective of the fact that there are always differences between the mask shapes and the shapes on the wafer.

For the purposes of the manufacturability discussion in [Table 5-1](#), the lowest PVI for the layout is referred to as PVImin:

Table 5-1. Manufacturability Based on Variability Data

PVI	DVI	Manufacturability	
PVImin	0	Desirable	The process is stable and the design is manufacturable.
PVImin	> 0	Design Limited	The process is stable and the design is not manufacturable.
Approaching PVImin	0	Process Limited	The process is unstable but the design is manufacturable.
Approaching PVImin	> 0	Undesirable	The process is unstable and the design is not manufacturable.

Viewing PV-Band Data

PV-band data (if saved) is saved in the form of an ASCII database called the Bands database. You can view the PV-bands in your layout viewer by opening the Bands database in Calibre RVE and highlighting the individual database objects that represent the PV-bands.

Prerequisites

- You have the layout open in a layout viewer that is supported by Calibre RVE.
- You have the [Indexing Database](#) open in Calibre RVE.

Procedure

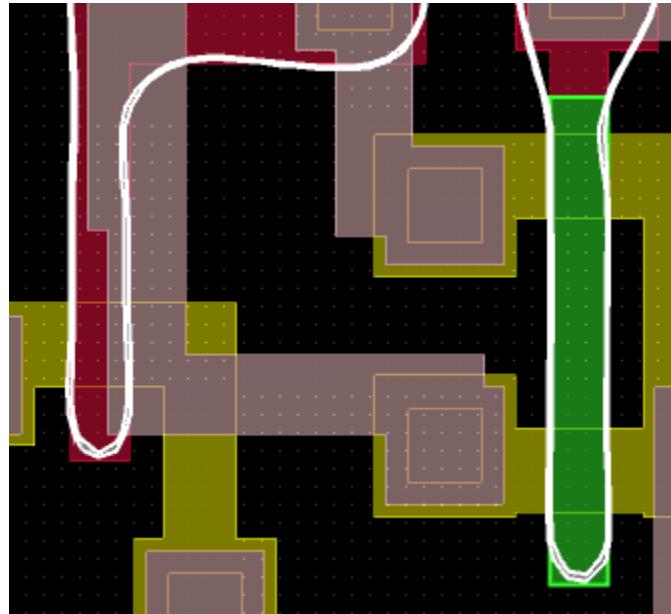
1. Configure Calibre RVE to allow multiple highlights:
 - a. Choose **Setup > Options**, to display the Setup DRC-RVE Options pane.
 - b. Uncheck **Clear existing highlights** before showing new highlights in the **Highlight** tab.

2. In Calibre RVE, choose **File > Open Database**.
3. In the Open Database dialog box, use the **Browse** button to browse through to the Bands database, and then click **OK**. The database opens in a new pane.

Note

The hierarchy on the left side of the new Calibre RVE window shows the sets of bands available for viewing. Expanding one of these sets of bands shows a separate Calibre RVE object for each feature's PV-band.

4. Select the check that represents bands you want to visualize. This requires that you know the layer and the experiment that generated the bands.
5. Right-click the check and select **Highlight**. Calibre RVE instructs the viewer to display the PV-band data over the existing highlights.
6. Zoom to an area that interests you.



7. Repeat for other errors as desired.

Related Topics

[Running Calibre LFD with Calibre Interactive](#)

[Calibre RVE User's Manual](#)

HTML Reporting for Calibre LFD Results

DRC HTML reporting in Calibre RVE creates an HTML-formatted report based on views in Calibre RVE for DRC. An HTML report can be created from Calibre LFD output files (RDB

and Summary Report) and the layout file. You can include hotspot errors, their properties, as well as snapshots of the hotspots from the layout. You can display results tables, result highlights, histograms, and colormaps.

Configuration controls are included for you to customize the grouping, filtering, and sorting the results, and for customizing the HTML report format.

To create an HTML report based on the view displayed in Calibre RVE, see this topic:

- “[Creating a DRC HTML Report from Calibre RVE for DRC](#)” in the *Calibre RVE User’s Manual*

For complete information on creating HTML reports, see this topic:

- “[DRC HTML Reporting](#)” in the *Calibre RVE User’s Manual*

For an example, see this topic:

- “[DRC HTML Report for Calibre LFD](#)” in *Calibre Solutions for Physical Verification*

The following is required to run HTML reporting:

- A Calibre RVE license.
- A results database.
- An HTML report configuration file.
- File permissions on `/tmp/.X11-unix` and `/tmp/.X11-pipe` must be set to 1777.
- A Calibre DESIGNrev license is required if layout images are included in the report.
- A GDS or OASIS layout is required if layout images are included in the report.

Related Topics

[Viewing Calibre LFD Check Results](#)

[Viewing Design Variability Index \(DVI\) Data](#)

[Viewing PV-Band Data](#)

About Model-Based Hints (MBH)

The MBH function of Calibre LFD generates an output file that provides the designer with repair hints on how to modify the layout to fix lithographic hotspot regions.

MBH functionality is integrated within the Calibre LFD flow. This integration occurs early enough in the design process for the designer to address hotspot issues when they are less complex to repair.

MBH Operation and Flow	127
Running Calibre LFD with MBH	128
Viewing MBH Results	129

MBH Operation and Flow

The MBH simulation runs on selected layers of IP design blocks and standard cell geometries. Typical layers for analysis can include active, poly, contact, metal1, and metal2. Multilayer knowledge in conjunction with basic DRC checks such as minimum spacing and width constraints are accounted for during MBH analysis to prevent the hints from introducing new errors.

There are two modes of MBH operation:

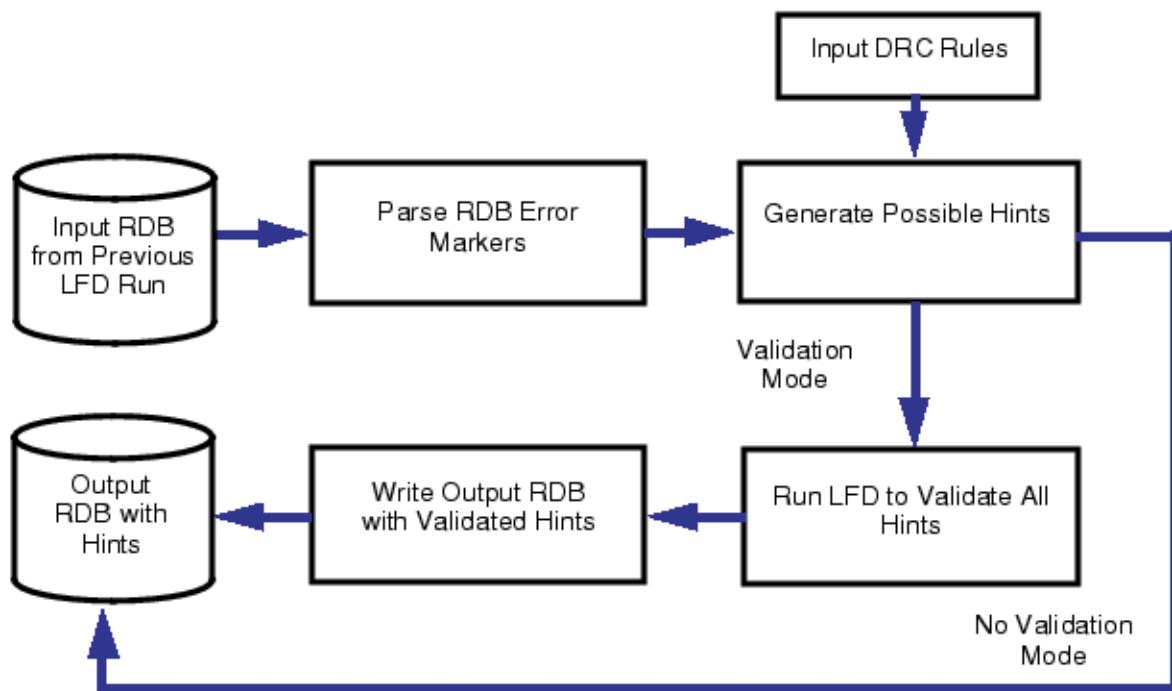
- **Validation Mode** — Hints are verified by running Calibre LFD on the regions around each error marker after applying the hint. This mode is recommended to verify that the hint is Calibre LFD clean and a valid fix.
- **No-Validation Mode** — A faster mode of operation that runs MBH simulation and generates hints, but does not run Calibre LFD validation on the hint sites. The electrical connectivity and DFM DRC integrity of the specified layers are maintained.

Note



While hints are DRC clean and do not create new errors, unless the run includes validation mode the hint may not fully solve the hotspot.

Figure 5-5. The MBH Flow



Hints are derived by moving edges or groups of edge candidates in the problematic regions of the layout until a possible solution is found. MBH analysis of the specified layers and neighboring features in the lithographic hotspot regions results in a repair hint generated for each error marker in the Calibre LFD RDB file.

Using Calibre LFD with MBH generation reduces the amount of time the layout designer spends in repairing lithographic hotspots.

Running Calibre LFD with MBH

You can run Calibre LFD with MBH functionality to generate suggested repair hints for the identified lithography hotspots in the layout.

The input to MBH is the RDB file generated from a previous Calibre LFD run. This RDB file contains error markers showing the problematic areas in the layer shapes. MBH is used to simulate the region around the error markers for possible localized design changes that can fix these hotspots.

The output from MBH is a new RDB file that contains the same error markers with added Calibre LFD hints for the designer for possible fixes of the hotspots. The designer has the option to modify the layout using the hint information as a repair reference.

Prerequisites

- Path to a valid Calibre LFD software tree.
- Process-specific Calibre LFD kit.
- MBH TVF rule file with layer information, parameters, and arguments used by the function LFD::[GenerateHints](#). See “[Sample MBH TVF Rule File](#)” on page 132.
- MBH map file that maps RDB check names to original layer names (non-derived layers). See “[Sample MBH Map File](#)” on page 135.
- Layout database in GDS or OASIS format.
- Calibre LFD RDB file from a previous LFD run with hotspot error markers.

Procedure

1. Run Calibre LFD with MBH functionality and control the processing through the MBH TVF rule file. This runs LFD with MBH by calling the function LFD::[GenerateHints](#).
2. When the processing completes, view the hints in the RDB hint file, as described in “[Viewing MBH Results](#)” on page 129.

Examples

Here is a command line example of running LFD with MBH:

```
calibre -lfd -hier -mbh mbh_tvf_file_name
```

Related Topics

[About Model-Based Hints \(MBH\)](#)

[GenerateHints](#)

[calibre -lfd -mbh](#)

[Sample Calibre LFD Rule File and PDK](#)

Viewing MBH Results

This procedure shows you how to view the suggested hotspot fixes from running Calibre LFD with MBH. You display the design database in a layout viewer or editor and use Calibre RVE to view the hint sites.

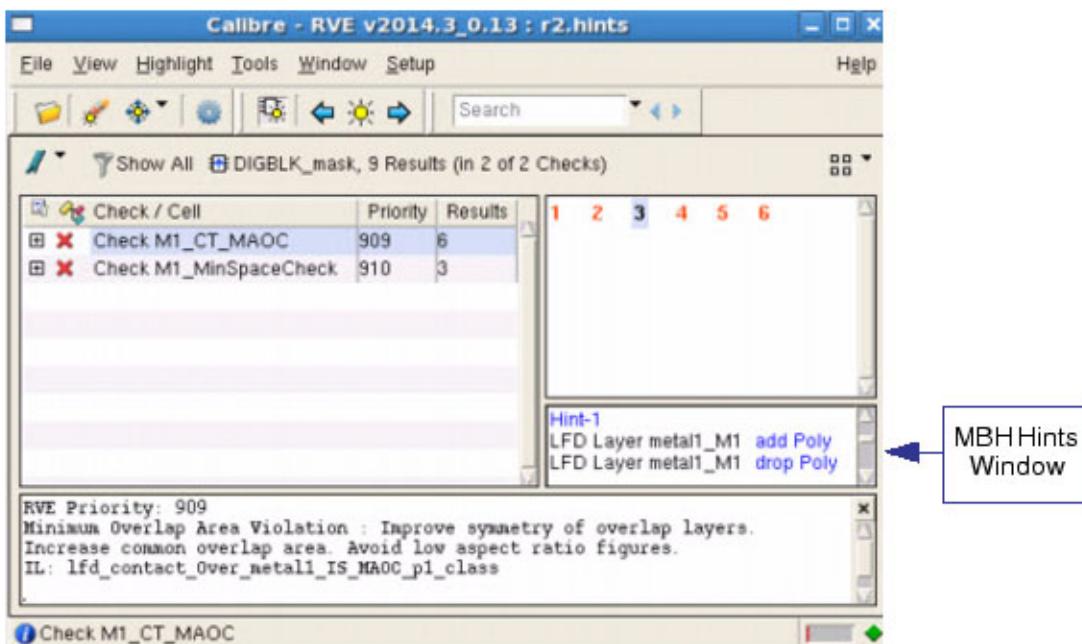
Prerequisites

- The Calibre DESIGNrev or Calibre WORKbench layout viewer. From other viewers, invoke Calibre Interactive from the Calibre menu.
- The layout database in GDS or OASIS format.
- The RDB hint file from “[Running Calibre LFD with MBH](#)” on page 128.

Procedure

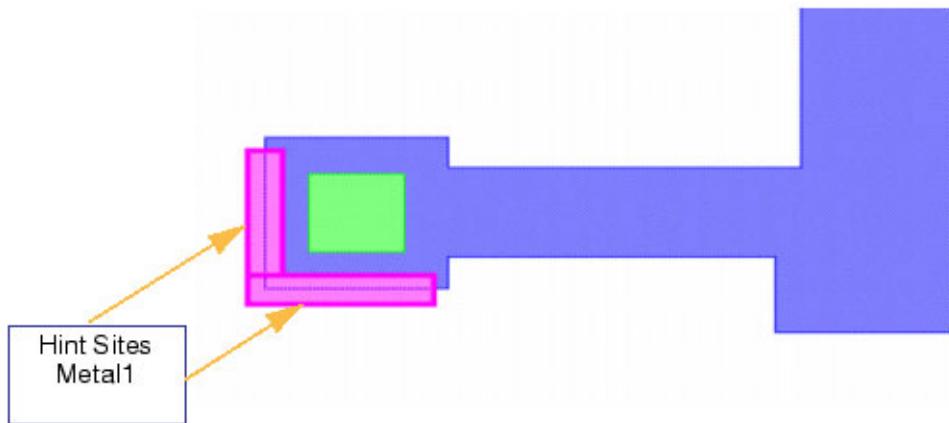
1. Load the MBH RDB hint file in Calibre RVE:
 - a. Open the layout database in your layout viewer/editor.
 - b. In Calibre DESIGNrev or Calibre WORKbench, choose **Verification > Start RVE**.
 - c. In the Calibre RVE dialog box, select **DRC/ERC** database type, supply the path to the MBH RDB hint file, and then click **Open**.
2. View the suggested layout fixes:
 - a. In the upper right of the Calibre RVE window, click the result number for a specific check to select it.
 - b. Highlight the selected site with the highlight function of Calibre RVE in one of the following ways:
 - o Click the right mouse button and select **Highlight** from the pop-up menu.
 - o Click the **Highlight Selected Result** icon in the toolbar.
 - c. Scroll down in the Calibre RVE window directly below the numbered check sites to view the hint text. There may be more than one hint per site.
 - d. In the graphic window of the layout viewer, ensure that the **Layers** browser selection shows the targeted MBH layers.
 - e. Click on the numbered **Hint** text in the Calibre RVE window to view the potential fix in the layout viewer.
 - f. Toggle the views of the suggested modifications to the layout shapes by clicking on the hints text in the MBH Hints Window. For example in [Figure 5-6](#), click on **add Poly** or **drop Poly**.

Figure 5-6. MBH Hints



3. View the hint sites in the graphic window of the layout viewer.

Figure 5-7. MBH Metal1 Hints



Related Topics

[Calibre RVE User's Manual](#)

[About Model-Based Hints \(MBH\)](#)

MBH Rule, Map, and Transcript Files

Running Calibre LFD with MBH functionality requires TVF rule and error map (map) files with specific formatting and syntax. Run information is available through the MBH transcript file.

The MBH TVF rule and MBH map files can be included with the other rules used to run Calibre LFD. See also “[Sample Calibre LFD Rule File and PDK](#)” on page 519. The sample MBH TVF rule and MBH map file provided are for reference. Each MBH TVF rule and MBH map file must be customized with the check requirements and specifications that are unique to a design.

Sample MBH TVF Rule File	132
Sample MBH Map File	135
MBH Map File Format	139
MBH Transcript File Information	149

Sample MBH TVF Rule File

The MBH TVF rule file is necessary when running LFD with MBH. This Tcl Verification Format (TVF) file calls the MBH function.

For a complete discussion of TVF, refer to “[Tcl Verification Format](#)” in the *Standard Verification Rule Format (SVRF) Manual*. For MBH syntax functionality, see “[GenerateHints](#)” on page 374.

```
#! tvf
# ****
# REQUIRED
=====
namespace import tvf::*
package require CalibreDFM_LFD
namespace import LFD::*
=====

# Specify LFD related output databases
set lfdErrorDatabase      "./lfd_Errors.rdb"
set lfdBandDatabase        "./lfd_Bands.rdb"
set lfdTopClassification   "./lfd_results.gds"
set lfdSummaryReport       "./lfd_results.summary"
set lfdPrecision           1000
=====
# Start the actual recipe
VERBATIM "
LAYOUT SYSTEM GDSII
LAYOUT PATH \"./inputs/generic_layout.gds\
LAYOUT PRIMARY \"*\\""

FLAG SKEW YES
LAYOUT ERROR ON INPUT YES
PRECISION $lfdPrecision
RESOLUTION 1

// MAP EXISTING LAYERS

LAYER active          200
LAYER active_M1        201
LAYER active_M2        202
LAYER active_opc         203
LAYER active_opc1       204
LAYER active_opc2       205
LAYER active_band        206
LAYER active_con1       207
LAYER active_con2       208
LAYER active_min         209
LAYER active_max         210

LAYER poly              300
LAYER poly_M1            301
LAYER poly_M2            302
LAYER poly_opc             303
LAYER poly_opc1           304
LAYER poly_opc2           305
LAYER poly_band            306
LAYER poly_con1           307
LAYER poly_con2           308
LAYER poly_min             309
LAYER poly_max             310

LAYER contact            400
```

```
LAYER contact_M1          401
LAYER contact_M2          402
LAYER contact_opc          403
LAYER contact_opc1         404
LAYER contact_opc2         405
LAYER contact_band         406
LAYER contact_con1         407
LAYER contact_con2         408
LAYER contact_min          409
LAYER contact_max          410

//LAYER metal1              500
LAYER metal1_M1             501
LAYER metal1_M2             502
LAYER metal1_opc            503
LAYER metal1_opc1           504
LAYER metal1_opc2           505
LAYER metal1_band           506
LAYER metal1_con1           507
LAYER metal1_con2           508
LAYER metal1_min            509
LAYER metal1_max            510

LAYER marker                700
gate=poly AND active
metal1=metal1_M1 OR metal1_M2
metal_IS= COPY metal1_M1

// Output Section. ****
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "\"$lfdTopClassification\" GDSII
DRC SUMMARY REPORT "\"$lfdSummaryReport\""
DRC MAXIMUM VERTEX 199
DRC KEEP EMPTY YES

//
=====
// Create pv-Bands
//
=====
"
LFD::Begin

LFD::GenerateHints \
-inputRDB ./lfd_Errors.rdb \
-errorMapFile map \
-LFDRuleFile rules \
-validatedHintsNum 5 \
-minDRCWidth 0.1 \
-minDRCSpace 0.07 \
-haloSize 1.5 \
-simHaloSize 3 \
-hintsNum 5 \
-dfmDB dfm.rdb \
```

```
-outputHintsFile ./r2.hints
LFD::End
```

Related Topics

[Running Calibre LFD with MBH](#)

[Sample MBH Map File](#)

Sample MBH Map File

The MBH map file, also known as the MBH error map file, defines the layer, check, and interlayer spacing DRC values that are used by MBH generation to produce hints that eliminate hotspots found from a Calibre LFD run.

```
# Layer Definitions Section:  
# Specify layer names and specifications.  
# There are two exposures for metal(metal1,metal2).  
Layer  
    -layerName          metal1  
    -minDRCWidth       0.032  
    -lineEndLength     0.04  
    -minLETLSpacing    0.09  
    -minETESpacing     0.1  
    -minETLSpacing     0.11  
    -enablingVar       {RUN_metal1}  
  
Layer  
    -layerName          metal2  
    -minDRCWidth       0.034  
    -lineEndLength     0.030  
    -minLETLSpacing    0.09  
    -minETESpacing     0.1  
    -minETLSpacing     0.11  
    -enablingVar       {RUN_metal2}  
  
Layer  
    -layerName          poly  
    -minDRCWidth       0.024  
    -minDRCSpace        0.08  
    -enablingVar       {RUN_poly}  
  
Layer  
    -layerName          contact  
    -minDRCWidth       0.020  
    -minDRCSpace        0.07  
    -enablingVar       {RUN_contact}  
  
Layer  
    -layerName          via  
    -minDRCWidth       0.020  
    -minDRCSpace        0.07  
    -enablingVar       {RUN_via}  
  
Layer  
    -layerName          diff  
    -minDRCWidth       0.024  
    -minDRCSpace        0.08  
    -enablingVar       {RUN_diff}  
  
# Check Definitions Section:  
# Specify check names and include validation layers for LFD and DRC  
# cleanliness. Layers -layer1 and -layer2 are maintained DRC clean by  
# default.  
  
# MinWidthCheck on layer metal1.  
# Double patterning hints can be of type layer metal1 or metal2.  
Check  
    -checkName          mwc_metal1  
    -errorType          width  
    -layer1             metal1  
    -drcCleanLayers    {contact}  
    -validationLayers  out_mwc_m1
```

```

# InterLayerSpaceCheck on layer metal1 and metal2.
# Double patterning hints can be of type layer metal1 or metal2.
Check
  -checkName          ilsc_metal1_metal2
  -errorType          stitching
  -layer1             metal1
  -layer2             metal2
  -drcCleanLayers    {contact }
  -validationLayers   out_ilsc_m1_m2

# MinWidthCheck on layer poly.
# Hints generated for this check must be clean on layer poly(by default).
Check
  -checkName          mwc_poly
  -errorType          width
  -layer1             poly
  -validationLayers   out_mwc_poly

# InterLayer Spacing Section:
# Specify interlayer minimum enclosure and spacing distance between
# -layer1 and -layer2.

# Use for check mwc_metal1, ilsc_metal1_metal2.
InterLayer
  -layer1              metal1
  -layer2              contact
  -minDRCenc1          0.02

# Use for check mwc_poly.
InterLayer
  -layer1              poly
  -layer2              contact
  -minDRCenc1          0.02

# Use for check ilsc_metal1_metal2.
InterLayer
  -layer1              metal1
  -layer2              metal2
  -minDRCSpace         0.03

# Specify interlayer multiple via enclosure rules with enclosure values
# and width-range constraints using comparison operators.
InterLayer
  -layer1              via
  -layer2              metal1
  -minDRCenc1          0.125
  -minDRCenc1Sides     {{==0.25 0} {>=0.2 <0.4 0.6}}
  -minDRCenc1LineEnds  {{==0.25 0} {>=0.2 <0.4 0.6}}

```

Related Topics

[Running Calibre LFD with MBH](#)

[GenerateHints](#)

[Sample MBH TVF Rule File](#)

MBH Map File Format

MBH Map File Format

Required input file for LFD::[GenerateHints](#).

The MBH map file is divided into three sections that define the layer, check, and interlayer DRC values that are used when running LFD with MBH functionality.

Three types of sections are required and are listed in the order that they appear in the MBH map file:

- **Layer Definitions** — Includes the name and minimum DRC width and space of all layers used in subsequent checks.
- **Check Definitions** — Includes parameters for single and multilayer checks.
- **Interlayer Spacing** — Includes the minimum DRC space between any two different layers or the minimum enclosure between two different layers.

Each of these sections may be specified multiple times as needed.

Format

An MBH map file must conform to the following formatting and syntax rules:

- All layers must be defined before the check or interlayer spacing.
- Each parameter entry must be defined on one line.
- Tcl comments are supported for lines beginning with a (#) character.

See “[Sample MBH Map File](#)” on page 135.

Syntax

Layer

```
-layerName layer_name
-minDRCWidth min_drc_width_value
[-lineEndLength line_end_length]
[-lineEndWidth line_end_width]
{-minDRCSpace min_drc_space |
-minLETLESpacing min_letle_spacing
-minETESpacing min_ete_spacing
-minETLESpacing min_etle_spacing}
```

[-enablingVar ‘{’*env_var* [*env_varN* [0]]...‘}’]

Check

-checkName *check_name*

-errorType {width | space | stitching}

-layer1 *layer1_name*

[-layer2 *layer2_name*]

[-drcCleanLayers ‘{’*layer3_name* *layer4_name* *layer5_name*...*layerN_name*‘}’]

[-validationLayers

‘{’*validationLayer1* *validationLayer2* *validationLayer3*...*validationLayerN*‘}’]

InterLayer

-layer1 *layer1_name*

-layer2 *layer2_name*

{

-minDRCencl *min_drc_encl*

[-minDRCenclSides ‘{’‘{’*width_constraint min_encl_side*‘}’

[‘{’*width_constraint min_encl_side*‘} ...] ‘}’]

[-minDRCenclLineEnds ‘{’‘{’*width_constraint min_encl_end*‘}’

[‘{’*width_constraint min_encl_end*‘} ...] ‘}’]

|

-minDRCSpace *min_drc_space*

|

-minLETLESpacing *min_letle_spacing*

-minETESpacing *min_ete_spacing*

-minETLESpacing *min_etle_spacing*

}

```
[-stitching {'list_of_width_length_pairs'} |  
'default_length' |  
{'list_of_width_length_pairs'}' default_length'})}]
```

Parameters

Layer Definitions

- **Layer**

Required keyword used to define the starting point for the layer operations section of the map file before check or interlayer space operations can be specified.

- **-layerName *layer_name***

Required keyword and argument specifying the original name of a layer.

- **-minDRCWidth *min_drc_width_value***

Required keyword and argument specifying the minimum DRC width in microns of a layer. Use this keyword to ensure that the MBH generated hints do not violate the minimum DRC width rules.

- **-minDRCSpace *min_drc_space***

Keyword and argument specifying the minimum DRC space in microns between same layer exterior edges. Use this keyword to ensure that the MBH generated hints do not violate the minimum DRC spacing rule.

Either -minDRCSpace or the keyword set (-minLETLESpacing, -minETESpacing, -minETLESpacing) must be used, but not both.

- **-minLETLESpacing *min_letle_spacing***

Keyword and argument specifying the minimum spacing in microns between same layer exterior line ends. Use this keyword to ensure that the MBH generated hints do not violate this spacing rule.

Either -minDRCSpace or the keyword set (-minLETLESpacing, -minETESpacing, -minETLESpacing) must be used, but not both.

- **-minETESpacing *min_ete_spacing***

Keyword and argument specifying the minimum spacing in microns between same layer exterior edges (not a line-end). Use this keyword to ensure that the MBH generated hints do not violate this spacing rule.

Either -minDRCSpace or the keyword set (-minLETLESpacing, -minETESpacing, -minETLESpacing) must be used, but not both.

- **-minETLESpacing *min_etle_spacing***

Keyword and argument specifying the minimum spacing in microns between same layer exterior edges and line ends. Use this keyword to ensure that the MBH generated hints do not violate this spacing rule.

Either -minDRCSpace or the keyword set (-minLETLESpacing, -minETESpace, -minETLESpacing) must be used, but not both.

- **-lineEndLength *line_end_length***

Optional keyword and argument that provide the distance criteria in microns used to determine if an edge is on a line end or not. A line end is defined as an edge that is less than or equal to -lineEndWidth and located between two convex corner shapes, each of which is at least -lineEndLength long.

The default value is $1.5 * \text{-minDRCWidth}$.

- **-lineEndWidth *line_end_width***

An optional keyword and argument that provide the distance criteria in microns used to determine if an edge is on a line end or not. A line end is defined as an edge that is less than or equal to -lineEndWidth and located between two convex corner shapes, each of which is at least -lineEndLength long.

The default value is -lineEndLength.

- **-enablingVar {’env_var [env_varN [0]]...‘’}**

Optional keyword and list argument used to specify the environment variable argument for each layer defined within the layer section of the MBH map file. Using this keyword turns off the simulation of any unnecessary layers (DRC clean layers that were not moved by MBH during the validation run). The -enablingVar keyword does not accept an empty list argument. More than one environment variable can be specified.

For example, there are two scenarios in the case of using -enablingVar {RUN_metal1_M1}:

- metal1_M1 is a DRC clean layer used in a given check. The hint being validated resulted in the movement of fragments belonging to that layer.
- metal1_M1 belongs to either -layer1 or -layer2.

In both cases, during hint validation the environment variable RUN_metal1_M1 is automatically set to a value of 1 (simulation), else it is set to 0 (no simulation).

You may specify a value of 0 for the environment variable to disable that layer simulation. However, an environment variable for a layer must be enabled if that layer is required for the simulation of any of the checks in the input RDB file. For example:

`-enablingVar {RUN_M1 RUN_M2 0}`

Setting the environment variable RUN_M2 to a value of 0 disables RUN_M2 for the entire run. This provides a mechanism to switch off layer simulation from within the MBH map file.

Additionally, you must specify the following when using the -enablingVar:

- Define the -enablingVar keyword and environment variable(s) argument for each layer in the layer operations section of the MBH map file.

- Define the environment variables in the contour generation section of your LFD rule file.
- Define the environment variables in the run file or user shell before invoking MBH, unless the -inputLayersList keyword is specified in the MBH TVF rule file.

Check Definitions

- **Check**

Required keyword that defines the starting point for the check operations section of the map file.

- **-checkName *check_name***

Required keyword and argument specifying the check name as it is explicitly stated in the previously generated input RDB file with error markers. A warning message is issued when a check in the map file does not exist in the input RDB file.

- **-errorType {width | space | stitching}**

Required keyword and argument specifying the error marker type. You specify one of three error marker types. The **stitching** argument is only applied to checks that use -layer2, as in the case of dual layer checks.

- **-layer1 *layer1_name***

Required keyword and argument specifying the original name of the layer used in the single layer check, and the first layer original name in case of a dual layer check.

- **-layer2 *layer2_name***

Optional keyword and argument specifying the second layer original name in the case of a dual layer check.

- **-drcCleanLayers {'*layer3_name layer4_name layer5_name...layerN_name*'}**

Optional keyword and space-separated list argument specifying original layer names that the MBH hints generation should maintain DRC clean. MBH generates hints that are DRC clean on **-layer1** and -layer2 (if it exists), but other layers are not necessarily DRC clean unless specified with this optional keyword and list argument.

If the -drcCleanLayers keyword and list argument are not defined, MBH generates hints that are clean on the layers that the specific check is operating on by default.

- **-validationLayers {'*validationLayer1 validationLayer2 validationLayer3...validationLayerN*'}**

Optional keyword and space-separated list argument specifying the layers used in the validation mode to verify that the MBH generated hints are LFD clean. The validated hints verify LFD cleanliness and ensure that no new hotspots are introduced. In addition, this keyword checks that no valid hints are claimed as invalid.

Interlayer Spacing

- **InterLayer**

Required keyword that defines the starting point for the interlayer spacing operations of the map file. Interlayer spacing is used to maintain interlayer DRC cleanliness, and is used for single-patterning and multi-patterning specifications as well.

Interlayer spacing rules for any pair of layers work in three ways:

- Define an enclosure distance
- Define an interlayer space
- Define both enclosure and interlayer space

For each layer defined in the -drcCleanLayers option of a check, there must be an interlayer spacing between the layer and the layer(s) used for the check (layer1 and layer2, if it exists).

For multi-patterning there must be an interlayer spacing defined between the two layers used for the check (layer1 and layer2).

- **-layer1 *layer1_name***

Required keyword and argument specifying the first layer original name. This is the enclosed layer name in the case that the -minDRCEncl keyword is defined.

- **-layer2 *layer2_name***

Required keyword and argument specifying the second layer original name. This is the surrounding layer name in the case that the -minDRCEncl keyword is defined.

- **-minDRCEncl *min_drc_encl***

Keyword and argument specifying the minimum enclosure distance in microns between layer1 and layer2. Use this keyword to ensure that the MBH generated hints do not violate this spacing rule.

At least one of -minDRCSpace, -minDRCEncl, or the spacing keyword set (-minLETLESpacing, -minETESpacing, -minETLESpacing) must be defined.

- **-minDRCenclSides ‘{’‘{’*width_constraint min_encl_side*‘}’[‘{’*width_constraint min_encl_side*‘}’ ...] ‘}’**

Optional keyword and parameter defining enclosure rules for enclosing layer sides between layer1 and layer2, where the enclosure rule can depend on the width of the layer1 shape.

- *width_constraint* — Defines the widths that the rule applies to. The width constraint is defined with the comparison operators “==”, “>”, “<”, “>=”, and “<=”. Open-ended width constraints are not allowed. For example, “> 0.005 < 0.03” is allowed, but “< 0.03” is not allowed.
- *min_encl_side* — A required parameter giving the minimum side enclosure distance in microns for the given *width_constraint*.

For example, this specification:

```
-minDRCenclSides {{== 0.2 0.1} {>= 0.1 < 0.2 0.07}}
```

causes this behavior:

For widths equal to 0.2, apply an enclosure side constraint of 0.1.

For widths ≥ 0.1 and < 0.2 , apply an enclosure side constraint of 0.07.

- `-minDRCenclLineEnds {{'width_constraint min_encl_end}}[{{'width_constraint min_encl_end}} ...]'`

Optional keyword and parameter defining enclosure rules for enclosing layer line ends between layer1 and layer2, where the enclosure rule can depend on the width of the layer1 shape.

- *width_constraint* — Defines the widths that the rule applies to. The width constraint is defined with the comparison operators “==”, “>”, “<”, “ \geq ”, and “ \leq ”. Open-ended width constraints are not allowed. For example, “ $> 0.005 < 0.03$ ” is allowed, but “ < 0.03 ” is not allowed.
- *min_encl_end* — A required parameter giving the minimum line end enclosure distance in microns for the given *width_constraint*.

For example, this specification:

```
-minDRCenclLineEnds {{==0.2 0.1} {>=0.1 <0.2 0.07}}
```

causes this behavior:

For widths equal to 0.2, apply an enclosure line end constraint of 0.1.

For widths ≥ 0.1 and < 0.2 , apply an enclosure line end constraint of 0.07.

- `-minDRCSpace min_drc_space`

Keyword and argument specifying the minimum DRC space in microns between different layers' exterior edges. Use this keyword to ensure that the MBH generated hints do not violate this spacing rule.

At least one of `-minDRCSpace`, `-minDRCEncl`, or the spacing keyword set (`-minLETLESpacing`, `-minETESpacing`, `-minETLESpacing`) must be defined.

The `-minDRCSpace` keyword cannot be specified with the spacing keyword set (`-minLETLESpacing`, `-minETESpacing`, `-minETLESpacing`).

- `-minLETLESpacing min_letle_spacing`

Keyword and argument specifying the minimum spacing in microns between different layers' exterior line ends. Use this keyword as part of the spacing keyword set (`-minLETLESpacing`, `-minETESpacing`, `-minETLESpacing`) to ensure that the MBH generated hints do not violate the spacing rules.

This keyword cannot be defined if `-minDRCSpace` is defined and must be defined if `-minDRCSpace` or `-minDRCencl` is not defined.

- `-minETESpacing min_ete_spacing`

Keyword and argument specifying the minimum spacing in microns between different layers' exterior edges (not a line-end). Use this keyword as part of the spacing keyword set (`-minLETLESpacing`, `-minETESpacing`, `-minETLESpacing`) to ensure that the MBH generated hints do not violate spacing rules.

This keyword cannot be defined if `-minDRCSpace` is defined and must be defined if `-minDRCSpace` or `-minDRCencl` is not defined.

- `-minETLESpacing min_etle_spacing`

Keyword and argument specifying the minimum spacing in microns between different layers' exterior edges and line ends. Use this keyword as part of the spacing keyword set (`-minLETLESpacing`, `-minETESpacing`, `-minETLESpacing`) to ensure that the MBH generated hints do not violate spacing rules.

This keyword cannot be defined if `-minDRCSpace` is defined and must be defined if `-minDRCSpace` or `-minDRCencl` is not defined.

- `-stitching { {'list_of_width_length_pairs'} | {'default_length'} | {'{'list_of_width_length_pairs'} default_length'}}`

Optional keyword and argument specifying the stitching rules applied between stitch fragments. The list argument options are width-length pair(s) followed by a default length constraint. Units are microns.

Note



The `-stitching` keyword option is only applied to checks that use `-layer2`.

How stitching rules are applied between stitch fragments:

- **Default Length Only** — Default length constraint value is applied to all fragment widths.
- **Width-Length Pairs Only** — No rules are applied to any fragment widths that are greater than the greatest value in the pair list. However, the length rules are applied to the widths defined in the pair list.
- **Both Default Length and Width-Length Pairs** — All defined width-length pair length values are applied. The default length constraint applies to all fragment widths greater than the greatest value in the pairs list.
- **Stitching Length-Width** — Minimum length check of both of the stitching edges after the hint movement takes place. The corresponding width-length rule that matches the new length is then applied.

For example, given the stitching rules `{0.05 0.12} {0.06 0.15}` and the following conditions:

- Before hint movement, the lengths of the stitching edges are 0.06 and 0.06.
- After hint movement, the lengths of the stitching edges are 0.05 and 0.06.

The first rule {0.05 0.12} is applied.

Examples

Layer Section Example 1

The -enablingVar keyword option and argument is defined for all layers specified in the layer section of the map file.

```
Layer
  -layerName metall_M1
  -minDRCWidth 0.025
  -minDRCSpace 0.06
  -enablingVar    {RUN_M1}

Layer
  -layerName metall_M2
  -minDRCWidth 0.025
  -minDRCSpace 0.06
  -enablingVar    {RUN_M1}

Layer
  -layerName contact
  -minDRCWidth    0.07
  -minDRCSpace    0.02
  -enablingVar    {RUN_C}
```

Layer Section Example 2

The environment variables for the -enablingVar keyword option are defined in the contour generation section of the LFD rule file.

```
LFD::Begin
LFD::loadPDK...

#Register layers

If ${::env(RUN_metal1_M1)} {
LFD::RegisterLayer -DesignLayer metall_M1...
}
If ${::env(RUN_C)} {
LFD::RegisterLayer -Designlayer contact...
}

If ${::env(RUN_M1)} {
LFD::MinSpaceCheck "-layer metall..."
}
If ${::env(RUN_C)} {
LFD::MinWidthCheck "-layer contact..."
}
LFD::End
```

Layer Section Example 3

The environment variables for the -enablingVar keyword option must be defined in the run file or user shell before invoking MBH, unless the -inputLayersList keyword is used.

```
#!/bin/sh
export RUN_metal1 M1=1
export RUN_metal1_M2=1
export RUN_C=1
export RUN_M2=1
```

Check Definitions Section Example 1

The check definitions section of the map file is used to specify check names and include validation layers for LFD and DRC. Layers -layer1 and -layer2 (if exists) are maintained DRC clean by default.

```
Check
  -checkName          M1_MinSpaceCheck
  -errorType          space
  -layer1             metal1_M1
  -validationLayers   {M1_MSC_ALL}
  -drcCleanLayers     {contact}
```

Interlayer Spacing Section Example 1

The interlayer section of the map file can be used to specify via enclosure rules.

```
InterLayer
  -layer1 via
  -layer2 metal
  -minDRCenc1 0.001
  -minDRCenc1Sides {{==0.032 0.000} {>0.032<=0.070 0.012}}
  -minDRCenc1LineEnds {{==0.032 0.025} {>0.032<=0.070 0.012}}
```

Where the -minDRCenc1Sides values are applied as follows:

For a width that is 0.032, apply an enclosure side constraint of 0.000.

For a width that is > 0.032 and <= 0.070, apply an enclosure side constraint of 0.012.

Likewise, the -minDRCenc1LineEnds values are applied as follows:

For a width that is 0.032, apply an enclosure line-end constraint of 0.025.

For a width that is > 0.032 and <= 0.070, apply an enclosure line-end constraint of 0.012.

Interlayer Spacing Section Example 2

The interlayer section of the map file can be used to specify stitching width-length pairs and a default length constraint.

```
-stitching { {0.12 0.12} {0.52 0.32} 0.4 }
```

For a width that is ≤ 0.12 , apply a length constraint of 0.12.

For a width that is ≤ 0.52 , and > 0.12 apply a length constraint of 0.32.

For a width that is greater than 0.52, apply a default length constraint of 0.4.

Interlayer Spacing Section Example 3

The interlayer section of the map file can be used to specify stitching width-length pairs without a default length constraint.

```
-stitching { {0.12 0.12} {0.52 0.32} }
```

For a width that is ≤ 0.12 , apply a length constraint of 0.12.

For a width that is ≤ 0.52 and > 0.12 , apply a length constraint of 0.32.

For a width that is greater than 0.52, no rule is applied.

Interlayer Spacing Example 4

The interlayer section of the map file can be used to specify only a default length constraint.

```
-stitching {0.4}
```

For all widths, the 0.4 length constraint is applied.

Related Topics

[Running Calibre LFD with MBH](#)

[GenerateHints](#)

[Sample MBH Map File](#)

MBH Transcript File Information

The MBH transcript file outputs run-specific information that is useful for debugging and reviewing MBH run results.

When you complete an MBH run, entries appear at the bottom of the transcript file that show MBH runtime and hint statistics.

For example:

```
MBH: TOTAL HINT VALIDATION TIME(s) = 54
MBH: CUMMULATIVE HINT GENERATION TIME(s) = 267
MBH: CUMMULATIVE HINT VALIDATION TIME(s) = 3666
MBH: PERCENTAGE OF VALID HINTS/GENERATED HINTS: 12.38%
Hint 1 was valid -> 18 times.
Hint 2 was valid -> 6 times.
Hint 3 was valid -> 6 times.
Hint 4 was valid -> 6 times.
Hint 5 was valid -> 3 times.
MBH: GENERATE HINTS REAL TIME = 3936
Warning 2: No database changes. Nothing to do.
Elapsed Time = 0:0:0:1:7:32
```

Modifying the Design

After reviewing the Calibre LFD results and deciding which errors you want to fix first, you can modify the design to correct the problems. Working interactively allows you to re-run Calibre LFD to see how effective your modifications are.

Prerequisites

- You are using a layout viewer supported by Calibre RVE.

Procedure

1. Open the design database in your layout editor.
2. In the layout viewer, choose **Verification > Start RVE** (for Calibre viewers) or **Calibre > Start RVE** (for most other design tools).
3. In the Calibre RVE dialog box, select the **DRC/ERC** database type, supply the path to the [Check Database](#), and then click **Open**.
4. Expand the hierarchy on the left side of the Calibre RVE window to highlight the error you want to fix in the viewer.
5. In Calibre RVE, enable **View > Check Text Pane** and review the check text in the lower portion of the window. This text describes:
 - The check that was violated.
 - Recommendations for fixing the problem.

For example:

Minimum Space Violation: Improve symmetry. If feature is dense, increase spacing. Reduce surrounding feature width.

6. If you need additional information, display the PV-bands as described in “[Viewing PV-Band Data](#)” on page 124.

7. In the layout editor, modify the problem feature.
8. To see whether or not the change corrected the problem, you can re-run Calibre LFD as described in “[Re-Running Calibre LFD Checks on a Modified Design](#)” on page 151.
9. Repeat for other errors.

Re-Running Calibre LFD Checks on a Modified Design

Using Calibre Interactive, you can check your design modifications before you commit to them by saving the design. You do this by running the Calibre LFD checks on a selected portion of the layout.

Prerequisites

- You are in the middle of an editing session, with your layout editor, Calibre RVE, and Calibre Interactive all running and configured properly.

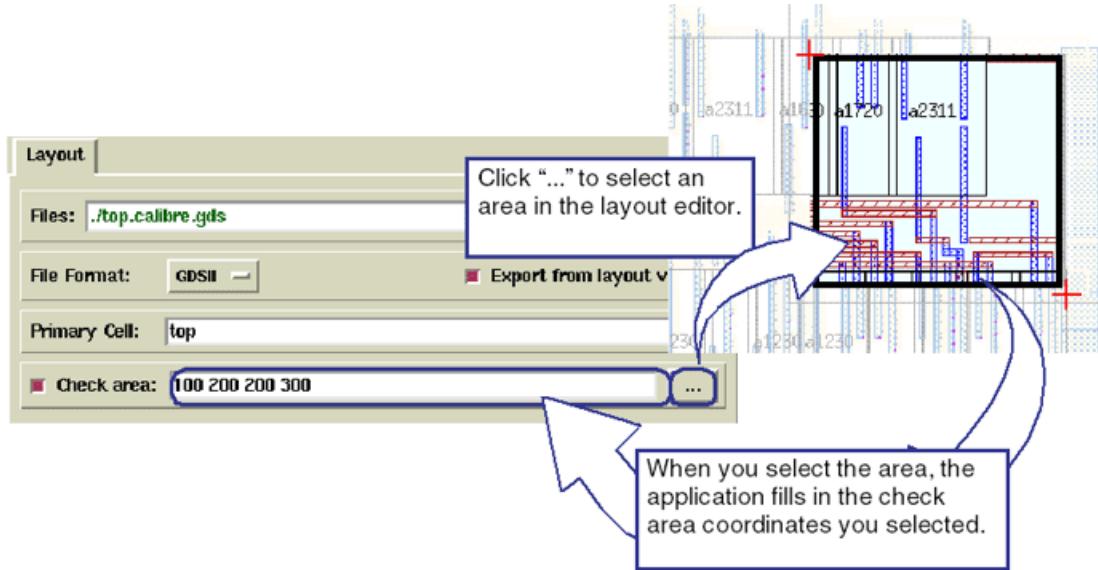
Note

 You do not need to save your modifications before re-running the checks. This allows you to check your modifications before you commit to them.

Procedure

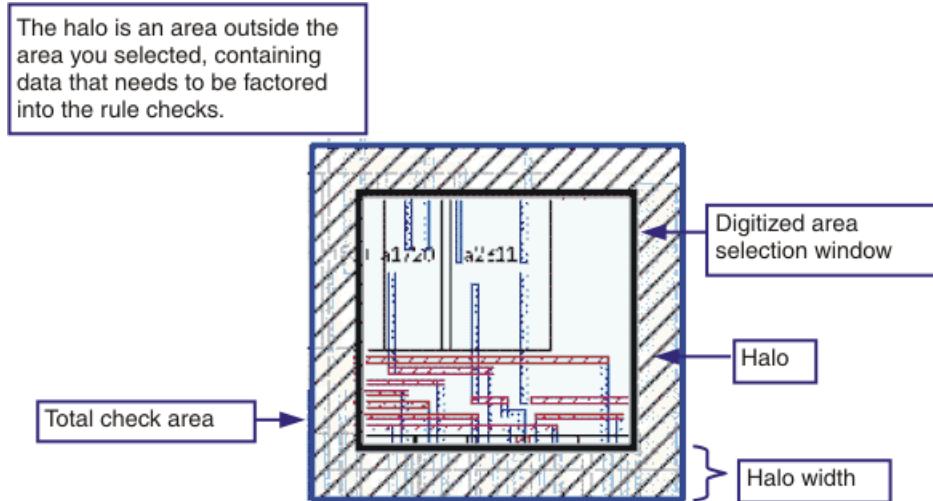
1. In Calibre Interactive, click **Inputs** on the left panel and do the following:
 - a. Make sure “Litho Friendly Design (LFD)” is selected for the Run type.
 - b. Enable “Export from layout viewer.”
2. Enable the “Area” check box on the Inputs pane.
3. Click the browser (...) button next to the “Area” entry field.

4. In the layout editor, click and drag out an area selection rectangle. The coordinates of the area selection rectangle appear in the “Area” section of the Inputs pane.



When performing Calibre LFD checks, the area that is actually checked is the specified area plus a halo region around the area.

5. To control the halo size, do the following:
 - Select **Setup > DFM Options** to open the DFM Options pane,
 - Select the **Area DFM** tab.
 - Adjust the “Halo width for area DFM” setting.



Note

 When importing layout data from the layout editor, Calibre Interactive clips polygons that overlap the area boundary, ignoring those portions of clipped polygons that fall outside of the boundary.

See the section “[Performing Area DRC](#)” in the *Calibre Interactive User’s Manual* for complete details on the options available for Area DFM runs.

6. Click **Run LFD** to run the full set of Calibre LFD checks.

rdb_flattener

The stand-alone rdb_flattener utility provides RDB file flattening functionality from the command line. Typically, this utility is used by designers in the place and route stage of the design flow.

Usage

```
rdb_flattener
  -rdb input_rdb_file
  -layout input_layout_file
  -o output_rdb_file
  [-rdb_topcell | -topcell topcell_name_to_flatten | -layout_topcell]
```

Description

You can use the **rdb_flattener** utility from the command line to output a flattened RDB file. By default, the **rdb_flattener** flattens the input RDB file with respect to the top cell name specified in the input RDB file. The flattened RDB output file maintains all the shape information in the context of the layout top cell space. Additional options support flattening of the input RDB file with respect to a user specified top cell name or an input layout top cell name.

Arguments

- **-rdb *input_rdb_file***

Required keyword and argument that specify the name of the input RDB file to be flattened.

- **-layout *input_layout_file***

Required keyword and argument that specify the name of the input layout file. The format of the layout file must be either OASIS or GDS.

- **-o *output_rdb_file***

Required keyword and argument that specify the name of the flattened output RDB file.

- **-rdb_topcell**

Optional keyword specifying that the input RDB file be flattened with respect to the top cell name in the input RDB file. This is the default if the -topcell or -layout_topcell options are not specified.

- **-topcell *topcell_name_to_flatten***

Optional keyword and argument that specify that the input RDB file be flattened with respect to a user specified top cell name. If the specified top cell name is not in the layout, an error is generated and the utility terminates.

- **-layout_topcell**

Optional keyword specifying that the input RDB file be flattened with respect to the top cell name in the input layout file (must be a unique name).

Examples

Example 1

In this example, the input RDB file is flattened in the context of the top cell name that is in the RDB input file.

```
% rdb_flattener -rdb my_input.rdb -layout my_input.layout -o my_flat.rdb
```

Example 2

In this example, the input RDB file is flattened in the context of the top cell name that is in the RDB input file using the optional `-rdb_topcell` default argument. The behavior is the same as that of Example 1.

```
% rdb_flattener -rdb my_input.rdb -layout my_input.layout -o my_flat.rdb  
-rdb_topcell
```

Example 3

In this example, the input RDB file is flattened in the context of the user specified top cell name using the optional `-topcell` argument.

```
% rdb_flattener -rdb my_input.rdb -layout my_input.layout -o my_flat.rdb  
-topcell my_top_cell_name
```

Example 4

In this example, the input RDB file is flattened in the context of the layout top cell name using the optional `-layout_topcell` argument.

```
% rdb_flattener -rdb my_input.rdb -layout my_input.layout -o my_flat.rdb  
-layout_topcell
```


Chapter 6

For Designers — Incorporating Calibre LFD Into Your Timing Simulations

As a designer, you can capture process variations and improve design robustness by combining the lithography and edge simulation data from Calibre LFD tools with timing analysis.

About Model Based Extraction	157
The Importance of MBE	158
MBE and Your Timing Analysis Flow.....	159
About the CSG Function.....	161
About the CSI Function	163
Performing Model Based Extraction	166

About Model Based Extraction

The process of combining Litho-Friendly Design (LFD) simulations with timing simulations is referred to as Calibre Model Based Extraction (MBE).

You can do the following with Calibre Model Based Extraction:

- Predict timing and performance based on simulated silicon images rather than drawn shapes.
- Evaluate a design's performance through process variations.
- Make circuit simulation-based comparisons of multiple versions of a layout with respect to variability through process window.
- Extract parasitics for simulated silicon images rather than drawn shapes.

Model-based extraction is supported by two functions within the Calibre LFD software that translate process variation data into data you can feed into your extraction tools:

- **Contour Simplification for Gates (CSG)** — Calculates gate L adjustment and gate W adjustment to feed into Calibre® nmLVS™.
- **Contour Simplification for Interconnect (CSI)** — Calculates RC interconnect adjustment (rectangles) to feed into parasitic extraction tools.

Related Topics

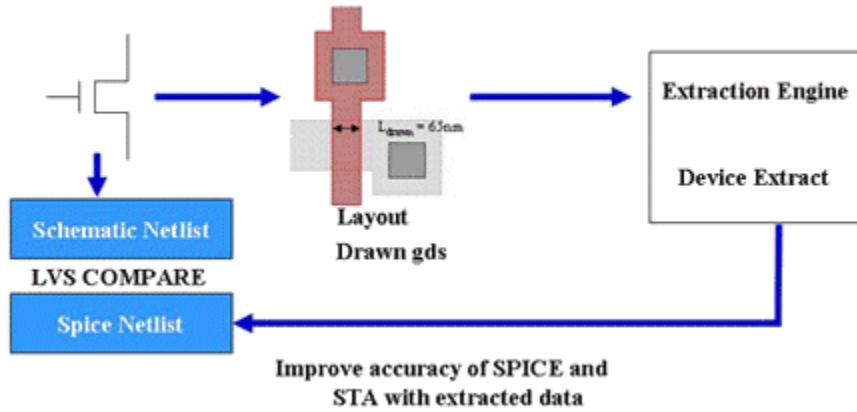
[CSG](#)

CSI

The Importance of MBE

In the typical timing analysis flow, extraction is performed based on the drawn GDS layout.

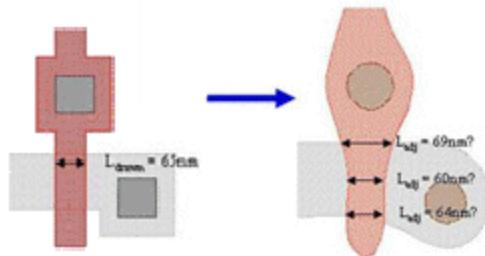
Figure 6-1. Typical Timing Analysis Flow



In the real world, what is printed and etched (polygons printed in silicon) can be quite different from drawn shapes. This difference is even greater when you take into account process variation and context dependency.

- *Process variation* refers to the fact that the actual process conditions vary from one die to another.
- *Context dependency* refers to the fact that identical cells printed in different locations actually print differently, and hence behave as if they are two different cells.

Figure 6-2. How Printing Affects Drawn Shapes



At smaller technology nodes, the differences between the drawn shapes and silicon are much more significant. Producing robust designs with satisfactory performance and accuracy requires a solution such as the Calibre LFD Model-Based Extraction flow, which allows designers to

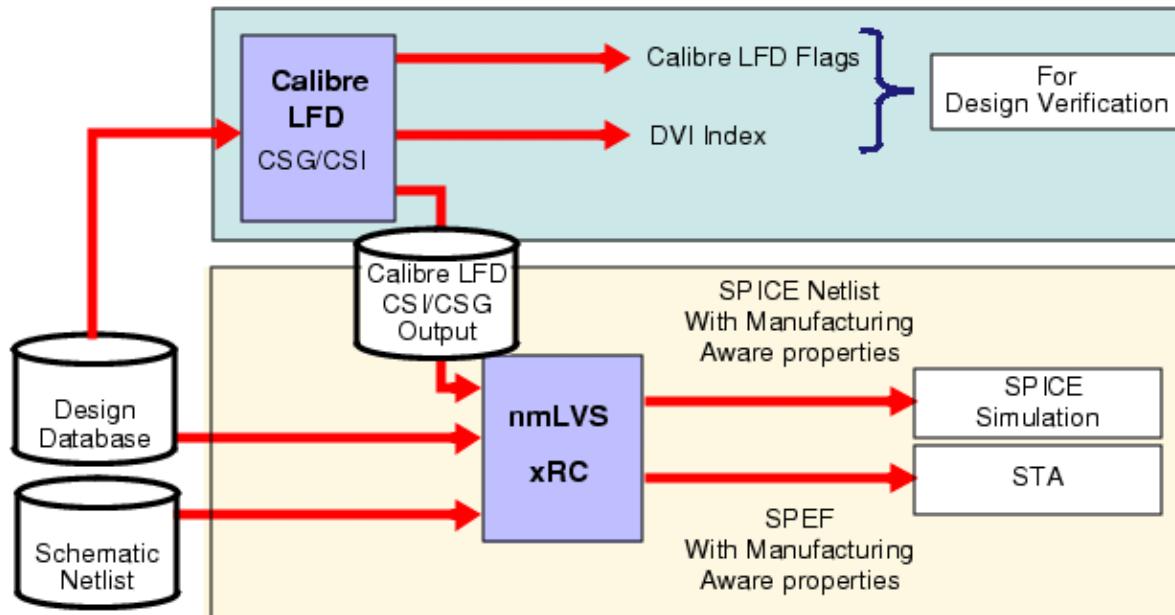
anticipate the impact of manufacturing on designs. Model-based extraction captures the silicon contour information and allows you to perform the following actions:

- Manufacturing-aware SPICE simulations for timing and power.
- Manufacturing-aware interconnect extraction for accurate timing closure (STA and SSTA).

MBE and Your Timing Analysis Flow

Feeding Calibre LFD contour data forward into timing and power analysis is made possible by the Calibre LFD functions CSG (Contour Simplification for Gates) and CSI (Contour Simplification for Interconnect). These functions translate process variation data into the data that you can feed directly into your extraction tools.

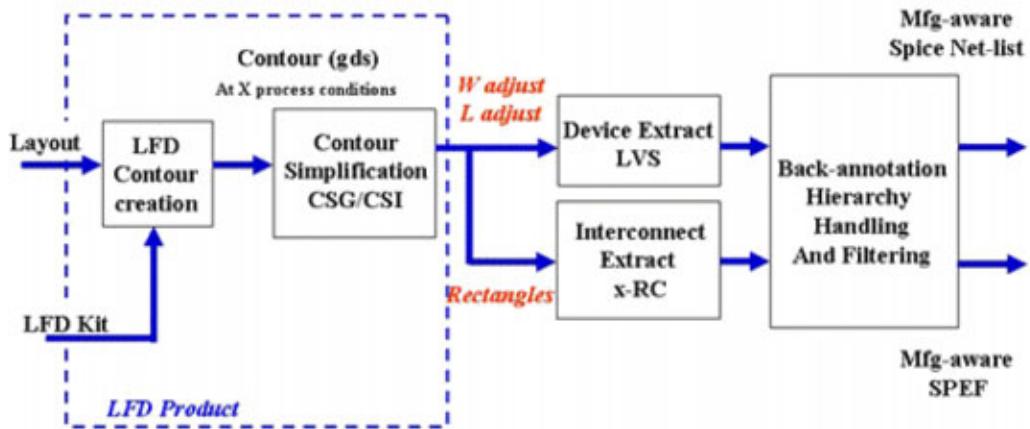
Figure 6-3. Calibre LFD to Calibre nmLVS Flow



The CSG function calculates the gate L adjustment and gate W adjustment to feed into Calibre nmLVS. The CSI function calculates the RC interconnect adjustment (rectangles) to feed into parasitic extraction tools.

Inputs to the Calibre LFD portion of the flow include your layout, as well as a Calibre LFD kit specific to your foundry and process. Based on these inputs, the Calibre LFD toolset generates contours for each of the process conditions. The CSG function calculates the W adjusted (W_{adj}) values and L adjusted (L_{adj}) values, and writes data as files that can be used as input to your LVS tools. The CSI function generates rectangles for the interconnect layer, and writes that data as layers that can be used as input to your parasitic extraction tools.

Figure 6-4. CSG / CSI Inputs and Outputs



Output files have been carefully designed to feed directly into existing extraction tools. Using the Calibre LFD MBE flow *does not* require re-qualification of your LVS and extraction tools.

Note

 Calibre LFD is performed at the RET database unit level to ensure accuracy. This translates into a grid that is much finer than the design grid.

Related Topics

[CSG](#)

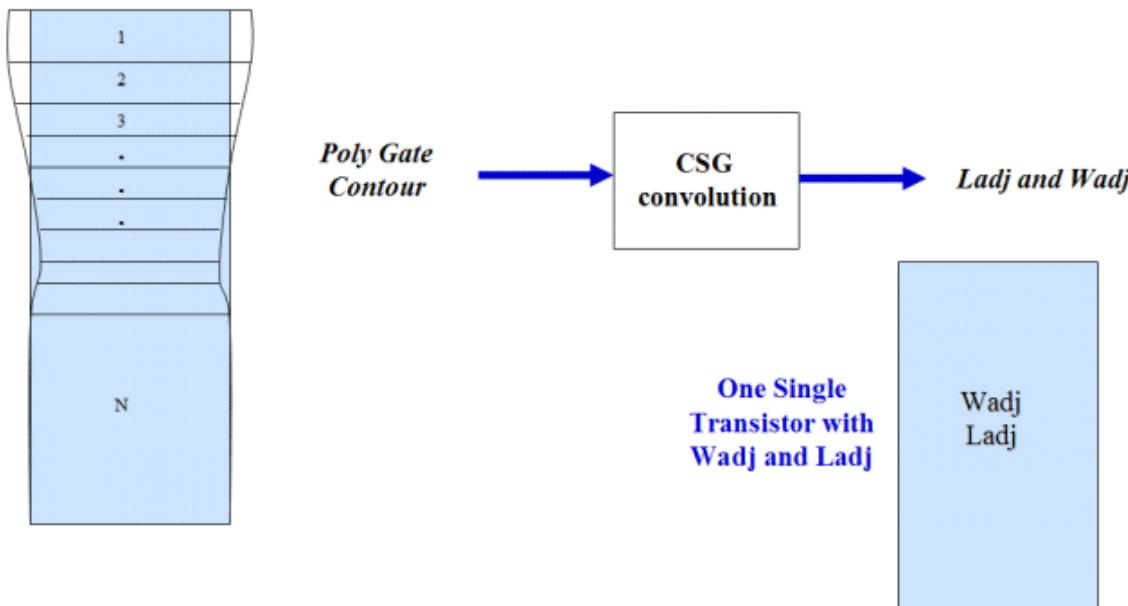
[CSI](#)

About the CSG Function

The CSG function generates an output file that fills the gap between Calibre LFD and Calibre nmLVS. The gap exists because LVS tools do not recognize the irregular contours generated by Calibre LFD.

The **CSG** function analyzes each gate portion in a Calibre LFD contour that is within the area under investigation. The process begins by identifying the transistor contours, then dividing the contour for the transistor into subsections. The tool then evaluates each subsection of the transistor to derive an L_{adj} and W_{adj} for the device. You control the number of subsections into which the transistor is divided through the **-resolution** command keyword.

Figure 6-5. Understanding L_{adj} and W_{adj}



Note

- The assumption when running CSG is that your design is Calibre LFD clean and contains no hard pinch in the transistor. The transistor must stay within range of the SPICE model.
-

CSG Data and SPICE Model Formulas	161
Files Generated by the CSG Function	162

CSG Data and SPICE Model Formulas

To understand how CSG data is fed forward into the extraction tools, you can look at how this data fits into the SPICE model formulas for L_{eff} and W_{eff} .

The SPICE model formula for L_{eff} is based on two components:

- L_{DRAWN}
- XL — accounts for channel length offset due to lithography (mask) and etch effects.

$$L_{\text{eff}} = L_{\text{DRAWN}} + XL$$

Similarly, the SPICE model formula for W_{eff} is based on two components:

- W_{DRAWN}
- WL — accounts for channel width offset due to lithography (mask) and etch effects.

$$W_{\text{eff}} = W_{\text{DRAWN}} + WL$$

Related Topics

[CSG](#)

Files Generated by the CSG Function

The output of the CSG function is an ASCII file containing valid SVRF LAYOUT TEXT commands that you include in your Calibre nmLVS run. A properly designed Calibre LFD CSG rule file generates one output file for each poly condition / active condition combination. A results database (RDB) file can also be generated.

The CSG File

Each line of the CSG output file contains one **LAYOUT TEXT** statement, where each statement represents either the L_{adj} or W_{adj} for an individual transistor. You include the files in your Calibre nmLVS run using the Include statement.

The basic syntax follows:

LAYOUT TEXT *text* *x* *y* *layer* *cell_name*

where:

- *text*
Required argument defining the value computed by CSG. This is supplied in quotes.
- *x* *y*
Required argument defining the pair of coordinates specifying the center of the transistor.
- *layer*

Required argument defining the name of the layer with which the text are associated.
This is either L_{adj} or W_{adj} .

- `cell_name`

Optional argument defining the name of the cell in which the transistor was found.

Results Database Output

A results database (RDB) file can be generated if the following option is used:

CSG

-outputFile *file_name*

{ -property {yes | no}

{ -database *db_name* | -layerOut *csg_layer* | -database *db_name* -layerOut *csg_layer* }

where:

- `-property {yes | no}`

Optional keyword used to enable or disable the generation of the properties *ladj* and *wadj*. When “-property yes” is specified, you must also specify at least one of -database or -layerOut. By default, properties are disabled. This argument is only used with “-method default”.

- `-database db_name`

Optional keyword and argument for use with *-property yes*, defining the database to which output data are written.

- `-layerOut csg_layer`

Optional keyword and argument for use with *-property yes*, instructing the tool to generate an output layer containing geometries on the seed layer, with properties attached.

Related Topics

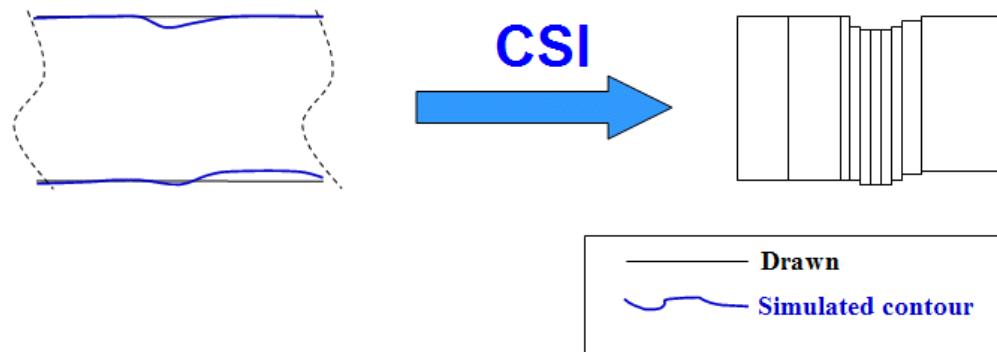
[CSG](#)

[Generating CSI Layers and CSG Files](#)

About the CSI Function

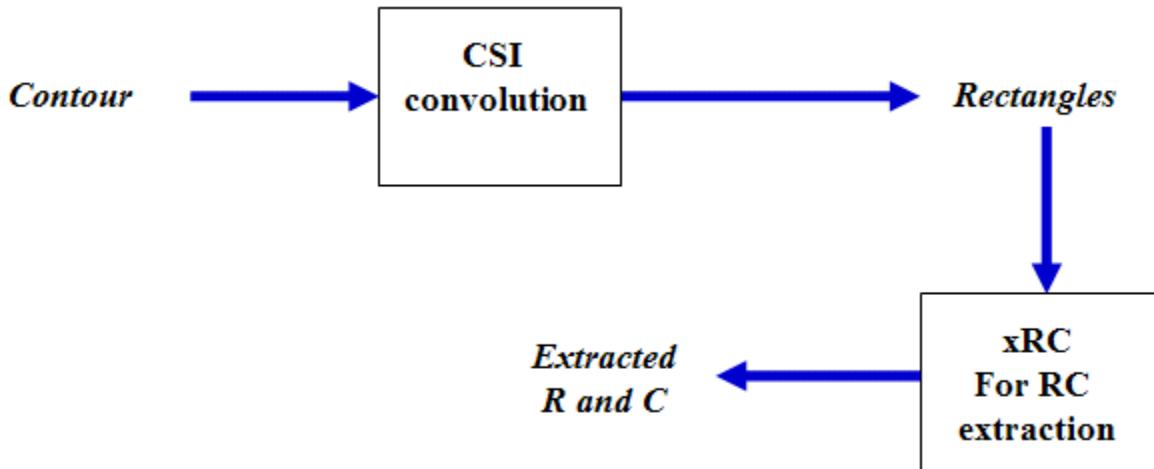
The CSI function takes simulated contours for each interconnect layer and breaks them down into component rectangles.

Figure 6-6. Contour Simplification



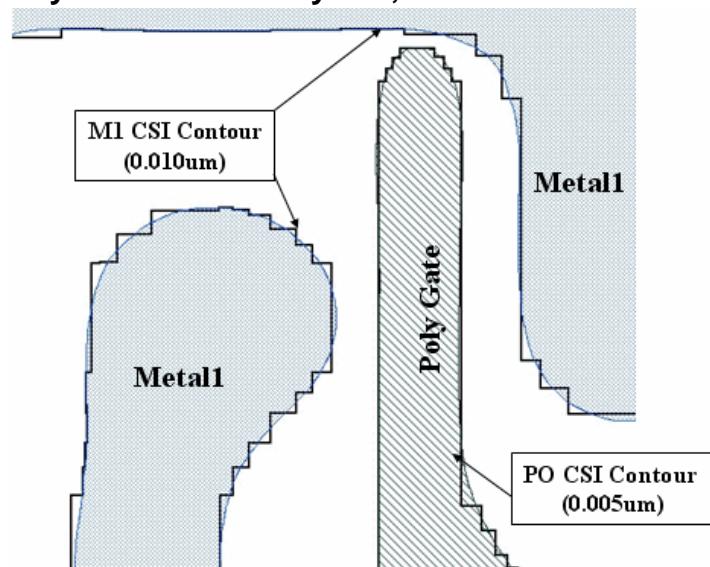
The output from **CSI** is a new layer containing the simplified versions of the contours generated for the input layer. These output layers can be fed directly into the parasitic extraction tools, in place of the drawn layers.

Figure 6-7. Using Simplified Contours for RC Extraction



You define the level of accuracy needed to obtain the desired speed and optimization. You do this by defining the sizes of the steps through the -deviation keyword. Larger -deviation values result in coarser simplified contours, as illustrated in [Figure 6-8](#).

Figure 6-8. Layers Generated by CSI, Shown over Associated Contours



Typically, you generate one simplified contour for each layer and process condition combination.

Related Topics

[Generating CSI Layers and CSG Files](#)

Performing Model Based Extraction

The typical timing analysis flow with Model Based Extraction includes Calibre LFD, Calibre nmLVS, and parasitic extraction.

Figure 6-9 shows a typical timing analysis flow using Model Based Extraction. Table 6-1 provides a description of each extraction type.

Figure 6-9. Timing Analysis Flow Using MBE

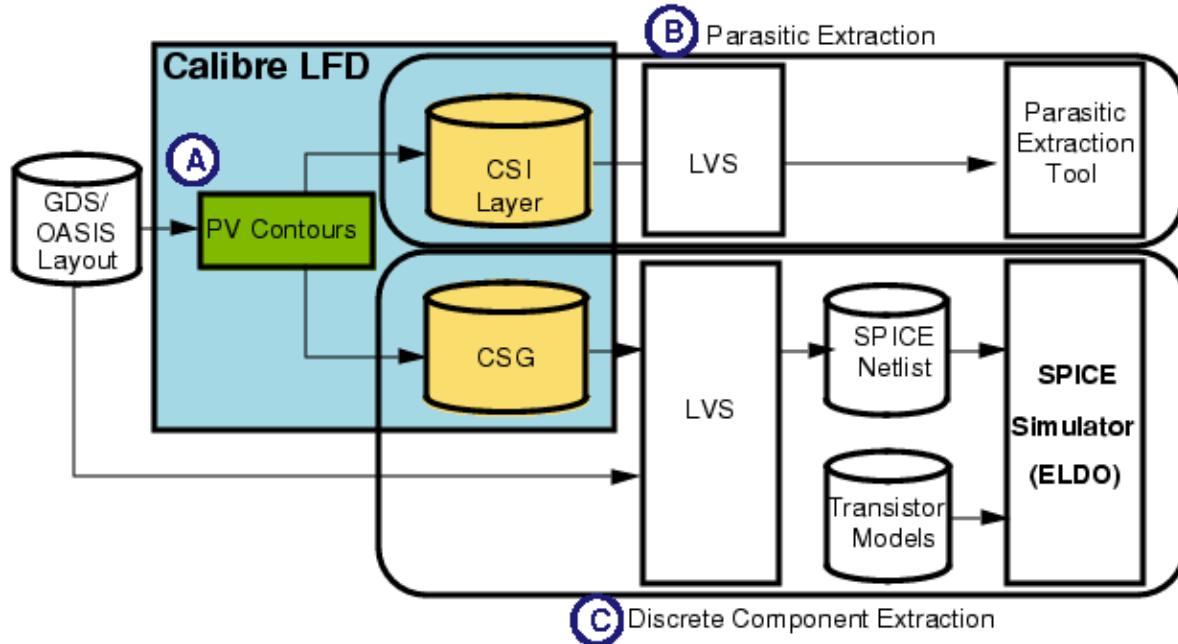


Table 6-1. The Timing Analysis Flow Using MBE

Type of Extraction	Description
A	Run Calibre LFD to generate the CSI layers and the CSG files for each poly condition/active condition combinations.
B	Run parasitic extraction on the simplified interconnect (CSI) layers.
C	Run discrete component extraction on each based on a CSG file containing the measurements that represent the adjusted dimensions of the transistor.

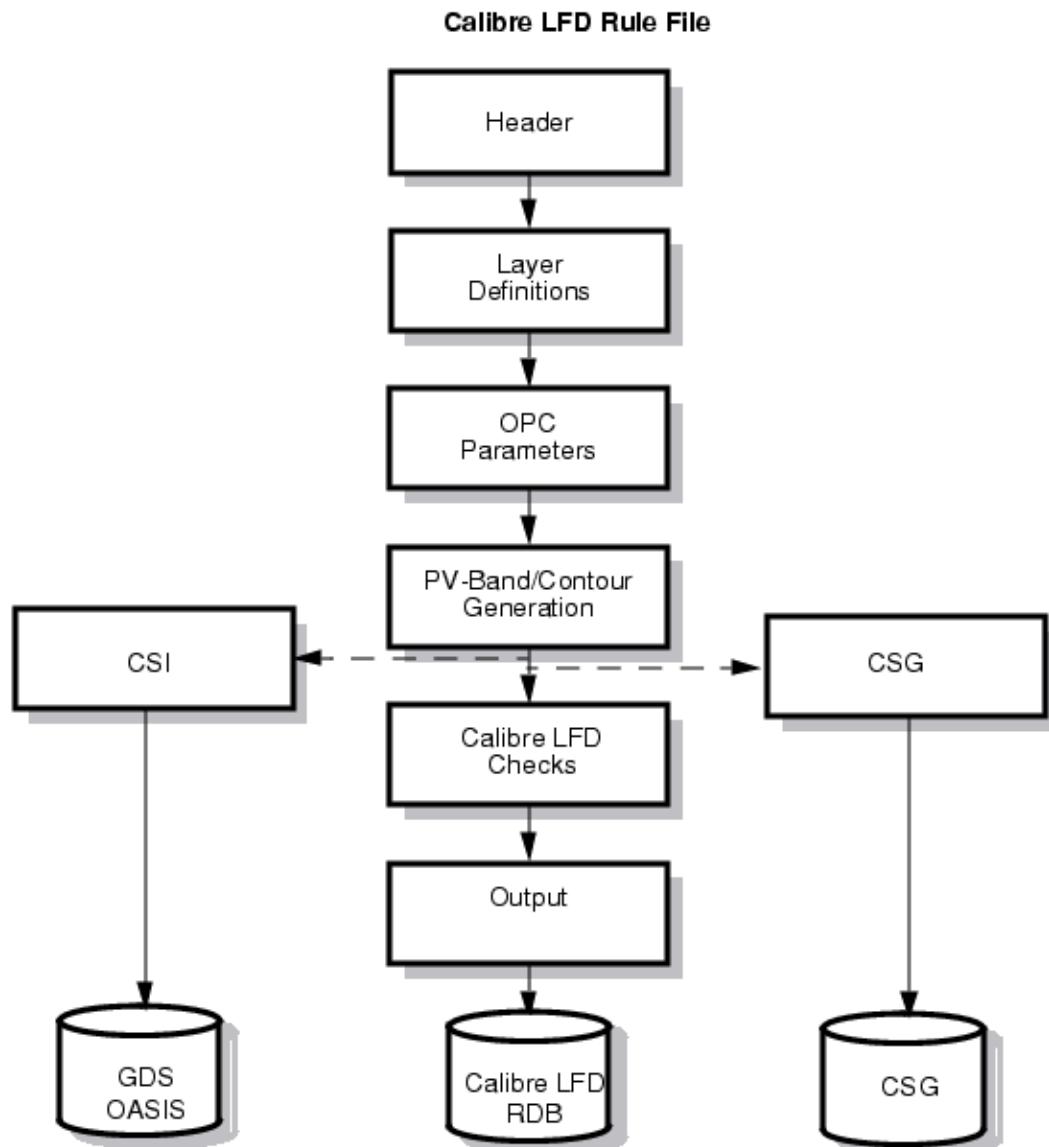
Generating CSI Layers and CSG Files	167
Sample MBE Calibre LFD Rule File	169

Generating CSI Layers and CSG Files

CSI layers and CSG files are two different outputs generated based on Calibre LFD contours. Typically, you create a single rule file that produces both these outputs. When these functions are incorporated into a standard Calibre LFD rule file, you can perform standard Calibre LFD checks at the same time that it generates the simplified contours and the CSG file.

This procedure describes a basic procedure for creating such a rule file based on an existing Calibre LFD rule file, and then using it to generate the outputs required by MBE.

Figure 6-10. Calibre LFD Rule File for Running CSI and CSG



Prerequisites

- The layout database for the design.

- The Calibre LFD kit specific to the foundry and process to be used.

Procedure

1. Edit the TVF rule file as needed, based on these guidelines:
 - a. Use standard TVF statements to generate a “seed” layer identifying the gate regions.
 - b. Make sure the rule file contains the LFD::PVband functions needed to generate PV-bands for each process condition on each layer (typically poly, active and metal).
 - c. Include the CSG function once for each poly condition / active condition pair, writing the output of each function to a separate file.
If you are investigating N process conditions for the poly layer and M process conditions for the active layer, you issue the CSG function N*M times.
- d. Include the CSI function once for each process condition for the poly layer, using the -layerOut to write the output to a unique layer. (Optional)
- e. Include the CSI function once for each process condition for the metal layer(s). using -layerOut to write the output to a unique layer.(Optional)
- f. Include DRC CHECK MAP statements to write each of the CSI output layers to the CSI GDS or OASIS database.
- g. For each layer under investigation, make sure there are DRC CHECK MAP statements to write the original drawn layers and the layers containing the contours at nominal conditions to the RDB.
- h. Create ASCII summary file of all transistor, all process conditions, both designs, for spreadsheet analysis. (Optional)

Note

 See “[Sample MBE Calibre LFD Rule File](#)” on page 169 for a typical Model Based Extraction rule file.

2. Run Calibre LFD using the rule file you modified in step 1. Be sure to supply the -lfd and -hier switches. In most cases you also want to run using -turbo and -turbo_litho.

When the run completes you have the following files:

- One CSG file for each poly condition/ active condition pair. These files are located in the output directory.
- An RDB containing the CSI results layers, drawn layers, and nominal contour layers.
- An ASCII summary file summarizing the results for future use in spreadsheet analysis. This file contains descriptions of all transistors and all process conditions. This is optional.

Sample MBE Calibre LFD Rule File

The sample Model Based Extraction rule file produces CSI layers and CSG files.

```
#! tvf
namespace import tvf::*
package require CalibreDFM_LFD

#####
# Specify Calibre LFD related output databases
#####
set Output          "\$RDB_DIR/lfd_nominal.oas"

#####
# Start the Header Section
#####

LAYOUT SYSTEM GDSII
LAYOUT PATH "./gds/5x6_orig.gds"
LAYOUT PRIMARY "*"

LAYOUT PROCESS BOX RECORD YES
FLAG SKEW YES
FLAG ACUTE NO
FLAG OFFGRID YES

LAYOUT ERROR ON INPUT NO
DRC CELL NAME YES XFORM ALL
PRECISION 2000
RESOLUTION 1

LAYOUT MAGNIFY 2

# Output Section. ****
DRC MAXIMUM RESULTS all
DRC RESULTS DATABASE "\$RDB_DIR/default.rdb" ASCII DRC SUMMARY REPORT \
"lfdsum.cal"
DRC KEEP EMPTY YES

#####
# Setup for OPC Run
#####
VERBATIM "
INCLUDE ./lfd/OPC_new.svrfe
"

#####
# Load Resist Models for LFD::PVband call
#####
source lfd/resist.tvf

#####
# Start of Calibre LFD Calls for CSG/CSI
#####
LFD::Begin

#####
# Create Contours needed for CSG/CSI using LFD::PVband call
#####
LFD::PVband "-layer    DIFF \
             -layerRET   {DIFF_OPc} \
```

```

-foreground      dark \
-background     clear \
-pixel          0.010 \
-modelDir       ${\$\:\:\env\ (MODEL_DIR\)} \
-resistFile     ${DIFF_resist_mod} \
-opticalSpanList {{andF APDF ANOM ANOM ANOM andF APDF}} \
-doseSpanList   {{0.95 0.95 0.95 1.00 1.05 1.05 1.05}} "

LFD:::PVband  "-layer      POLY \
    -layerRET    ${POLY_OPC POLY_SBAR} \
    -foreground   {{attenuated 0.06} {attenuated 0.06}} \
    -background   clear \
    -pixel        0.010 \
    -modelDir     ${\$\:\:\env\ (MODEL_DIR\)} \
    -resistFile   ${POLY_resist_mod} \
    -opticalSpanList {{PNDF PPDF PNOM PNOM PNOM PNDF PPDF}} \
    -doseSpanList {{0.95 0.95 0.95 1.00 1.05 1.05 1.05}} "

LFD:::PVband  "-layer      M1 \
    -layerRET    M1OPC \
    -foreground   {{clear}} \
    -background   {{attenuated 0.06}} \
    -pixel        0.010 \
    -modelDir     ${\$\:\:\env\ (MODEL_DIR\)} \
    -resistFile   ${M1_resist_mod} \
    -opticalSpanList {{MNDF MPDF MNOM MNOM MNOM MNDF MPDF}} \
    -doseSpanList {{0.95 0.95 0.95 1.00 1.05 1.05 1.05}} "

#####
# Run CSG, using TVF Loop though all conditions
#####
setlayer lfd_gate = POLY and DIFF

set DIFF_COND(DFDD) [list ANOM 1.00 0.0]
set DIFF_COND(DFND) [list ANOM 0.95 0.0]
set DIFF_COND(DFPD) [list ANOM 1.05 0.0]
set DIFF_COND(NFND) [list andF 0.95 0.0]
set DIFF_COND(NFPD) [list andF 1.05 0.0]
set DIFF_COND(PFND) [list APDF 0.95 0.0]
set DIFF_COND(PFPD) [list APDF 1.05 0.0]

set POLY_COND(DFDD) [list PNOM 1.00 0.0]
set POLY_COND(DFND) [list PNOM 0.95 0.0]
set POLY_COND(DFPD) [list PNOM 1.05 0.0]
set POLY_COND(NFND) [list PNDF 0.95 0.0]
set POLY_COND(NFPD) [list PNDF 1.05 0.0]
set POLY_COND(PFND) [list PPDF 0.95 0.0]
set POLY_COND(PFPD) [list PPDF 1.05 0.0]

set MET1_COND(DFDD) [list MNOM 1.00 0.0]
set MET1_COND(DFND) [list MNOM 0.95 0.0]
set MET1_COND(DFPD) [list MNOM 1.05 0.0]
set MET1_COND(NFND) [list MNDF 0.95 0.0]
set MET1_COND(NFPD) [list MNDF 1.05 0.0]
set MET1_COND(PFND) [list MPDF 0.95 0.0]
set MET1_COND(PFPD) [list MPDF 1.05 0.0]

foreach AAcond [array names DIFF_COND] {

```

```
foreach POcond [array names POLY_COND] {  
    set csgfile $AAcond  
    append csgfile "_" $POcond ".csg"  
    set csgname $AAcond  
    append csgname "_" $POcond  
  
    LFD::CSG "-poly POLY \  
        -polyCondition {$POLY_COND($POcond)} \  
        -active DIFF \  
        -activeCondition {$DIFF_COND($AAcond)} \  
        -outputFile {$env(RDB_DIR)}/$csgfile} \  
        -checkName {$csgname} \  
        -seedLayer lfd_gate \  
        -resolution 0.0005 \  
        -maxGate 1.0 \  
        -maxSearch 0.06"  
}  
}  
  
#####  
# Run CSI, using TVF Loop though each condition  
#####  
set lcnt 255  
foreach POcond [array names POLY_COND] {  
    set csilayer PO_$POcond  
    append csilayer "_" "csi"  
  
    LFD::CSI "-layer POLY \  
        -layerCondition {$POLY_COND($POcond)} \  
        -deviation 0.005 \  
        -layerOut {$csilayer} "  
  
        RULECHECK $csilayer {COPY $csilayer}  
        DRC CHECK MAP $csilayer OASIS 17 $lcnt \"$Output\"  
    set lcnt [expr ($lcnt -1)]  
}  
  
set lcnt 255  
foreach M1cond [array names MET1_COND] {  
    set csilayer M1_$M1cond  
    append csilayer "_" "csi"  
  
    LFD::CSI "-layer M1 \  
        -layerCondition {$MET1_COND($M1cond)} \  
        -deviation 0.010 \  
        -layerOut {$csilayer} "  
  
        RULECHECK $csilayer {COPY $csilayer}  
        DRC CHECK MAP $csilayer OASIS 31 $lcnt \"$Output\"  
    set lcnt [expr ($lcnt -1)]  
}  
  
#####  
# Capture Nominal Contours for reference  
#####  
LFD::CaptureContour "-layer DIFF -layerOut AA_NOM -optical ANOM -dose  
1.00"  
LFD::CaptureContour "-layer POLY -layerOut PC_NOM -optical PNOM -dose
```

```
1.00"
LFD::CaptureContour "-layer M1      -layerOut M1_NOM -optical MNOM -dose
1.00"

LFD::End

#####
# Create Output File
#####
RULECHECK DIFF {COPY DIFF}
DRC CHECK MAP DIFF OASIS 1 0 \"$Output\
RULECHECK POLY {COPY POLY}
DRC CHECK MAP POLY OASIS 2 0 \"$Output\
RULECHECK MET1 {COPY M1}
DRC CHECK MAP MET1 OASIS 3 0 \"$Output\


RULECHECK AA_NOM      {COPY AA_NOM}
DRC CHECK MAP AA_NOM    OASIS 253 253 \"$Output\
RULECHECK PO_NOM      {COPY PC_NOM}
DRC CHECK MAP PO_NOM    OASIS 254 253 \"$Output\
RULECHECK M1_NOM      {COPY M1_NOM}
DRC CHECK MAP M1_NOM    OASIS 255 253 \"$Output\


#####
#End of CSG/CSI Demo Deck
#####
```


Chapter 7

Calibre LFD Syntax Descriptions

Calibre LFD includes an extensive set of commands to define and run LFD checks. Most checks produce system-generated properties, scores, and priority values.

Calibre LFD Command Summary Tables	175
Calibre LFD Check Properties.....	179
Alternate Syntax Options for PDK Usage.....	179
Calibre LFD Checks	181
Calibre LFD General-Purpose Commands.....	304
System-Generated Names, Priorities, and Scores.....	495

Calibre LFD Command Summary Tables

Calibre LFD includes commands for performing LFD checks, general purpose commands, and commands for use with PDKs.

- [Table 7-1](#), Calibre LFD Check Commands
- [Table 7-2](#), Calibre General Purpose Commands
- [Table 7-3](#), Calibre LFD Commands for use with PDKs

Table 7-1. Calibre LFD Check Commands

Function	Definition
AddCustomCheck	Registers a user-defined check with the Calibre LFD check registry.
AddCustomOPCVCheck	Performs custom checks and measurement operations on PV-bands using Calibre OPCverify tool syntax.
AreaCheck	Performs area checks for a specified layer.
AreaOverlayCheck	Checks the size of the predicted overlay between two layers.
EndCapCheck	Checks the endcap enclosure for poly over active.
InterLayerSpaceCheck	Performs space checks between two separate layers.
LineEndCheck	Checks the distance by which a line-end deviates from the target edge.
MaxAreaVariabilityCheck	Performs variability checks on interactions between the two specified layers.

Table 7-1. Calibre LFD Check Commands (cont.)

Function	Definition
MaxCDVariabilityCheck	Performs variability checks on the change in CD between simulated and drawn features.
MinAreaOverlapCheck	Performs overlap checks between the two specified layers.
MinOverlayCheck	Performs an enclosure check between two separate layers where the target layer geometries on the surrounding layer name encloses the geometry on the enclosed layer name.
MinSpaceCheck	Performs spacing checks for the specified layer.
MinWidthCheck	Performs width checks for the specified layer.
NonPrintingCheck	Finds areas where PV-band and target layer do not interact.
PrintableCheck	Categorizes error markers defined in prior checks into either layout-problem or OPC-problem categories.
ViaWidthCheck	Performs check on width of a printed via.

Table 7-2. Calibre LFD General Purpose Commands

Function	Definition
Anchor	Generates anchor points that are candidates for litho hotspots and problematic patterns.
AnchorRule	Overrides LFD::Anchor arguments and defines layers and spacing for multi-pattern and multilayer anchor points.
Band	Creates PV-bands with two input contours.
Begin	Command used to mark the start of the Calibre LFD processing block.
CaptureContour	Stores a previously generated contour as a derived layer.
ClassifyConfig	Configures common user-defined classification options.
Contour	Creates contours which can be used to create other contours, PV-bands, or to perform checks.
CSG	Generates a .csg file that can be used as input to Calibre nmLVS.
CSI	Generates a new layer that is a simplified version of the input contour layer.
Drawn2Contour	Generate empirically calibrated PV-bands using Square Directional Kernels (SDK) when access to the exact OPC recipe and models are not available.
End	Closing command used to signal the end of the Calibre LFD block.

Table 7-2. Calibre LFD General Purpose Commands (cont.)

Function	Definition
FrameCellOptimizer	Pre-processing that reduces the area to be analyzed by removing those areas that do not require processing by Calibre LFD because they have already been checked or because they are IP that the designer cannot modify.
GenerateHints	Runs Calibre LFD with MBH and generates a hints file for fixing lithographic hotspots.
GetMacroLayers	Extracts layers from the process-correction macro.
GetOPCInputLayers	Returns intermediate OPC layers.
GetOPCLayers	Returns OPC layer and visible layer.
GetRetargetLayers	Returns retarget layers.
GetVisibleLayers	Returns visible layers.
IncrementalSelect	Performs intelligent clipping of one or more input layers based on your marker layers.
LayoutOptimizer	Defines problem areas based on geometric rules.
LFDregion	Defines regions within the layout, which can then be used to limit checking to those areas.
LoadPDK	Loads the specified PDK and makes it available to Calibre LFD command calls.
Macro	Used to define macros.
MLDataGen	Samples and translates features for input to a machine learning training model in the preprocessing and data generation phase of the Calibre LFD deep neural network (DNN) flow.
MLOptimizer	Performs hotspot prediction on a new design using a machine learning trained model in the prediction phase of the Calibre LFD deep neural network (DNN) flow.
OutputBands	Writes PV-band data to the specified database.
OverlayBand	Creates overlay PV-bands with two input contours.
PrintableBand	Simulate expected lithographic PV-bands in the absence of an RET recipe.
ProcessCorrection	Applies a process correction recipe to the specified design layer.
PVband	Generates PV-band data for the specified layer.
ReadECO	Provides layer name for GDS ECO layers.
ReadHIF	Reads a HIF file and maps all the polygons from the checks for a given layer onto a single output layer.

Table 7-2. Calibre LFD General Purpose Commands (cont.)

Function	Definition
ReadiLPC	Reads an iLPC file and maps all the polygons and areas for a given layer onto a single output layer.
ReadRDB	Reads an RDB file and maps the polygons from the checks for a given layer onto an output layer.
RegisterBands	Makes PV-band data generated in a previous run available for analysis in the current run.
RegisterContour	Makes contour data generated in a previous run available for analysis in the current run.
RegisterLayer	Registers a design layer with the Calibre LFD processing engine.
RemoveErrors	Removes error markers from the output layer of a Calibre LFD check.
SetSTOMode	Sets the Structure Optimizer (STO) mode.
SimConfig	Configures the simulations to be performed as part of Calibre LFD.
StructureOptimizer	Reduces layout data set being passed into Calibre LFD simulation, improving overall runtime performance.
StructureOutput	Takes hotspot and target layers and creates a pattern description file.
TopCellOptimizer	Aids checking geometries at top level of the design hierarchy.
VariabilityIndices	Generates variability indexes (scores) for the specified layer.
WriteHIF	Writes a Litho hotspot file that adheres to the Cadence Hotspot Interface Format (HIF).
WriteICC	Writes a Litho hotspot file that adheres to the Synopsys IC Compiler format (ICC).

Table 7-3. Calibre LFD Commands for use with PDKs

Function	Definition
addPDKCheck	Creates a check template and assigns it a name by which it can be referenced.
addPDKLayer	Defines a layer process to use when investigating a design.
addPDKSTO	Defines a Structure Optimizer (STO) template and assigns it a name by which it can be referenced.
createPDK	Registers the PDK, setting up the appropriate namespace within Calibre LFD.

Calibre LFD Check Properties

Calibre LFD outputs system-generated properties for most of the Calibre LFD checks.

Table 7-4 defines the naming scheme for system-generated properties:

Table 7-4. Set of Properties

Check	Default Properties	Additional Available Properties
AreaCheck	AreaRatio Area	
AreaOverlayCheck	AverageArea MinArea MaxArea	
EndCapCheck	PVI	Max
InterLayerSpaceCheck	PVI	Space
LineEndCheck	PVI	Max
MaxAreaVariabilityCheck	PVI	
MaxCDVariabilityCheck	Min	None, Max, MaxRatio, TargetCD, All
MinAreaOverlapCheck	Area AreaRatio	
MinOverlayCheck	PVI	Min
MinSpaceCheck	PVI	Space, SpaceRatio, MaxRunLength
MinWidthCheck	PVI	MinCD, MinCDRatio, MaxRunLength
NonPrintingCheck		
PrintableCheck		Intensity, IntensErr
ViaWidthCheck		Min
CSG	Leff & Weff	Wadj_max, Wadj_min, Ladj_max, Ladj_min, Wadj, Ladj, Wdrawn, Ldrawn, delta_Ion, delta_Ioff, delta_L, delta_W

Alternate Syntax Options for PDK Usage

Some commands have alternate syntax options if a PDK is used. Some options may be required rather than optional and additional options may be available.

When a command description in this chapter lists *two* usage definitions, the second description represents the PDK usage. For example, in the following usage statements for the

AddCustomCheck command, the PDK usage requires the **-layer**, **-pdkCheckName**, and **-database** arguments. Also, the PDK usage allows the additional options found in the first usage statement:

For example:

Syntax 1

AddCustomCheck

```
{-layer input_layer_name | {-layer1 layer1_name |
-layer2 layer2_name} |
-customCheck derived_layer_name
-subwindow expr_number
[-checkName cName]
[-priority cPriority]
[-comment comment_text]
{-database db_name | -layerOut return_layer_name |
-database db_name -layerOut return_layer_name}
```

Syntax 2

AddCustomCheck

```
-layer input_layer_name
pdkCheckName check_template
-database db_name
[additional_options]
```

The second syntax description is required to have an associated LFD::[addPDKCheck](#) statement in the PDK file.

The PDK *additional_options* can only be used if the kit developer has not protected it with securities in the corresponding LFD::[addPDKLAYER](#)

Calibre LFD Checks

LFD checks look at the interrelationships between PV-bands (or the underlying inner and outer contour layers) or between PV-bands and target features.

Designers run Calibre LFD checks on their works-in-progress to identify and address problems while it is still relatively easy to correct them. PV-bands are context-sensitive. The PV-band for a feature in one location is different from the PV-band for the same feature somewhere else in the design with different neighboring features.

Note

 For additional commands, see the [Calibre LFD General-Purpose Commands](#) and [PDK Commands](#) sections.

Table 7-5. Calibre LFD Check Commands

Function	Definition
AddCustomCheck	Registers a user-defined check with the Calibre LFD check registry.
AddCustomOPCVCheck	Performs custom checks and measurement operations on PV-bands using Calibre OPCverify tool syntax.
AreaCheck	Performs area checks for a specified layer.
AreaOverlayCheck	Checks the size of the predicted overlay between two layers.
EndCapCheck	Checks the endcap enclosure for poly over active.
InterLayerSpaceCheck	Performs space checks between two separate layers.
LineEndCheck	Checks the distance by which a line-end deviates from the target edge.
MaxAreaVariabilityCheck	Performs variability checks on interactions between the two specified layers.
MaxCDVariabilityCheck	Performs variability checks on the change in CD between simulated and drawn features.
MinAreaOverlapCheck	Performs overlap checks between the two specified layers.
MinOverlayCheck	Performs an enclosure check between two separate layers where the target layer geometries on the surrounding layer name encloses the geometry on the enclosed layer name.
MinSpaceCheck	Performs spacing checks for the specified layer.
MinWidthCheck	Performs width checks for the specified layer.
NonPrintingCheck	Finds areas where PV-band and target layer do not interact.

Table 7-5. Calibre LFD Check Commands (cont.)

Function	Definition
PrintableCheck	Categorizes error markers defined in prior checks into either layout-problem or OPC-problem categories.
ViaWidthCheck	Performs check on width of a printed via.

AddCustomCheck

Registers a user-defined check with the Calibre LFD check registry.

Usage

Syntax 1

AddCustomCheck

```
{-layer input_layer_name | {-layer1 layer1_name | -layer2 layer2_name} }  
-customCheck derived_layer_name  
{-subwindow expr_number} | {-subwindow1 expr_number -subwindow2 expr_number}  
[-checkName cName]  
[-priority cPriority]  
[-security {no | yes}]  
[-comment comment_text]  
[-dependentCheck yes]  
{-database db_name | -layerOut return_layer_name  
/ -database db_name -layerOut return_layer_name}
```

Syntax 2

AddCustomCheck

```
-layer input_layer_name  
-pdkCheckName check_template  
-database db_name  
[additional_options]
```

Description

Registers a custom (user-defined) check with the Calibre LFD check registry, which integrates its results seamlessly within the Calibre LFD database and factors the results into the DVI calculations.

If used with a PDK, this function calls one of the custom checks defined in the PDK and runs it for the specified layer. Outputs of the check are written to the specified database using the check name you supply.

Arguments

- **-layer *input_layer_name***

Required keyword and argument, only used for single layer checks, specifying the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-layer1 *layer1_name***

Required keyword and argument, only used for dual layer checks, specifying the name of the first layer you are checking. This is the first layer for which PV-bands are generated and is typically a poly, active, or metal layer.

- **-layer2 *layer2_name***

Required keyword and argument, only used for dual layer checks, specifying the name of the second layer you are checking. This is the second layer for which PV-bands are generated, and is typically a contact or a via layer.

- **-customCheck *derived_layer_name***

Required keyword and argument defining the name of the derived layer.

- **-subwindow *expr_number*, -subwindow1 *expr_number1*, -subwindow2 *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. **-subwindow** is used for single layer checks and both **-subwindow1** and **-subwindow2** are used for dual layer checks.

You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, *expr_number* refers to an index to a list of experiments.

- **-checkName *cName***

Optional keyword and argument specifying the name to use for the check in the results database (RDB). If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions performing multiple checks of this type.

- **-priority *cPriority***

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the **-subwindow** value and default ranking of 3, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

Note

 Setting “-security yes” does not prevent custom SVRF from appearing in the transcript. If you do not want the custom SVRF to be displayed in the transcript, use the following lines:

```
tvf:::VERBATIM "#PRAGMA ENCRYPTED"  
tvf:::_suppress_svrf_echo 1  
    " Check implementation"  
tvf:::VERBATIM "#PRAGMA ENDCRYPT"  
tvf:::_suppress_svrf_echo 0
```

- **-comment *comment_text***

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. Each comment results in one line of check text. There is no default value for this argument.

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces ({}). For example:

```
-comment {This is my comment.}
```

- **-dependentCheck yes**

Optional keyword and argument that indicates this custom check is post-processing for other checks. Specifying “-dependentCheck yes” enables chopping of contours in the Calibre OPCverify tool, preventing total contours being exported from the Calibre OPCverify setup file if LFD::CaptureContour and LFD::OutputBands are defined with the *errorFilterSize* option.

- **-database db_name**

Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut return_layer_name**

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName check_template**

Required keyword and argument specifying the name of the check template defining how the check is performed.

- **additional_options**

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK AddCustomCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

```
tvf::VERBATIM " gate = poly and active "
LFD::AddCustomCheck -layer poly \
    -customCheck gate \
    -subwindow 1 \
    -checkName Add_Custom_Check_poly1 \
    -database $lfdErrorDatabase \
    -layerOut add_custom_layer
```

AddCustomOPCVCheck

Performs custom checks and measurement operations on PVbands using Calibre OPCverify tool syntax.

Usage

Syntax 1

AddCustomOPCVCheck

```
{-layer input_layer_name | {-layer1 layer1_name -layer2 layer2_name} }  
{-subwindow expr_number} | -subwindow1 expr_number -subwindow2 expr_number}  
-layerOut out_layers  
-checkSyntax opcverifySyntax_list  
[-supportLayers supportLayer_list]  
[-security {no | yes}]  
[-property {property_list}]  
[-markerLayer layer_name]  
[-layerContoursHandle contour_handle]  
| [-layer1ContoursHandle contour1_handle -layer2ContoursHandle contour2_handle]  
[-database db_name]  
[-checkName cName]  
[-priority cPriority]  
[-comment comment_text]  
[-classify {classifyHandle1, classifyHandle2, ..., classifyHandleN}]  
[-layerContourCondition contour_condition_list]  
| {-layer1ContourCondition contour_condition_list}  
| {-layer2ContourCondition contour_condition_list} ]
```

Syntax 2

AddCustomOPCVCheck

```
{-layer input_layer_name | {-layer1 layer1_name -layer2 layer2_name} }  
-pdkCheckName check_template  
[additional_options]
```

Description

Performs custom checks and measurement operations on PV-bands using Calibre OPCverify tool syntax. This allows you to use the encapsulation capabilities of Calibre LFD, which is useful as it limits the number of cases where a full contour needs to be output. You are assumed to have knowledge of the Calibre OPCverify syntax.

The output is one or more SVRF layers. You have the freedom to use the layer(s) in operations or pass them to [AddCustomCheck](#) to output to the specified results database (RDB).

The command only operates on minimum or maximum PV-bands. It can perform checks on single or dual layers.

Note

-  For PDK usage, the process layer security is fully enabled (read and write) to be able to define this check.
-

Arguments

- **-layer *input_layer_name***

Required keyword and argument, used only in single layer checks, specifying the name of the layer for which the custom checks are to be performed.

- **-layer1 *layer1_name***

Required keyword and argument, used only in dual layer checks, specifying the name of the first layer for which the custom checks are to be performed. This is the first layer for which PV-bands are generated and is typically a poly, active, or metal layer.

- **-layer2 *layer2_name***

Required keyword and argument, used only in dual layer checks, specifying the name of the second layer for which the custom checks are to be performed. This is the second layer for which PV-bands are generated, and is typically a contact or via layer.

- **-subwindow *expr_number*, -subwindow1 *expr_number1*, -subwindow2 *expr_number***

Required keyword and argument defining the process variation experiment to which this custom check applies. **-subwindow** is used for single layer checks and both **-subwindow1** and **-subwindow2** are used for dual layer checks.

You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

- **-layerOut *out_layers***

Required keyword and argument specifying a name for the layer to contain the simplified contours.

- **-checkSyntax *opcverifySyntax_list***

Required keyword and list argument used to specify one or more commands. It is your responsibility to ensure layers names are unique. The following Tcl variables are locally defined by the function:

- \$layer — The layer name.

\$layer1, \$layer2 — The layer names in dual layer checks.

- \$layer_minband, \$layer_maxband — Inner and outer PV-band edges.

\$layer1_minband, \$layer1_maxband, \$layer2_minband, \$layer2_maxband — Inner and outer PV-bands edges in dual layer checks.

- \$support_layer_name — The name of a passed support layer (that is, \$contact, \$support1, ...).
- \$layer_target, \$layer1_target, \$layer2_target — Targeting layers.
- \$classify_block1, \$classify_block2, ... \$classify_blockN — Classification blocks.

Note

 The classification blocks are a mapping of the classification handles defined in -classify.

- \$layer_contour1, \$layer_contour2, ..., \$layer_contourN
- \$layer1_contour1, \$layer1_contour2, ..., \$layer1_contourN
- \$layer2_contour1, \$layer2_contour2, ..., \$layer2_contourN

Note

 The layer contour variables are a mapping of the contour names defined in the -layerContourCondition, -layer1ContourCondition, and -layer2ContourCondition lists.

- **-supportLayers *supportLayers_list***

Optional keyword and Tcl list argument used to specify one or more layers containing supplementary mask data (features that have a lithographic impact on the input layer). These can be original layers or derived polygon layers.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file are encrypted in the transcript.

Note

 If you attempt to read a target layer in your check syntax and target layer security defined for this layer in the PDK is 00 or 01 (read access denied), then an error is reported. If you attempt to use a check already defined in the PDK, and the latter has one or more of its check syntax commands trying to read the target layer, you are allowed to read it, because the PDK has allowed it.

- **-property {*property_list*}**

Optional keyword set instructing the check to store the measured properties in the RDB. The *property_list* must be a Tcl list containing the layerOut properties required to be stored in the RDB.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on the layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer** or a circular layer definition results.

- **-layerContoursHandle** *contour_handle*

Optional keyword and argument, used only in single layer checks, used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-layer1ContoursHandle** *contour1_handle*

Optional keyword and contour (or list of contours), used only in dual layer checks, specifically for the first layer, used to perform Calibre LFD checks on specific contour handles.

This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-layer2ContoursHandle** *contour2_handle*

Optional keyword and contour (or list of contours), used only in dual layer checks, specifically for the second layer, used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-database** *db_name*

Optional keyword and argument defining the RDB to which the contours generated by the check are written.

- **-checkName** *cName*

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name. Use the keyword to avoid name collisions if performing multiple checks of this type.

- **-priority** *cPriority*

Optional keyword and argument specifying a priority for this check.

- **-comment** *comment_text*

Optional keyword and argument used with the **-database** argument for defining the comment text to be reported in the RDB if the check encounters a violation of this type.

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-classify** {*classifyHandle1*, *classifyHandle2*, ..., *classifyHandleN*}

Optional keyword and argument that holds classification blocks names defined by [LFD::ClassifyConfig](#). You can use these classification blocks to classify your Calibre OPCverify layers.

- **-layerContourCondition** *contour_condition*

Optional keyword and argument to define a **-layer** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be a Tcl ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

- **{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}**

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- **-layer1ContourCondition** *contour_condition1*

Optional keyword and argument to define **-layer 1** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

- **{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}**

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- **-layer2ContourCondition** *contour_condition2*

Optional keyword and argument to define **-layer 2** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- *min* — Specifies the command works on the inner PV-band contour.
- *max* — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{*optical1 dose1 size1 [resist1 etch1]* [*optical2 dose2 size2 [resist2 etch2]*]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-pdkCheckName** *check_template*

Required keyword and argument specifying the name of the check template defining how the check is performed.

- *additional_options*

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK AddCustomOPCVCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

Example 1

Access to PV-band edges is granted by pre-defined handle names (that is, *\$layer1_minband*, *\$layer2_minband*).

```
LFD::AddCustomOPCVCheck -layer1 poly -layer2 contact
    -subwindow1 1 -subwindow2 1
    -layerOut {errorMarkerDual contactInnerBandDual polyInnerBandDual}
    -property {area_ratio {} {}} -database $lfdbErrorDatabase
    -checkName DualAreaRatioCheck -checkSyntax {
        setlayer polyInnerBandDual = copy $layer1_minband
        setlayer contactInnerBandDual = copy $layer2_minband
        setlayer errorMarkerDual = area_ratio contactInnerBandDual \
            $layer2_minband <= 0.3 max_extent 3 property { area_ratio }
    }
```

Example 2

Out layers need to be defined inside the checkSyntax block.

```
LFD::AddCustomOPCVCheck -layer poly -subwindow 1
    -layerOut {errorMarker choppedContact polyInnerBand}
    -database $lfdbErrorDatabase
    -checkName PoorAreaRatioCheck -supportLayers {contact}
    -property {area_ratio {} {}} -checkSyntax {
        setlayer choppedContact = cornerchop $contact 0.1 0.1
        setlayer polyInnerBand = copy $layer_minband
        setlayer errorMarker = area_ratio choppedContact \
            polyInnerBand <= 0.3 max_extent 3 property { area_ratio }
    }
```

Example 3

Classification layerOut:

```
LFD::AddCustomOPCVCheck -layer poly -subwindow 1
    -layerOut {errorMarker choppedContact polyInnerBand} \
    -checkName PoorAreaRatioCheck -supportLayers {contact} \
    -property {area_ratio} -database $lfdbErrorDatabase \
    -classify {myClass myClass2} -checkSyntax {
        setlayer choppedContact = cornerchop $contact 0.1 0.1
        setlayer polyInnerBand = copy $layer_minband $classify_block1
        setlayer errorMarker = area_ratio choppedContact \
            polyInnerBand <= 0.3
        max_extent 3 property { area_ratio } $classify_block2
    }
```

Example 4

Passing two contours to layer1ContoursHandle and layer2ContoursHandle:

```
LFD::AddCustomOPCVCheck -layer1 poly -layer2 contact \
    -layerOut {out1 out2 out3 out4 out5 out6} \
    -pdkCheckName ACOPCVC -checkSyntax {
        setlayer out1 = copy \$layer1_maxband
        setlayer out2 = copy \$layer1_contour1
        setlayer out3 = copy \$layer1_contour2
        setlayer out4 = copy \$layer2_maxband
        setlayer out5 = copy \$layer2_contour1
        setlayer out6 = copy \$layer2_contour2
    } -database $lfdErrorDatabase -checkName r1

LFD::addPDKCheck { -name ACOPCVC -pdk generic65 -type AddCustomOPCVCheck \
    -security 11 -definition { -layer1ContoursHandle {max_poly c1} \
    -layer2ContoursHandle {max_contact c1}}}
```

Example 5:

Using contours:

```
LFD::AddCustomOPCVCheck -layer poly -subwindow 1 \
    -layerOut {errorMarker choppedContact polyInnerBand} \
    -checkName PoorAreaRatioCheck -supportLayers {contact} \
    -property {area_ratio} -database $lfdErrorDatabase \
    -layerContourCondition {2, min, {P1W_D0 1 0}} \
    -checkSyntax {
        setlayer choppedContact = cornerchop $contact 0.1 0.1
        setlayer polyInnerBand = copy $layer_contour1
        setlayer errorMarker = area_ratio choppedContact \
            polyInnerBand <= 0.3 \
        max_extent 3 property { area_ratio }
    }
```

AreaCheck

Performs area checks for a specified layer.

Usage

Syntax 1

AreaCheck

```
-layer input_layer_name
-subwindow expr_number
{-minLFDarea area | -maxLFDarea area}
[-mode {absolute | ratio}]
[-map {OVERLAP | INTERSECT}]
[-security {no | yes}]
[-markerLayer layer_name]
[-referenceLayer "%drawn" | "%retarget" | layer_name]
[-anchorLayer "%reference" | "%drawn" | "%retarget" | layer_name]
[-layerContourCondition contour_condition]
[-maxExtent extent_size]
[-checkName cName]
[-comment comment_text]
[-priority cPriority]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contourHandle contour_handle]
{-database db_name | -layerOut return_layer_name
 | -database db_name -layerOut return_layer_name}
```

Syntax 2

AreaCheck

```
-layer input_layer_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

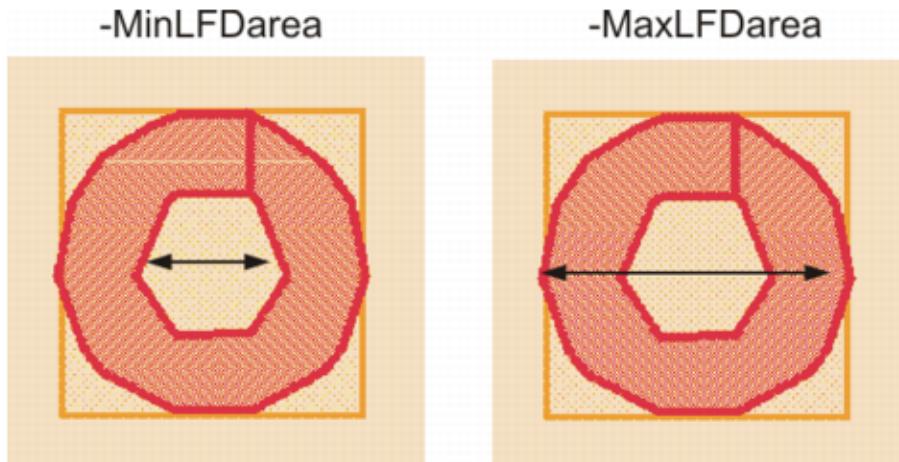
Description

Performs an area check on PV-band geometries for the specified layer. If -markerLayer is specified, the check is limited to PV-band data that lies inside geometries on -markerLayer. The AreaCheck looks at the area inside a Calibre LFD-generated contour and flag it as a violation if the area does not comply with a constraint. This check is used to identify problems such as minimum area violations on contacts/vias, or posts. It can also be used on metal slots and donut-like shapes.

AreaCheck (AC) checks the area inside PV-bands as shown in Figure 7-1. If **-minLFDarea area** is specified, the check measures the area inside the inner edge of the PV-band and flags PV-bands having an area less than or equal to the specified value. If **-maxLFDarea area** is

specified, the check measures the area inside the outer edge of the PV-band and flags PV-bands having an area greater than or equal to the specified value.

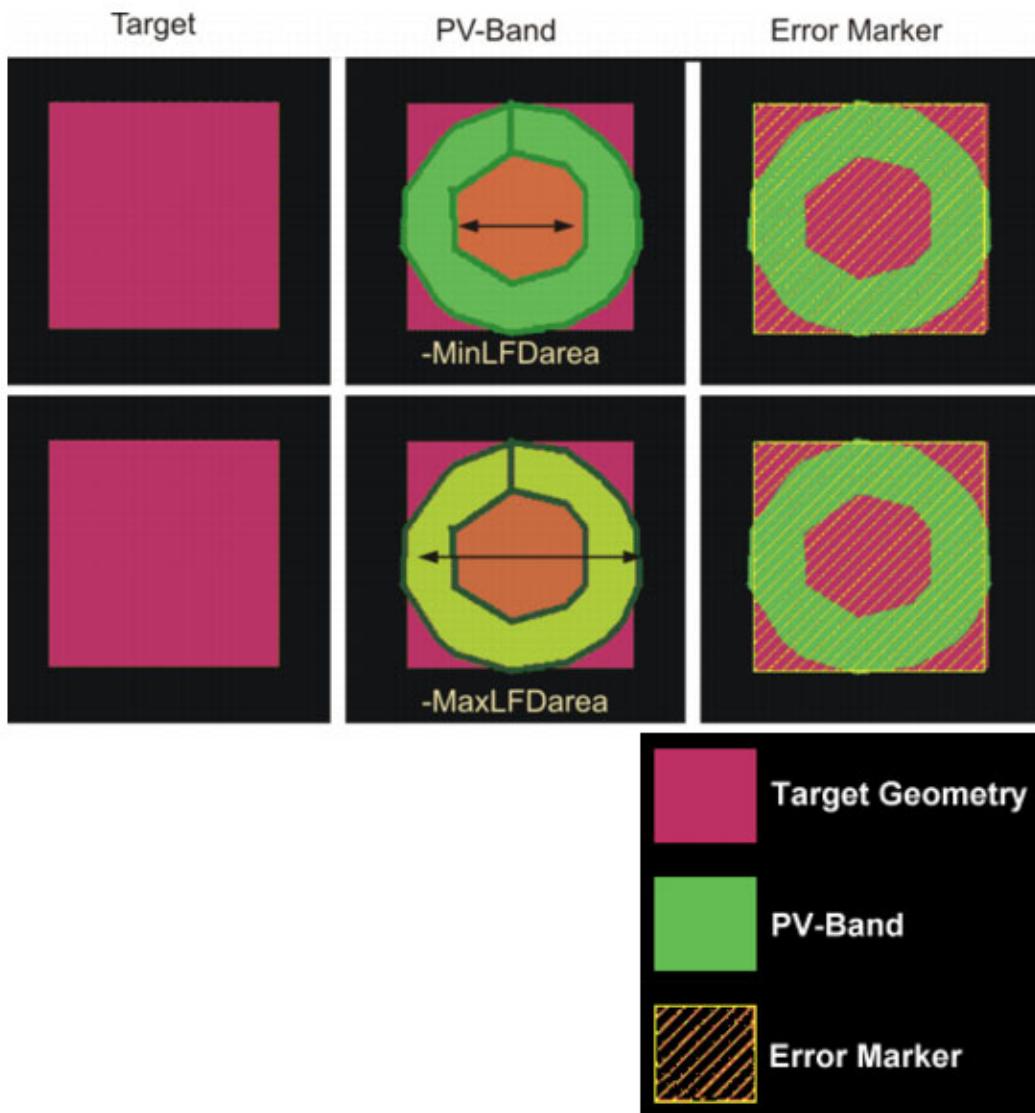
Figure 7-1. AreaCheck



This function writes all errors discovered by the check to the Calibre nmDRC results database (RDB). It also associates a score to each error and writes it to the [Check Database](#) specified by the **-database** argument. The score is calculated as the area of the model-based violation.

If used with a PDK, this function calls an AreaCheck defined in the PDK and runs it for the specified layer, writing check results to the specified database.

Figure 7-2. How AreaCheck Works



Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, -subwindow 1_N causes the check to only run on the north shift).

- **-minLFDarea area | -maxLFDarea area**

Required keyword and argument defining where to measure the area and what values to flag as errors. You must specify either **-minLFDarea area** or **-maxLFDarea area**.

- **-minLFDarea area** — Instructs the check to measure the area inside the inner edge of the PV-band and defines the lower bound for the area considered to be passing. Any contours having an area less than or equal to this value are flagged as an error.
- **-maxLFDarea area** — Instructs the check to measure the area inside the outer edge of the PV-band and defines the upper bound for the area considered to be passing. Any contours having an area greater than or equal to this value are flagged as an error.

- **-mode {absolute |ratio}**

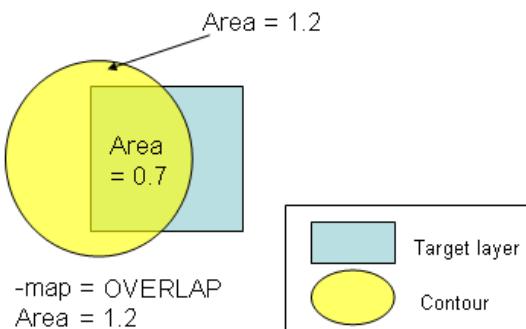
Optional keyword and argument specifying the mode used to evaluate critical dimension (CD) variability. The option outputs Area and AreaRatio properties. The default mode is “absolute”.

- **absolute** — Calculates the absolute area of the PV-band, and highlights the contacts where their PV-band areas are violating the area constraint.
- **ratio** — Calculates the area ratio of the PV-band with respect to the drawn contacts, and highlights the contacts where their ratios violate the area constraint.

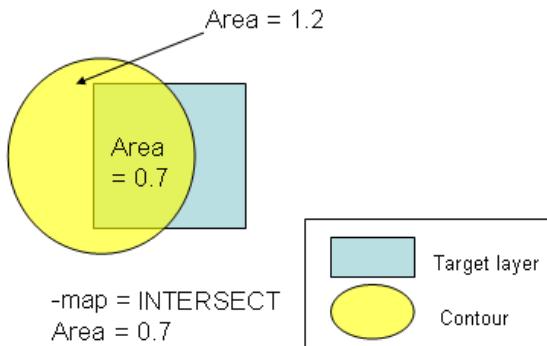
- **-map {OVERLAP | INTERSECT}**

Optional keyword and argument used to specify how the area is calculated. You can choose either of the following value types, OVERLAP and INTERSECT. The default is INTERSECT.

- **OVERLAP** — Area is the total area of the contour that overlaps the feature on the target layer.



- **INTERSECT** — Area is the portion of the contour that intersects the feature on the target layer.



- **-security {no | yes}**
Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.
- **-markerLayer *layer_name***
Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on the layer.
A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer** or a circular layer definition results.
- **-referenceLayer “%drawn” | “%retarget” | *layer_name***
Optional keyword and argument to have the check measurements calculated on a different layer than the input layer to the checks for which the PV-bands have been generated.
You can provide one of the following options as an input to this argument:
 - “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer**.
 - “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - **layer_name** — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-anchorLayer “%reference” | “%drawn” | “%retarget” | *layer_name***
Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the input drawn layer of the checks, and different from the reference layer input to the check.
You can provide one of the following options as an input to this argument:
 - “%reference” — Default. The output error markers are anchored to the layer input to **-referenceLayer**.

- “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer**.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - **layer_name** — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-layerContourCondition** *contour_condition*
- Optional keyword and argument to define a **-layer** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.
- You can provide one of the following options as an input to this argument:
- min — Specifies the command works on the inner PV-band contour.
 - max — Specifies the command works on the outer PV-band contour.
 - *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
 - *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{*optical1 dose1 size1 [resist1 etch1]* [*optical2 dose2 size2 [resist2 etch2]*]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note



This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-maxExtent** *extent_size*
- Optional keyword and argument used to modify the Calibre OPCVerify tool’s max_extent value option to the area_compute control inside of a package. The default *extent_size* is 3.0 microns. This is useful as the large default max_extent value can affect Calibre LFD performance.
- **-checkName** *cName*
- Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in

Table 7-14. Use this keyword to avoid name collisions if performing multiple checks of this type.

- **-priority *cPriority***

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 3, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-comment *comment_text***

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. Each comment results in one line of check text. If not specified, the comment for an AC violation is “Polygon Area Violation: Improve / Deprove the original geometry size. Increase / Decrease proximity spacing to near by polygons.”

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-classify *handle***

Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.

- **-appendMarker *extra_markers_layer***

Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).

- **-contourHandle *contour_handle***

Optional keyword and argument used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-database *db_name***

Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how the check is performed.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK AreaCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

Verify that the area of individual contacts are above 0.003 across all the conditions specified by subwindow 1. All violations are then reported to *areaCheck.rdb*:

```
LFD::AreaCheck -layer contact -subwindow 1 -minLFDarea 0.003 \
    -database areaCheck.rdb
```

AreaOverlayCheck

Checks the size of the predicted overlay between two layers.

Usage

Syntax 1

AreaOverlayCheck

```
-layer1 layer1_name -layer2 layer2_name
-subwindow expr_number
-overlayError shift
-minOverlapArea min_area_size
-overlapAreaType {Average | MinArea | MaxArea}
[-security {no | yes}]
[-markerLayer layer_name]
[-referenceLayer1 "%drawn" | "%retarget" | layer_name]
[-referenceLayer2 "%drawn" | "%retarget" | layer_name]
[-anchorLayer1 "%reference" | "%drawn" | "%retarget" | layer_name]
[-anchorLayer2 "%reference" | "%drawn" | "%retarget" | layer_name]
[-layer1ContourCondition contour_condition1]
[-layer2ContourCondition contour_condition2]
[-checkName cName]
[-comment comment_text]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contour1Handle contour1_handle -contour2Handle contour2_handle]
-database db_name | -layerOut return_layer_name
| -database db_name -layerOut return_layer_name
```

Syntax 2

AreaOverlayCheck

```
-layer1 layer1_name -layer2 layer2_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

Description

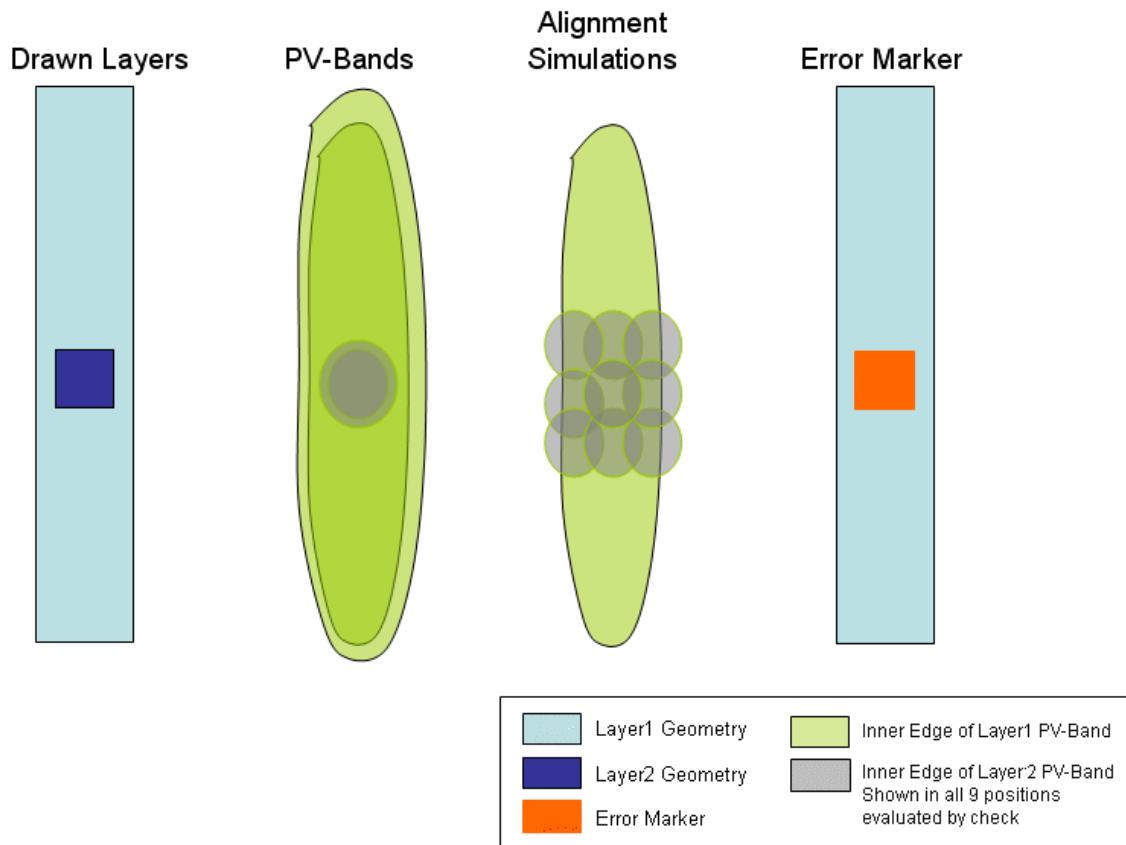
Compares the area inside the inner edge of a PV-band of one layer with the area inside the inner edge of a PV-band of a second layer. This information is then used to calculate the size of the overlay that can be expected, given that the two layers can be slightly misaligned.

AreaOverlayCheck identifies potential problems with connectivity. It assumes that the two layers under investigation are likely to be misaligned with respect to one another, and simulates the possible misalignment as part of its computation. An error is flagged if this predicted overlay is less than the -minOverlapArea.

This check is typically used to predict the overlay of a contact or via layer (**-layer2**) over a poly, active or metal layer (**-layer1**). The check simulates misalignment by shifting the **-layer2** PV-band by the **-overlayError** amount. It evaluates eight different shifts, one in each of the following directions: top, bottom, left, right, top-left, top-right, bottom-left and bottom-right.

If used with a PDK, this function calls an AreaOverlayCheck defined in the PDK and runs it for the specified layers, writing check results to the specified database.

Figure 7-3. AreaOverlayCheck



The check identifies nine separate overlap areas in all, one for the original position and one for each of the eight shifted positions. It then calculates the specified type of overlap value:

- Average — The average of the nine individual overlap areas.
- MinArea — The smallest of the nine individual overlap areas.
- MaxArea — The largest of the nine individual overlap areas.

The check outputs a property called *AverageArea* if the overlap type is average, outputs a property called *MinArea* if the overlap type is the smallest, or outputs a property called *MaxArea* if the overlap type is the largest.

Flagged errors are identified by error markers located at the intersection of **-layer1** and **-layer2**.

Arguments

- **-layer1 layer1_name**

Required keyword and argument defining the name of the first layer you are checking using **AreaOverlayCheck**. This is the first layer for which PV-bands are generated and is typically a poly, active or metal layer.

- **-layer2 layer2_name**

Required keyword and argument defining the name of the second layer you are checking using **AreaOverlayCheck**. This is the second layer for which PV-bands are generated, and is typically a contact or via layer.

- **-subwindow expr_number**

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, **expr_number** refers to an index to a list of experiments.

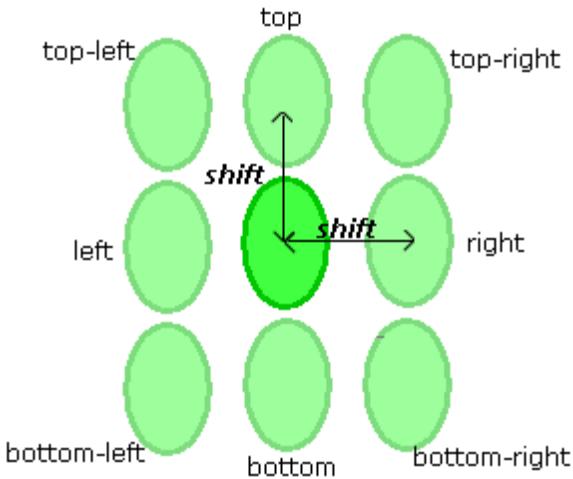
The check uses the contours defined by the inner PV-bands of this subwindow for identifying problem spots.

Setting **expr_number** to a value of “**expr_number_shift**” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, **-subwindow 1_N** causes the check to only run on the north shift).

- **-overlayError shift**

Required keyword and argument defining the shift distance, expressed as a real number. The shift distance represents the maximum distance by which you expect **-layer2** to be misaligned with respect to **-layer1**.

The check simulates misalignment by shifting polygons formed inside the inner edge of a **-layer2** PV-band by the specified distance in eight directions: top, bottom, left, right, top-left, top-right, bottom-left and bottom-right.

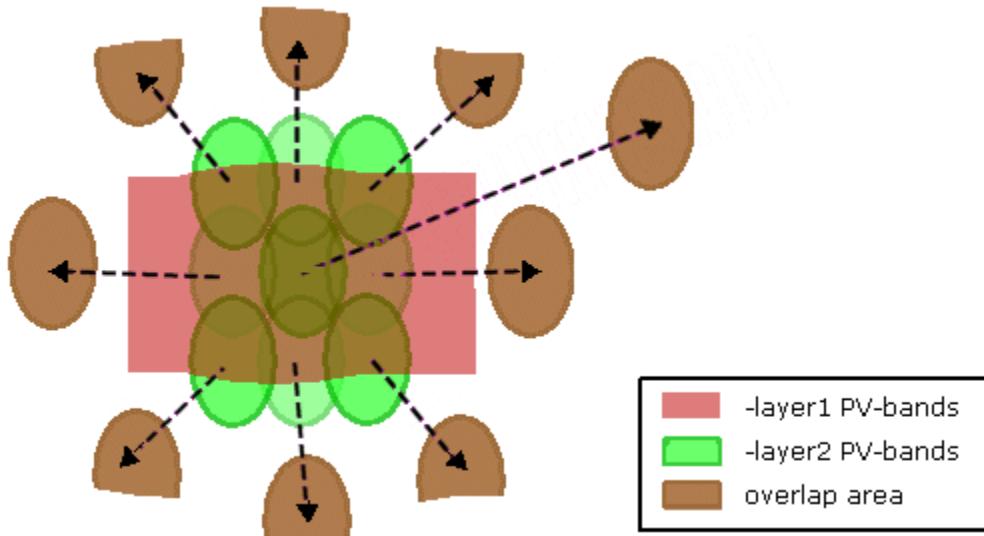


- **-minOverlapArea *min_area_size***

Required keyword and argument defining the minimum acceptable overlap, expressed as a real number. Any overlap smaller than the ***min_area_size*** is flagged as an error.

- **-overlapAreaType { Average | MinArea | MaxArea }**

Optional keyword and argument used to define how the check calculates the overlap value based on the nine separate overlay configurations shown below.



The overlap type must be one of:

- **Average** — The average of the nine individual overlap areas.
- **MinArea** — The smallest of the nine individual overlap areas.
- **MaxArea** — The largest of the nine individual overlap areas.

The check outputs a property called *AverageArea* if the overlap type is average, outputs a property called *MinArea* if the overlap type is the smallest, or outputs a property called *MaxArea* if the overlap type is the largest.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on the layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-referenceLayer1 "%drawn" | "%retarget" | *layer_name***

Optional keyword and argument to have the check measurements calculated on a different layer than the first drawn layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer1**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -referenceLayer2 “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the second drawn layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer2**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -anchorLayer1 “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the first drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer1.
- “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer1**.
- “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-anchorLayer2** “%reference” | “%drawn” | “%retarget” | *layer_name*
Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the second drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer2.
 - “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer2**.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-layer1ContourCondition** *contour_condition1*

Optional keyword and argument to define **-layer1** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- **-layer2ContourCondition** *contour_condition2*

Optional keyword and argument to define **-layer2** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer*— This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-checkName *cName***

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions if performing multiple checks of this type.

- **-priority *cPriority***

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 2, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-comment *comment_text***

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of the type. There is no default value for this argument.

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-classify *handle***
Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.
- **-appendMarker *extra_markers_layer***
Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).
- **-contour1Handle *contour1_handle***
Optional keyword and argument specifying the name of the first layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.
- **-contour2Handle *contour2_handle***
Optional keyword and argument specifying the name of the second layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.
- **-database *db_name***
Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.
- **-layerOut *return_layer_name***
Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.
- **-pdkCheckName *check_template***
Required keyword and argument specifying the name of the check template defining how the check is performed.
- ***additional_options***
Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for the command. These can be any of the options for the non-PDK AreaOverlayCheck command. Any additional options specified overrides the options defined within the PDK.

EndCapCheck

Checks the endcap enclosure for poly over active.

Usage

Syntax 1

EndCapCheck

```
-layer1 poly_layer
-layer2 active_layer
-subwindow expr_number
-minDRCEndCap DRC_value
-minLFDEndCap LFD_value
[-maxPolyWidth max_poly_width]
[-referenceLayer1 layer1_reference_layer]
[-referenceLayer2 layer2_reference_layer]
[-classify classify_block_name]
[-property Max]
[-checkName check_name]
[-markerLayer layer_name]
[-comment comment_text]
[-suppressPVI yes]
[-layer1ContourCondition poly_contour_condition]
[-layer2ContourCondition active_contour_condition]
[-anchorLayer1 "%reference" | "%drawn" | "%retarget" | layer_name]
[-indexFilter value]
[-jogLength max_jog_length]
[-appendMarker extra_markers_layer]
[-contour1Handle contour1_handle -contour2Handle contour2_handle]
{-database db_name | -layerOut return_layer_name
 / -database db_name -layerOut return_layer_name}
```

Syntax 2

EndCapCheck

```
-layer1 poly_layer
-layer2 active_layer
-pdkCheckName check_template
-database db_name
[additional_options]
```

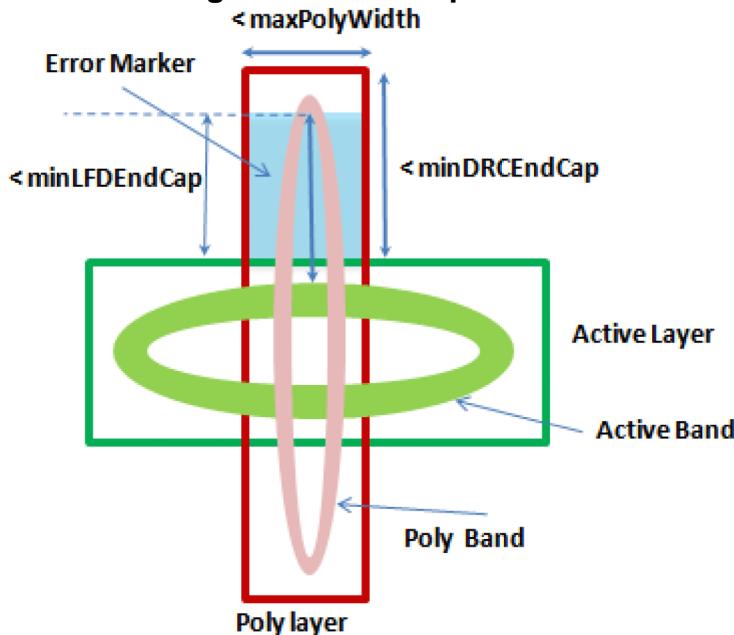
Description

Measures the maximum enclosure of edges on **-layer2** (active layer) by edges on **-layer1** (poly layer) contour, within the search region defined around each **-layer2** edge and bounded by **-layer1**. EndCapCheck (ECC) is used to check the endcap enclosure, and should only be used for poly over active overlap checks. Insufficient endcap coverage impacts transistor leakage current, and hence the leakage power of the chip.

Note

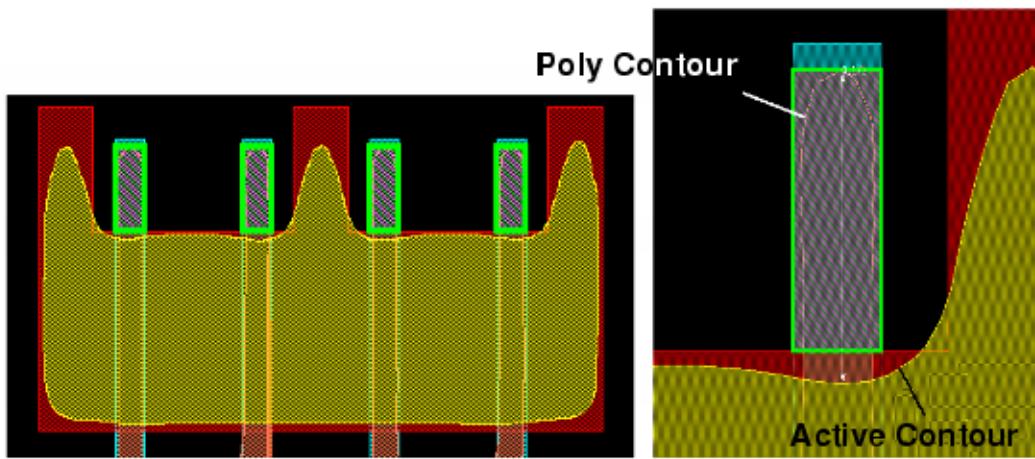
 Jogs are excluded from this check. The command only measures poly gates with line-ends facing active.

Figure 7-4. EndCapCheck



If used with a PDK, this function calls an EndCapCheck defined in the PDK and runs it for the specified layers, writing check results to the specified database.

Figure 7-5. Expected Output



Arguments

- **-layer1** *poly_layer*

Required keyword and argument defining the poly layer you are checking.

- **-layer2 active_layer**

Required keyword and argument defining the name of the active layer that is going to be enclosed.

- **-subwindow expr_number**

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, **expr_number** refers to an index to a list of experiments.

Setting **expr_number** to a value of “*subwindow_C*” causes the check to operate on the center PV-band rather than all PV-bands.

- **-minDRCEndCap DRC_value**

Required keyword and argument. Poly gate extensions longer than **DRC_value** are not checked.

- **-minLFDEndCap LFD_value**

Required keyword and argument. Poly contours with enclosure for the active contour smaller than **LFD_value** are flagged by error markers.

- **-maxPolyWidth max_poly_width**

Optional keyword and argument defining the maximum poly gate widths that are subjected to be checked. Poly gates with widths larger than *max_poly_width* are not checked.
max_poly_width must be less than 0.5. Default value is 0.2.

- **-referenceLayer1 layer1_reference_layer**

Optional keyword and argument to have the check measurements calculated on a different layer than the *poly* layer input to the checks for which the PV-bands have been generated.

You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-referenceLayer2 layer2_reference_layer**

Optional keyword and argument to have the check measurements calculated on a different layer than the active layer input to the checks for which the PV-bands have been generated.

You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-classify classify_block_name**

Optional keyword and argument used to define a handle to point to an LFD::**ClassifyConfig** object.

- **-property Max**

Optional keyword and argument specifying an optional property operation for the returned output of the command. Only the maximum length can be computed for this command.

- **-checkName** *check_name*

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-markerLayer** *layer_name*

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on the layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-comment** *comment_text*

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of the type. The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-suppressPVI** yes

Optional keyword and argument specifying whether or not to suppress calculation of PVI. When “**-suppressPVI yes**” is specified, the check suppresses calculation of PVI. By default, PVI calculation is not suppressed.

- **-layer1ContourCondition** *poly_contour_condition*

Optional keyword and argument to define **-layer1** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- *min* — Specifies the command works on the inner PV-band contour.
- *max* — Specifies the command works on the outer PV-band contour.
- *integer* — Integer defining the order of the contour to be checked in the subwindow.

- **-layer2ContourCondition** *active_contour_condition*

Optional keyword and argument to define **-layer2** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- *min* — Specifies the command works on the inner PV-band contour.
- *max* — Specifies the command works on the outer PV-band contour.
- *integer* — Integer defining the order of the contour to be checked in the subwindow.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-anchorLayer1** “%reference” | “%drawn” | “%retarget” | *layer_name*
Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the input drawn layer of the checks, and different from the reference layer input to the check. -anchorLayer1 is associated with the first drawn layer input to the check.
You can provide one of the following options as an input to this argument:
 - “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer1.
 - “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer1**.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-indexFilter** *value*
Optional keyword and value used to filter out inconsequential errors. Only errors having an index greater than or equal to this value are written to the [Check Database](#).

The index is defined as:

$$\frac{\text{area}(I_{abs})}{\text{area}(I_{marker})}$$

where:

I_{abs} = sum of the absolute PV-bands (for both layers) within the area of interest

I_{marker} = area of the associated error marker

For this check, the area of interest is defined relative to critical dimension (CD).

- **-jogLength *max_jog_length***
Optional keyword and argument used to have poly gates having jogs with length less than *max_jog_length* included in checking. By default, *max_jog_length* is 0.02.
- **-appendMarker *extra_markers_layer***
Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::StructureOptimizer.
- **-contour1Handle *contour1_handle***
Optional keyword and argument specifying the name of the first layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.
- **-contour2Handle *contour2_handle***
Optional keyword and argument specifying the name of the second layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.
- **-database *db_name***
Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.
- **-layerOut *return_layer_name***
Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

Note



A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***
Required keyword and argument specifying the name of the check template defining how the check is performed.
- ***additional_options***
Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for the command. These can be any of the options for the non-PDK EndCapCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

```
LFD::EndCapCheck \
    -layer1 poly -layer2 active -subwindow 1 \
    -minDRCEndCap 0.15 -minLFDEndCap 0.1 \
    -maxPolyWidth 0.07 -property Max \
    -database checks.rdb
```

InterLayerSpaceCheck

Performs space checks between two separate layers.

Usage

Syntax 1

InterLayerSpaceCheck

```
-layer1 layer1_name -layer2 layer2_name
-subwindow expr_number
-minDRCspace target_dist
-minLFDspace space
[-security {no | yes}]
[-markerLayer layer_name]
[-referenceLayer1 "%drawn" | "%retarget" | layer_name]
[-referenceLayer2 "%drawn" | "%retarget" | layer_name]
[-anchorLayer1 "%reference" | "%drawn" | "%retarget" | layer_name]
[-anchorLayer2 "%reference" | "%drawn" | "%retarget" | layer_name]
[-layer1ContourCondition contour_condition1]
[-layer2ContourCondition contour_condition2]
[-ext_side ext_side_value]
[-property Space]
[-checkName cName]
[-priority cPriority]
[-indexFilter value]
[-minMarkerWidth width_size]
[-suppressPVI {no | yes}]
[-comment comment_text]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contour1Handle contour1_handle -contour2Handle contour2_handle]
{-database db_name | -layerOut return_layer_name
 | -database db_name -layerOut return_layer_name}
```

Syntax 2

InterLayerSpaceCheck

```
-layer1 layer1_name -layer2 layer2_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

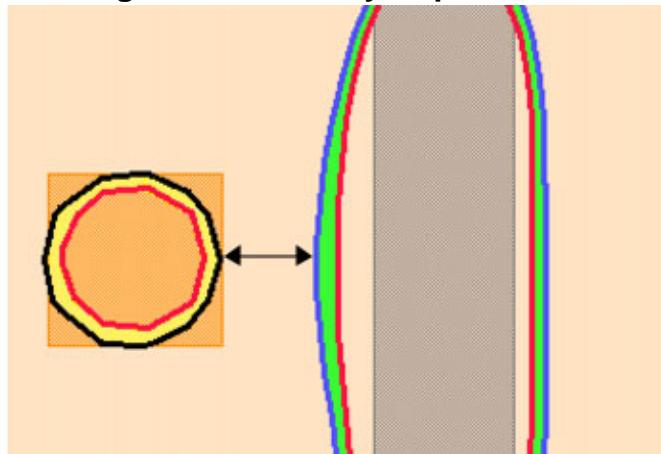
Description

Performs a space check between PV-bands between two separate layers in areas where target layer geometries are within a **-minDRCspace** from one another. If -markerLayer is specified, the check is limited to PV-band data that lies inside geometries on -markerLayer. This check is

used to identify problems such as gate to contact over active shorting and contact/via to near-by metal proximity issues. This check does not look at contacts or vias over poly or metal.

InterLayerSpaceCheck (ILSC) measures the distance from the outside edge of the PV-band for the geometry on the first input layer to the outside edge of the PV-band for the geometry on the second input layer. A violation is flagged when the distance is less than -minLFDspace.

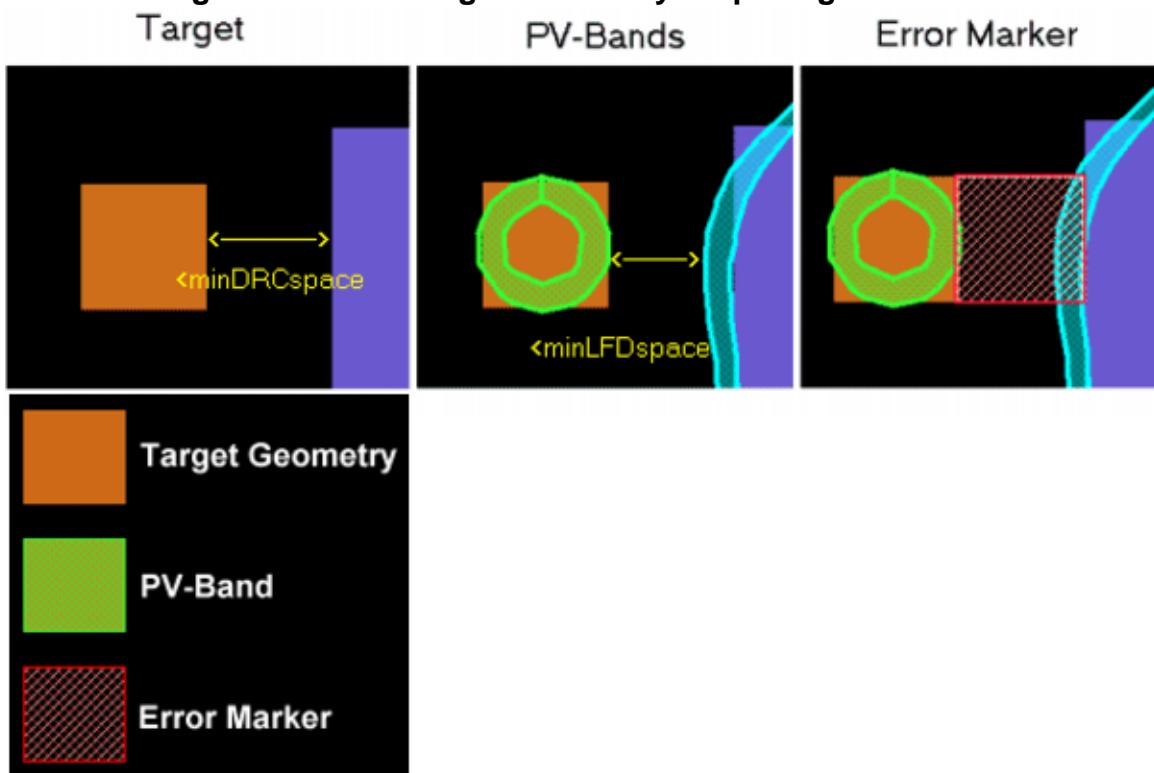
Figure 7-6. InterLayerSpaceCheck



This function writes all errors discovered by the check to the Calibre nmDRC RDB. It also associates a score to each error and writes it to the [Check Database](#) specified by the **-database** argument. The score is calculated as the area of the model-based violation.

If used with a PDK, this function calls an InterLayerSpaceCheck defined in the PDK and runs it for the specified layers, writing check results to the specified database.

Figure 7-7. Checking for Inter-Layer Spacing Problems



Arguments

- **-layer1 *layer1_name***

Required keyword and argument defining the name of the first layer you are checking. This is the first layer for which PV-bands are generated.

- **-layer2 *layer2_name***

Required keyword and argument defining the name of the second layer you are checking. This is the second layer for which PV-bands are generated.

- **-subwindow *expr_number***

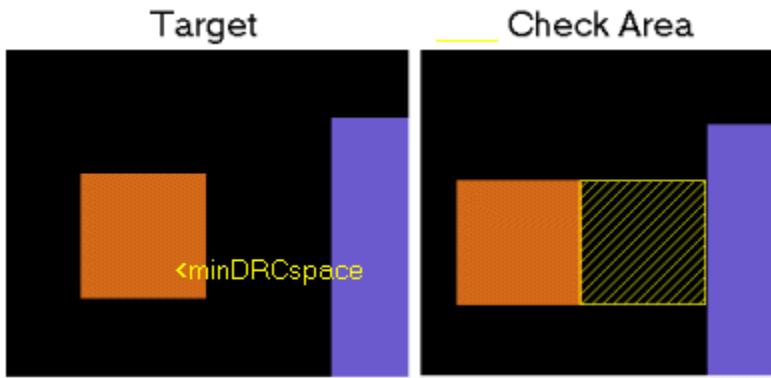
Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, **-subwindow 1_N** causes the check to only run on the north shift).

- **-minDRCspace** *target_dist*

Required keyword and argument used to constrain the checks to those areas where target layer geometries are close enough to one another to be at risk. The function ignores areas where the distance between target features is greater than *target_dist*.

The value *target_dist* is greater than the minimum spacing constraint for layer but should not be larger than the minimum spacing plus two times the minimum width.



- **-minLFDspace** *space*

Required keyword and argument defining the PV-band spacing violations. PV-band geometries whose external PV-band edges are \leq *space* microns away from the external edges of PV-band for the other input layer are flagged as an error.

- **-security** {no | yes}

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer** *layer_name*

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on the layer.

In the case of a Calibre LFD region, if a marker layer is specified, then this marker layer is combined with the Calibre LFD region. If a marker layer is not specified, the Calibre LFD region is the default marker layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-referenceLayer1** “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the first drawn layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer1**.

- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -referenceLayer2 “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the second drawn layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer2**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -anchorLayer1 “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the first drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer1.
- “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer1**.
- “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-anchorLayer2** “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the second drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer2.
 - “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer2**.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-layer1ContourCondition** *contour_condition1*

Optional keyword and argument to define **-layer1** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- **-layer2ContourCondition** *contour_condition2*

Optional keyword and argument to define **-layer2** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-ext_side *ext_side_value***

Optional keyword and argument defining how drawn layer edges are selected to satisfy the minDRCspace condition, expressed as a real number. This is useful in cases where the tool could miss errors due to non-overlapping edges. The default value for *ext_side_value* is minLFDspace/2.

- **-property *Space***

Optional keyword and argument specifying whether or not the check should save error measurement data to the RDB database. When *-property Space* is specified, the check reports the distance from the target line-end to the tip of feature. By default, property values are not saved.

- **-checkName *cName***

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-priority *cPriority***

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 2, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-indexFilter *value***

Optional keyword and value used to filter out inconsequential errors. Only errors having an index greater than or equal to this value are written to the [Check Database](#).

The index is defined as:

$$\frac{area(I_{abs})}{area(I_{orig})}$$

where:

I_{abs} = sum of the absolute PV-bands (for both layers) within the area of interest

I_{marker} = area of the associated error marker

For this check, the area of interest is defined relative to critical dimension (CD).

- **-minMarkerWidth *width_size***
Optional keyword and argument defining the minimum width for error markers sent to the [Check Database](#). Any error for which the error marker is smaller than the width is ignored. *width_size* must be specified in user units (um). By default, *width_size* is 0. This option allows you to remove thin width error markers from Calibre LFD check results, as thin, long errors can negatively affect Calibre LFD runtime.
- **-suppressPVI { no | yes }**
Optional keyword and argument specifying whether or not to suppress calculation of PVI. When “-suppressPVI yes” is specified, the check suppresses calculation of PVI. By default, PVI calculation is not suppressed.
- **-classify *handle***
Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.
- **-comment *comment_text***
Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of the type. If not specified, the comment for an ILSC violation is “Minimum Inter Layer Space Violation: Improve symmetry. If feature is dense, increase spacing. Reduce surrounding feature width.”

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-appendMarker *extra_markers_layer***
Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::StructureOptimizer.
- **-contour1Handle *contour1_handle***
Optional keyword and argument specifying the name of the first layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.
- **-contour2Handle *contour2_handle***
Optional keyword and argument specifying the name of the second layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.
- **-database *db_name***
Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.
- **-layerOut *return_layer_name***
Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

Note

 A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***
Required keyword and argument specifying the name of the check template defining how the check is performed.
- ***additional_options***
Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for the command. These can be any of the options for the non-PDK InterLayerSpaceCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

Verify that all polysilicon-to-contact spacing between the outer edges of their outer PV-bands are at least 0.060 apart across all the conditions specified by subwindow 1. All violations have

an original geometric separation value of less than or equal to 0.10 and are reported to *minInterLayerSpaceCheck.rdb*:

```
LFD::InterLayerSpaceCheck -layer1 poly -layer2 contact -subwindow 1 \
    -minDRCspace 0.10 -minLFDspace 0.06 \
    -database minInterLayerSpaceCheck.rdb
```

LineEndCheck

Checks the distance by which a line-end deviates from the target edge.

Usage

Syntax 1

LineEndCheck

```
-layer input_layer_name
-subwindow expr_number
-maxLFDdistance deviation_dist
[-displacement {min | max}]
-maxLineEnd line_end_width
[-minLEAlength min_length]
[-security {no | yes}]
[-layer1 support_layer1 [-layer2 support_layer2]]
[-markerLayer layer_name]
[-markerLength marker_len]
[-anchorLayer "%reference" | "%drawn" | "%retarget" | layer_name]
[-referenceLayer "%drawn" | "%retarget" | layer_name]
[-layerContourCondition contour_condition]
[-property {no | yes | Max}]
[-checkName cName]
[-priority cPriority]
[-indexFilter value]
[-comment comment_text]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contourHandle contour_handle]
{-database db_name | -layerOut return_layer_name
 / -database db_name -layerOut return_layer_name}
```

Syntax 2

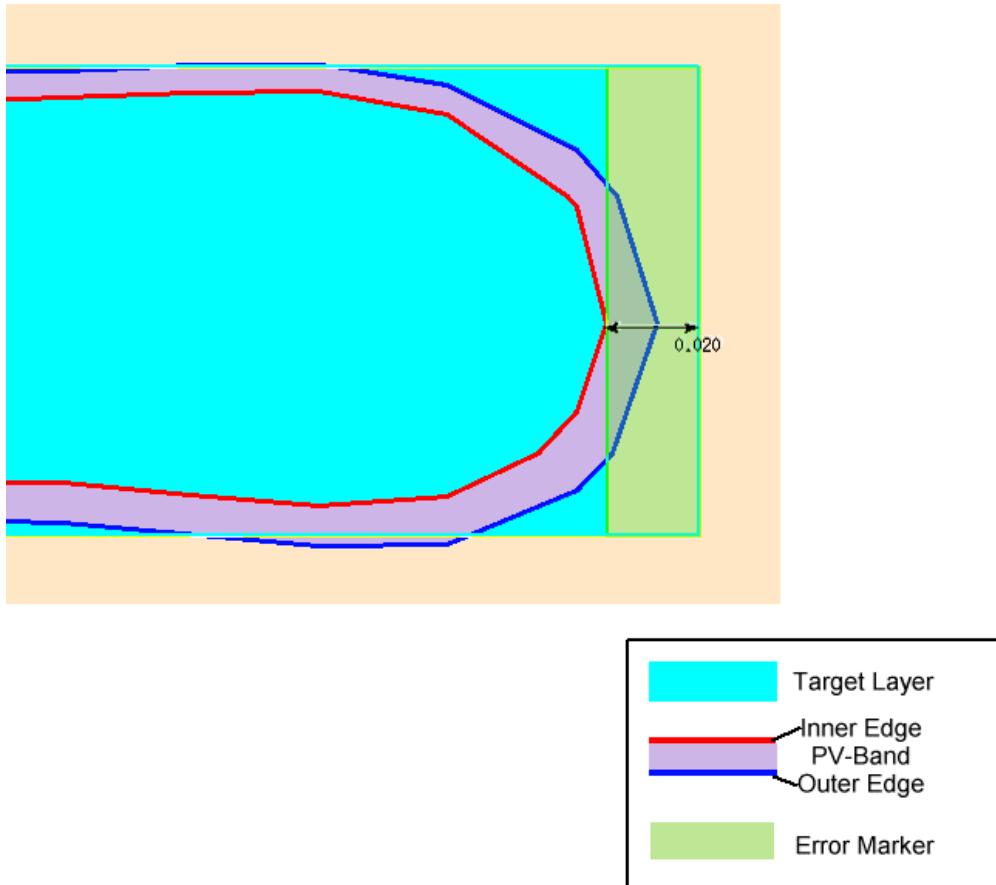
LineEndCheck

```
-layer input_layer_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

Description

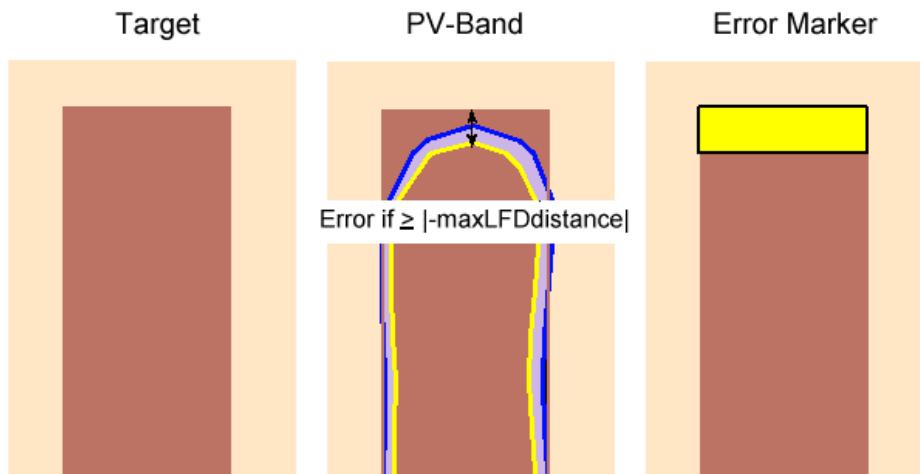
LineEndCheck (LEC) measures the distance by which a line-end deviates from the target edge and reports errors when this deviation is greater than the specified amount. This check is useful for evaluating line-end pullbacks, for which the PV-band is inside the target edge, and line-end extensions, for which the PV-band is outside the target edge.

This check evaluates only the line-ends on a single layer. Supply the names of contact or via layers to constrain the check to only those line-ends adjacent to a contact or via.



The LineEndCheck function writes all errors discovered by the check to the Calibre nmDRC RDB. It also associates a score to each error and writes it to the [Check Database](#) specified by the **-database** argument. The score is calculated as the area of the model-based violation.

If used with a PDK, this function calls a LineEndCheck defined in the PDK and runs it for the specified layer, writing check results to the specified database.

Figure 7-8. How LineEndCheck Works

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is a layer for which PV-bands are generated and must contain the line ends to check.

- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, **-subwindow 1_N** causes the check to only run on the north shift).

- **-maxLFDdistance *deviation_dist***

Required keyword and argument defining the maximum allowed deviation from the target. Any line-end for which the PV-band is farther than the distance from the target line-end is considered an error.

- When the ***deviation_dist*** is a negative number, the check identifies problems caused by line-end shortening, or pullbacks. In this case, the PV-band for the line-end is found inside of the target edge.
- When the ***deviation_dist*** is a positive number, the check identifies problems caused by lineend lengthening, or extensions. In this case, the PV-band for the line-end is found outside of the target edge.

- **-displacement {min | max}**

Optional keyword used to specify which edge of the PV-band to use in measuring line-end displacement. The default varies to the kind of line-end under investigation:

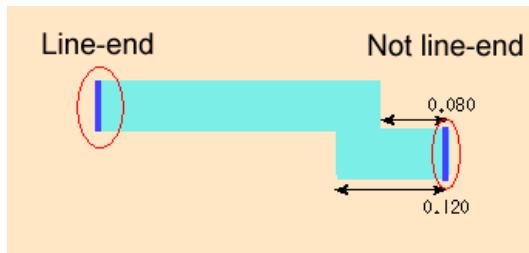
- For pullbacks, the default is “min,” which measures to the inner edge of the PV-band.
- For extensions, the default is “max,” which measures to the outer edge of the PV-band.

- **-maxLineEnd *line_end_width***

Required keyword and argument used to constrain the checks to line-ends having a width less than or equal to ***line_end_width***.

- **-minLEAlength *min_length***

Optional keyword used to constrain the check to those line-ends having both adjacent edges equal to or greater than the specified length. The default value is $2 * \text{maxLineEnd}$.



Assume **-minLEAlength = 0.010**

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-layer1 *support_layer1* -layer2 *support_layer2***

A pair of optional keywords and arguments used to supply one or two additional layers against which to check line ends. These layers must be original layers (not PV-band layers) and should contain either contacts or vias.

The check evaluates only those line-ends within a distance of **-minLEAlength** from at least one of the contacts or vias on these layers.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on ***layer_name***. The function ignores areas outside polygons on the layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-markerLength** *marker_len*

Optional keyword and argument defining the length of the markers to create for each error. The default marker length is 10*dbu. One database unit equals one micrometer times the precision.

- **-anchorLayer** “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the input drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to **-referenceLayer**.
 - “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer**.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-referenceLayer** “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the input layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer**.
 - “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-layerContourCondition** *contour_condition*

Optional keyword and argument to define a **-layer** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.

- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-property { no | yes | Max }**

Optional keyword and argument specifying whether or not the check should save error measurement data to the RDB database.

- no — The check does not report the error measurement. By default, property values are not saved.
- yes — The check reports the error measurement.
- Max — The check reports the maximum distance from the target line-end to the tip of feature.

- **-checkName *cName***

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-priority *cPriority***

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 1, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-indexFilter *value***

Optional keyword and value used to filter out inconsequential errors. Only errors having an index greater than or equal to the value are written to the [Check Database](#).

The index is defined as:

$$\frac{area(I_{abs})}{area(I_{target})}$$

where:

I_{abs} = the area of the absolute PV-band.

I_{target} = the area of the target geometry.

- -comment *comment_text*

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of the type. If not specified, the comment is “LineEndCheck Violation: Try increasing the width of the line or increasing the spacing between facing features.”

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- -classify *handle*

Optional keyword and argument used to define a handle to point to an LFD::ClassifyConfig object.

- -appendMarker *extra_markers_layer*

Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::StructureOptimizer.

- -contourHandle *contour_handle*

Optional keyword and argument used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-database** *db_name*

Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how the check is performed.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK LineEndCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

```
LFD:::LineEndCheck \
    -layer poly \
    -maxLFDdistance 0.15 \
    -maxLineEnd 0.01 \
    -subwindow 1 \
    -database $lfdErrorDb
```

MaxAreaVariabilityCheck

Performs variability checks on interactions between the two specified layers.

Usage

Syntax 1

```
MaxAreaVariabilityCheck
  -layer1 layer1_name
  -layer2 layer2_name
  -subwindow expr_number
  -minAreaChange area_size
  [-security {no | yes}]
  [-markerLayer layer_name]
  [-referenceLayer1 "%drawn" | "%retarget" | layer_name]
  [-minContourCondition contour_condition]
  [-maxContourCondition contour_condition]
  [-checkName cName]
  [-priority cPriority]
  [-comment comment_text]
  [-classify handle]
  [-appendMarker extra_markers_layer]
  [-maxContourHandle max_contour_handle -minContourHandle min_contour_handle]
  {-database db_name / -layerOut return_layer_name
   / -database db_name -layerOut return_layer_name}
```

Syntax 2

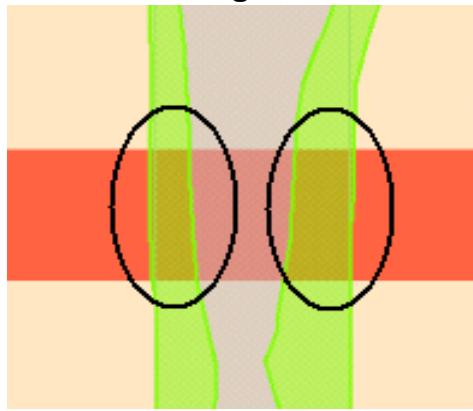
```
MaxAreaVariabilityCheck
  -layer1 layer1_name
  -layer2 layer2_name
  -pdkCheckName check_template
  -database db_name
  [additional_options]
```

Description

Performs a variability check on PV-band geometries in areas where the target **-layer1** overlaps the target **-layer2**. If the **-markerLayer** option is used, the check is limited to PV-band data which lies inside geometries on the marker layer. This check is used to identify problems such as large resistance fluctuations and poly gate critical dimension (CD) control issues.

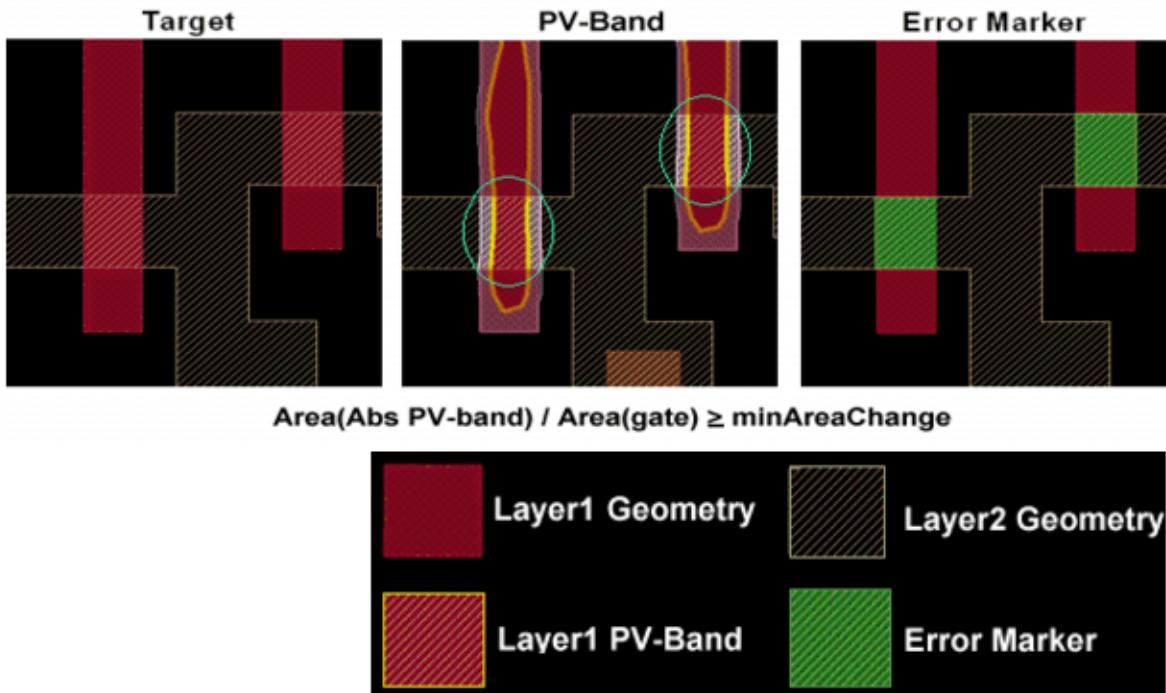
MaxAreaVariabilityCheck (MAVC) checks the area between the inside and outside edges of a PV-band. A violation exists when the area of the absolute PV-band, normalized by the **-layer1** and **-layer2** intersection area, is greater than or equal to that specified by the **-minAreaChange** option. In situations such as is shown in [Figure 7-38](#), the check sums the two areas, and then normalizes that value by the **-layer1** and **-layer2** intersection. The check flags violations only on areas in which the PV-band data for **-layer1** intersects **-layer2**.

Figure 7-9. Checking PV-Band Variability



This check writes all discovered errors to the Calibre nmDRC RDB. The function associates a score with each error, and writes it to the [Check Database](#) specified by the **-database** argument. The score is calculated as the area of the model-based violation.

Figure 7-10. How MaxAreaVariabilityCheck Works



If used with a PDK, the function calls a `MaxAreaVariabilityCheck` defined in the PDK and runs it for the specified layers, writing check results to the specified database.

Note

 This check is not commutative with respect to **-layer1** and **-layer2**. The two operations shown below produce different results:

```
MaxAreaVariabilityCheck -layer1 A -layer2 B -subwindow 1 ...
MaxAreaVariabilityCheck -layer1 B -layer2 A -subwindow 1 ...
```

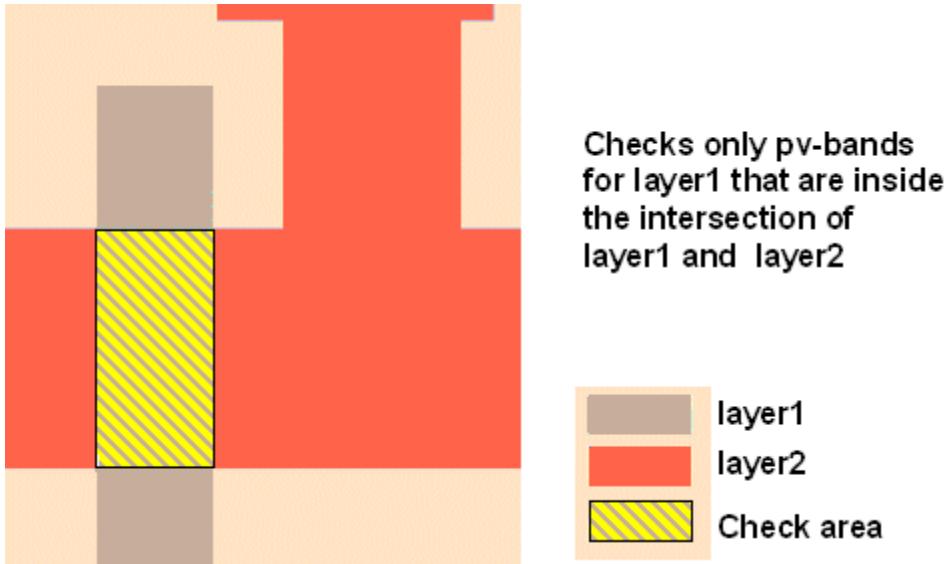
Arguments

- **-layer1 *layer1_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-layer2 *layer2_name***

Required keyword and argument defining an additional target layer, with respect to which **-layer1** is checked. This function evaluates those areas where **-layer1** overlaps **-layer2**.



- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, **-subwindow 1_N** causes the check to only run on the north shift).

- **-minAreaChange *area_size***

Required keyword and argument defining the minimum variability considered to be problematic. Variability is expressed as a ratio of Absolute PV-band area to interaction area. This check reports on variability that is greater than the value specified for this keyword.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to contours that lie within polygons on *layer_name*. The check ignores areas outside polygons on the layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to -markerLayer, or a circular layer definition results.

- **-referenceLayer1 “%drawn” | “%retarget” | *layer_name***

Optional keyword and argument to have the check measurements calculated on a different layer than the input layer to the checks for which the PV-bands have been generated. The option is associated with the first drawn layer input to the check.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-maxContourCondition *contour_condition***

Optional keyword and argument used to define certain contours to be used as the maximum contour for **-layer1**. The *contour_condition* can be defined as follows:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer defines the order of the contour to be checked in the subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{*optical1 dose1 size1 [resist1 etch1]* [*optical2 dose2 size2 [resist2 etch2]*]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- **-minContourCondition** *contour_condition*

Optional keyword and argument used to define certain contours to be used as the minimum contour for **-layer1**. The *contour_condition* can be defined as follows:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer defines the order of the contour to be checked in the subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-checkName** *cName*

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-priority** *cPriority*

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 3, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-comment** *comment_text*

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of the type. If not specified, the comment for a MAVC violation is “Area Variability Violation: Improve symmetry. Avoid low aspect ratio figures. Use long rectangular structures.”

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-classify handle**
Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.
- **-appendMarker extra_markers_layer**
Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).
- **-maxContourHandle max_contour_handle**
Optional keyword and argument specifying the name of the maximum layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.
- **-minContourHandle min_contour_handle**
Optional keyword and argument specifying the name of the minimum layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.
- **-database db_name**
Required keyword and argument defining the RDB to which spacing values for violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.
- **-layerOut return_layer_name**
Required keyword and argument defining the name of a derived layer to which the violations identified by the check is written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.
- **-pdkCheckName check_template**
Required keyword and argument specifying the name of the check template defining how the check is performed.
- **additional_options**
Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for the command. These can be any of the options for the non-PDK MaxAreaVariabilityCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

Verify that polysilicon gates do not vary more than 20% with respect to design intent across all conditions specified by subwindow 2:

```
LFD::MaxAreaVariabilityCheck -layer1 poly -layer2 active -subwindow 2 \
    -minAreaChange 0.20 -database maxAreaVariability.rdb
```

MaxCDVariabilityCheck

Performs variability checks on the change in CD between simulated and drawn features.

Usage

Syntax 1

```
MaxCDVariabilityCheck
  -layer input_layer_name
  -subwindow expr_number
  -minDRCwidth min_target_width
  -maxDRCwidth max_target_width
  -mode {absolute | ratio}
  -maxCDvariability max_var
  [-security {no | yes}]
  [-markerLayer layer_name]
  [-referenceLayer "%drawn" | "%retarget" | layer_name]
  [-anchorLayer "%reference" | "%drawn" | "%retarget" | layer_name]
  [-maxContourCondition contour_condition]
  [-minContourCondition contour_condition]
  [-minLineEnd line_end_dist]
  [-checkName cName]
  [-property {None | All | Max | MaxRatio | TargetCD}]
  [-priority cPriority]
  [-minMarkerWidth width_size]
  [-comment comment_text]
  [-classify handle]
  [-appendMarker extra_markers_layer]
  {-database db_name | -layerOut return_layer_name
   / -database db_name -layerOut return_layer_name}
```

Syntax 2

```
MaxCDVariabilityCheck
  -layer input_layer_name
  -pdkCheckName check_template
  -database db_name
  [additional_options]
```

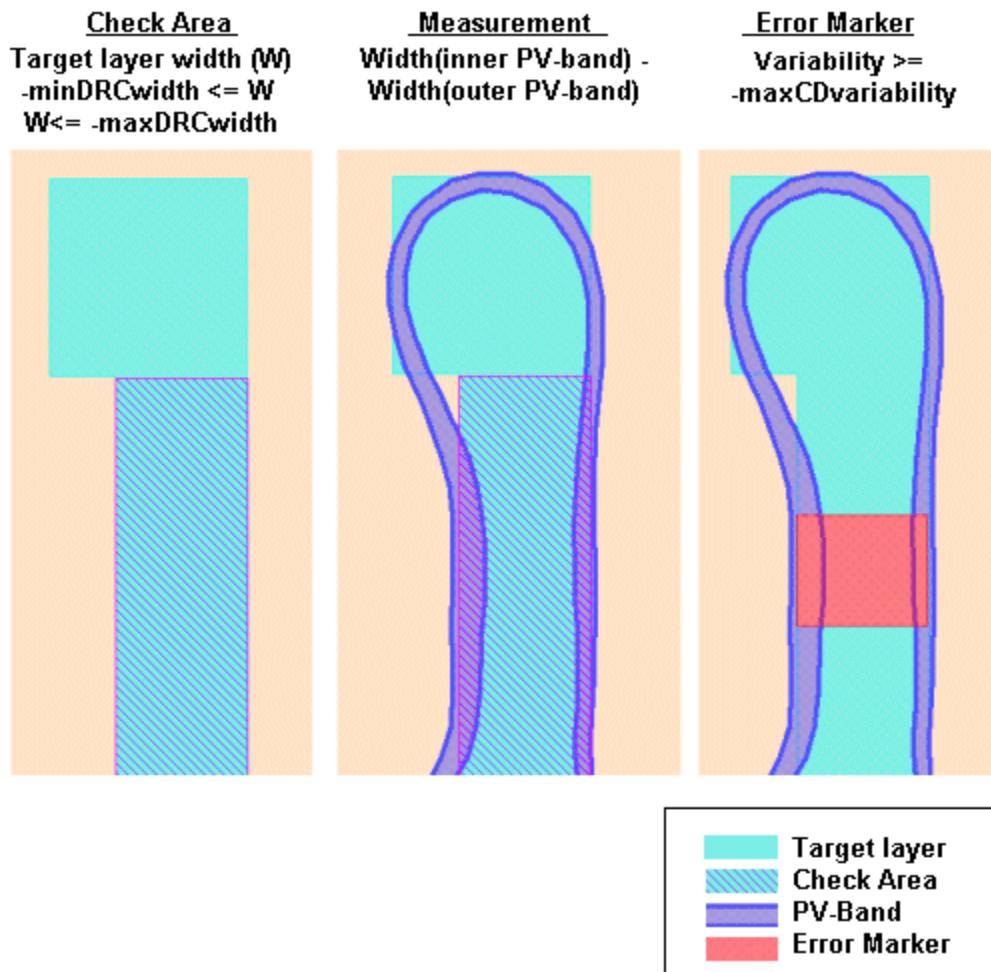
Description

For sensitive areas, as defined by a minimum and maximum width on the target layer, MaxCDVariabilityCheck checks how much the critical dimension (CD) changes over the process window. Violations represent areas on the PV-band where the variability is greater than **-maxCDvariability** or the PDK-specified amount.

This check is used to identify portions of the design that are most sensitive to process variations.

If used with a PDK, this function calls a MaxCDVariabilityCheck defined in the PDK and runs it for the specified layer, writing check results to the specified database.

Figure 7-11. Checking CD Variability

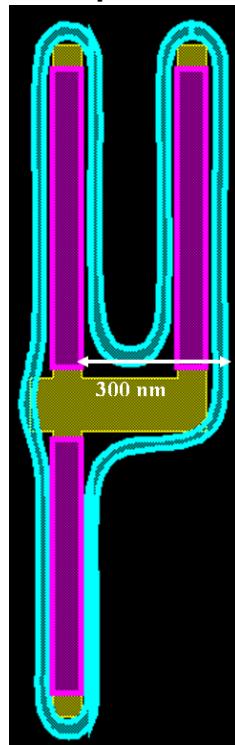


LFD::MaxCDVariabilityCheck generates two types of error markers:

- Typical error markers that show a CD variability violation.
- Exceptions-only error markers which indicate that the check was not able to find the layer's exterior PV-band within the maximum search distance from the target edge.

Properties attached to the exceptions-only error marker have a value of zero (0). [Figure 7-12](#) shows an example of an exceptions-only violation. The distance from the drawn poly edge to the nearest poly maximum PV-band contour edge is more than the maximum search distance used in this case (which is 0.07). The default maximum search value for MaxCDVariabilityCheck is maxDRCwidth * 2.

Figure 7-12. Exceptions-Only Violation



Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, **-subwindow 1_N** causes the check to only run on the north shift).

- **-minDRCwidth *min_target_width***

Required keyword and argument used to constrain the checking to features having a width that is within a specified range. The function ignores areas where the width of the target features is less than ***min_target_width***.

- **-maxDRCwidth *max_target_width***

Required keyword and argument used to constrain the checking to features having a width that is within a specified range. Generally those areas where target layer geometries are

wide enough to be at risk. The function ignores areas where the width of the target feature is greater than *max_target_width*.

- **-mode {absolute | ratio}**

Required keyword specifying the mode to use to evaluate CD variability.

- **absolute** — express CD variability in microns:

$$Variation = \text{width}(PVband_{inner}) - \text{width}(PVband_{outer})$$

- **ratio** — express CD variability as a ratio:

$$Variation = \frac{\text{width}(PVband_{inner}) - \text{width}(PVband_{outer})}{\text{width}(\text{target})}$$

- **-maxCDvariability *max_var***

Required keyword and argument defining the maximum variability allowed.

- When the mode is **absolute**, this is expressed as a real number representing the maximum change in width allowed, in microns. This check flags an error when the variation is greater than this value.

- When the mode is **ratio**, and variability is expressed as the ratio, $\frac{\Delta PV - band}{target}$ this is the maximum ratio allowed. This check flags an error when the variation is greater than this value.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on the layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

Use of this keyword to limit the check to potential problem areas is highly recommended.

- **-referenceLayer “%drawn” | “%retarget” | *layer_name***

Optional keyword and argument to have the check measurements calculated on a different layer than the input layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer**.

- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -anchorLayer “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the input drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer.
 - “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with -layer.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
 - -maxContourCondition *contour_condition*
- Optional keyword and argument used to define certain contours to be used as the maximum contour for -layer1. The *contour_condition* can be defined as follows:
- min — Specifies the command works on the inner PV-band contour.
 - max — Specifies the command works on the outer PV-band contour.
 - *integer* — This integer defines the order of the contour to be checked in the subwindow.
 - *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- **-minContourCondition** *contour_condition*

Optional keyword and argument used to define certain contours to be used as the minimum contour for **-layer1**. The *contour_condition* can be defined as follows:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer defines the order of the contour to be checked in the subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note



This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-minLineEnd** *line_end_dist*

Optional keyword and argument providing the definition of a line end error distance. Any error this close to a line end, or closer, is considered a line-end error, and its error marker is removed from the final error group. The default value for *line_end_dist* is -minDRCwidth.

- **-checkName** *cName*

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

-property {None | All | Max | MaxRatio | TargetCD}

Optional keyword and argument specifying whether or not the check should save error measurement data to the RDB database. The check reports the specified property measurements for the feature as a property. The argument to this keyword must be one of the following:

- None — Default. Error measurement data is not reported.
- All — Reports Max, MaxRatio, and TargetCD.
- Max — The maximum change in width measured for the feature.
- MaxRatio — The ratio of the PV-band variability to the TargetCD.

- TargetCD — The CD of the drawn feature.
- **-priority *cPriority***
Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 3, as defined in [Table 7-15](#). *cPriority* must be an integer value.
- **-minMarkerWidth *width_size***
Optional keyword and argument defining the minimum width for error markers sent to the [Check Database](#). Any error for which the error marker is smaller than this width is ignored. *width_size* must be specified in user units (um). By default, *width_size* is 0. This option allows you to remove thin width error markers from Calibre LFD check results, as thin, long errors can negatively affect Calibre LFD runtime.
- **-comment *comment_text***
Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. There is no default value for this argument.
The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```
- **-classify *handle***
Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.
- **-appendMarker *extra_markers_layer***
Optional keyword and argument to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).
- **-database *db_name***
Required keyword and argument defining the RDB to which spacing values for violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.
- **-layerOut *return_layer_name***
Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.
A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how the check is performed.

- *additional_options*

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK MaxCDVariabilityCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

```
LFD::MaxCDVariabilityCheck \
-layer metall \
-subwindow 1 \
-mode absolute \
-minDRCwidth 0.15 \
-maxDRCwidth 0.05 \
-maxCDvariability absolute \
-database minWidthCheck.rdb
```

MinAreaOverlapCheck

Performs overlap checks between the two specified layers.

Usage

Syntax 1

MinAreaOverlapCheck

```
-layer1 layer1_name
-layer2 layer2_name
-subwindow expr_number
-minOverlapArea area_size
[-minLFDfilter filter]
-areaType {absolute | relative}
[-security {no | yes}]
[-markerLayer layer_name
[-referenceLayer1 "%drawn" | "%retarget" | layer_name]
[-referenceLayer2 "%drawn" | "%retarget" | layer_name]
[-anchorLayer1 "%reference" | "%drawn" | "%retarget" | layer_name]
[-anchorLayer2 "%reference" | "%drawn" | "%retarget" | layer_name]
[-layer1ContourCondition contour_condition1]
[-layer2ContourCondition contour_condition2]
[-maxExtent extent_size]
[-priority cPriority]
[-checkName cName]
[-comment comment_text]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contour1Handle contour1_handle -contour2Handle contour2_handle]
{-database db_name | -layerOut return_layer_name
 / -database db_name -layerOut return_layer_name}
```

Syntax 2

MinAreaOverlapCheck

```
-layer1 layer1_name
-layer2 layer2_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

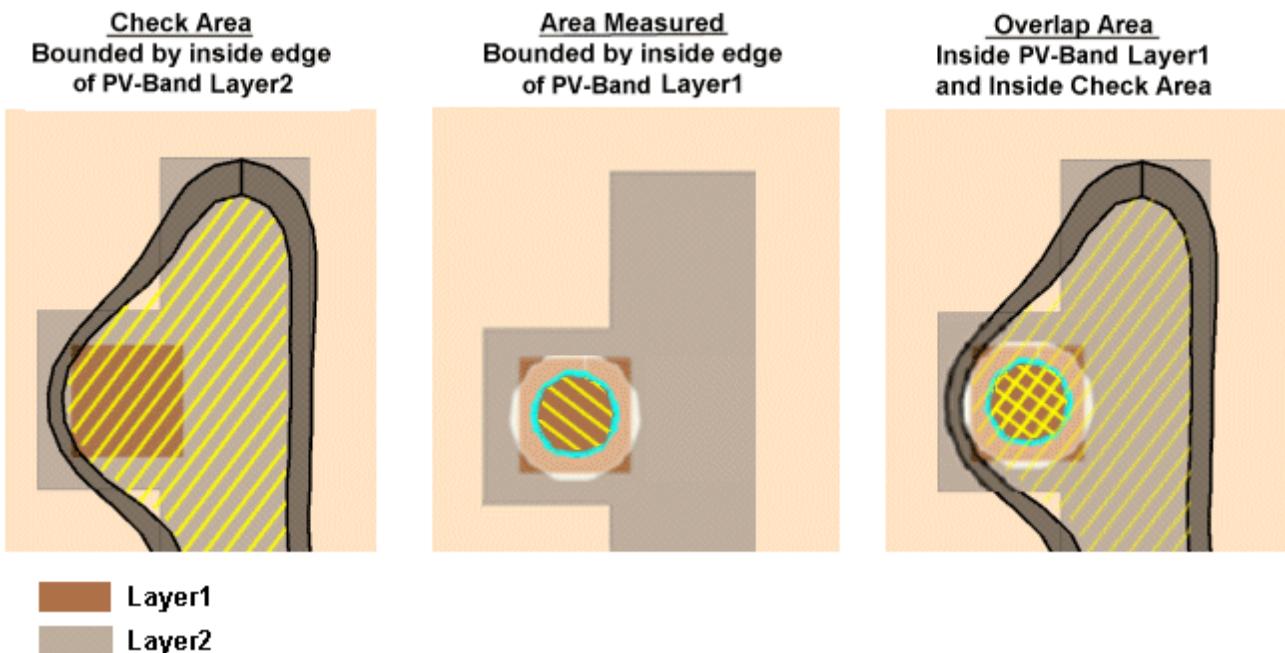
Description

Performs an overlap check on PV-band geometries in areas where target *layer1_name* overlaps target *layer2_name*. If -markerLayer is specified, the check is limited to PV-band data that lies inside geometries on the -markerLayer. This check flags violations only on the PV-band data for *layer1_name*. Violations represent areas on the PV-band that overlap PV-band data for *layer2_name* by-minOverlapArea or less.

This check is used to identify problems with resistance.

MinAreaOverlapCheck (MAOC) checks the area bounded by inside edge of a PV-band. You can write this check either in terms of **absolute** overlap areas or in terms of the PV-band overlap **relative** to the target layer overlap.

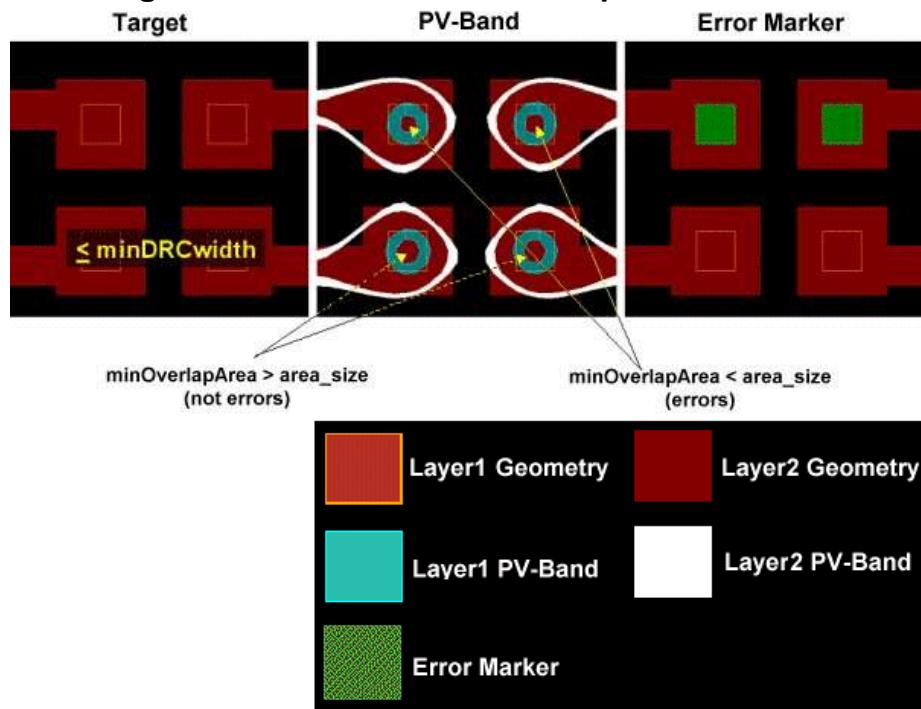
Figure 7-13. Checking PV-Band Overlap



This function writes all errors discovered by the check to the Calibre nmDRC RDB. It also associates a score to each error and writes it to the [Check Database](#) specified by the **-database** argument. The score is calculated as the area of the model-based violation.

If used with a PDK, this function calls a MinAreaOverlapCheck defined in the PDK and runs it for the specified layers, writing check results to the specified database.

Figure 7-14. How MinAreaOverlapCheck Works



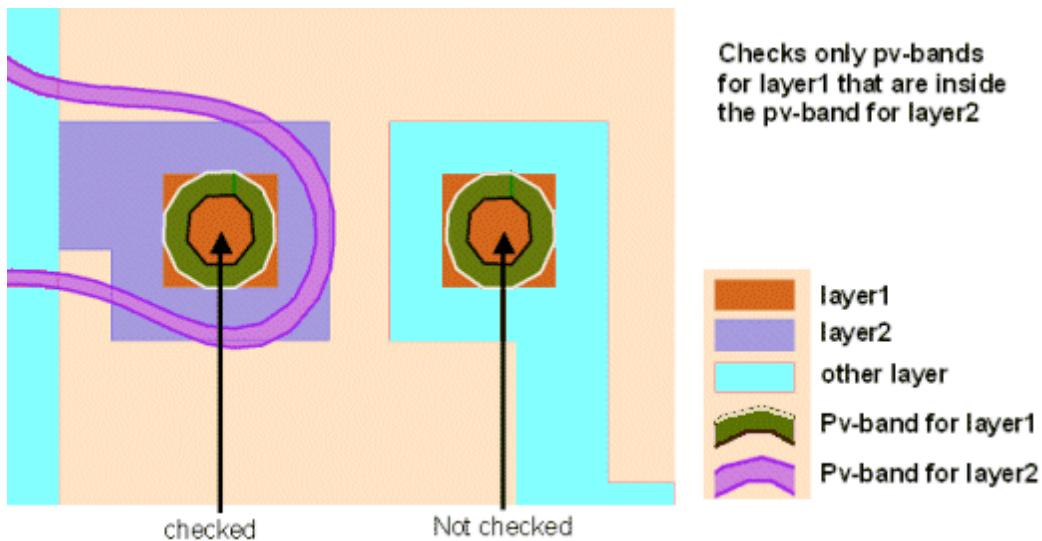
Arguments

- **-layer1 *layer1_name***

Required keyword and argument defining the name of the first design layer you are checking. This is the layer for which PV-bands are generated.

- **-layer2 *layer2_name***

Required keyword and argument defining an additional design layer, with respect to which *layer1_name* is checked. This is another layer for which PV-bands are generated. This function evaluates those areas where *layer1_name* overlaps *layer2_name*.



- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments. This value is used with both ***layer1_name*** and ***layer2_name***. For example, if ***expr_number*** = 3, the check evaluates the PV-band for ***layer1_name***, experiment 3 against the PV-band for ***layer2_name***, experiment 3.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, -subwindow 1_N causes the check to only run on the north shift).

- **-minOverlapArea *area_size***

Required keyword and argument defining the minimum size of overlap area to check for. You must take **-areaType** into consideration when choosing the value for ***area_size***.

- **-minLFDfilter *filter***

Optional keyword and value used to filter out geometries that are non-resolving. If the area bounded by inside edge of a PV-band for ***layer1_name*** is \leq ***filter***, that polygon is ignored by the check. The ***filter*** is expressed in square microns.

- **-areaType {absolute | **relative**}**

Required keyword and argument defining how area is calculated:

- absolute — The actual size of the overlap, expressed as microns square.
- relative — The size of the overlap relative to the area being examined, calculated as the ratio of <size of PV-band overlap>/<design layer overlap>.

Overlap is calculated based on the inner edge of the PV-bands. Output properties are Area and AreaRatio.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on ***layer_name***. The function ignores areas outside polygons on the layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-referenceLayer1 “%drawn” | “%retarget” | *layer_name***

Optional keyword and argument to have the check measurements calculated on a different layer than the first drawn layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer1**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -referenceLayer2 “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the second drawn layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer2**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -anchorLayer1 “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the first drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer1.
- “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer1**.
- “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-anchorLayer2** “%reference” | “%drawn” | “%retarget” | *layer_name*
Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the second drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer2.
 - “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer2**.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-layer1ContourCondition** *contour_condition1*

Optional keyword and argument to define **-layer1** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- **-layer2ContourCondition** *contour_condition2*

Optional keyword and argument to define **-layer2** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-maxExtent *extent_size***

Optional keyword and argument used to modify the Calibre OPCverify tool's max_extent value option to the area_compute control inside of a package. The default *extent_size* is 3.0 microns. This is useful as the large default max_extent value can affect Calibre LFD performance.

- **-checkName *cName***

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-priority *cPriority***

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 2, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-comment *comment_text***

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. If not specified, the comment for a MAOC violation is "Minimum Overlap Area Violation: Improve symmetry of overlap layers. Increase common overlap area. Avoid low aspect ratio figures."

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-classify handle**

Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.

- **-appendMarker *extra_markers_layer***

Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).

- **-contour1Handle *contour1_handle***

Optional keyword and argument specifying the name of the first layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-contour2Handle *contour2_handle***

Optional keyword and argument specifying the name of the second layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-database *db_name***

Required keyword and argument defining the RDB to which spacing values for violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how the check is performed.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK MinAreaOverlapCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

Verify that all contacts overlap at least 0.003 micron square with the poly layer across all the conditions specified by subwindow 1:

```
LFD::MinAreaOverlapCheck -layer1 contact -layer2 poly -subwindow 1 \
    -minOverlapArea 0.003 -areaType absolute \
    -database minAreaOverlapCheck.rdb
```

MinOverlayCheck

Performs an enclosure check between two separate layers where the target layer geometries on the surrounding layer name encloses the geometry on the enclosed layer name.

Usage

Syntax 1

MinOverlayCheck

```
-layer1 enclosed_layer_name
-layer2 surrounding_layer_name
-subwindow expr_number
[-subwindow2 expr_number]
-minDRCencl encl
-minLFDencl encl
[-overlayError shift]
[-security {no | yes}]
[-markerLayer layer_name]
[-referenceLayer1 "%drawn" | "%retarget" | layer_name]
[-referenceLayer2 "%drawn" | "%retarget" | layer_name]
[-anchorLayer1 "%reference" | "%drawn" | "%retarget" | layer_name]
[-anchorLayer2 "%reference" | "%drawn" | "%retarget" | layer_name]
[-layer1ContourCondition contour_condition1]
[-layer2ContourCondition contour_condition2]
[-property Min]
[-checkName cName]
[-priority cPriority]
[-indexFilter value]
[-minMarkerWidth width_size]
[-suppressPVI {no | yes}]
[-comment comment_text]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contour1Handle contour1_handle -contour2Handle contour2_handle]
{-database db_name | -layerOut return_layer_name
 / -database db_name -layerOut return_layer_name}
```

Syntax 2

MinOverlayCheck

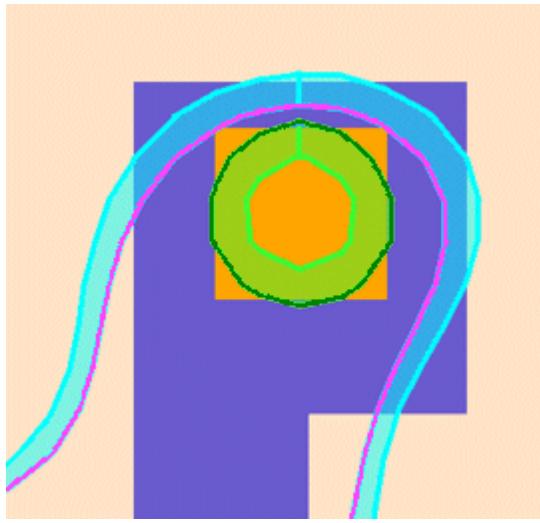
```
-layer1 layer1_name -layer2 layer2_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

Description

Performs an enclosure check between two PV-bands on separate layers where the target layer geometries are oriented such that the geometry on -layer2 surrounds the geometry on -layer1 and the enclosure distance is less than or equal to -minDRCencl. If -markerLayer is specified, the check is limited to PV-band data that lies inside geometries on -markerLayer. This check is designed to identify alignment as well as overlay errors between metal and poly as well as enclosed vias and contacts.

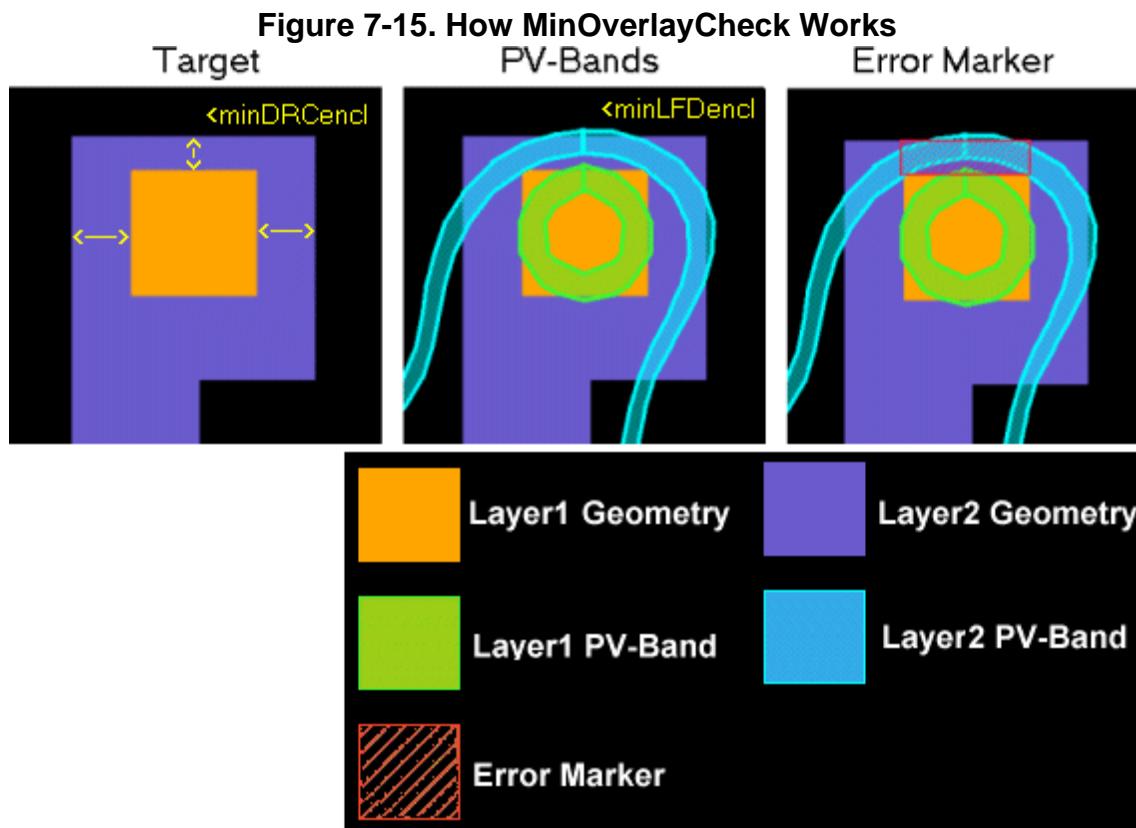
MinOverlayCheck (MOC) measures the distance between the outer edge of the PV-band for **-layer1** and the inner edge of the PV-band for **-layer2**. An error is flagged if this distance is less than or equal to -minLFDencl.

This check is not commutative. Results differs if **-layer1** and **-layer2** are reversed.



This function writes all errors discovered by the check to the Calibre nmDRC RDB. It also associates a score to each error and writes it to the [Check Database](#) specified by the **-database** argument. The score is calculated as the area of the model-based violation.

If used with a PDK, this function calls a MinOverlayCheck defined in the PDK and runs it for the specified layers, writing check results to the specified database.



Arguments

- **-layer1 *enclosed_layer_name***

Required keyword and argument defining the name of the first layer you are checking. Geometries on this layer are enclosed by -layer2. This is typically a via or contact layer.

- **-layer2 *surrounding_layer_name***

Required keyword and argument defining the name of the second layer you are checking. Geometries on this layer surround geometries on -layer1. This is typically a metal or poly layer.

- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, **-subwindow 1_N** causes the check to only run on the north shift).

- **-subwindow2** *expr_number*

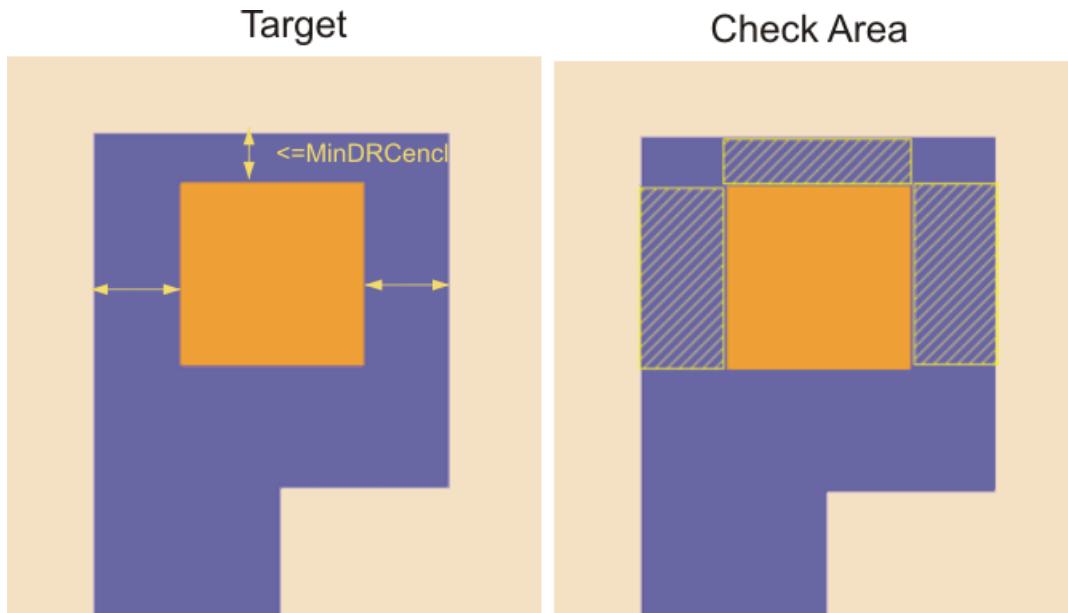
Optional keyword and argument defining the process variation experiment to which layer 2 in this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, *expr_number* refers to an index to a list of experiments. Setting *expr_number* to a value of “*subwindow#_shift*” causes the check to operate on shifts of an overlaid double-patterned **PVband**. If not specified, the check uses the value given to the **-subwindow** keyword by default.

In the following example, the check calculates the distance between a north shift in the contact layer and a south shift in the poly layer:

```
LFD:::MinOverlayCheck -layer1 contact -layer2 poly -minDRCencl 0.05 \
    -minLFDencl 0.04 -overlayError 0.01 -subwindow 1_N \
    -subwindow2 1_S -checkName MOC_1 -property min \
    -database $lfdErrorDatabase
```

- **-minDRCencl** *encl*

Required keyword and argument used to constrain the checks to those areas where problems are most likely to occur. The function ignores areas where the distance between the outer edge of the geometry on **-layer1** and the inner edge of the geometry on **-layer2** and the enclosure distance is greater than **-minDRCencl**.



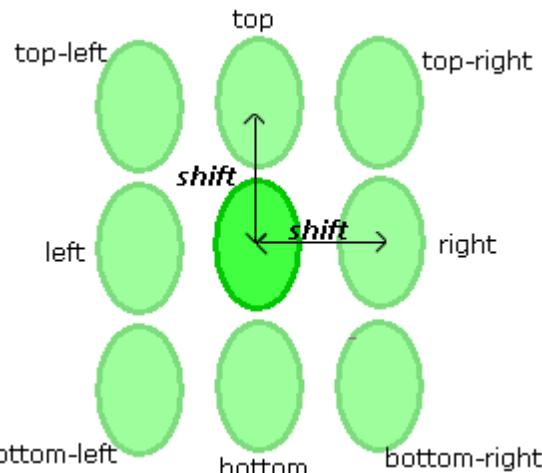
- **-minLFDencl** *encl*

Required keyword and argument defining the enclosure threshold value between the outer edge of **-layer1** and the inner edge of **-layer2**. PV-band geometries whose external PV-band edges are \leq *encl* microns away from the internal edge of the PV-band for the surrounding geometry are flagged as an error.

- **-overlayError *shift***

Optional keyword and argument defining a shift distance, expressed as a real number. The check evaluates PV-bands in the original position shifted by the specified amount. The shift distance represents the maximum distance by which to expect **-layer2** to be misaligned in respect to **-layer1**.

The check simulates misalignment by shifting polygon formed inside the inner edge of a **-layer2** PV-band by the specified distance in eight directions: top, bottom, left, right, top-left, top-right, bottom-left and bottom-right.



- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on this layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-referenceLayer1 “%drawn” | “%retarget” | *layer_name***

Optional keyword and argument to have the check measurements calculated on a different layer than the first drawn layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer1**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.

- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-referenceLayer2** “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the second drawn layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer2**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-anchorLayer1** “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the first drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to **-referenceLayer1**.
 - “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer1**.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
 - **-anchorLayer2** “%reference” | “%drawn” | “%retarget” | *layer_name*
- Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the second drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer2.
- “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer2**.
- “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -layer1ContourCondition *contour_condition1*

Optional keyword and argument to define **-layer1** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- -layer2ContourCondition *contour_condition2*

Optional keyword and argument to define **-layer2** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as

extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.

- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-property Min**

Optional keyword and argument specifying whether or not the check should save error measurement data to the RDB database. When “-property Min” is specified, the check reports the minimum change in width measured for the feature. By default, property values are not saved.

- **-checkName *cName***

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-priority *cPriority***

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 2, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-indexFilter *value***

Optional keyword and value used to filter out inconsequential errors. Only errors having an index greater than or equal to this value are written to the [Check Database](#).

The index is defined as:

$$\frac{\text{area}(I_{abs})}{\text{area}(I_{marker})}$$

where:

I_{abs} = sum of the absolute PV-bands (for both layers) within the area of interest

I_{marker} = area of the associated error marker

For this check, the area of interest is defined relative to overlay area.

- **-minMarkerWidth *width_size***
Optional keyword and argument defining the minimum width for error markers sent to the [Check Database](#). Any error for which the error marker is smaller than this width is ignored. *width_size* must be specified in user units (um). By default, *width_size* is 0. This option allows you to remove thin width error markers from Calibre LFD check results, as thin, long errors can negatively affect Calibre LFD runtime.
- **-suppressPVI { no | yes }**
Optional keyword and argument specifying whether or not to suppress calculation of PVI. When “-suppressPVI yes” is specified, the check suppresses calculation of PVI. By default, PVI calculation is not suppressed.
- **-comment *comment_text***
Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. If not specified, the comment for a MOC violation is “Minimum Overlay Violation: Increase the area of the enclosure layer.”

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-classify *handle***
Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.
- **-appendMarker *extra_markers_layer***
Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).
- **-contour1Handle *contour1_handle***
Optional keyword and argument specifying the name of the first layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-contour2Handle *contour2_handle***

Optional keyword and argument specifying the name of the second layer used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-database *db_name***

Required keyword and argument defining the RDB to which spacing values for violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how this check is performed.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK MinOverlayCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

Verify that all contacts enclosed by poly have an enclosure value of at least 0.055 across all the conditions specified by subwindow 1. All violations have an original geometric separation value of less than or equal to 0.10 and are reported to minOverlayCheck.rdb:

```
LFD::MinOverlayCheck -layer1 contact -layer2 poly -subwindow 1 \
                     -minDRCencl 0.10 -minLFDencl 0.055 -database minOverlayCheck.rdb
```

MinSpaceCheck

Performs spacing checks for the specified layer.

Usage

Syntax 1

MinSpaceCheck

```
-layer input_layer_name
-subwindow expr_number
-minDRCspace target_dist
{ -minLFDspace space | -LFDspaceRange from_space to_space }
[-security {no | yes}]
[-markerLayer layer_name]
[-referenceLayer "%drawn" | "%retarget" | layer_name]
[-anchorLayer "%reference" | "%drawn" | "%retarget" | layer_name]
[-layerContourCondition contour_condition]
[-checkName cName]
[-priority cPriority]
[-indexFilter value]
[-minMarkerArea area_size]
[-minMarkerWidth width_size]
[-bandType {absolute | regular}]
[-extendMarker extend_dist]
[-minSpaceEnd space_end_dist]
[-ext_side ext_side_value]
[-ratio yes]
[-property {property_list}]
[-comment comment_text]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contourHandle contour_handle]
{-database db_name | -layerOut return_layer_name
 / -database db_name -layerOut return_layer_name}
```

Syntax 2

MinSpaceCheck

```
-layer input_layer_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

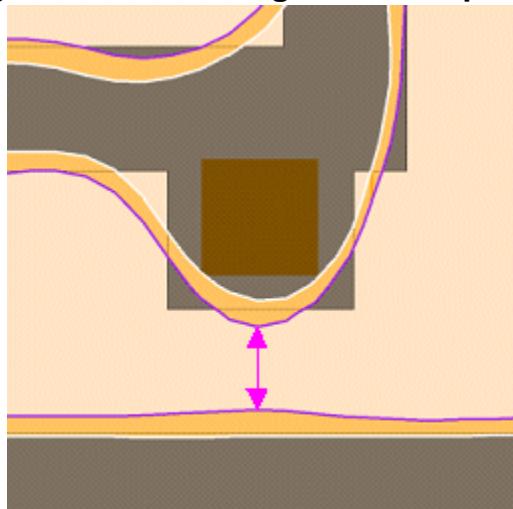
Description

Performs a spacing check between PV-band geometries in areas where target layer geometries are within a **-minDRCspace** from one another. If -markerLayer is specified, the check is limited

to PV-band data that lies inside geometries on -markerLayer. This check is used for identifying problems such as resist bridging for clear field mask and resist pinching for dark field mask.

MinSpaceCheck (MSC) checks the distance between outside edges of PV-bands. An error is flagged if this distance is less than or equal to **-minLFDspace**. When *space* < 0, this check also catches situations in which two PV-bands overlap.

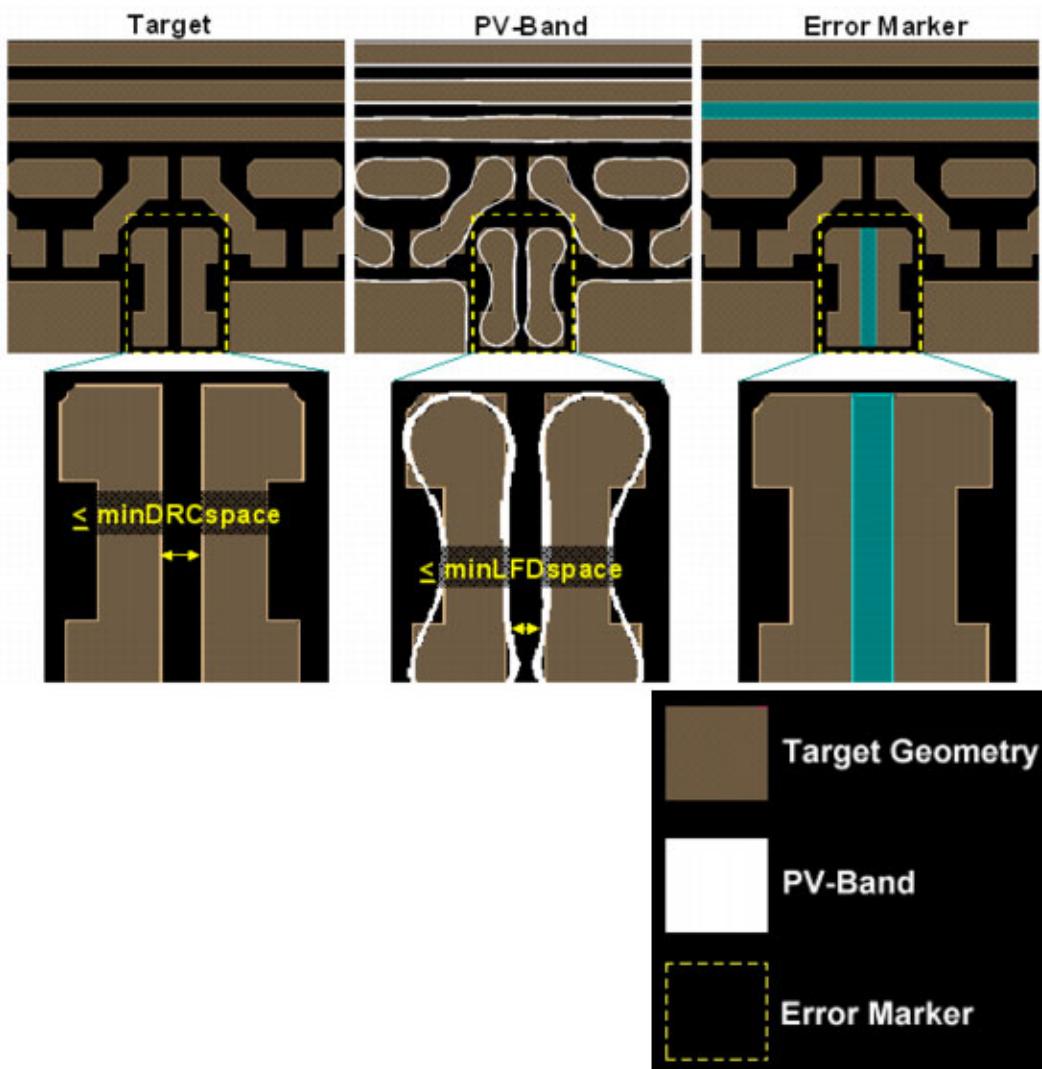
Figure 7-16. Checking PV-Band Spacing



This function writes all errors discovered by the check to the Calibre nmDRC RDB. It also associates a score to each error and writes it to the [Check Database](#) specified by the **-database** argument. The score is calculated as the area of the model-based violation.

If used with a PDK, this function calls a MinSpaceCheck defined in the PDK and runs it for the specified layer, writing check results to the specified database.

Figure 7-17. How MinSpaceCheck Works



Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-subwindow *expr_number***

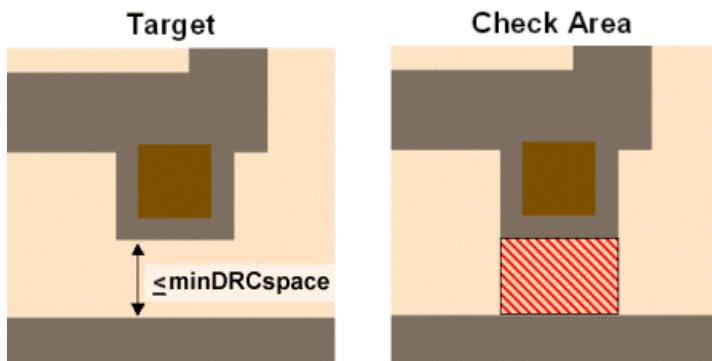
Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, -subwindow 1_N causes the check to only run on the north shift).

- **-minDRCspace *target_dist***

Required keyword and argument used to constrain the checks to those areas where target layer geometries are close enough to one another to be at risk. The function ignores areas where the distance between target features is greater than ***target_dist*** (at least 1.5 to 2 times greater).

The value ***target_dist*** is greater than the minimum spacing constraint for layer but should not be larger than the minimum spacing plus two times the minimum width.



- **-minLFDspace *space***

Required keyword and argument defining the PV-band spacing violations. PV-band geometries whose external PV-band edges are \leq ***space*** microns away from other external PV-band edges are flagged as an error.

It measures from one outer edge to the outer edge on the adjacent PV-band and flags all PV-bands closer than this as errors.

The -minLFDspace and -LFDspaceRange keywords and arguments are mutually exclusive.

- **-LFDspaceRange *from_space to_space***

Required keyword and argument set defining the PV-band spacing violations. PV-band geometries whose external PV-band edges are $>$ ***from_space*** and \leq ***to_space*** microns away from other external PV-band edges are flagged as an error.

It measures from one outer edge to the outer edge on the adjacent PV-band and flags all PV-bands closer than this as errors.

The -LFDspaceRange and -minLFDspace keywords and arguments are mutually exclusive.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer** *layer_name*

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on this layer.

In the case of a Calibre LFD region, if a marker layer is specified, then the marker layer is combined with the Calibre LFD region. If a marker layer is not specified, the Calibre LFD region is the default marker layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-referenceLayer** “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the input layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-anchorLayer** “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the input drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to **-referenceLayer**.
- “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer**.
- “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-layerContourCondition** *contour_condition*

Optional keyword and argument to define a **-layer** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- **min** — Specifies the command works on the inner PV-band contour.
- **max** — Specifies the command works on the outer PV-band contour.
- **integer** — This integer is the order of the experiment in the subwindow. In the LFD::[PVband](#) command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- **process condition list** — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{**optical1 dose1 size1** [resist1 etch1] [**optical2 dose2 size2** [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::[PVband](#) or LFD::[RegisterContour](#) commands for the layer.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-checkName** *cName*

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-priority** *cPriority*

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 1, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-indexFilter** *value*

Optional keyword and value used to filter out inconsequential errors. Only errors having an index greater than or equal to this value are written to the [Check Database](#).

The index is defined as:

$$\frac{area(I_{abs})}{area(I_{orig})}$$

where:

I_{orig} = the original geometry within the area of interest

I_{abs} = the absolute PV-band within the area of interest

For this check, the area of interest is defined relative to the minimum space.

- `-minMarkerArea area_size`

Optional keyword and argument defining the minimum size for error markers sent to the [Check Database](#). Any error for which the error marker is smaller than this size is ignored. *area_size* must be specified in square user units (um^2). By default, *area_size* is 0.

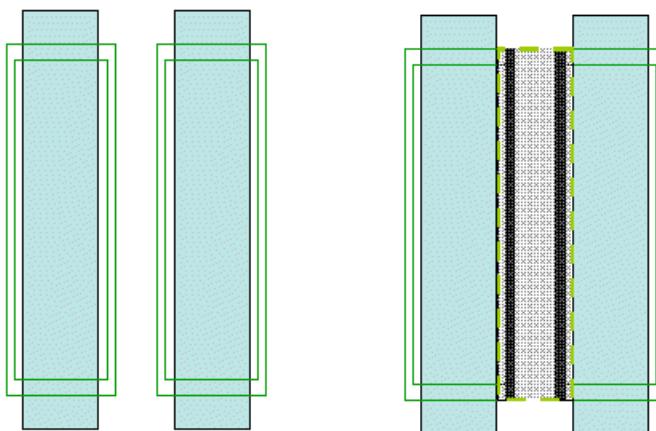
- `-minMarkerWidth width_size`

Optional keyword and argument defining the minimum width for error markers sent to the scores database. Any error for which the error marker is smaller than this width is ignored. *width_size* must be specified in user units (um). By default, *width_size* is 0. This option allows you to remove thin width error markers from Calibre LFD check results, as thin, long errors can negatively affect Calibre LFD runtime.

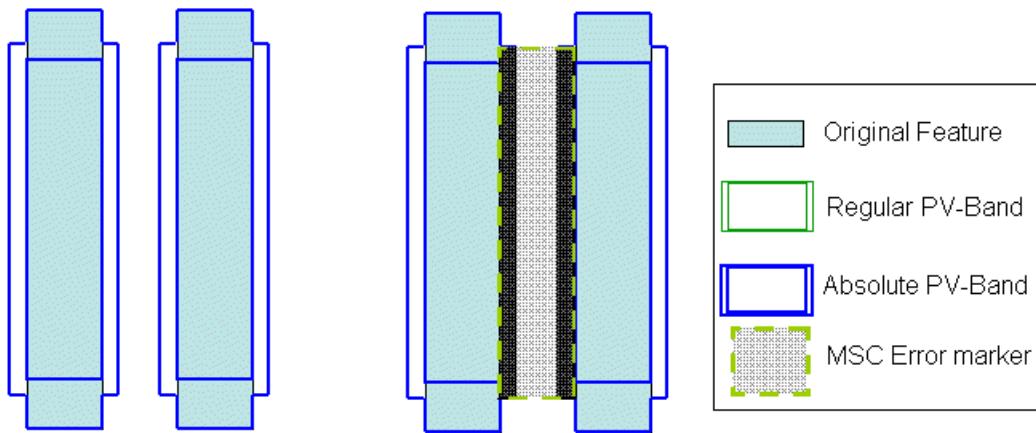
- `-bandType {absolute | regular}`

Optional keyword and argument defining the type of PV-band to use when checking for minimum space violations. By default this is the absolute PV-band.

MSC: with –bandType regular

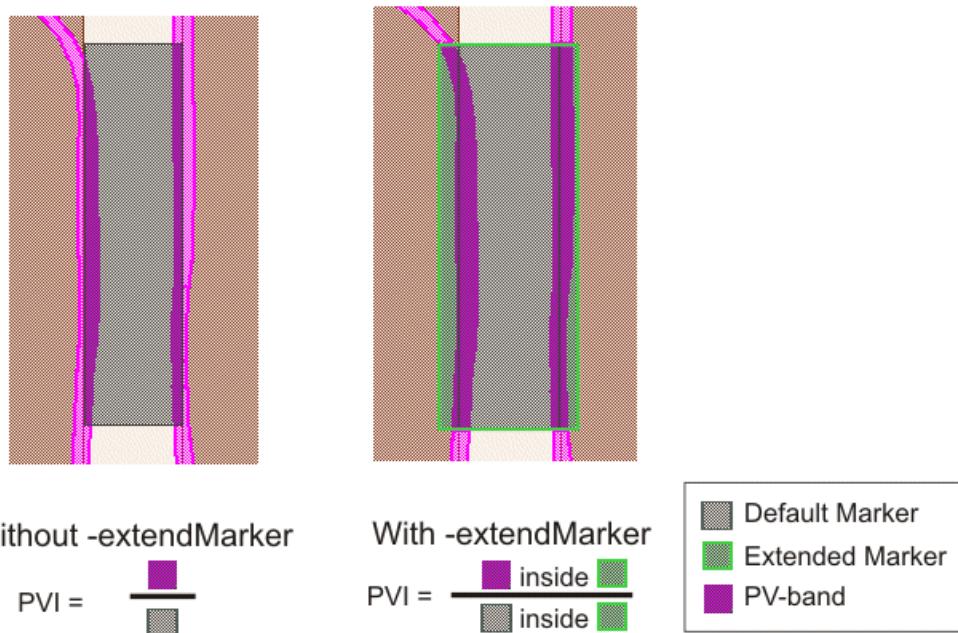


MSC: with –bandType absolute (default)



- `-extendMarker extend_dis t`

Optional keyword and argument defining an extension, specified in user units, to be applied to error markers in the directions of feature edges. By default the `extend_dist` is 0. Use this keyword when you need to ensure that PV-bands outside the default marker area are factored into PVI calculations.



- **-minSpaceEnd space_end_dist**
Optional keyword and argument providing the definition of an inverse line end error distance (also called space line-end). Any error this close to an inverse line end, or closer, is considered an inverse line-end error, and its error marker are removed from the final error group. The default value for *space_end_dist* is **-minDRCspace**.
- **-ext_side ext_side_value**
Optional keyword and argument defining how drawn layer edges are selected to satisfy the minDRCspace condition, expressed as a real number. This is useful in cases where the tool could miss errors due to non-overlapping edges. The default value for *ext_side_value* is **-minLFDspace** divided by two.
- **-ratio yes**
Optional keyword and argument set to “yes” to work in relative mode. If not set, the check works in absolute mode.
- **-property {property_list}**
Optional keyword set instructing the check to store the measured properties in the RDB. The *property_list* must be a Tcl list containing Space, SpaceRatio, MaxRunLength, or any combination of these properties. For each error generated, four properties are calculated. PVI is the ratio of the PV-band area associated to the error marker, and the area of the target layer. This property is the default, and it cannot be turned off. Space is the minimum distance between two exterior PV-band edges. SpaceRatio is the ratio between the Space measurement and the space between target layer features. MaxRunLength is the maximum target edge run length.

- **-comment** *comment_text*

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. If not specified, the comment for a MSC violation is “Minimum Space Violation: Improve symmetry. If feature is dense, increase spacing. Reduce surrounding feature width.”

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-classify** *handle*

Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.

- **-appendMarker** *extra_markers_layer*

Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).

- **-contourHandle** *contour_handle*

Optional keyword and argument used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-database** *db_name*

Required keyword and argument defining the RDB to which spacing values for violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut** *return_layer_name*

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName** *check_template*

Required keyword and argument specifying the name of the check template defining how this check is performed.

- *additional_options*

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the

non-PDK MinSpaceCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

Verify that all spaces in the polysilicon layer that are smaller or equal than 100nm are at least 60nm apart across all the conditions specified by subwindow 1:

```
LFD::MinSpaceCheck -layer poly -subwindow 1 -minDRCspace 0.10 \
                     -minLFDspace 0.06 -database minSpaceCheck.rdb
```

MinWidthCheck

Performs width checks for the specified layer.

Usage

Syntax 1

MinWidthCheck

```
-layer input_layer_name
-subwindow expr_number
-minDRCwidth | target_width
{ -minLFDwidth width | -LFDwidthRange from_width to_width }
[-security {no | yes}]
[-markerLayer layer_name]
[-referenceLayer "%drawn" | "%retarget" | layer_name]
[-anchorLayer "%reference" | "%drawn" | "%retarget" | layer_name]
[-layerContourCondition contour_condition]
[-checkName cName]
[-priority cPriority]
[-indexFilter value]
[-minMarkerArea area_size]
[-minMarkerWidth width_size]
[-bandType {absolute | regular}]
[-extendMarker extend_dist]
[-minLineEnd line_end_dist]
[-ratio yes]
[-property {property_list}]
[-comment comment_text]
[-concaveOffset offset_size]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contourHandle contour_handle]
{-database db_name | -layerOut return_layer_name
 / -database db_name -layerOut return_layer_name}
```

Syntax 2

MinWidthCheck

```
-layer input_layer_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

Description

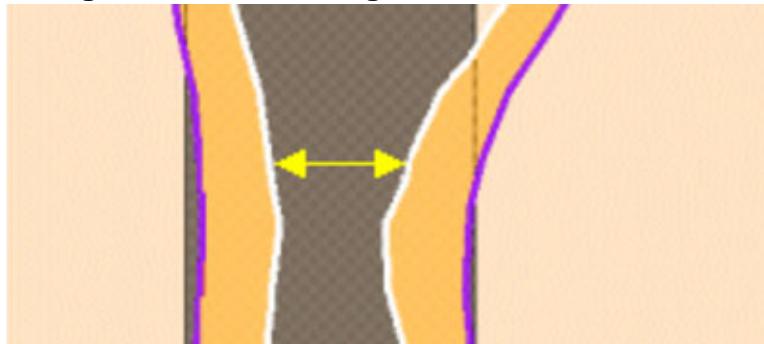
Performs a width check on PV-band geometries for target layer geometries no wider than **-minDRCwidth**. If -markerLayer is specified, the check is limited to PV-band data that lies

inside geometries on -markerLayer. This check is used to identify problems such as resist pinching in clear field mask as well as resist bridging in dark field masks.

MinWidthCheck (MWC) checks the width from the inside edge of a PV-band to the facing inside edge. This check catches situations in which the two inside edges overlap (*width* < 0).

This check should not be used for features that print as circular shapes, such as contacts and vias. Circular features are analyzed using [MinAreaOverlapCheck](#).

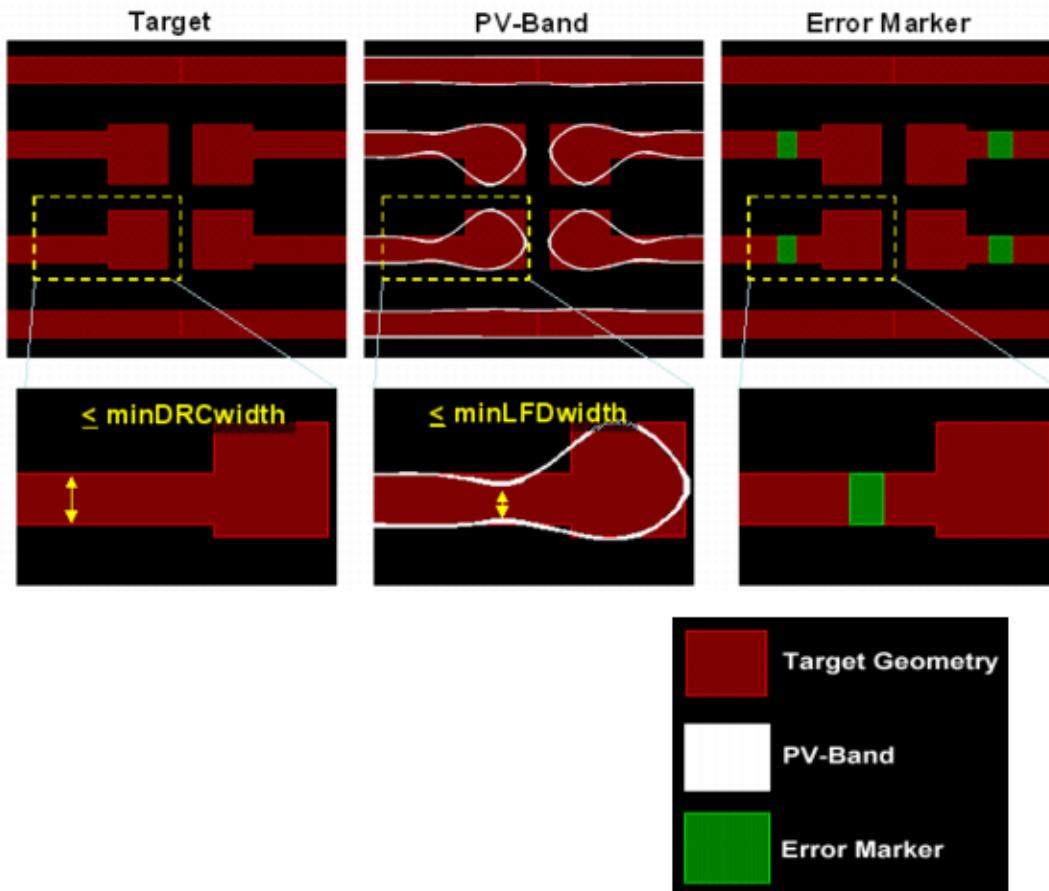
Figure 7-18. Checking PV-Band Width



This function writes all errors discovered by the check to the Calibre nmDRC RDB. It also associates a score to each error and writes the score to the [Check Database](#) specified by the **-database** argument. The score is calculated as the area of the model-based violation.

If used with a PDK, this function calls a MinWidthCheck defined in the PDK and runs it for the specified layer, writing check results to the specified database.

Figure 7-19. How MinWidthCheck Works



Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-subwindow *expr_number***

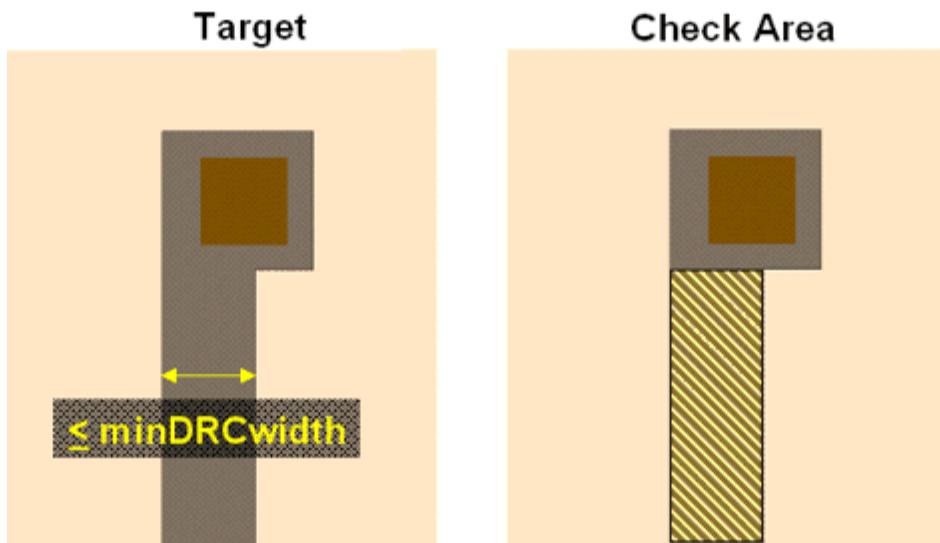
Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, **-subwindow 1_N** causes the check to only run on the north shift).

- **-minDRCwidth *target_width***

Required keyword and argument used to constrain the checks to those areas where target layer geometries are narrow enough to one another to be at risk. The function ignores PV-bands for target features that are wider than *target_width*.

The value *target_width* is significantly greater than the minimum width constraint for layer (at least 1.5 to 2 times greater).



- **-minLFDwidth *width***

Required keyword and argument defining the minimum distance between the internal PV-band edges. Layout configurations with internal PV-band edges closer or equal to *width* microns are flagged as an error.

The -minLFDwidth and -LFDwidthRange keywords and arguments are mutually exclusive.

- **-LFDwidthRange *from_width to_width***

Required keyword and argument set defining the minimum distance between the internal PV-band edges. Layout configurations with internal PV-band edges >*from_width* and <= *to_width* microns are flagged as an error.

The -LFDwidthRange and -minLFDwidth keywords and arguments are mutually exclusive.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on this layer.

In the case of a Calibre LFD region, if a marker layer is specified, then the marker layer is combined with the Calibre LFD region. If a marker layer is not specified, the Calibre LFD region is the default marker layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-referenceLayer** “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the input layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-anchorLayer** “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the input drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to **-referenceLayer**.
- “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer**.
- “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-layerContourCondition** *contour_condition*

Optional keyword and argument to define a **-layer** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.

- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- -checkName *cName*

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in [Table 7-14](#). Use this keyword to avoid name collisions when performing multiple checks of this type.

- -priority *cPriority*

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 1, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- -indexFilter *value*

Optional keyword and value used to filter out inconsequential errors. Only errors having an index greater than or equal to this value are written to the [Check Database](#).

The index is defined as:

$$\frac{\text{area}(I_{abs})}{\text{area}(I_{orig})}$$

where:

I_{abs} = the absolute PV-band within the area of interest

I_{orig} = the original geometry within the area of interest

For this check, the area of interest is defined relative to critical dimension (CD).

- **-minMarkerArea *area_size***

Optional keyword and argument defining the minimum size for error markers sent to the [Check Database](#). Any error for which the error marker is smaller than this size is ignored. *area_size* must be specified in square user units (μm^2). By default, *area_size* is 0.

- **-minMarkerWidth *width_size***

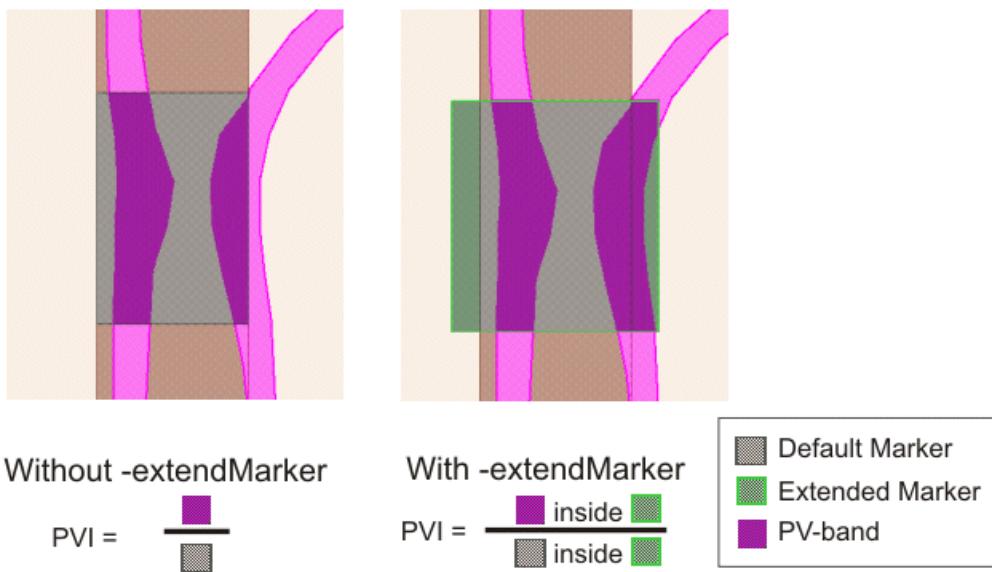
Optional keyword and argument defining the minimum width for error markers sent to the scores database. Any error for which the error marker is smaller than this width is ignored. *width_size* must be specified in user units (μm). By default, *width_size* is 0. This option allows you to remove thin width error markers from Calibre LFD check results, as thin, long errors can negatively affect Calibre LFD runtime.

- **-bandType {absolute | regular}**

Optional keyword and argument defining the type of PV-band to use when checking for minimum width violations. By default this is the absolute PV-band.

- **-extendMarker *extend_dist***

Optional keyword and argument defining an extension, specified in user units, to be applied to error markers in the directions of feature edges. By default the *extend_dist* is 0. Use this keyword when you need to ensure that PV-bands outside a feature are factored into PVI calculations.



- `-minLineEnd line_end_dist`

Optional keyword and argument providing the definition of a line end error distance. Any error this close to a line end, or closer, is considered a line-end error, and its error marker is removed from the final error group. The default value for *line_end_dist* is `-minDRCwidth`.

- `-ratio yes`

Optional keyword and argument used set to “yes” to work in relative mode. If not set, the check works in absolute mode.

- `-property {property_list}`

Optional keyword set instructing the check to store the measured properties in the RDB. The *property_list* must be a Tcl list containing MinCD, MinCDRatio, MaxRunLength, or any combination of these properties. For each error generated, four properties are calculated. PVI is the ratio of the PV-band area associated to the error marker, and the area of the target layer. This property is the default, and it cannot be turned off. MinCD is the minimum distance between two interior PV-band edges. MinCDRatio is the ratio between the MinCD measurement and the width of the target layer. MaxRunLength is the maximum target edge run length.

- `-comment comment_text`

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. If not specified, the comment for a MWC violation is “Minimum Width Violation: Improve symmetry. If feature isolated, make dense. If feature dense, separate.”

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- `-concaveOffset offset_size`

Optional keyword and argument which causes centerline shortening near concave corners and centerline extension near convex corners. *offset_size* must be specified in user units (um), and by default *offset_size* is 0.

- `-classify handle`

Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.

- `-appendMarker extra_markers_layer`

Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).

- **-contourHandle *contour_handle***

Optional keyword and argument used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-database *db_name***

Required keyword and argument defining the RDB to which spacing values for violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how this check is performed.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK MinWidthCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

Verify that all metal1 widths smaller or equal than 150nm are at least 50nm across all the conditions specified by subwindow:

```
LFD:::MinWidthCheck -layer metal1 -subwindow 1 -minDRCwidth 0.150 \
                     -minLFDwidth 0.05 -database minWidthCheck.rdb
```

NonPrintingCheck

Finds areas where PV-band and target layer do not interact.

Usage

Syntax 1

NonPrintingCheck

```
-layer input_layer_name
-subwindow expr_number
[-checkName cName]
[-security {no | yes}]
[-markerLayer layer_name]
[-markerSize size]
[-referenceLayer "%drawn" | "%retarget" | layer_name]
[-layerContourCondition contour_condition]
[-maxExtent extent_size]
[-priority cPriority]
[-comment comment_text]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contourHandle contour_handle]
{-database db_name | -layerOut return_layer_name
 | -database db_name -layerOut return_layer_name}
```

Syntax 2

NonPrintingCheck

```
-layer input_layer_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

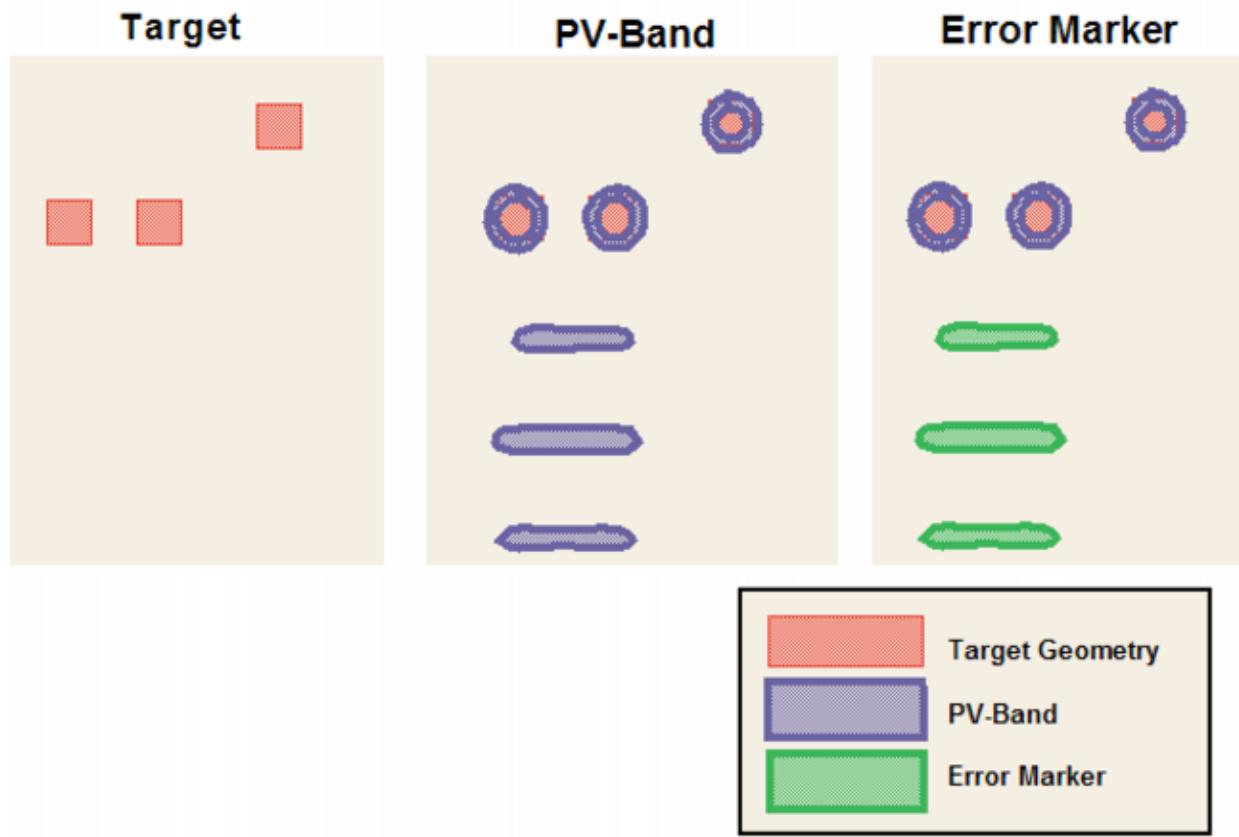
Description

Performs a comparison of the target layer and the PV-band and returns areas where the two do not interact.

This check is to detect side-lobes or printing assist features. There is no PVI associated with this check.

If used with a PDK, this function calls a NonPrintingCheck defined in the PDK and runs it for the specified layer, writing check results to the specified database.

Figure 7-20. NonPrintingCheck



Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Setting ***expr_number*** to a value of “***expr_number_shift***” causes the check to operate on a certain shift for a double-patterned PV-band with overlay (for example, **-subwindow 1_N** causes the check to only run on the north shift).

- **-checkName *cName***

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name as defined in

Table 7-14. Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-priority *cPriority***

Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 3, as defined in [Table 7-15](#). *cPriority* must be an integer value.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on this layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-markerSize *size***

Optional keyword and argument used to define the size of marker generated to display errors. The default value is 2 dbu.

- **-referenceLayer “%drawn” | “%retarget” | *layer_name***

Optional keyword and argument to have the check measurements calculated on a different layer than the input layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with **-layer**.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — The check measurements are done with respect to the specified layer name. This argument can be used when no PDK is used, and you know the name of the retarget layer.

- **-layerContourCondition *contour_condition***

Optional keyword and argument to define a **-layer** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.

- *integer* — This integer is the order of the experiment in the subwindow. In the LFD::PVband command, by default subwindows are assumed to be constructed as extensions to previously-created subwindows, so the order of experiments is counted from the first subwindow. If the LFD::PVband -independentWindows option is set, the order of experiments is only counted in the defined subwindow.
- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{*optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]*}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

Note

 This switch is useful when variations between layers are well-controlled, and you are interested in considering process variations of one layer over the nominal behavior of another.

- **-maxExtent *extent_size***

Optional keyword and argument used to distinguish between edge placement errors (EPE) due to the Calibre OPCverify tool's tiling methodology and reported errors. The default value of *extent_size* argument is 0.15 microns.

- **-comment *comment_text***

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. Each comment results in one line of check text. There is no default value for this argument.

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-classify *handle***

Optional keyword and argument used to define a handle to point to an LFD::ClassifyConfig object.

- **-appendMarker *extra_markers_layer***

Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::StructureOptimizer.

- **-contourHandle *contour_handle***

Optional keyword and argument used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-database *db_name***

Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how this check is performed.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK NonPrintingCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

```
LFD:::NonPrintingCheck \
-layer poly \
-subwindow 1 \
-database npc.rdb
```

PrintableCheck

Categorizes error markers defined in prior checks into either layout-problem or OPC-problem categories.

Usage

Syntax 1

PrintableCheck

```
-layer input_layer_name
-subwindow expr_number
-printableModel {model_path | model_handle | inline_model}
[-modelDir models_directory]
[-checkList error_layer_list]
[-property {Intensity| IntensErr}]
[-halo halo_value]
[-security {no | yes}]
[-comment comment_text]
{-database db_name | -layerOut return_layer_name
 / -database db_name -layerOut return_layer_name}
```

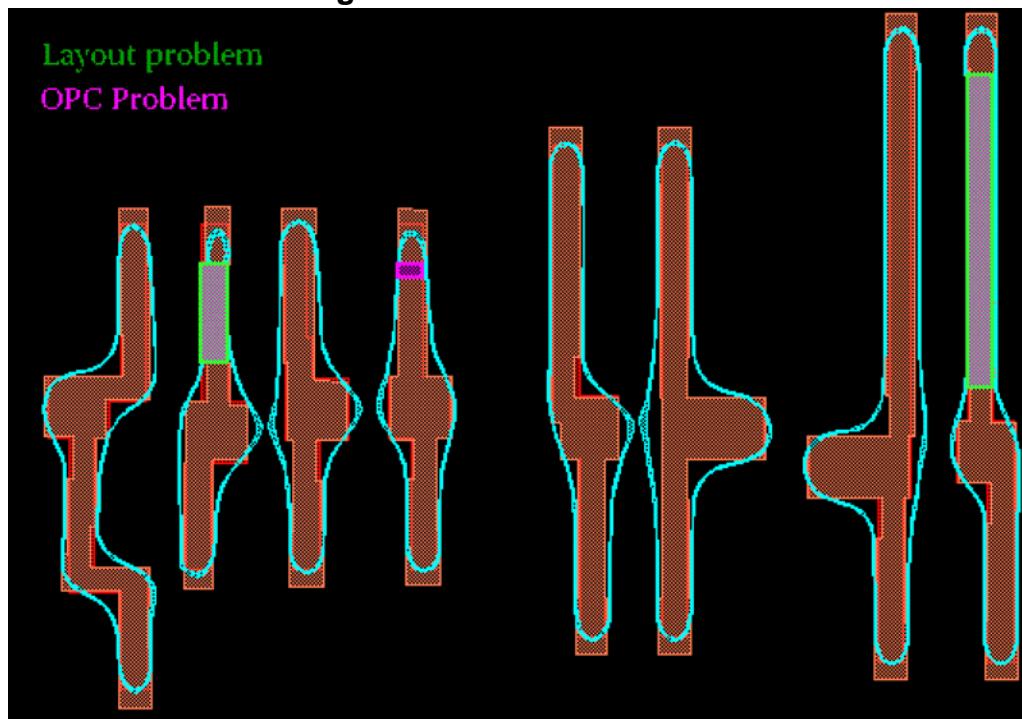
Syntax 2

PrintableCheck

```
-layer input_layer_name
-pdkCheckName check_template
[additional_options]
```

Description

Categorizes error markers defined in previous Calibre LFD checks into two main categories: *layout-problem* or *OPC-problem*. Error markers output by this check indicate that the error is due to the drawn layout. Error markers that pass through this printability checking indicate that this shape is printable if the foundry modifies the RET recipes.

Figure 7-21. PrintableCheck

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the target layer you are checking.

- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies.

- **-printableModel {model_path | model_handle | inline_model}**

Required keyword followed by “**model_path**”, “**inline_model**” or “**model_handle**”, defining the printable model. If the model is defined as a **model_path**, the model must reside within the model directory. If the model is defined as a **model_handle**, this handle must be defined in the PDK. For PDK usage, this option is defined as a list containing the printable model name and model definition.

- **-modelDir *models_directory***

Optional keyword and argument specifying the path of the printable models. Used only when **-printableModel** is set to **model_path**.

- **-checkList *error_layer_list***

Optional keyword and argument defining the layers needed to be checked. *error_layer_list* is a list containing the error layer names. If not defined, the error markers of all previous checks are used.

- **-property {Intensity | IntensErr}**

Optional keyword set instructing the check to store the measured properties in the RDB. The property must be a Tcl list containing “Intensity” or “IntensErr or” both.

- **-halo *halo_value***

Optional keyword and argument used to set the simulation halo. This value is used to size up the error layer sent to the printable check. The default value is 0.5 um.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-comment *comment_text***

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type.

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-appendMarker *extra_markers_layer***

Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an [LFD::StructureOptimizer](#).

- **-database *db_name***

Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK LFD::PrintableCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

```
LFD::PrintableCheck \
-layer poly \
-modelDir ./models \
-printableModel pm \
-subwindow 1 \
-database $lfdErrors.rdb
```

ViaWidthCheck

Performs check on width of a printed via.

Usage

Syntax 1

ViaWidthCheck

```
-layer input_layer_name
-subwindow expr_number
-minViaWidth min_width_value
[-layerContourCondition contour_condition]
[-security yes]
[-markerLayer layer_name]
[-referenceLayer "%drawn" | "%retarget" | layer_name]
[-anchorLayer "%reference" | "%drawn" | "%retarget" | layer_name]
[-checkName cName]
[-comment comment_text]
[-suppressPVI yes]
[-priority cPriority]
[-indexFilter value]
[-property Min]
[-classify handle]
[-appendMarker extra_markers_layer]
[-contourHandle contour_handle]
{-database db_name | -layerOut layer_name
 | -database db_name -layerOut layer_name}
```

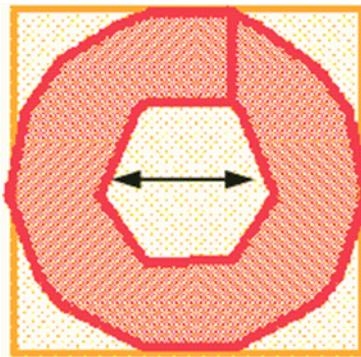
Syntax 2

ViaWidthCheck

```
-layer input_layer_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

Description

Performs a check on the width of square polygons, such as vias and contacts. This check measures the diameter of the simulated contour. The measurement is made in four directions: horizontal, vertical, 45-degree slope, and negative 45-degree slope passing by the simulated contour center. If any of the four diameter lengths is smaller than *min_width_value*, the contact is flagged as an error.

Figure 7-22. ViaWidthCheck

This command writes all errors discovered by the check to the Calibre nmDRC results database (RDB). It also associates a score to each error and writes the score to the Check Database specified by the **-database** argument.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated. It is a contact or via layer.

- **-subwindow *expr_number***

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the LFD::PVband command used to generate the PV-band data being checked. Thus, *expr_number* refers to an index to a list of experiments.

Setting *expr_number* to a value of “*subwindow_shift*” causes the check to operate on a certain shift for an overlaid, double-patterned PV-band.

- **-minViaWidth *min_width_value***

Required keyword and argument defining the minimum width for the contact contour.

- **-layerContourCondition *contour_condition***

Optional keyword and argument to define a **-layer** check on any defined contour condition in the PV-band, as opposed to only on the inner or outer PV-band contours.

You can provide one of the following options as an input to this argument:

- min — Default. Specifies the command works on the inner PV-band contour.
- max — Specifies the command works on the outer PV-band contour.
- *integer* — This integer is the order of the experiment in the subwindow, or the number of contours defined by LFD::RegisterContour. In the LFD::PVband command, by default subwindows are assumed to have been constructed as extensions to previously-created subwindows, so the order of experiments is counted

from the first subwindow. If the option -independentWindows in LFD::PVband is set, the order of experiments is counted in the defined subwindow.

- *process condition list* — This must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{*optical1 dose1 size1 [resist1 etch1]* [*optical2 dose2 size2 [resist2 etch2]*]}

These values must define a process condition (dose and focus settings) that is one of the conditions evaluated by the LFD::PVband or LFD::RegisterContour commands for the layer.

- -security yes

Optional argument defining security privileges. If defined, the setup file is encrypted in the transcript.

- -markerLayer *layer_name*

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on this layer.

A -layerOut layer or a derivation of a -layerOut layer should not be used as the input to -markerLayer, or a circular layer definition results.

- -referenceLayer “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the check measurements calculated on a different layer than the input layer to the checks for which the PV-bands have been generated.

You can provide one of the following options as an input to this argument:

- “%drawn” — Default. The check measurements are calculated with respect to the drawn layer input to the check with -layer.
- “%retarget” — The check measurements are calculated with respect to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
- *layer_name* — You can define any layer name to have the check measurements done with respect to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- -anchorLayer “%reference” | “%drawn” | “%retarget” | *layer_name*

Optional keyword and argument to have the output error markers from the checks anchored on a different layer than the input drawn layer of the checks, and different from the reference layer input to the check.

You can provide one of the following options as an input to this argument:

- “%reference” — Default. The output error markers are anchored to the layer input to -referenceLayer.

- “%drawn” — The output error markers of the check are anchored to the drawn layer input to the check with **-layer**.
 - “%retarget” — The output error markers of the check are anchored to the retarget layer of the input drawn layer of the check. The check stores the retarget layer name in the PDK.
 - *layer_name* — You can define any layer name to have the output error markers anchored to it. This argument can be used when no PDK is used, and you know the name of the retarget layer.
- **-checkName** *cName*
Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name. Use this keyword to avoid name collisions when performing multiple checks of this type.
 - **-comment** *comment_text*
Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. If not specified, the comment for a VWC violation is “Via Width Violation”.
The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```
 - **-suppressPVI** *yes*
Optional keyword and argument specifying whether or not to suppress calculation of PVI. When “-suppressPVI yes” is specified, the check suppresses calculation of PVI. By default, PVI calculation is not suppressed.
 - **-priority** *cPriority*
Optional keyword and argument specifying a priority for this check. If not specified, the check in the RDB is assigned a system-generated priority based on the -subwindow value and default ranking of 3. *cPriority* must be an integer value.
 - **-indexFilter** *value*
Optional keyword and value used to filter out inconsequential errors. Only errors having an index greater than or equal to this value are written to the [Check Database](#).

The index is defined as:

$$\frac{\text{area}(I_{abs})}{\text{area}(I_{orig})}$$

where:

I_{abs} = the absolute PV-band within the area of interest

I_{orig} = the original geometry within the area of interest

- -property Min

Optional keyword and argument specifying whether or not the check should save error measurement data to the RDB database. When “-property Min” is specified, the check reports the minimum change in width measured for the feature. By default, property values are not saved.

- -classify *handle*

Optional keyword and argument used to define a handle to point to an LFD::[ClassifyConfig](#) object.

- -appendMarker *extra_markers_layer*

Optional keyword and argument used to add the polygons in the *extra_markers_layer* layer to the output of the check. The output retains the properties on the *extra_markers_layer* layer supported by the check. This option is not allowed for checks that are tied to an LFD::[StructureOptimizer](#).

- -contourHandle *contour_handle*

Optional keyword and argument used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- -database *db_name*

Required keyword and argument defining the RDB to which violations identified by the check are written. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

- -layerOut *return_layer_name*

Required keyword and argument defining the name of a derived layer to which the violations identified by the check are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- -pdkCheckName *check_template*

Required keyword and argument specifying the name of the check template defining how this check is performed.

- *additional_options*

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK LFD::ViaWidthCheck command. Any additional options specified overrides the options defined within the PDK.

Examples

```
LFD::ViaWidthCheck \
-layer contact \
-subwindow 1 \
-minViaWidth 0.04 \
-database viaWidthCheck.rdb
```

Calibre LFD General-Purpose Commands

This reference section contains an alphabetized list of the supported general-purpose Calibre LFD commands.

Note

 For additional commands, see the [Calibre LFD Checks](#) and [PDK Commands](#) sections.

Table 7-6. Calibre LFD General Purpose Commands

Function	Definition
Anchor	Generates anchor points that are candidates for litho hotspots and problematic patterns.
AnchorRule	Overrides LFD::Anchor arguments and defines layers and spacing for multi-pattern and multilayer anchor points.
Band	Creates PV-bands with two input contours.
Begin	Command used to mark the start of the Calibre LFD processing block.
CaptureContour	Stores a previously generated contour as a derived layer.
ClassifyConfig	Configures common user-defined classification options.
Contour	Creates contours which can be used to create other contours, PV-bands, or to perform checks.
CSG	Generates a .csg file that can be used as input to Calibre nmLVS.
CSI	Generates a new layer that is a simplified version of the input contour layer.
Drawn2Contour	Generate empirically calibrated PV-bands using Square Directional Kernels (SDK) when access to the exact OPC recipe and models are not available.
End	Closing command used to signal the end of the Calibre LFD block.
FrameCellOptimizer	Pre-processing that reduces the area to be analyzed by removing those areas that do not require processing by Calibre LFD because they have already been checked or because they are IP that the designer cannot modify.
GenerateHints	Runs Calibre LFD with MBH and generates a hints file for fixing lithographic hotspots.
GetMacroLayers	Extracts layers from the process-correction macro.
GetOPCInputLayers	Returns intermediate OPC layers.

Table 7-6. Calibre LFD General Purpose Commands (cont.)

Function	Definition
GetOPCLayers	Returns OPC layer and visible layer.
GetRetargetLayers	Returns retarget layers.
GetVisibleLayers	Returns visible layers.
IncrementalSelect	Performs intelligent clipping of one or more input layers based on your marker layers.
LayoutOptimizer	Defines problem areas based on geometric rules.
LFDregion	Defines regions within the layout, which can then be used to limit checking to those areas.
LoadPDK	Loads the specified PDK and makes it available to Calibre LFD command calls.
Macro	Used to define macros.
MLDataGen	Samples and translates features for input to a machine learning training model in the preprocessing and data generation phase of the Calibre LFD deep neural network (DNN) flow.
MLOptimizer	Performs hotspot prediction on a new design using a machine learning trained model in the prediction phase of the Calibre LFD deep neural network (DNN) flow.
OutputBands	Writes PV-band data to the specified database.
OverlayBand	Creates overlay PV-bands with two input contours.
PrintableBand	Simulate expected lithographic PV-bands in the absence of an RET recipe.
ProcessCorrection	Applies a process correction recipe to the specified design layer.
PVband	Generates PV-band data for the specified layer.
ReadECO	Provides layer name for GDS ECO layers.
ReadHIF	Reads a HIF file and maps all the polygons from the checks for a given layer onto a single output layer.
ReadiLPC	Reads an iLPC file and maps all the polygons and areas for a given layer onto a single output layer.
ReadRDB	Reads an RDB file and maps the polygons from the checks for a given layer onto an output layer.
RegisterBands	Makes PV-band data generated in a previous run available for analysis in the current run.
RegisterContour	Makes contour data generated in a previous run available for analysis in the current run.

Table 7-6. Calibre LFD General Purpose Commands (cont.)

Function	Definition
RegisterLayer	Registers a design layer with the Calibre LFD processing engine.
RemoveErrors	Removes error markers from the output layer of a Calibre LFD check.
SetSTOMode	Sets the Structure Optimizer (STO) mode.
SimConfig	Configures the simulations to be performed as part of Calibre LFD.
StructureOptimizer	Reduces layout data set being passed into Calibre LFD simulation, improving overall runtime performance.
StructureOutput	Takes hotspot and target layers and creates a pattern description file.
TopCellOptimizer	Aids checking geometries at top level of the design hierarchy.
VariabilityIndices	Generates variability indexes (scores) for the specified layer.
WriteHIF	Writes a Litho hotspot file that adheres to the Cadence Hotspot Interface Format (HIF).
WriteICC	Writes a Litho hotspot file that adheres to the Synopsys IC Compiler format (ICC).

Anchor

Generates anchor points that are candidates for litho hotspots and problematic patterns.

Usage

Anchor

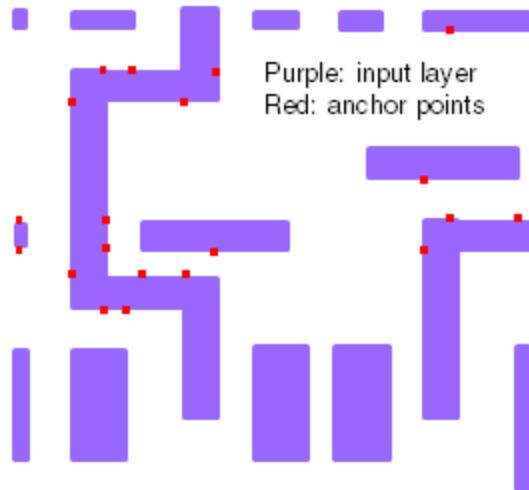
```
-layer {input_layer_name...input_layer_nameN}  
-width width_value  
{-space space_value /  
 {-length LE_length LE_value -space LE_space LE_value  
 -space_ LE_E space_E_value -space_ E_E space_E_value}}}  
-merge_distance merge_distance_value  
-layer_out layer_out_name  
[-parallel_lines_length parallel_lines_length_value -parallel_lines_space  
 parallel_lines_space_value]  
[-mergeTileLength merge_tile_length_value -maxAnchorPoints  
 max_anchor_points_value]  
[-classification_halo classification_halo_value]
```

Description

The Calibre LFD::Anchor command identifies areas of the layout that are potential litho hotspots. Anchor points are generated at critical sites near corners and parallel line structures that meet certain criteria. These sites are marked with anchor points on the output layer. If you specify LFD::[AnchorRule](#) before LFD::Anchor in the Calibre LFD rule file, spacing and width arguments are overridden by corresponding arguments specified in LFD::AnchorRule.

The anchor points are used as input to the capture function Calibre LFD::[StructureOutput](#) to generate a pattern description file. The pattern description file can be passed to the function Calibre LFD::[StructureOptimizer](#), where simulation and pattern matching determine if the anchor points predict a litho hotspot that results in a problematic pattern.

Figure 7-23. Anchors Generated

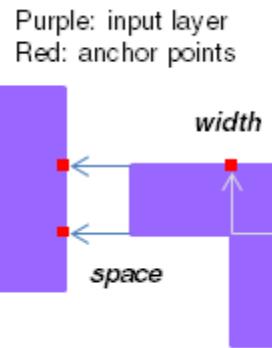


When specifying the criteria for anchor points, both width and space values are required. For example, if there is a concave corner, then any polygon shape within this concave corner with a width less than the specified **-width** value generates an anchor point. Likewise, for a convex corner, any polygon shape within a distance less than the specified **-space** value generates an anchor point.

Note

 The LFD::Anchor supports only Manhattan (orthogonal) corners.

Figure 7-24. Anchors Generated Using width and space Values

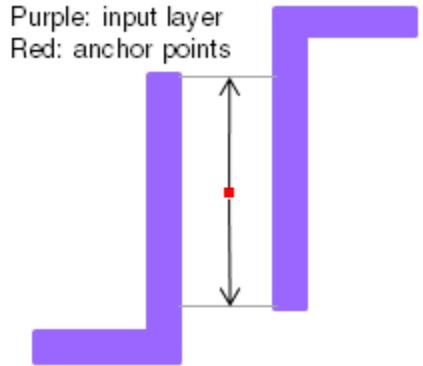


There are two conditions that determine if an area of the layout receives an anchor point.

- If the nearby polygon structures are close enough for a corner to have an influence on them, where “nearby” is related to the minimum width and spacing of the technology.
- If the location has parallel line shapes that meet both of the following constraints:
 - The common length of the parallel line shapes is greater than or equal to the *parallel_lines_length_value* given in the command.

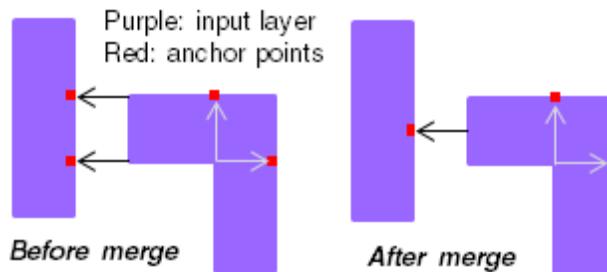
- The separation between the parallel line shapes is less than or equal to the *parallel_lines_space_value* given in the command.

Figure 7-25. Anchors Generated from Parallel Lines



In the case of very dense designs, it is desirable to reduce the number of anchor points, thus reducing the number of close anchor points that are later translated to similar patterns. The required keyword **-merge_distance** is provided to reduce the number of collinear anchor points. Anchor points that are in intervals less than or equal to the specified *merge_distance_value* are grouped together in a single anchor point. The size of the pattern is a factor in identifying the merge distance between anchor points. If the merge distance is too big, two distant anchor points are merged into one anchor point, possibly missing a candidate for a problematic pattern. The parameter *merge_distance_value* applies when merging collinear anchor points.

Figure 7-26. Anchors Before and After Merge



For anchor points that are non-collinear, an optional merging algorithm can be used to reduce the number of non-collinear anchor points by specifying the optional keywords **-mergeTileLength** and **-maxAnchorPoints**. These two keywords must be specified together. This merging mechanism partitions the design into $L \times L$ merging tiles, where L is the length of the tile edge defined by **-mergeTileLength**.

If the **-classification_halo** optional keyword is also specified, then the effect of the keywords **-mergeTileLength** and **-maxAnchorPoints** is applied to the anchor points before the **-classification_halo** keyword.

Arguments

- **{-layer *input_layer_name* ...*input_layer_nameN*}**

Required keyword and argument specifying the input layer name. A list argument can be used to specify multiple input layer names.

- **-width *width_value***

Required keyword and argument that defines the width of the input layer shape that is critical. This value gives the constraint for anchors near to concave corners. Specify this value in user units. Widths at or below this value are considered for anchor point sites.

- **-space *space_value***

Required keyword and argument that defines the space between shapes of the input layer that is critical. This value gives the constraint for anchors near to convex corners. Spaces at or below this value are considered for anchor point sites. Specify this value in user units.

- **-length_LE *length_LE_value* -space_LE_LE *space_LE_LE_value* -space_LE_E *space_LE_E_value* -space_E_E *space_E_E_value***

Required keyword set and arguments specifying the critical length and space constraints for generating anchor points. This keyword set is used in place of the **-space** argument.

-length_LE *length_LE_value*

Required keyword and argument specifying the maximum length of an edge that can be defined as a line-end. Any length greater than this value is defined as an edge. Specify this value in user units.

Figure 7-27. LFD::Anchor Length LE



-space_LE_LE *space_LE_LE_value*

Required keyword and argument specifying the maximum space between two line-ends that can result in anchor points. Parallel line-ends with separation less than or equal to **-space_LE_LE** result in an anchor point between the two parallel line-ends. Specify this value in user units.

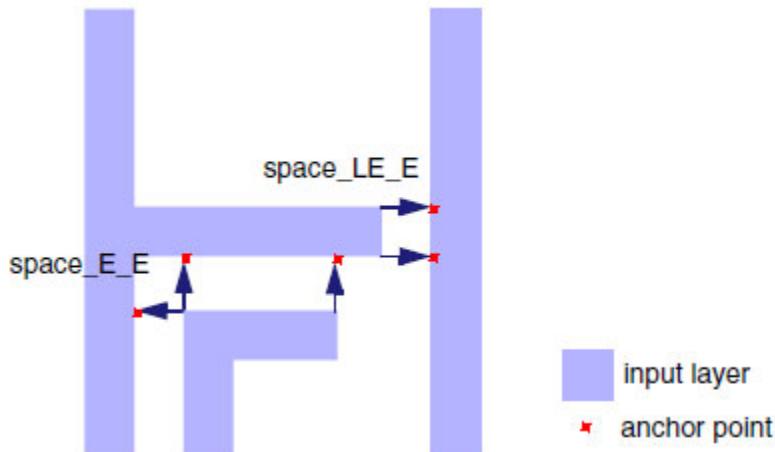
Figure 7-28. LFD::Anchor Space LE_LE



-space_LE_E *space_LE_E_value*

Required keyword and argument specifying the maximum space between a line-end shape and an edge that can result in anchor points. Only line-end corners with edge distances that are less than or equal to this value are considered for anchor points. Specify this value in user units.

Figure 7-29. LFD::Anchor Space LE_E and E_E



-space_E_E space_E_E_value

Required keyword and argument specifying the maximum space between edges that can result in anchor points. Only edges whose distances are less than or equal to this space are considered for anchor points. Specify this value in user units.

- **-merge_distance merge_distance_value**

Required keyword and argument that specifies how collinear anchor points are merged. Anchor points separated by a distance less than or equal to *merge_distance_value* are merged into one anchor point. Specify this value in user units.

- **-layer_out layer_out_name**

Required keyword and argument that specifies the name of the output layer that contains the anchor points.

- **-parallel_lines_length parallel_lines_length_value**

Optional keyword and argument that specifies that an anchor point is created if the common length of the parallel line shapes is greater than or equal to *parallel_lines_length_value*. If specified, the -parallel_lines_space keyword and value must also be specified. Specify this value in user units.

- **-parallel_lines_space parallel_lines_space_value**

Optional keyword and argument that specifies that an anchor point is created if the space between the parallel line shapes is less than or equal to *parallel_lines_space_value*.

Resulting anchors are located at the midpoint between the parallel line shapes. If specified, the -parallel_lines_length keyword and value must also be specified. Specify this value in user units.

- **-mergeTileLength** *merge_tile_length_value*

Optional keyword and argument that specifies the length of the tile edge that is used for merging non-collinear anchor points. Use this keyword together with the **-maxAnchorPoints** keyword. Specify this value in user units.

- **-maxAnchorPoints** *max_anchor_points_value*

Optional keyword and argument that specifies the maximum number of anchor points within a merging tile. If the number of anchor points is greater than or equal to this value, then the anchor points within a merging tile are replaced by a single anchor point in the center of the tile. The **-maxAnchorPoints** and the **-mergeTileLength** keywords are used together. Express this value as a positive integer.

- **-classification_halo** *classification_halo_value*

Optional keyword and argument that reduces the number of anchor points by using the Calibre OPCverify tool classify command to check the anchor point and the surrounding halo region for similarities. Only the unique anchor points are displayed. Specify this value in user units.

Examples

Example 1

The following is an example of the LFD::Anchor command using anchor point merging options:

```
LFD::Anchor
  -layer poly \
  -width 0.001 \
  -space 0.001 \
  -merge_distance 0.001 \
  -parallel_lines_length 0.7 \
  -parallel_lines_space 0.7 \
  -classification_halo 0.005 \
  -maxAnchorPoints 40 \
  -mergeTileLength 30 \
  -layer_out anchors
```

Example 2

In the following example, the **-space** keyword is not specified in LFD::Anchor, so the spacing rules are defined by the keyword set **-length_LE**, **-space_LE_LE**, **-space_LE_E**, and **-space_E_E**:

```
LFD::Anchor
  -layer {M1_1 M1_2} \
  -width 0.05 \
  -length_LE 0.05 -space_LE_LE 0.02 -space_LE_E 0.01 -space_E_E 0.05 \
  -parallel_lines_length 0.7 \
  -parallel_lines_space 0.03 \
  -merge_distance 0.05 \
  -layer_out anchors "
```

Related Topics

[StructureOutput](#)

AnchorRule

Overrides LFD::Anchor arguments and defines layers and spacing for multi-pattern and multilayer anchor points.

Usage

AnchorRule

```
-layer1 layer1_name
[-layer2 layer2_name]
{-space space_value /
  {-lengthLE lengthLE_value -spaceLE spaceLE_value
   -spaceLE_E spaceLE_E_value -spaceE_E spaceE_E_value} }
-width1 width1_value
[-width2 width2_value]
[-parallel_lines_length parallel_lines_length_value
  -parallel_lines_space parallel_lines_space_value]
```

Description

You can use LFD::AnchorRule before LFD::Anchor to define the anchor point rules for multilayer constraints in the case of multi-pattern and via layers. The LFD::AnchorRule command is specified for an input layer or layer pair and must precede the LFD::Anchor command in the Calibre LFD rule file. The LFD::AnchorRule command can be specified multiple times in the Calibre LFD rule file before specifying the LFD::Anchor command. Duplicate layer specifications are not supported within LFD::AnchorRule.

The relationship between LFD::AnchorRule and LFD::Anchor specifications is as follows:

- Arguments for input layers specified in LFD::AnchorRule must also be specified as input layer arguments in LFD::Anchor.
- Arguments specified in LFD::AnchorRule override corresponding arguments specified in LFD::Anchor for a specific input layer or input layer combination.
- Arguments not specified in LFD::AnchorRule default to argument values specified in LFD::Anchor for a specific input layer or input layer combination.

You can specify a single layer or layer pair (same-layers or different layers) as an input to LFD::AnchorRule. However, a rule must be defined for the single layer or layer pair combination.

Overlapping layers, as in the case of stitches (multi-pattern) or vias (multilayer) that meet width constraints, receive a center anchor point in the middle of the overlap area.

Arguments

- **-layer1 *layer1_name***

Required keyword and argument specifying the name of the first input layer.

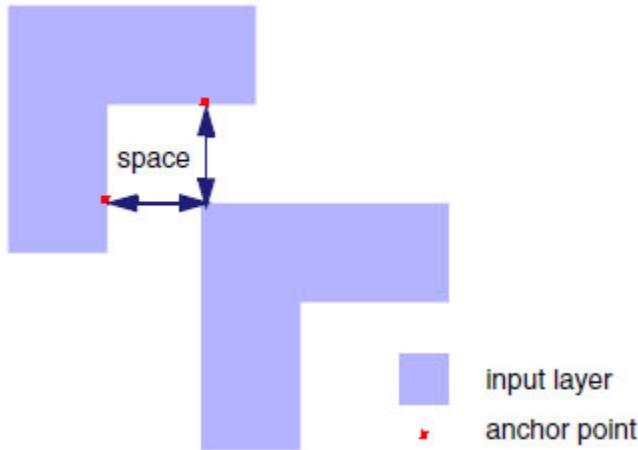
- **-layer2 layer2_name**

Optional keyword and argument specifying the name of a second input layer in the case of multi-pattern or multilayer (via) anchor points. If the -layer2 keyword is specified, the -width2 keyword must also be specified.

- **-space space_value**

Required keyword and argument specifying the critical space between shapes of the input layer or layers. This value gives the constraint for anchors near convex corners. Spaces at or below this value are considered for anchor point sites. Specify this value in user units.

Figure 7-30. LFD::AnchorRule Space



- **-length_LE length_LE_value -space_LE_LE space_LE_LE_value -space_LE_E space_LE_E_value -space_E_E space_E_E_value**

Required keyword set and arguments specifying the critical length and space constraints for generating anchor points. This keyword set is used in place of the **-space** argument.

-length_LE length_LE_value

Required keyword and argument specifying the maximum length of an edge that can be defined as a line-end. Any length greater than this value is defined as an edge. Specify this value in user units.

Figure 7-31. LFD::AnchorRule Length LE



-space_LE_LE space_LE_LE_value

Required keyword and argument specifying the maximum space between two line-ends that can result in anchor points. Parallel line-ends with separation less than or equal to **-space LE LE** result in an anchor point between the two parallel line-ends. Specify this value in user units.

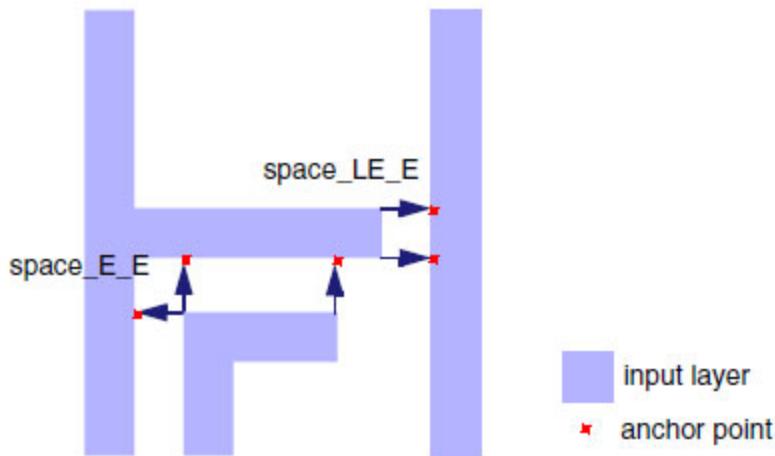
Figure 7-32. LFD::AnchorRule Space LE LE



-space LE_E space LE_E value

Required keyword and argument specifying the maximum space between a line-end shape and an edge that can result in anchor points. Only line-end corners with edge distances that are less than or equal to this value are considered for anchor points. Specify this value in user units.

Figure 7-33. LFD::AnchorRule Space LE_E and E_E



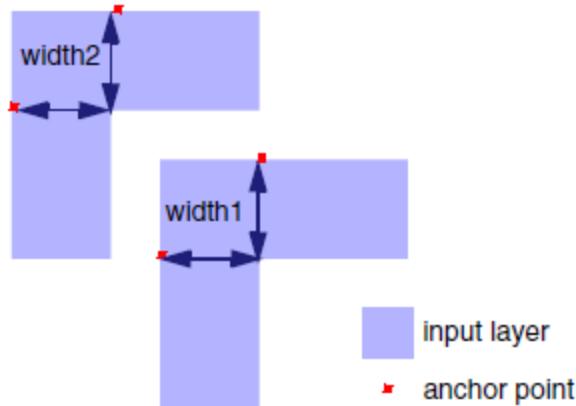
-space E_E space E_E value

Required keyword and argument specifying the maximum space between edges that can result in anchor points. Only edges whose distances are less than or equal to this space are considered for anchor points. Specify this value in user units.

• **-width1 width1 value**

Required keyword and argument specifying the critical width of the first input layer shape. Edges within the critical distance from concave corners receive anchor points. Specify this value in user units.

Figure 7-34. LFD::AnchorRule Widths

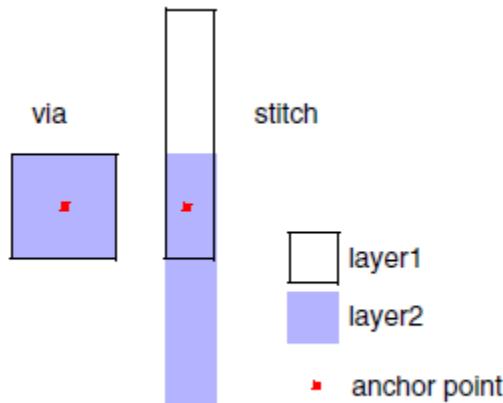


- **-width2 *width2_value***

Optional keyword and argument specifying the critical width of the second input layer shape. If the -width2 keyword is specified, the -layer2 keyword must also be specified. Edges within the critical distance from concave corners receive anchor points. Specify this value in user units.

For a stitch or via, an anchor point is generated in the center of the overlap region. The anchor point site is determined by the width value(s) specified in LFD::AnchorRule. If not specified, the width value specified in LFD::Anchor is used.

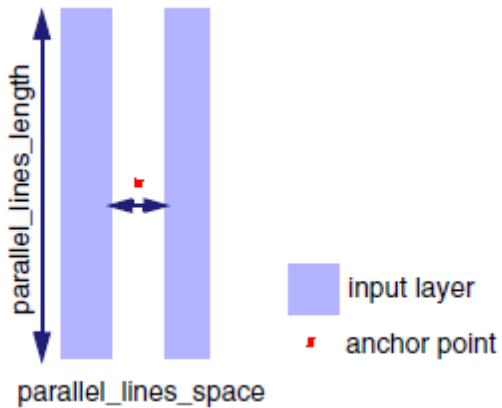
Figure 7-35. LFD::AnchorRule Stitch and Via Anchor Points



- **-parallel_lines_length *parallel_lines_length_value***

Optional keyword and argument specifying the minimum common length for edges to be considered parallel line shapes. If specified, the -parallel_lines_space keyword and value must also be specified. Specify this value in user units.

Figure 7-36. LFD::AnchorRule Parallel Lines Length and Space



- `-parallel_lines_space parallel_lines_space_value`

Optional keyword and argument specifying the maximum space between common length edges to be considered for anchor point sites. Resulting anchors are located at the midpoint between the parallel line shapes. If specified, the `-parallel_lines_length` keyword and value must also be specified. Specify this value in user units.

Examples

Example 1

The LFD::AnchorRule command must precede the LFD::Anchor command in the Calibre LFD rule file. In this example, the `-space` keyword is not specified in LFD::AnchorRule, so the spacing rules are defined by the keyword set `-length_LE`, `-space_LE_LE`, `-space_LE_E`, and `-space_E_E`.

LFD::AnchorRule spacing rules override corresponding LFD::Anchor spacing rules, likewise, LFD::AnchorRule width rules (`-width1` and `-width2`) override LFD::Anchor `-width` rule.

```
LFD::AnchorRule \
-layer1 M1_1 \
-layer2 M1_2 \
-length_LE 0.06 \
-space_LE_LE 0.04 \
-space_LE_E 0.05 \
-space_E_E 0.06 \
-width1 0.04 \
-width2 0.04

LFD::Anchor
-layer {M1_1 M1_2} \
-width 0.05 \
-space 0.05 \
-parallel_lines_length 0.7 \
-parallel_lines_space 0.03 \
-merge_distance 0.05 \
-layer_out anchors
```

Example 2

In this example, the input layer M1_2 is not specified in LFD::AnchorRule. Both the argument values for M1_2 and the layer combination {M1_1 M1_2} default to the values specified in LFD::Anchor.

```
LFD::AnchorRule \
-layer1 M1_1 \
-length_LE 0.06 \
-space_LE_LE 0.04 \
-space_LE_E 0.05 \
-space_E_E 0.06 \
-width1 0.04

LFD::Anchor
-layer {M1_1 M1_2} \
-width 0.05 \
-space 0.05 \
-parallel_lines_length 0.7 \
-parallel_lines_space 0.03 \
-merge_distance 0.05 \
-layer_out anchors
```

Band

Creates PV-bands with two input contours.

Usage

Band

```
-handle handle_name
-layer input_layer_name
-subwindow process_window_number
-minBandEdge inner_contour_layer
-maxBandEdge outer_contour_layer
```

Description

Creates PV-bands with two input contours. One contour is the minimum PV-band edge and the other is the maximum PV-band edge. These contours are either layers (original or derived) or contour handles generated using the LFD::[Contour](#) command.

Given two input contours, Calibre LFD generates the PV-band and absolute PV-band layers as follows:

- BandLayer = NOT max_band_edge min_band_edge
- AbsBand layer = and input_layer_name BandLayer

You can use the PV-bands generated with this command with Calibre LFD checks. Each PV-band generated with the LFD::Band command requires a unique handle per layer. If this handle has the value of “default”, the PV-band is treated as if it was created using the LFD::[PVband](#) command.

Note

 This command is part of the Customizable PV-bands interface, which uses the LFD::Contour, LFD::Band, and LFD::[OverlayBand](#) commands to provide you flexibility in generating simulation contours and PV-bands, and the ability to use your custom-generated contours with the [Calibre LFD Checks](#). See the [Customizable PV-Bands](#) section for additional information.

Arguments

- **-handle *handle_name***

Required keyword and argument specifying a PV-band handle which must be unique per layer. This argument along with the -layer argument gives the PV-band a unique name that can be used to reference the PV-band in further operations.

- **-layer *input_layer_name***

Required keyword and argument specifying the name of the design layer for this PV-band. The layer passed to this argument is a registered layer. If it is not registered, this command registers the layer. This option is not for use in a PDK.

- **-subwindow *process_window_number***

Required argument and integer, defining the severity of the generated PV-band. This option is not for use in a PDK.

- **-minBandEdge *inner_contour_layer***

Required keyword and argument defining the minimum PV-band edge of the PV-band. The argument accepts either a contour handle generated previously during the run or a layer name (original or derived).

- **-maxBandEdge *outer_contour_layer***

Required keyword and argument defining the maximum PV-band edge of the PV-band. The argument should accept either a contour handle generated previously during the run or a layer name (original or derived).

Examples

This example creates a PV-band for the contours created in the section, [Example 1: Generating Simulation Contours with LFD::Contour](#):

```
LFD:::Band -handle band1 -layer poly -subwindow 1 -minBandEdge min_edge \
-maxBandEdge max_edge
```

Note

 You can use the PV-bands created with LFD::Band in Calibre LFD checks, as you use PV-bands created with the LFD::PVband command.

Begin

Command used to mark the start of the Calibre LFD processing block.

Usage

Begin

[-encrypted yes]

Description

Trigger command used to set off the Calibre LFD processing block. The processing block is the portion of the rule file that calls the Calibre LFD package, and it must be separated from the rest of the rule file using the LFD::Begin and LFD::End functions. The LFD::Begin function is not for use in a PDK, and the function must be paired with LFD::End call.

The LFD::Begin command must be placed before the first call to any other Calibre LFD function.

Note

 See [Defining the Calibre LFD Block](#) for more information on delineating a Calibre LFD processing block.

Arguments

- -encrypted yes

Optional keyword and argument used to allow encryption of the rule file. This option is only used when encrypting the whole rule file using caltvfencrypt as follows:

```
caltvfencrypt -id rulesID rules_file_name encrypted_rule_file_name
```

and then executing the encrypted rule file.

Examples

This example shows the pairing of LFD::Begin and LFD::End:

```
// Begin Calibre LFD Block
LFD::Begin
LFD::PVband ...
LFD::MinWidthCheck ...
...
LFD::CaptureContour ...
...
LFD::End
// End Calibre LFD Block
```

CaptureContour

Stores a previously generated contour as a derived layer.

Usage

Syntax 1

CaptureContour

```
-layer input_layer_name
-layerOut return_layer_name
{ {-dose real_number [-dose2 real_number]
    -optical model1 [-optical2 model2]
    [-size size1 [-size2 size2]]
    [-subwindow expr_number] } |
  {-displacement {max | min}
    -reference {regular | absolute}
    -subwindow expr_number} }
[-errorFilterSize size_filter]
[-resist resistFile1 [-resist2 resistFile2]]
[-etch etchFile1 [-etch2 etchFile2]]
[-contourHandle contour_handle]
```

Syntax 2

CaptureContour

```
-layer input_layer_name
-layerOut return_layer_name
-pdkCheckName check_template
[additional_options]
```

Description

Saves a previously calculated contour to a derived layer with the specified name. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You can use this command to capture any of the following types of contours:

- Printing contour for a specific focus and dose condition.
- Regular maximum (outer) or minimum (inner) contour for a specific experiment.
- Absolute maximum (outer) or minimum (inner) contour for a specific experiment.

Each command captures one contour. You can issue this command as often as is needed.

If used with a PDK, this function calls one of the contour definitions in the PDK to generate a PV-band contour for the specified layer and save it as a new layer with the specified name.

Note

 For PDK usage, this command is only available if the security options are set such that writing contour data is allowed.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the captured contour is written. If you issue the CaptureContour command more than once, each *return_layer_name* must be unique.

- **-dose *real_number* -dose2 *real_number***

Required keyword and argument used to identify which contour to capture. Each contour can be uniquely identified by the dose, focus, and mask bias used to generate the contour. The -dose keyword defines the dose condition that is of interest. If you are using a double exposure process, you must also specify the dose for the second exposure.

The dose value must be one of the dose values investigated through the LFD::PVband command.

You must specify either (-dose and -optical) OR (**-displacement**, **-reference**, and **-subwindow**).

- **-optical *model1* -optical2 *model2***

Required keyword and argument used to identify which contour to capture. Each contour can be uniquely identified by the dose, focus, and mask bias used to generate the contour. The -optical keyword defines the optical model of interest. If you are using a double exposure process, you must also specify the optical model for the second exposure.

The model must be one of the models used to generate the contour through the LFD::PVband command.

You must specify either (-dose and -optical) OR (**-displacement**, **-reference**, and **-subwindow**).

- **-size *size1* -size2 *size2***

Optional keyword and argument used to identify which contour to capture. Each contour can be uniquely identified by the dose, focus, and mask bias used to generate the contour. The -size keyword defines the mask bias that is of interest. The mask bias must be expressed as real numbers representing the bias size.

If you are using a double exposure process and you specify the mask bias for the first exposure, you must also specify the mask bias for the second exposure.

The mask bias must be the exact mask bias used when generating the contour with the dose and focus specified by the -dose and -optical keywords.

This keyword can be used only when -dose and -optical are also specified.

- **-displacement {max/ min}**

Required keyword and argument specifying which edge of the PV-band to capture:

- **max** — Specifies the command works on the outer edge.
- **min** — Specifies the command works on the inner edge.

You must specify either (-dose and -optical) OR (**-displacement**, **-reference** and -subwindow).

If you are using this command to save contours for use with RegisterBands in some future run, you *must* use **-displacement**.

- **-reference {regular | absolute}**

Required keyword and argument defining the type of PV-band contour to capture:

- **regular** — Represents the maximum and minimum edge displacement.
- **absolute** — Represents the maximum and minimum deviation from the target layout. The regular PV-bands are always within the absolute PV-bands.

You must specify either (-dose and -optical) OR (**-displacement**, **-reference** and -subwindow).

- **-subwindow expr_number**

Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the LFD::PVband command used to generate the PV-band data being checked. Thus, *expr_number* refers to an index to a list of experiments.

The **-subwindow** option is required when you specify **-displacement** and **-reference**. If -dose, -optical, or -size are specified, then you have the option to specify **-subwindow**. If **-subwindow** is not specified with this setup, then by default the command looks in the first subwindow that is listed in either the LFD::PVband call, or the [addPDKLayer](#) first subwindow section in the PDK. In this mode, when you request a subwindow, you may want to specify LFD::PVband -independentWindows, depending on your mode of operation.

You must specify either (-dose and -optical) OR (**-displacement**, **-reference**, and -subwindow).

- **-errorFilterSize size_filter**

Optional keyword and argument used to control which portions of the PV-band are important enough to write to the output database.

- This operation outputs only the PV-bands around the error markers generated for this layer within a window defined by the *size_filter*.
- When not specified, this command writes the entire PV-band to the specified database.

The *size_filter* must be a positive real number expressed in microns. If *size_filter* is less than 0.01, the tool issues an error message and abort.

Only the error markers for checks performed prior to this command being issued are used in this filtering.

- **-resist *resistFile1* -resist2 *resistFile2***

Optional argument used to define the pathname to the resist model if -resistSpanList was used in creating PV-bands for the input layer. If -resistSpanList2 was used in creating PV-bands for the input layer, then -resist2 should also be specified.

- **-etch *etchFile1* -etch2 *etchFile2***

Optional argument used to define the pathname to the etch model if -etchSpanList was used in creating PV-bands for the input layer. If -etchSpanList2 was used in creating PV-bands for the input layer, then -etch2 should also be specified.

- **-contourHandle *contour_handle***

Optional keyword and argument used to perform Calibre LFD checks on specific contour handles. This option is for use with the contour handle generated using the [Customizable PV-Bands](#) flow.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how this check is performed.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK CaptureContour command. Any additional options specified overrides the options defined within the PDK.

Examples

Example 1

This example saves previously calculated contours to derived layers:

```
LFD::CaptureContour -layer active -layerOut contour_1 -dose 0.9500 \
    -optical P1W_D0
LFD::CaptureContour -layer active -layerOut max_abs_contour3 \
    -displacement max -reference absolute -subwindow 3
```

Example 2

This example uses multi-patterning:

```
LFD::CaptureContour -layer poly -layerOut nominalPoly \
-dose 1.00 -dose2 1.00 -optical P1W_D0 -optical2 P1W_D0 \
-errorFilterSize 0.1
```

Example 3

This example uses CaptureContour within a PDK:

```
LFD::addPDKCheck {
    -name Active_CNTR_nom
    -pdk lfd_demo
    -type CaptureContour
    -security 11
    -definition {
        -dose 1.0
        -optical ACTV_D0
    }
}
```

And the following is defined in your TVF code:

```
LFD::CaptureContour -pdkCheckName Active_CNTR_nom -layer active \
-layerOut Active_CNTR_nom
```

ClassifyConfig

Configures common user-defined classification options.

Usage

ClassifyConfig

```
-name classifyBlock_name
-halo halo_size
[-coarse_match coarse_match_size]
[-maxsize max_extent]
[-maximum_error_number error_count]
[-duplicates { suppress | keep }]
[-anchor layer [-anchor_max_snap anchor_snap_size] [-anchor_halo anchor_halo_size]]
[-reflections_rotations_match { decide | yes | no | reflections }]
[-score_bin_size score_delta]
[-score_property property_name]
[-score_order { smallest | largest }]
[-maximum_worst_error_number error_count]
```

Description

Configures common user-defined error classification options to be performed as part of Calibre LFD. For each check to use these settings, the check call must set its -classify option to the name of the LFD::ClassifyConfig command defined previously.

The output contains unique instances of errors providing a mechanism to prioritize those errors that occur more frequently in the layout. Using the ClassifyConfig command can reduce the database size by reducing the number of database error markers.

For a classified check, the result information is attached to error markers as CLASS_FLAT_COUNT and CLASS_ID properties.

If suppressing duplicates:

- Unique errors: CLASS_FLAT_COUNT encodes the flat count of instances of this particular error, including the duplicates and the unique error itself.
- Unclassified errors: CLASS_FLAT_COUNT is set to the total placement count of the pseudo-cell that contains the unclassified errors.

If keeping duplicates:

- Duplicate errors: CLASS_FLAT_COUNT is set to 0.
- Duplicate errors have their CLASS_ID set to the identifier of the corresponding unique error.

Unclassified errors have their CLASS_ID set to 0.

This command is enabled for the following checks:

- [AddCustomOPCVCheck](#)
- [AreaCheck](#)
- [AreaOverlayCheck](#)
- [InterLayerSpaceCheck](#)
- [LineEndCheck](#)
- [MaxAreaVariabilityCheck](#)
- [MaxCDVariabilityCheck](#)
- [MinAreaOverlapCheck](#)
- [MinOverlayCheck](#)
- [MinSpaceCheck](#)
- [MinWidthCheck](#)
- [NonPrintingCheck](#)

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 $2 * \text{anchor_snap_size}$ option.

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

Arguments

- **-name** *classifyBlock_name*

Required keyword and argument which defines the ClassifyConfig name that is used by the checks.

- **-halo** *halo_size*

Required keyword and argument specifying a halo distance around an error, used as the classification region around each error. A square of size $2 * \text{halo_size}$ is clipped out around each error center. This value is expressed in microns.

- **-coarse_match** *coarse_match_size*

Optional keyword and argument that triggers approximate matching. When this option is specified, some patterns that differ by less than *coarse_match_size* are classified as duplicates. This value is expressed in microns, and the default value is 0.

- `-maxsize max_extent`

Optional keyword and argument that sets the maximum extent of errors. Errors with an extent exceeding *max_extent* are considered unclassified. This value is expressed in microns, and the default value is 0.5.

- `-maximum_error_number error_count`

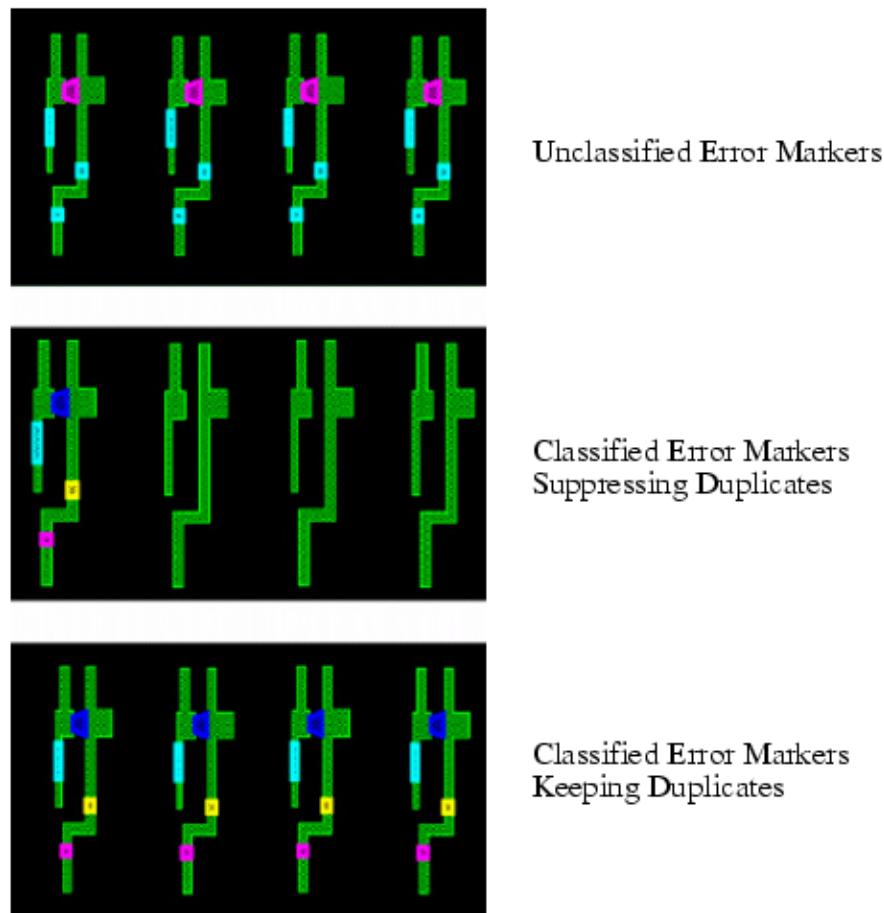
Optional keyword and argument that turns off classification after counting *error_count* errors. The command then marks errors in the remaining tiles as unclassified. This value is an integer, and the default is to classify all errors.

- `-duplicates { suppress | keep }`

Optional keyword which allows for the output of only unique and unclassified errors (using `-duplicates suppress`). Setting `-duplicates keep` outputs all errors. By default duplicates are suppressed.

Note

 Classification with `-duplicates suppress` affects the results provided by the LFD::VariabilityIndices function. The DVI is only a metric of the unique errors present in the layout, and PVI is only calculated in a smaller area delimited by the unique errors and not all errors.

Figure 7-37. Classification Duplicates

- **-anchor *layer***

Optional keyword and argument specifying a layer to use in anchoring error extent centers. This makes classification less sensitive to the location of the error's center point, reducing the number of unique errors. Use the drawn (target) layer as the anchor layer. A layer must be a drawn layer. The default is the anchor layer defined in the check.

- **-anchor_max_snap *anchor_snap_size***

Optional keyword and argument specifying the maximum distance which the center point of an error extent can move to snap to a grid induced by the closest vertex on the anchor layer (within the halo region from the error's center point). Larger values yield fewer unique errors. This value is expressed in microns, and the default value is 0.2.

- **-anchor_halo *anchor_halo_size***

Optional keyword and argument that allows for the use of a region larger than *halo_size* for finding anchoring vertices. This value is expressed in microns, and the default is the value for *halo_size*.

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

Optional keyword that can be used to override default reflection behavior:

- “decide” — Lets the tool determine behavior.
- “yes” — Forces classification of rotated or mirrored patterns as duplicates.
- “no” — Forces classification of rotated or mirrored patterns as non-duplicates.
- “reflections” — Forces classification of X- and Y-axis reflected patterns as duplicates.

Caution



Setting the value to “no” causes the tool to clone all mirrored and rotated cell placements during the creation of the database, which may result in slower runtimes.

Setting the value to “reflections” causes the tool to clone all rotated cell placements during the creation of the database, which may result in slower runtimes (but still faster than the “no” option which clones *both* rotated and mirrored placements).

- `-score_bin_size score_delta`

Optional keyword and argument defining the difference between the scores of two errors that can still be considered duplicates if their geometric contexts match. This value is expressed in microns. This option can be used only if the classified check has a property defined in it.

- `-score_property property_name`

Optional keyword and argument that accepts a property name. This property is defined as an argument in the check the uses this classify block. The default is the first property defined in the property list of the check.

- `-score_order {smallest | largest}`

Optional keyword which specifies whether the smallest or largest score is the worst.

- `-maximum_worst_error_number error_count`

Optional keyword and argument that sets the maximum number of errors to output. This value is an integer.

Note



The scoring options, `-score_bin_size`, `-score_property`, `-score_order`, and `-maximum_worst_error_number` are not used with the following checks:
[AreaCheck](#), [AreaOverlayCheck](#), [MaxAreaVariabilityCheck](#), [MaxCDVariabilityCheck](#), [NonPrintingCheck](#).

`MinAreaOverlapCheck` accepts `-score_bin_size`, `-score_property`, `-score_order`, and `-maximum_worst_error_number`. The value defined by the `-areaType` option is considered the property.

Examples

Example 1

```
LFD::ClassifyConfig -name classifyBlock_name -halo 3 \
                     -maximum_error_number 1000 -duplicates keep

LFD::MinWidthCheck .... -classify classifyBlock_name
```

Example 2

```
LFD::ClassifyConfig -name cname -halo 3 -score_bin_size 0.1 \
                     -property MinCD
LFD::MinWidthCheck .... -classify cname -property MinCD
```

Contour

Creates contours which can be used to create other contours, PV-bands, or to perform checks.

Usage

Syntax 1

Contour

```
-handle handle_name
-layer input_layer_name
-operation operation_type
-arguments argument_list
-inputLayers layer_list
-inputModels model_list
[-lithoModel {"litho_model_name" {"lithomodel"} "}]"
[-imageOptions {"image_options_name" {"image_options_block"} "}]"
[-opticalModels {"optical_model_name1" {"optical_model1"} ... "}]"
[-resistModel {"resist_model_name" {"resist_model"} "}]"
[-etchModel {"etch_model_name" {"etch_model"} "}]"
[-ddmModel {"ddm_model_name" {"ddm_model"} "}]"
```

Syntax 2

Contour

```
{-inputModels model_list | -lithoModel litho_model}
-pdkCheckName check_template
[additional_options]
```

Description

Creates contours which can be used to create other contours, PV-bands, or to perform checks.
Each contour represents a single Calibre OPCverify operation, such as:

- Logical operations
- The shift operation
- The size operation
- The simulation operation

You specify a handle for each defined contour. This handle must be unique per layer. For example, the layer poly can have only one contour handle c1, but both layer poly and layer active may have a contour handle c1.

You define the required input layers for each operation. The input layers for each operation must be one of the following:

- Original GDS layers
- Derived layers

- Previously-defined contour handles

For operations that require arguments, you specify the needed input arguments of the operation. For example, the “xshift” and “yshift” values in the case of the shift operation, and the “sizeValue” in the case of the size operation.

For operations that require input models, you must specify the full path to those models. Models supported by the interface are these:

- Optical models
- Resist models
- Etch models
- DDM models

For simulation operation and user-defined operation, you must specify a definition of those operations. This definition is treated as a template with a placeholder for layers and models. The placeholders are later replaced with the corresponding model and layer names.

This command is part of the customizable PV-bands interface, which uses the LFD::Contour, LFD::Band, and LFD::OverlayBand commands to provide you flexibility in generating simulation contours and PV-bands, and the ability to use your custom-generated contours with the [Calibre LFD Checks](#). See the [Customizable PV-Bands](#) section for additional information.

For PDK usage, see the [Example 5: PDK Implementation of LFD::Contour](#) section.

LFD::Contour Litho Model Support

Using the LFD::Contour function with litho models (litho model mode), requires the inputs to have the necessary information for creating the Calibre OPCverify setup file. This information includes the model definitions.

- **Litho Model Definition** — Litho models can be defined either inline or within a litho model file that resides in the model directory (non-inline). In the case of inline litho models, the model components are replaced with the corresponding symbolic names of the models. Inline litho models must be previously defined by “*_model_load” commands that load each component model. In the case of non-inline litho models, the model components are defined in a litho model file “Lithomodel” that resides in a litho model directory.

Refer to the *Calibre OPCverify User’s and Reference Manual* for a complete description of the litho model and OPCverify setup files.

- **Inline Litho Model Usage** — The inline definition of a litho model is intended only for use in creating encrypted setup files for inclusion in encrypted TVF and PDKs. Do not use this in standard Calibre OPCverify or Calibre nmOPC setup files. The litho model

inline definition 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_DEVEL  
OPMENT=1
```

Refer to “[Encrypting the Calibre LFD Rules](#)” on page 510 for information on how to encrypt the rule files that perform OPC and Calibre LFD.

- **PDK Litho Model Definition** — Using the LFD::Contour function with litho models (litho model mode) in a PDK is similar to the non-PDK command call. Both inline and non-inline litho models are only defined in the -processInfo subsection of the PDK. Model definitions including resist, etch, optical, and DDM are mandatory only if the litho model is inline.

Arguments

- **-handle *handle_name***

Required keyword and argument specifying a handle for a defined contour. This handle must be unique per layer. For example, the layer poly can have only one contour handle c1, but both layer poly and layer active may have a contour handle c1. This argument along with the -layer argument give the contour a unique name that can be used to reference the contour in other operations.

- **-layer *input_layer_name***

Required keyword and argument specifying the name of the design layer for this contour. The layer passed to this argument is a registered layer. If the layer is not registered, this command registers the layer. This argument is not for use in a PDK.

- **-operation *operation_type***

Required keyword and argument defining how the contour is generated. This argument accepts one of the following keywords:

- or, and, xor, not
- shift
- size
- simulation
- user_defined

According to the value of ***operation_type***, the following arguments are either required or not.

- In case of the logical operations (or, and, xor, not) — Only **-inputLayers** is required.
- In case of the shift and size operations — **-inputLayers** and **-arguments** is required.
- In case of the simulation operation — **-inputLayers**, **-arguments**, and **-inputModels** are required.

- In case of the user_defined operation — **-inputLayers** and **-arguments** are required, and **-inputModels** is optional or required based on the operation type.
- **-arguments argument_list**

This option is required for the following *operation_type* settings — size, shift, simulation, and user_defined.

This option takes a list of arguments that depend on the value of *operation_type*.

- size operation — **-arguments** accepts a list of two elements: {-sizeVal value}
For example: **-arguments** {-sizeVal 0.01}
- shift operation — **-arguments** accepts a list of four elements: {-xshift *shift_value* -yshift *shift_value*}
For example: **-arguments** {-xshift 0.01 -yshift 0.02}
- The simulation and user_defined operations — **-arguments** accepts a list of two elements: {-definition *definition*}

The definition passed to this argument must be a Calibre OPCverify setlayer command syntactically, with all of the layers replaced by the placeholder (\$layeri) and all of the model names replaced by (\$modeli).

- **-inputLayers layer_list**

Required argument that specifies the list of input layers or contour handles required to perform the Calibre OPCverify setlayer operation.

This argument is not required if the -imageOptions argument is specified in litho model mode.

The layers passed to this argument can be either an original GDS layer, a derived layer, or the handle of a contour previously defined in the Calibre LFD kit.

The number of layers required in the layer list depends on the operation type:

- Operations (or, and) — **-inputLayers** accept two or more layers.
- Operations (not, xor) — **-inputLayers** accepts only two layers.
- Operation shift — **-inputLayers** accepts only one layer.
- Operation size — **-inputLayers** accepts only one layer.
- The simulation and user_defined operations — The number of layers passed to this operation is equal to the number of layer placeholders specified in the definition.

If the input layer to the operation is a previously-defined contour for a different layer than the one specified for this contour, then the contour handle is scoped by the layer name using the scope operator (::).

For example, say that layer poly has contours C1, C2. Also, layer contact has contour C1, C2. We want to define contour C3 for layer poly, and this contour represents the Poly C1 not contact C1.

```
LFD::Contour -layer poly -handle c1 ...
LFD::Contour -layer poly -handle c2 ...
LFD::Contour -layer contact -handle c1 ...
LFD::Contour -layer contact -handle c2 ...
LFD::Contour -layer poly -handle polyC1NotcontactC1 -operation not
    -inputLayers {c1 contact::c1}
```

- **-inputModels *model_list***

This argument is required for simulation and user_defined operations. The argument specifies a list of models (inline models, absolute paths of models). The models specified with this list replaces the placeholders in the operation definition. This argument is not required in litho model definitions.

The number of models specified must be equivalent to the number of models placeholders specified in the definition of the operation.

- **-lithoModel “{”*litho_model_name* “{”*lithomodel* “}” “}”**

Optional argument that specifies the name of the litho model followed by the litho model definition (inline) or the litho model path (non-inline), where the litho model path is the name of a directory. This argument is required in litho model mode.

- **-imageOptions “{”*image_options_name* “{”*image_options_block* “}” “}”**

Optional argument that specifies the name of the image options followed by the image options block that is used for simulation. This argument is used in litho model mode, and if specified, replaces the **-inputLayers** argument. For more information on defining the **-imageOptions** argument, refer to the *OPCverify User’s and Reference Manual*.

- **-opticalModels “{”*optical_model_name1* “{”*optical_model1* “}”...“}”**

Optional argument that specifies the name of the optical model followed by the optical model that is used for simulation. Multiple models can be specified. This argument is used in litho model mode. In the case of using inline litho models, this argument must be defined.

- **-resistModel “{”*resist_model_name* “{”*resist_model* “}” “}”**

Optional argument that specifies the name by which the resist model is referenced followed by the resist model that is used for simulation. This argument is used in litho model mode. In the case of using inline litho models, this argument must be defined.

- **-etchModel “{”*etch_model_name* “{”*etch_model* “}” “}”**

Optional argument that specifies the name by which the etch model is referenced followed by the etch model that is used for simulation. This argument is used in litho model mode. In the case of using inline litho models, this argument must be defined if an etch model is required.

- **-ddmModel** “{”*ddm_model_name* “{”*ddm_model* “}” “}”

Optional argument that specifies the name of the Domain Decomposition Method (DDM) model followed by the DDM model that is used for simulation. This argument is used in litho model mode. In the case of using inline litho models, this argument must be defined if a DDM model is required.

- **-pdkCheckName** *check_template*

Required keyword and argument specifying the name of the check template defining how this check is performed.

- *additional_options*

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK Contour command. Any additional options specified override the options defined within the PDK.

Examples

Here are some examples of generating contours.

Example 1: Generating Simulation Contours with LFD::Contour

This example generates simulation contours for layer Poly:

- Optical Span List {Poly_50 Poly_0 Poly_50}
- Dose Span List {0.98 1.00 1.02}
- Number of masks = 1
- Resist model: poly.mod
- Etch model: etch.mod
- RETLayers: poly_opc
- Background: clear
- Foreground: attenuated 0.06

To generate the first contour in Calibre OPCverify, we use the image command with the following syntax:

```
"image mask0 optical Poly_50 background dark layer poly_opc attenuated
0.06 dose 0.98 size 0 resist_model poly.mod etch_model etch.mod"
```

This definition is supplied to the “**-arguments** {-definition *definition*}” argument in the LFD::Contour command, with some modifications:

- No layer names are mentioned in the definition. All layers are replaced by a \\$layeri placeholder.

- No models are mentioned in the definition. All models are replaced by `\$model1` placeholder.

The template for the definition is written as:

```
"image mask0 optical \$model1 background dark layer \$layer1 attenuated  
0.06 dose 0.98 size 0 resist_model \$model2 etch_model \$model3"
```

The corresponding names and paths for the layers and models are supplied to the Contour command using the **-inputLayers** and **-inputModels** arguments.

For this contour:

- **-inputLayers** {poly_opc}
- **-inputModels** {Poly_50 poly.mod etch.mod}

The LFD::Contour command is written as follows:

```
LFD::Contour  
-layer poly  
-handle c1  
-operation simulation  
-arguments {-definition { image mask0 optical \$model1 background dark  
layer \$layer1 attenuated 0.06 dose 0.98 size 0 resist_model  
\$model2 etch_model \$model3 }}  
-inputLayers {poly_opc}  
-inputModels {Poly_50 poly.mod etch.mod}
```

The three contours representing the process window in the example can be written as follows:

```
set opticalList {Poly_50 Poly_0 Poly_50}  
set doseList {0.98 1.00 1.02}  
set counter 1  
foreach optical $opticalList dose $doseList {  
    LFD::Contour -layer poly -handle c$counter -operation simulation  
    -arguments [list -definition [list image mask0 optical \$model1  
        background dark layer \$layer1 attenuated 0.06 dose $dose size  
        0 resist_model \$model2 etch_model \$model3]]  
    -inputLayers {poly_opc}  
    -inputModels {$optical poly.mod etch.mod}  
    incr counter  
}
```

If you need to calculate the maximum edge contour and the minimum edge contour for this process window:

```
LFD::Contour -layer poly -handle min_edge -operation and \  
    -inputLayers {c1 c2 c3}  
LFD::Contour -layer poly -handle max_edge -operation or \  
    -inputLayers {c1 c2 c3}
```

The following example shows how to use those two contours to represent a PV-band.

Example 2: Double Patterning Example of Generating Simulation Contours with LFD::Contour

The multi-patterning example also generates simulation contours for layer Poly:

```
Double patterning split
Process window:
Optical Span List 1 {Poly1_50 Poly1_0 Poly1_50}
Dose span list 1 {0.89 1.00 0.89}
Optical span list 2 {Poly2_50 Poly2_0 Poly1_50}
Dose span list 2 {0.89 1.00 0.89}
Resist model 1: poly1.mod
Etch model 1: poly1_etch.mod
Resist model 2: poly2.mod
Etch model 2: poly2_etch.mod
RETLayers 1: poly_opc_mask1
RETLayers 2: poly_opc_mask2
Foreground 1: attenuated 0.06
Foreground 2: attenuated 0.06
```

To create one contour for this layer, this example creates one contour to simulate the first mask:

```
LFD::Contour
-layer poly
-handle cl_mask1
-operation simulation
-arguments {-definition { image mask0 optical \$model1 background
    dark layer \$layer1 attenuated 0.06 dose 0.98 size 0
    resist_model \$model2 etch_model \$model3 }}
-inputLayers {poly_opc_mask1}
-inputModels {Poly1_50 poly1.mod poly1_etch.mod}
```

And to simulate the second mask:

```
LFD::Contour
-layer poly
-handle cl_mask2
-operation simulation
-arguments {-definition { image mask0 optical \$model1 background
    dark layer \$layer1 attenuated 0.06 dose 0.98 size 0
    resist_model \$model2 etch_model \$model3 }}
-inputLayers {poly_opc_mask2}
-inputModels {Poly2_50 poly2.mod poly2_etch.mod}
```

The final LFD::Contour command is implemented as:

```
LFD::Contour
-layer poly
-handle cl
-operation or
-inputLayers {cl_mask1 cl_mask2}
```

This example code generates the three contours representing this process window:

```
set opticalList1 {Poly1_50 Poly1_0 Poly1_50}
set doseList1 {0.98 1.00 1.02}
set opticalList2 {Poly2_50 Poly2_0 Poly2_50}
set doseList2 {0.98 1.00 1.02}
set counter 1
foreach optical1 $opticalList1 dose1 $doseList1 {
    LFD::Contour -layer poly -handle c$counter\_mask1 -operation simulation
        -arguments [list -definition [list image mask0 optical \$model1
            background dark layer \$layer1 attenuated 0.06 dose \$dose1 size 0
            resist_model \$model12 etch_model \$model3]]
        -inputLayers {poly_opc_mask1}
        -inputModels {$optical1 poly1.mod poly1_etch.mod}
    LFD::Contour -layer poly -handle c$counter\_mask2 -operation simulation
        -arguments [list -definition [list image mask0 optical \$model1
            background dark layer \$layer1 attenuated 0.06 dose \$dose2 size 0
            resist_model \$model12 etch_model \$model3]]
        -inputLayers {poly_opc_mask2}
        -inputModels {$optical2 poly2.mod poly2_etch.mod}
    LFD::Contour -layer poly -handle c$counter -operation or
        -inputLayers {c$counter\_mask1 c$counter\_mask2}
incr counter
}
```

If you need to calculate the maximum edge contour and the minimum edge contour for this process window:

```
LFD::Contour -layer poly -handle min_edge -operation and
    -inputLayers {c1 c2 c3}
LFD::Contour -layer poly -handle max_edge -operation or
    -inputLayers {c1 c2 c3}

LFD::Band -handle band1 -layer poly -subwindow 1
    -minBandEdge min_edge -maxBandEdge max_edge
```

Example 3: LFD::Contour Inline Litho Model Definition

This example defines inline litho models and generates contours:

```
set ::env(litho_model) {{
    version 1
    resist contact_mod
    mask 0 {
        background clear
        mask_layer 0 TRANS atten 0.06
        optical mCA_D0 focus 0nm
        optical mCA_D50 focus 50nm
    }
}}
LFD::Begin
LFD::SimConfig -layer contact \
-modelpath "./models"
set doseList {0.9800 1.0000}
set focusList {0nm 50nm}
set i 1
foreach fo $focusList dose $doseList {
    LFD::Contour "-handle c1$i -layer contact -operation simulation \
        -inputLayers {contact_opc1} \
        -opticalModels { {mCA_D0 {CA_D0}} {mCA_D50 {CA_D50}} } \
        -resistModel {contact_mod {contact.mod}} \
        -arguments {-definition {image \$image_options mask0 \
            focus $fo dose $dose bias 0}} \
    -lithoModel {litho_model1 {$::env(litho_model)}} \
        -imageOptions {
            image_options_io1 { \n \
                layer \\\$layer1 visible mask 0\n \
                litho_model \\\$litho_model \n \
            }\n" \
        LFD::Contour "-handle c2$i -layer contact -operation simulation \
            -inputLayers {contact_opc2} -opticalModels { {mCA_D0 {CA_D0}} \
            {mCA_D50 {CA_D50}} } \
            -resistModel {contact_mod {contact.mod}} \
            -arguments {-definition {image \$image_options mask0 focus $fo \
                dose $dose bias 0}} \
            -lithoModel {litho_model1 {$::env(litho_model)}} \
                -imageOptions {
                    image_options_io2 { \n \
                        layer \\\$layer1 visible mask 0\n \
                        litho_model \\\$litho_model \n \
                    }\n" \
            incr i
}
```

Example 4: Non-inline Litho Model Definition LFD::Contour

This example defines non-inline litho models and generates contours:

```
LFD::Begin
LFD::SimConfig -layer contact \
-modelpath "./models"
set doseList {0.9800 1.0000}
set focusList {0nm 50nm}
set i 1
foreach fo $focusList dose $doseList {
    LFD::Contour "-handle c1$i -layer contact -operation simulation \
        -inputLayers {contact_opc1} \
        -arguments {-definition {image \$image_options mask0 \
            focus $fo dose $dose bias 0}} \
        -lithoModel {lm1} \
        -imageOptions { \
            image_options_io1 { \n \
                layer \\\$layer1 visible mask 0\n \
                litho_model \\\$litho_model \n \
            }\n}"
    LFD::Contour "-handle c2$i -layer contact -operation simulation \
        -inputModels {} -inputLayers {contact_opc2} \
        -arguments {-definition {image \$image_options mask0 \
            focus $fo dose $dose bias 0}} \
        -lithoModel {lm1} \
        -imageOptions { \
            image_options_io2 { \n \
                layer \\\$layer1 visible mask 0\n \
                litho_model \\\$litho_model \n \
            }\n}"
    incr i
}
```

Example 5: PDK Implementation of LFD::Contour

The customizable PV-bands interface supports the Calibre LFD PDK flow. You define contours, PV-bands, and overlay PV-bands for each subwindow in the LFD::[addPDKLayer PVbands](#) section. Each LFD::addPDKLayer call in the PDK defines a layer template, and the information in that template is used to generate OPC output, simulated contours, and checks.

Syntax of the customizable PV-bands commands used in the PDK is:

```
LFD::Contour
  -handle handle_name
  -operation operation_type
  -arguments argument_list
  -inputLayers layer_list
  -inputModels models_list | -lithoModel litho_model
  -imageOptions image_options
  -opticalModels optical_models_list
  -resistModel model_name
  -etchModel model_name
  -ddmModel model_name
LFD::Band
  -handle handle_name
  -minBandEdge inner_contour_layer
  -maxBandEdge outer_contour_layer
LFD::OverlayBand
  -handle handle_name
  -minBandEdge inner_contour_layer
  -maxBandEdge outer_contour_layer
  -shift shift_direction
  -origBandHandle band_handle
```

The arguments for these commands have the same description as the arguments used in the non-PDK flow.

These commands can be used in the addPDKLayer as follows:

```
addPDKLayer {
  ...
  PvBand {
    subwindow1 {
      contour { ... }
      contour { ... }
      contour { ... }
      band { ... }
    }
  }
}
```

Input models passed to simulation contours or user-defined contours should be a model handle defined earlier in the PDK.

Input layers to contours are either:

- A contour handle previously defined in the PDK.
- The placeholder of a layer defined in the PDK.

You use PDK layers in contour definition in the following manner, where N is a list index which takes the values from 0 to the length of the layers list. M is the mask index, and is a value between 0 to the number of masks defined:

- Drawn Layer — %drawn
- Target Layer — %target(N)
- Visible Layers — %visible(N)
- Opc In Layers — %opcInLayer(N)
- Aux Layers — %auxLayer(N)
- Opc Layers — %opcLayer(N)
- Input Design Layers — %designLayer(N)
- RET Layers of Mask M — %mask M (retLayers, N)

The contour definition also accepts the following keywords, which instruct Calibre LFD to use the transmission value defined in the mask M section in the PDK:

- %mask M (foreground, N)
- %mask M (background, N)

Contours of one layer can be used in another layer by using the LFD::[RegisterLayer](#) -inputDesignLayers argument which accepts a list of design layers that fit into the placeholders defined in contour and band definitions in the PDK.

Litho model definitions can only be specified in the -processInfo subsection of the PDK. For example,

```
LFD::addPDKLayer {
    ...
    -processInfo {
        Lithomodel {
            Litho_model_name "{$ litho_model_path "}" ...
        }
        imageOptions {
            image_options_name "{$"
            inline_image_options_definition
            "}"
            ...
        }
        ...
    }
}
```

Where:

- **Lithomodel** — Contains a list of litho models used in the contour definitions inside the subwindow. Each litho model is given a name and a path (non-inline) or a name and a definition (inline).
- **Litho_model_name** — Defines the name in the PDK which is used to reference a litho model. This name is passed to -lithoModel in the contour command.
- **litho_model_path** — Specifies the name of the folder containing the litho model file named "Lithomodel" inside the model directory (non-inline).
- **imageOptions** — Contains a list of image options blocks used in the contour definitions inside the subwindow. Each block is given a name and a definition. The name is passed to the -imageOptions in the Contour command.
- **image_options_name** — Defines the name in the PDK which is used to reference an image options block. This name is passed to -imageOptions in the Contour command.
- **inline_image_options_definition** — Defines the image options block (inline). The definition can contain place holders for layers named `\$layer<n>` where n is the order of the layer according to the -inputLayers list in the Contour command. Place holders can also be used for the litho models with syntax `\$litho_model`.

Example 6: PDK Definition of Contours and PV-band Generation

This example shows how to define contours and generate a PV-band:

```
LFD::addPDKLayer { -name contactHoles -pdk doublePatterning -processInfo {
    DrawnLayer contactI
    IslandLayers {}
    OpcLayers contact_opc
    OpticalModels {
        CA_0 {contact/models/CA_D0}
        CA_50 {contact/models/CA_D50}
        CA_75 {contact/models/CA_D75}
    }
    ProcessCorrection { CMACRO contact_mopc_macro}
    PvBands {
        subwindow1 {
            Contour {-handle c1 -operation simulation -inputLayers
                {%mask0(retLayers,0)} -inputModels {CA_0 contact_mod}
                -arguments {-definition {image mask0 optical \$model1
                    background dark layer \$layer1 clear dose 1.01 size 0
                    resist_model \$model2}}}
            Contour {-handle c2 -operation simulation -inputLayers
                {%mask0(retLayers,0)} -inputModels {CA_50 contact_mod}
                -arguments {-definition {image mask0 optical \$model1
                    background dark layer \$layer1 clear dose 1 size 0 resist_model
                    \$model2}}}
            Contour {-handle cMax -operation or -inputLayers
                {c1 c2}}
            Contour {-handle cMin -operation and -inputLayers
                {c1 c2}}
            Band {-handle default -subwindow 1 -minBandEdge cMin
                -maxBandEdge cMax}
        }
    }
    ResistModels {
        contact_mod {
            ...
        }
    }
    RetLayers {
        mask0 {
            background {attenuated 0.06}
            layers {contact_opc contact_sraf}
            transmission {clear clear}
        }
    }
    TargetLayer contactTarget
    VisibleLayers contact_sraf
    opcInLayers {contactTarget contact_sraf}
    tags2boxes {}
    -optimizerInfo {}
}
```

Example 7: PDK Use of Layer Contours

This example shows how to use contours from another layer:

```
LFD::addPDKLayer { -name contactHoles -pdk doublePatterning -processInfo {
    DrawnLayer contactI
    IslandLayers {}
    ...
    PvBands {
        subwindow1 {
            Contour {-handle c1 -operation simulation -inputLayers
                {%mask0(retLayers,0)} -inputModels {CA_0 contact_mod}
                -arguments {-definition {image mask0 optical \$model1 background dark
                    layer \$layer1 clear dose 1.01 size 0 resist_model \$model2}}}
            ...
        }
    }
    LFD::addPDKLayer { -name PC -pdk doublePatterning -processInfo {
        DrawnLayer poly
        IslandLayers {}
        inputDesignLayers {CA}
        ...
        PvBands {
            subwindow1 {
                Contour {-handle c1 -operation simulation -inputLayers
                    {%mask0(retLayers,0)} -inputModels {PC_0 pc_mod}
                    -arguments {-definition {image mask0 optical \$model1 background dark
                        layer \$layer1 clear dose 1.01 size 0 resist_model \$model2}}}
                Contour {-handle c2 -operation or -inputLayers
                    {%designLayer(0)::c1 c1} }
                ...
            }
        }
    }
}
```

In the rule file, the two layers are registered as follows:

```
LFD::RegisterLayer -Designlayer contact -Processlayer contactHoles \
    -pdk pdk1
LFD::RegisterLayer -Designlayer poly -Processlayer PC -pdk pdk1 \
    -inputDesignLayers contact
```

Example 8: PDK Inline Litho Model Definition with Contours

This example defines inline litho models in a PDK and generates contours:

```
LFD::addPDKLayer { -name diffusion -pdk generic65 -processInfo {
    DrawnLayer active_IS
    IslandLayers {}
    inputDesignLayers {contact}
    OpcLayers {active_opc1 active_opc2}
    OpticalModels {
        mOD_0 {OD_D0}
        mOD_50 {OD_D50}
        mOD_100 {OD_D100}
    }
    ResistModels {
        active_mod {
            type VT-5
            version 3
            sampleSpacing 0.02
            referenceThreshold 0.25
            searchRange 0.3
            bound IMAX 0.19223 0.798292
            bound SLOPE 0.898645 4.40989
            bound ISLOPE 0.17 0.17
            bound IMIN 0.00314129 0.147129
            bound FACTOR -1.0743 4.18486
            minThreshold 0.05
            maxThreshold 1.0
            minEigenval 0.002
            hoodpix 1.8
            kerngrid 0.01
            ttermCount 1
            TTERM 0.25
        }
    }
    doublePatterning split
    Lithomodel {
        od_inline {
            version 1
            resist active_mod
            mask 0 {
                background clear
                mask_layer 0 TRANS atten 0.06
                optical mOD_0 focus 0nm
                optical mOD_50 focus 50nm
                optical mOD_100 focus 100nm
            }
        }
    }
    imageOptions {
        img_opt_od_inline1 {
            layer \$layer1 visible mask 0
            litho_model \$litho_model
        }
        img_opt_od_inline2 {
            layer \$layer2 visible mask 0
            litho_model \$litho_model
        }
    }
}
```

```

    }
ProcessCorrection      {}

PvBands {
    subwindow1 {

Contour {-handle cdpa1_1 -operation simulation -arguments {-definition
{image \$image_options mask0 focus 0nm dose 0.98 bias 0.0 }}
-opticalModels {mOD_0 mOD_50} -resistModel {active_mod} -inputLayers
{%opcLayer(0) %opcLayer(1)} -lithoModel {od_inline} -imageOptions {
img_opt_od_inline1}

Contour {-handle cdpa1_2 -operation simulation -arguments {-definition
{image \$image_options mask0 focus 50nm dose 1.0 bias 0.0 }}
-opticalModels {mOD_0 mOD_50} -resistModel {active_mod} -inputLayers
{%opcLayer(0) %opcLayer(1)} -lithoModel {od_inline} -imageOptions {
img_opt_od_inline1}

Contour {-handle cdpa2_1 -operation simulation -arguments {-definition
{image \$image_options mask0 focus 0nm dose 0.98 bias 0.0 }}
-opticalModels {mOD_0 mOD_50} -resistModel {active_mod} -inputLayers
{%opcLayer(0) %opcLayer(1)} -lithoModel {od_inline} -imageOptions
{img_opt_od_inline2}

Contour {-handle cdpa2_2 -operation simulation -arguments {-definition
{image \$image_options mask0 focus 50nm dose 1.0 bias 0.0 }}
-opticalModels {mOD_0 mOD_50} -resistModel {active_mod} -inputLayers
{%opcLayer(0) %opcLayer(1)} -lithoModel {od_inline} -imageOptions
{img_opt_od_inline2}

Contour {-handle c1 -operation or -inputLayers {cdpa1_1 cdpa2_1}}
Contour {-handle c2 -operation or -inputLayers {cdpa1_2 cdpa2_2}}
Contour {-handle min -operation and -inputLayers {c1 c2}}
Contour {-handle max -operation or -inputLayers {c1 c2}}
Band {-handle default -subwindow 1 -minBandEdge min -maxBandEdge max}
}
}
RetLayers {
    mask0 {
        layers          active_opc1
    }
    mask1 {
        layers          active_opc2
    }
}
Security {
    OpticalModels 11
    TargetLayer   11
    PvBands       11
    ResistModels 11
    ProcessCorrection 11
    RetLayers 11
}
VisibleLayers{}
    opcInLayers{}
    tags2boxes{}
}
-optimizerInfo {}

```

}

Example 9: PDK Non-inline Litho Model Definition with Contours

This example defines non-inline litho models in a PDK and generates contours:

```
LFD:::addPDKLayer { -name diffusion -pdk generic65 -processInfo {
    DrawnLayer active_IS
    IslandLayers {}
    inputDesignLayers {contact}
    OpcLayers {active_opc1 active_opc2}
    modelDir {./models}

    Lithomodel {
        lmod {lm_od}
    }
    imageOptions {
        img_opt_od_inline1 {
            layer \$layer1 visible mask 0
            litho_model \$litho_model
        }
        img_opt_od_inline2 {
            layer \$layer2 visible mask 0
            litho_model \$litho_model
        }
    }
    ProcessCorrection {}
    PvBands {
        subwindow1 {

Contour {-handle cdpa1_1 -operation simulation -arguments {-definition
{image \$image_options mask0 focus 0nm dose 0.98 bias 0.0 }} -
inputLayers {opcLayer(0) opcLayer(1)} -lithoModel {lmod} -imageOptions
{img_opt_od_inline1}}

Contour {-handle cdpa1_2 -operation simulation -arguments {-definition
{image \$image_options mask0 focus 50nm dose 1.0 bias 0.0 }} -
inputLayers {opcLayer(0) opcLayer(1)} -lithoModel {lmod} -imageOptions
{img_opt_od_inline1}}

Contour {-handle cdpa2_1 -operation simulation -arguments {-definition
{image \$image_options mask0 focus 0nm dose 0.98 bias 0.0 }} -
inputLayers {opcLayer(0) opcLayer(1)} -lithoModel {lmod} -imageOptions
{img_opt_od_inline2}}

Contour {-handle cdpa2_2 -operation simulation -arguments {-definition
{image \$image_options mask0 focus 50nm dose 1.0 bias 0.0 }} -
inputLayers {opcLayer(0) opcLayer(1)} -lithoModel {lmod} -imageOptions
{img_opt_od_inline2}}

Contour {-handle c1 -operation or -inputLayers {cdpa1_1 cdpa2_1}}
Contour {-handle c2 -operation or -inputLayers {cdpa1_2 cdpa2_2}}

Contour {-handle min -operation and -inputLayers {c1 c2 }}
Contour {-handle max -operation or -inputLayers {c1 c2 }}
Band {-handle default -subwindow 1 -minBandEdge min -maxBandEdge max}
}
```

CSG

Generates a *.csg* file that can be used as input to Calibre nmLVS.

Usage

Syntax 1

CSG

```
-poly poly_layer_name
{-polyCondition poly_condition_list [poly2_condition_list] |
 -polyContourHandle poly_contour_handle_name}
-active active_layer_name
{-activeCondition active_condition_list [active2_condition_list] |
 -activeContourHandle active_contour_handle_name}
[-outputFile file_name]
[{-database db_name | -layerOut csg_layer |
 -database db_name -layerOut csg_layer}]
-seedLayer seed_layer
[-overlayError error]
[-resolution res]
[-security yes]
-maxGate gate_size
-maxSearch search_dist
[-woffset woffset_value]
[-loffset loffset_value]
[-shift shift_list]
[-markerLayer layer_name]
[-checkName cName]
{-method default
 [-property {no | yes}]} |
{-method standard
 -model csg_lut_table
 -von on_voltage
 -voff off_voltage}
```

Syntax 2

CSG

```
-poly poly_layer_name
-active active_layer_name
-pdkCheckName check_template
-outputFile file_name
[additional_options]
```

Description

Contour simplification for gates (CSG) generates adjusted layout data for Calibre nmLVS device extraction. The command enables litho-aware transistor parameters extraction for

Calibre LFD simulations where you have identified transistors that are sensitive to process variation. The CSG function extracts transistor parameters such as gate length and gate width from non-rectangular contours and PV-bands created during Calibre LFD simulations.

The adjusted data consists of rectangular features that represent the gates as they print on the chip. This data typically takes the form of the CSG file. It can also take the form of a new layer in a layout database. The CSG function generates an output file that fills the gap between the Calibre LFD and Calibre nmLVS applications. The gap exists because LVS tools do not recognize the irregular contours generated by Calibre LFD.

This command can be run in three ways:

- When **-method default** is specified, the command uses the default Siemens EDA algorithm. This method calculates Wadj and Ladj by performing an averaging of the non-rectangular gate dimensions. This geometry-based algorithm calculates the Wadj and Ladj, based on the length of each transistor slice.
- When **-method standard** is specified, the command uses a Look-Up Table (model) to calculate the effective total current, to obtain the effective Wadj and Ladj, and other output properties.

Table 7-7. CSG Variability Output Properties

Output	Value
Ladj, Wadj	Adjusted Length and Width calculated.
Lmax, Wmax	Maximum Length and Width estimated from the maximum simulated contour.
Lmin, Wmin	Minimum Length and Width estimated from the minimum simulated contour.
Ldrawn, Wdrawn	Drawn Length and Width used as a reference to measure relative variabilities.
Ion	Operating current when transistor is on.
Ioff	Leakage current when transistor is off.
delta_Ion	Percentage change in gate current due to lithographic variability. The delta_Ion property is calculated as follows: $(Ion - Ion_drawn) / Ion_drawn * 100$
delta_Ioff	Percentage change in leakage current due to lithographic variability, calculated as follows: $(Ioff - Ioff_drawn) / Ioff_drawn * 100$

Table 7-7. CSG Variability Output Properties (cont.)

Output	Value
delta_L	Percentage change in gate length due to lithographic variability, calculated as follows: $(\text{Ladj} - \text{Ldrawn}) / \text{Ldrawn} * 100$
delta_W	Percentage change in gate width due to lithographic variability, calculated as follows: $(\text{Wadj} - \text{Wdrawn}) / \text{Wdrawn} * 100$

The **standard** method is based on a rigorous algorithm to calculate the Wadj and Ladj by summing the total current of the different slices. The algorithm takes into consideration short channel effects, the transistor leakage current, and the transistor type (PMOS or NMOS). This method requires a setup step to generate the look-up table representing the transistor current plots.

The look-up table contains transistor transfer characteristics in the form of drain current versus transistor width plots for a range of gate lengths and applied gate voltages, obtained from SPICE simulations. This data should be in the following tabular form:

Table 7-8. CSG Look-Up Table

Gate Length in nm (L)	Voltage (V)	Slope in um (m)	Intercept (b)
60.0	1.0	2.8355e-09	-5.5280e-11
60.1	1.0	2.3425e-09	-6.5890e-11

Here is how to perform the setup step to generate the look-up table. The required data includes transistor transfer characteristics in the form of drain current (I) versus transistor width (W) plots for a range of gate lengths (L) and applied gate voltages (V). The CSG function requires that the SPICE simulation data be saved in a tabular ASCII file in the [Table 7-8](#) format. CSG requires designating values for two voltages that are used to calculate the equivalent parameters. Defined as Von and Voff, these two voltages are used to calculate the on and off currents for each non-rectangular gate, and to determine the final equivalent L and W values.

The pseudocode of an algorithm to obtain the look-up table data is:

```

for each L ( Lmin >= Li <= Lmax; step DL)
    for each V (Vmin >= Vj <= Vmax; step DV)
        SpiceSimulator: calculate I(W1) and I(W2)
        Solve system of equations for m and b:
            I1 = m W1 + b
            I2 = m W2 + b

```

Where:

- W1 and W2 — widths to approximate linear correlation of device width and current.
 - For 65 nm: W1 and W2 suggested values are 0.1 u and 1.0 u

- For 45 nm: W1 and W2 suggested values are 0.1 u and 0.5 u
- For 32 nm: W1 and W2 suggested values are 0.1 u and 0.3 u
- L1, L2, and DL — values that define the range of L values (in nm), L1 and L2, and the step DL
 - For 65 nm: L1, L2, and DL suggested values are 52 nm, 110 nm, and 0.1nm
 - For 45 nm: L1, L2, and DL suggested values are 36 nm, 80 nm, and 0.1 nm
 - For 32 nm: L1, L2, and DL suggested values are 25 nm, 56 nm, and 0.1 nm
- m — slope of the transfer characteristics curve
- b — off current at gate-source voltage of 0.0

You must switch off any SPICE parameters that handle the mask/etch process while doing this calibration step to avoid counting the litho effects twice when using Ladj and Wadj in a SPICE simulation. In Berkeley Short-channel IGFET Models (BSIM), these parameters are LX and WX, and they have a default value of 0.

The calculated Wadj and Ladj can be used during both timing analysis and current leakage analysis. Moreover the **standard** method outputs the effective Ion and Ioff of the transistor without SPICE simulation.

If used with a PDK, this command calls one of the LFD::CSG checks defined in the PDK and runs it for the specified layers.

The resulting .csg file describes the “adjusted” layout which contains rectangular features that represent the gates as they print on the chip.

Note

 The CSG command can extract transistor parameters for tapper gates (gate rectangles containing a jog).

Arguments

- **-poly poly_layer_name**

Required keyword and argument specifying the name of the layer containing the poly portion of the gates under investigation.

- **-polyCondition poly_condition_list [poly2_condition_list]**

Required keyword and argument defining the process condition to be evaluated for the poly layer.

Note

 In one LFD::CSG call, you can specify either **-polyCondition** or **-polyContourHandle** but not both.

The ***poly_condition_list*** must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{*optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]*}

These values define a process condition that is one of the conditions evaluated by the LFD::PVband command for the poly layer.

Alternatively the ***poly_condition_list*** must be an ordered list with 2 elements defining the edges of a PV-band which was registered using the [RegisterBands](#) command. The first element can be either “max” or “min” (to define the max or min displacement for the PV-band), and the second element is an integer defining the subwindow of interest. When using **-polyCondition** with double patterning, you do not need to specify a *poly2_condition_list*. For example, specifying {max 1} takes into account both exposure processes when getting the contour.

The optional *poly2_condition_list* defines the process condition for the second exposure of a double exposure process. It must follow the same conventions as ***poly_condition_list***.

- **-polyContourHandle** *poly_contour_handle_name*

Required keyword and argument, specifying a handle name for a defined contour for the poly layer.

Note

 In one LFD::CSG call, you can specify either **-polyContourHandle** or **-polyCondition** but not both.

The handle name must be unique per layer. For example, the poly layer can have only one contour handle c1, but both the poly layer and the active layer may have a contour handle c1. For information on Calibre LFD contour handle generation, refer to the “[Customizable PV-Bands](#)” flow.

- **-active** *active_layer_name*

Required keyword and argument specifying the name of the layer containing the active portion of the gates under investigation.

- **-activeCondition** *active_condition_list* [*active2_condition_list*]

Required keyword and argument defining the process condition to be evaluated for the active layer.

Note

 In one LFD::CSG call, you can specify either **-activeCondition** or **-activeContourHandle** but not both.

The ***active_condition_list*** must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

{*optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]*}

These values define a process condition that is one of the conditions evaluated by the LFD::PVband command for the active layer.

Alternatively the *active_condition_list* must be an ordered list with 2 elements defining the edges of a PV-band which was registered using the LFD::RegisterBands command. The first element can be either “max” or “min” (to define the max or min displacement for the PV-band), and the second element is an integer defining the subwindow of interest. When using **-polyCondition** with double patterning, you do not need to specify an *active2_condition_list*.

The optional *active2_condition_list* defines the process condition for the second exposure of a double exposure process. It must follow the same conventions as *active_condition_list*.

- **-activeContourHandle *active_contour_handle_name***

Required keyword and argument, specifying a handle name for a defined contour for the active layer.

Note



In one LFD::CSG call, you can specify either **-activeContourHandle** or **-activeCondition** but not both.

The handle name must be unique per layer. For example, the poly layer can have only one contour handle *c1*, but both the poly layer and the active layer may have a contour handle *c1*. For information on Calibre LFD contour handle generation, refer to the “[Customizable PV-Bands](#)” flow.

- **-outputFile *file_name***

Optional keyword and argument specifying the pathname for the output file containing annotations with new L and W values. The default file type is .csg.

- **-method {default | standard}**

Required keyword and argument defining what mode to operate in:

- When **default** is specified, uses the default Siemens EDA algorithm.
- When **standard** is specified, uses the Look Up Table algorithm.

Note



If **standard** is specified, you must also use the **-model csg_lut_file**, **-von**, and **-voff** options.

- **-database *db_name***

Optional keyword and argument for use with “-property yes”, defining the database to which the output data is written.

- **-layerOut *csg_layer***

Optional keyword and argument for use with “-property yes”, instructing the tool to generate an output layer containing geometries on the seed layer, with properties attached.

- **-seedLayer *seed_layer***

Required keyword and argument specifying the name of a polygon layer containing the device recognition shapes. It is also called the gate layer. Connectivity need not be established on this layer.

- **-overlayError *error***

Optional keyword instructing the tool to attach the ladj_min, ladj_max, wadj_min, and wadj_max properties to the output layer of CSG.

- **-resolution *res***

Optional keyword and argument defining the resolution used for calculating the adjusted length and width values, expressed in microns. The *res* refers to a change in critical dimension (CD) that you consider to be significant with respect to gate behavior. Lower values of *res* lead to longer computation times.

- **-security yes**

Optional keyword and argument defining security privileges. The only acceptable input value is “yes”. If defined, the setup file is encrypted in the transcript.

- **-maxGate *gate_size***

Required keyword and argument defining the maximum gate size to be processed, expressed in microns

- **-maxSearch *search_dist***

Required keyword and argument defining the maximum distance from a **-seedLayer** edge to be searched when finding gate contours. This value is expressed in microns.

- **-woffset *woffset_value***

Optional keyword and argument used to offset the calculated W value by the specified amount. The default value is 0.

Note

 CSG extracts transistors parameters from non-rectangular post-etch contours using suitable etch models and PV-bands created during Calibre LFD simulation. If these etch models are missing, then a constant etch bias can be added to these contours through the -loffset and -woffset parameters.

- **-loffset *loffset_value***

Optional keyword and argument used to offset the calculated L value by the specified amount. The default value is 0.

- **-shift {*shift_list*}**

Optional keyword and list argument used to shift poly contours with respect to the active contours by the distances specified by the two values in the *shift_list*. The *shift_list* must be

a Tcl list containing two values, x0 and y0. The default for both values is 0. Both x0 and y0 must be expressed in user units. For example:

```
-shift {0.010 -0.010}
```

- **-markerLayer *layer_name***

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on this layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-checkName *cName***

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name. Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-property {no | yes}**

Optional keyword used to enable or disable the generation of the Ldrawn and Wdrawn properties. When “-property yes” is specified, you must also specify at least one of -database or -layerOut. This argument is only used with **-method default**.

Note

By default, properties are disabled, however the *ladj* and *wadj* properties (which report the actual values calculated by the operation) are attached by default to the output layer regardless of whether the -property option is set.

- **-von *on_voltage***

Required keyword and argument specifying the on state of the transistor model. This is a required argument if using **-method standard**.

- **-voff *off_voltage***

Required keyword and argument specifying the off state of the transistor model. This is a required argument if using **-method standard**.

- **-model *csg_lut_table***

Required keyword and argument defining the transistor model. The model definition can be a model pathname or an inline model. The model defines the behavior of the transistor current as a function of gate length, width, and applied voltage. This is a required argument if using **-method standard**.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining which adjusted layout data is written to the output file.

- *additional_options*

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK CSG command. Any additional options specified overrides the options defined within the PDK.

Examples

Example 1

In this CSG command example, condition arguments for poly and active layers with other specifications are used to generate an output file with adjusted layout data for Calibre nmLVS device extraction.

```
LFD:::CSG \
    -method standard \
    -poly poly \
    -polyCondition { max 1 } \
    -active active \
    -activeCondition { max 1 } \
    -seedLayer nch \
    -checkName csg_max_sub_0 \
    -database $lfdErrorDatabase \
    -outputFile csg_max_0.txt \
    -model ./NMOSLUT \
    -von 1.2 \
    -voff 0 \
    -resolution 0.0005 \
    -maxGate 1.0 \
    -maxSearch 0.06
```

Example 2

In this CSG command example, contour handle arguments for poly and active layers with other specifications are used to generate an output file with adjusted layout data for Calibre nmLVS device extraction.

```
LFD:::CSG \
    -poly poly \
    -polyContourHandle p1 \
    -active active \
    -activeContourHandle c1 \
    -outputFile csg_out_1.txt \
    -seedLayer nch \
    -method standard \
    -model ./input/CSG2.0/NMOSLUT2 \
    -von 0.5 \
    -voff 0.0s \
    -resolution 0.0005 \
    -maxGate 1.0 \
    -maxSearch 0.06 \
    -checkName CSG_1 \
    -database $lfdErrorDatabase
```

Related Topics

[About the CSG Function](#)

[Generating CSI Layers and CSG Files](#)

CSI

Generates a new layer that is a simplified version of the input contour layer.

Usage

Syntax 1

CSI

```
-layer input_layer_name
-layerCondition layer_condition_list
-deviation step
-layerOut csi_layer
[-markerLayer layer_name]
[-checkName cName]
[-priority cPriority]
[-comment comment_text]
[-database db_name]
```

Syntax 2

CSI

```
-layer input_layer_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

Description

Generates a CSI layer containing the simplified contours from the input layer.

If used with a PDK, this function calls one of the CSI checks defined in the PDK and runs it for the specified layer.

Note

 The LFD::CSI command cannot be used with LFD::RegisterBands.

Arguments

- **-layer *input_layer_name***

Required keyword and argument specifying the name of the layer for which the simplified contours are to be generated.

- **-layerCondition *layer_condition_list***

Required keyword and argument defining the process condition to be evaluated for the input layer. The *layer_condition_list* must be an ordered list, with 3, 4, 5, 6, 8, or 10 elements defining an explicit process condition. The list must be supplied as follows:

```
{optical1 dose1 size1 [resist1 etch1] [optical2 dose2 size2 [resist2 etch2]]}
```

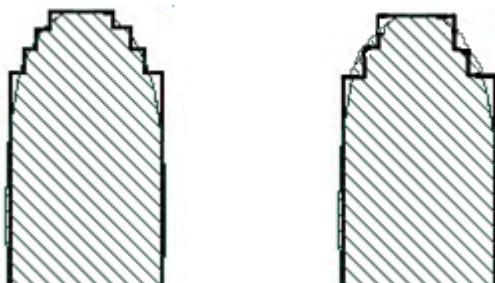
These values define a process condition that is one of the conditions evaluated by the **LFD::PVband** command for the input layer.

Alternatively the *layer_condition_list* must be an ordered list with two elements defining the edges of a PV-band which was registered using the **RegisterBands** command. The first element can be either “max” or “min” (to define the max or min displacement for the PV-band), and the second element is an integer defining the *subwindow* of interest.

- **-deviation step**

Required keyword and argument specifying the step size to use for the simplified contour. A smaller deviation step results in a finer simplification.

-deviation 0.5 -deviation 0.8



- **-layerOut csi_layer**

Required keyword and argument specifying a name for the layer to contain the simplified contours.

- **-markerLayer layer_name**

Optional keyword and argument used to constrain the check to those contours that lie within polygons on *layer_name*. The function ignores areas outside polygons on this layer.

A **-layerOut** layer or a derivation of a **-layerOut** layer should not be used as the input to **-markerLayer**, or a circular layer definition results.

- **-checkName cName**

Optional keyword and argument specifying the name to use for the check in the RDB. If not specified, the check in the RDB is assigned a system-generated name. Use this keyword to avoid name collisions when performing multiple checks of this type.

- **-priority cPriority**

Optional keyword and argument specifying a priority for this check.

- **-comment comment_text**

Optional keyword and argument used with **-database** for defining the comment text to be reported in the RDB if the check encounters a violation of this type. There is no default value for this argument.

The *comment_text* must be a single string, so comments containing spaces must be enclosed in braces. For example:

```
-comment {This is my comment.}
```

- **-database *db_name***

Optional keyword and argument defining the RDB to which the contours generated by the check are written.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining which adjusted contour to simplify and the settings to use for the simplification.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK CSI command. Any additional options specified overrides the options defined within the PDK.

Related Topics

[About the CSI Function](#)

[Generating CSI Layers and CSG Files](#)

Drawn2Contour

Generate empirically calibrated PV-bands using Square Directional Kernels (SDK) when access to the exact OPC recipe and models are not available.

Usage

Drawn2Contour

```
-layer drawn_layer_name
-subwindow number
-minBandFile {d2c_model | model_list}
-maxBandFile {d2c_model | model_list}
[-doublePatterning {split | trim}]
[-layerReference target_layer_name]
[-layerVisible visible_layer_list]
[-modelDir model_dir]
[-layerNegative negative_layer]
[-minBandTag {all | tag_name}]
[-maxBandTag {all | tag_name}]
[-security yes]
[-minBandSetup setup_file]
[-maxBandSetup setup_file]
```

Description

Generate empirically calibrated PV-bands using Square Directional Kernels (SDK) when access to the exact OPC recipe and models are not available.

You call Drawn2Contour for each PV-band subwindow. Each Drawn2Contour function call takes two Drawn-to-Contour (D2C) models, minBandModel and maxBandModel. In the case of asymmetric models, two D2C models must be provided to each of minBandModel and maxBandModel.

The function generates a PV-band object for the input layer with the specified subwindow.

You can run Calibre LFD checks on the PV-bands generated with this function, except certain modes for checks that deal with specific contour description (that is, [CaptureContour](#), [CSG](#), and [CSI](#)).

Arguments

- **-layer *drawn_layer_name***

Required keyword and argument specifying the name of the drawn layer.

- **-subwindow *number***

Required keyword and argument defining the severity of the PV-band to be generated.

- **-minBandFile {*d2c_model* | *model_list*}**

Required keyword and argument defining the D2C model to be used in generating the minimum PV-band contour. In case of asymmetric models, vertical and horizontal models are defined using this argument, using a list of two D2C models. The vertical and horizontal models are specified in order.

For example:

```
-minBandFile { vertical_d2c.mod horizontal_d2c.mod }
```

- **-maxBandFile {*d2c_model* | *model_list*}**

Required keyword and argument defining the D2C model to be used in generating the maximum PV-band contour. In case of asymmetric models, vertical and horizontal models are defined using this argument, using a list of two D2C models. The vertical and horizontal models are specified in order.

- **-doublePatterning {split | trim}**

Optional keyword and argument defining the type of processing to perform when specifying two mask layers and two sets of exposure settings.

The “-doublePatterning split” option refers to pitch-splitting techniques, and the “-doublePatterning trim” option refers to sacrificial printing-assist features, which 32 nm methods often use in favor of pitch-splitting.

- **-layerReference *target_layer_name***

Optional keyword and argument defining the target layer name.

- **-layerVisible *visible_layer_list***

Optional keyword and argument defining the list of input visible layers.

- **-modelDir *model_dir***

Optional keyword and argument defining the parent directory which contains the optical and resist models to be used for simulation. The default value for this argument is the value defined by LFD::[SimConfig](#) -modelpath.

- **-layerNegative *negative_layer***

Optional keyword and argument specifying the name of a negative layer, such as negative assist features. This layer is passed to OPCpro setupfile as an island layer. Island layers define areas of special interest or requiring special treatment.

- **-minBandTag {all | *tag_name*}**

Optional keyword and argument used to enable you to pass an input tag to the offset model. This allows you to manually adjust displacement in critical areas that are not captured correctly in the minimum PV-band model. The default argument is “all”.

- **-maxBandTag {all | tag_name}**

Optional keyword and argument used to enable you to pass an input tag to the offset model. This allows you to manually adjust displacement in critical areas that are not captured correctly in the maximum PV-band model. The default argument is “all”.

- **-security yes**

Optional argument defining security privileges. The only acceptable input value is “yes”. If defined, you are not able to write D2C models to SVRF.

- **-minBandSetup *setup_file***

Optional keyword and argument used to specify the Calibre ORC (Optical and Process Rule Checking) setup file to generate the coarse layers that are input to the Calibre OPCverify lfd_dtc (Drawn-to-Contour) command. The setup file can be defined inline or using a pathname. If not defined, a default setup file is used.

Input setup file can have the following place holders that are substituted: gridSize, maxDistance, largeFrag, minSpace, smallFrag, square, kerngrid, tileSize.

- **-maxBandSetup *setup_file***

Optional keyword and argument used to specify the Calibre ORC setup file to generate the coarse layers that are input to the Calibre OPCverify lfd_dtc (Drawn-to-Contour) command. The setup file can be defined inline or using a pathname. If not defined, a default setup file is used.

Input setup file can have the following placeholders that are substituted: gridSize, maxDistance, largeFrag, minSpace, smallFrag, square, kerngrid, tileSize.

Examples

Example 1

```
LFD::Drawn2Contour \
    -layer target_M1 \
    -subwindow 1 \
    -minBandFile DTC.fine.min.mod \
    -maxBandFile DTC.fine.max.mod \
    -layerNegative negative_M1 \
    -modelDir ./input/models
```

Example 2

This example uses Drawn2Contour within a PDK:

```
LFD::addPDKCheck {
    ...
    -processInfo {
        -security {
            Drawn2Contour 0X
        }
    }
}
```

And the following is defined in the TVF code:

```
LFD::LoadPDK example45nm.pdk
LFD::RegisterLayer -pdk ex45nm -Designlayer METAL1 -Processlayer pdklyr_m1
# (pdklyr_m1 layer is defined in example45nm.pdk)
LFD::Drawn2Contour \
    -layer METAL1 \
    -subwindow 1 \
    -security yes
```

End

Command used to signal the end of the Calibre LFD block.

Usage

End

Description

Closing command used to signal the end of the Calibre LFD block. This command must be placed after the last call to other Calibre LFD functions.

End is not for use in a PDK, and it must be paired with [Begin](#).

Arguments

None

Examples

```
// Begin Calibre LFD Block
LFD::Begin
    LFD::PVband ...
    LFD::MinWidthCheck ...
    ..
    LFD::CaptureContour ...
    ...
LFD::End
// End Calibre LFD Block
```

FrameCellOptimizer

Pre-processing that reduces the area to be analyzed by removing those areas that do not require processing by Calibre LFD because they have already been checked or because they are IP that the designer cannot modify.

Usage

Syntax 1

```
FrameCellOptimizer
  -layer input_layer_name
  -layerOut return_layer_name
  [-layerOut2 return_layer_name2]
  -halo halo_um
  [-minWidth width]
  -cells cell_names_list
```

Syntax 2

```
FrameCellOptimizer
  -layer input_layer_name
  -cells cell_names_list
```

Description

Analyzes the specified layer with respect to a set of cells that do not require processing by Calibre LFD, such as those that have already been checked or cells the designer cannot modify. This command provides a litho-aware filtering of cell data that takes into account any intruding polygons that exist outside of the cell. The command returns a derived polygon layer identifying areas requiring further investigation.

The function analyzes all placements of the specified cell or cells within the design. The function then characterizes the possible lithographic impact of those cell placements on the printability of neighboring cells or features. The portions of each cell placement that have little or no lithographic impact on their neighbors are marked as not requiring further investigation. Finally, results are negated from the extent of the input layer, generating a final layer region of interest where subsequent processing is to occur.

Using this command reduces the amount of area that is actually processed, which improves performance. Examples of cells that to consider using this function with include:

- Cells that are already Calibre LFD-clean.
- Cells with different design rules, such as SRAM.
- IP that you cannot modify.

If used with a PDK, this function calls the FrameCellOptimizer defined in the PDK to improve performance.

Arguments

- **-layer *input_layer_name***

Required keyword and argument specifying the name of the layer containing the design data under investigation. The *input_layer_name* can be an original layer or layer set, or a derived polygon layer.

- **-layerOut *return_layer_name***

Required keyword and argument specifying the name to assign to the primary output layer created by this operation. This return layer contains polygon regions defining the portions of the *input_layer_name* requiring further processing.

- **-layerOut2 *return_layer_name2***

Optional keyword and argument specifying the name to assign to the secondary output layer created by this operation. This layer contains original design data that requires further processing. That is, the layer contains the portions of the *input_layer_name* that are inside of the polygon regions as defined by the **-layerOut *return_layer_name***.

- **-halo *halo_um***

Required keyword and floating-point number, specified in user units, defining the distance the operation must investigate to identify the lithographic impact on each feature's neighbors. This value is equivalent to the optical diameter used during lithographic simulations.

- **-minWidth *width***

Optional keyword and argument used to turn on polygon sliver removal of design data written to *return_layer_name2*. The *width* defines minimum width criteria polygon data must meet if the data is to be written to *return_layer_name2*. Any polygons with a width less than this value are removed on the return layer.

The *width* value is a floating point number specified in user units. It is set 1-2 nm below the specified layer's minimum DRC requirement.

- **-cells *cell_names_list***

Required keyword and list of cell names that identify existing cells within the database which do not require Calibre LFD evaluation. The names in the *cell_names_list* can contain the wildcard character (*) which is used as defined by Calibre. The *cell_names_list* list must be enclosed in braces ({}).

GenerateHints

Runs Calibre LFD with MBH and generates a hints file for fixing lithographic hotspots.

Usage

GenerateHints

```
-inputRDB pathname_RDB_file
-errorMapFile pathname_error_map
-haloSize halo_size
-outputHintsFile pathname_hints_RDB_file
{-includeChecks list_of_check_names / -excludeChecks list_of_check_names}
[-LFDRuleFile pathname_LFD_kit]
[-inputLayersList input_layer_names]
[-simHaloSize sim_halo_size]
[-hintsNum hints_num]
[-validatedHintsNum valid_hints_num]
[-dfmDB pathname_generated_dfm_DB]
[-validateHints {yes | no}]
[-movementStep movement_step]
[-tempFilesDir pathname_temp_filesdir]
[-disableVictimEdgesHints {yes | no}]
[-disableGroupEdgesHints {yes | no}]
[-magnify magnify_value]
[-shrinkFactor shrink_factor_value]
[-hsd {yes | no}]
[-remoteNames machine_names]
[-envvarFile pathname_envvar_file]
[-parallelProcessesNum parallel_processes_num]
```

Description

Performs a litho-aware analysis by running Calibre LFD with MBH functionality. The input is a generated Calibre LFD RDB file with hotspot error markers.

An MBH run explores the local regions around the error markers and generates a hint or suggested repair for each error marker. In validation mode, the MBH analysis validates the hints by incrementally running Calibre LFD on the layout file for each hint to verify the hint fixes the error. The designer can use the suggested repair in the hint to fix the layout.

Arguments

- **-inputRDB *pathname_RDB_file***

Required keyword and path to an RDB file with error markers generated from a previously completed Calibre LFD run.

- **-errorMapFile *pathname_error_map***

Required keyword and path to an error map, also called a map file, that maps RDB check names to original layer names (non-derived layers). The error map file is also used to attach validation layer names to the RDB check names. Check names defined in the error map file and not in the RDB file are skipped, and a warning is displayed. See “[Sample MBH Map File](#)” on page 135.

- **-haloSize *halo_size***

Required keyword and floating-point number in microns specifying a halo region around the error marker. This region is also checked for any new error markers, after the Calibre LFD validation step. The default value is one micron.

- **-outputHintsFile *pathname_hints_RDB_file***

Required keyword and argument defining the path of the generated RDB file to which hints are written. By default, this location is the current working directory.

- **-includeChecks *list_of_check_names* | -excludeChecks *list_of_check_names***

Optional keyword and list argument specifying the checks to be included or excluded from the input RDB file for running MBH. You can specify either -includeChecks or -excludeChecks, but not both. If neither of these keywords and list arguments is specified, the default behavior is to include all the checks from the input RDB file. The use of wildcard characters (*) and (?) in the check names is supported. An empty list is not allowed. Enclose the *list_of_check_names* in braces ({}).

For example:

- -includeChecks {MWC_M1 MSC_M1} — Includes only the listed checks.
- -excludeChecks {MWC_M1 MSC_M1} — Excludes only the listed checks.

Do not use -excludeChecks simultaneously with -includeChecks, or an error message results.

- **-LFDRuleFile *pathname_LFD_kit***

Optional keyword and argument specifying the Calibre LFD kit or rule file used for generating the required layers for validation. This file sources other files and can include the [LoadPDK](#) command. Using this argument results in the MBH analysis running in validation mode.

- **-inputLayersList *input_layer_names***

Optional keyword and argument specifying the list of input layers used for the MBH TVF rule file if running without validation mode. Specify original layers in the list, otherwise the tool generates an error. You can specify the original layers, marker layers, and layers passed to the OPC recipe. These layers are copied to the generated DFM database.

If two original layers are mapped to the same GDS number, only the first mapped layer is copied into the DFM DB. An error is generated and the run aborts when the second mapped layer is referenced. For example, if POLY and POLY_COPY are both mapped to the same

GDS number (layer 10), references to POLY are handled properly. References to POLY_COPY generate an error, and the run aborts.

The -inputLayersList keyword is required if the MBH TVF rule file used to run MBH is encrypted.

- **-simHaloSize *sim_halo_size***

Optional keyword and floating-point number in microns specifying the simulation region around the error marker where the hint is evaluated. The Calibre LFD hints validation runs on this region to validate the hints. The default value is one micron.

Note

 Input to -simHaloSize should be greater than **-haloSize** by a value greater than the optical diameter used during lithographic simulations to ensure accurate simulation.

- **-hintsNum *hints_num***

Optional keyword argument that specifies the maximum number of hints to generate. The number of hints in the hints RDB file is less than or equal to hintsNum. The number is a positive integer.

- **-validatedHintsNum *valid_hints_num***

Optional keyword and argument that specifies the upper limit of hints to validate. Can be used to limit runtime in validation mode. The number is a positive integer less than -hintsNum.

- **-dfmDB *pathname_generated_dfm_DB***

Optional keyword and argument specifying the directory to store the DFM generated database used by the Calibre® YieldServer engine. The default location is *./out.db* if you do not specify a path.

- **-validateHints {yes | no}**

Optional keyword and argument used to control the MBH validation mode. Hints are automatically validated when you specify -LFDRuleFile. If hints are not validated, the -inputLayersList argument must be specified.

- **-movementStep *movement_step***

Optional keyword and argument in microns specifying how far to attempt movement of candidate edges to generate hints. Specify a value greater than 0.0 and less than 0.4. The range is enforced by the parser. The default value is 0.03 microns.

- **-tempFilesDir *pathname_temp_filesdir***

Optional keyword and directory name that specifies a temporary location for storing files created during simulation and hint generation. It is useful if you start more than one run from the same working directory. The default location is the current working directory.

- **-disableVictimEdgesHints {yes | no}**

Optional keyword and argument used to disable the generation of hints that move edges adjacent to the error marker. To turn this feature on use “yes”; to turn it off, use “no”. The default value is “no”; that is, a suggested fix may move edges adjacent to the marker.

- **-disableGroupEdgesHints {yes | no}**

Optional keyword and argument that specifies whether hints are generated using groups of edges or only single edges. To turn this feature on use “yes”; to turn it off, use “no”. The default value is “no”; that is, a suggested fix may involve more than one edge.

- **-magnify *magnify_value***

Optional keyword and floating-point number argument used when the SVRF command LAYOUT MAGNIFY is in the MBH TVF rule file. The *magnify_value* argument should have the same value of magnification. The default value is “1.0” (no magnification).

- **-shrinkFactor *shrink_factor_value***

Optional keyword and argument defining a shrink factor to support HSD kits. If a shrink factor value is passed to the SVRF command LAYOUT MAGNIFY, it should be passed to the LFD::GenerateHints command using this option. This value is a floating-point number.

- **-hsd {yes | no}**

Optional keyword and argument required when using an HSD validation kit. To enable this feature use “yes”. The default value is “no” (disable).

If -hsd “yes” is specified, the same -shrinkFactor or -magnify value used in the HSD validation kit needs to be used in the MBH TVF rule file. The magnify information also needs to be passed to the LFD::GenerateHints command.

- **-remoteNames *machine_names***

Optional keyword and argument instructing the command to run the generation of hints on the list of remote machines named by the argument.

The -envvarFile should be specified when -remoteNames is specified, or a warning is issued.

Two syntax options are supported:

The number of CPUs per machine is not specified:

- **-remoteNames {*machine_1 machine_2 machine_3...machine_N*}**.

The argument -parallelProcessesNum can be used with this syntax option.

The number of CPUs per machine is specified:

- **-remoteNames {{*machine_1 N1*} {*machine_2 N2*} {*machine_3 N3*}}**.

The argument -parallelProcessesNum cannot be used with this syntax option.

If the number of CPUs is specified, an error message is generated “Error: -parallelProcessesNum cannot be used when the numbers of CPUs/Machine are specified.”

For remote runs, if a host is not available, the job is added to the queue of pending jobs. When a CPU finishes an assigned job, a job from the queue is re-submitted. If the job fails to connect to a host again, the tool displays an error message and exits the run.

Note

 This error handling behavior is for connection problems with the remote hosts. If a job is submitted successfully and then fails to complete, it will not be re-submitted. In this case, the remaining jobs exit and an error message is displayed.

- **-envvarFile** *pathname_envvar_file*

Optional keyword and argument that should be used when -remoteNames is used. When a remote shell is opened to execute Calibre, all the environment variables used to run Calibre on the primary host must be exported. Include all environment variables that start with CALIBRE_ or MGC_.

The contents of this file should be in the csh shell format. For example:

```
#!/bin/csh
setenv CALIBRE_ENV_VAR yes
setenv RULE_FILE_VAR 1
```

Tip

 Include all of the environment variables in one file and source it in the batch run script. Then specify the same file as the argument to -envvarFile.

- **-parallelProcessesNum** *parallel_processes_num*

Optional keyword and argument that specifies the total number of available CPUs. Hotspots are divided equally among CPUs to run in parallel.

For example, -parallelProcessesNum 100 with 200 hotspots means that each CPU will process only two hotspots serially.

Examples

This example shows commands used in the MBH TVF rule file to run the generation of hints in validation mode.

```
LFD::GenerateHints \
    -inputRDB ./lfd_Errors.rdb \
    -errorMapFile map \
    -movementStep 0.05 \
    -haloSize 0.5 \
    -simHaloSize 1 \
    -hintsNum 5 \
    -dfmDB dfm.db \
    -outputHintsFile ./r2.hints \
    -LFDRuleFile rules \
    -validatedHintsNum 5
```

Related Topics

- [About Model-Based Hints \(MBH\)](#)
- [Sample MBH TVF Rule File](#)
- [Sample MBH Map File](#)
- [Running Calibre LFD with MBH](#)

GetMacroLayers

Extracts layers from the process-correction macro.

Usage

GetMacroLayers

```
-layer data_layer_name
-macroLayer original_macro_layer_list
-layerOut return_layer_list
```

Description

Extracts one or more layers from the process-correction macro and allows you to use these layers to debug the Calibre LFD flow. This function can be called more than once (with different **-macroLayer** and **-layerOut** arguments) for the same layer.

The macro must be defined using LFD::Macro (see the example below), and the security of the macro must not be “00”.

Arguments

- **-layer *data_layer_name***

Required keyword and argument specifying the name of the layer containing the data under investigation. The *data_layer_name* must be a registered layer using LFD::RegisterLayer.

- **-macroLayer *original_macro_layer_list***

Required keyword and list argument specifying the list of SVRF names of the layer output inside the Macro. The *original_macro_layer_list* is a Tcl list of such names. This layer cannot be one of the *data_layer_name* or *return_layer_list* layers.

- **-layerOut *return_layer_list***

Required keyword and list argument specifying the names to assign to the layers output by this operation. The *return_layer_list* is a Tcl list of such names.

Examples

In the PDK:

```
LFD::Macro name MX_MACRO -pdk LFD_DEMOPDK -pdkLayer PDKLYR_MX -recipe {
...
SOME_LAYER = SOME_OPERATION
...
}
```

In the rule file:

```
LFD::RegisterLayer -pdk LFD_DEMOPDK -ProcessLayer PDKLYR_MX \
    -Designlayer M3
LFD::GetMacroLayers -layer M3 -macroLayer SOME_LAYER \
    -layerOut M3_SOMELAYER_DEBUG
LFD::MinWidthCheck -pdkCheckName MX_MWC -layer M3 -checkName M3_MWC \
    -database lfd_errors.rdb
tvf::RULECHECK M3_SOMELAYER_DEBUG {COPY M3_SOMELAYER_DEBUG}
tvf::DRC RULE CHECK MAP M3_SOMELAYER_DEBUG OASIS 100 'debug.oas'
```

GetOPCInputLayers

Returns intermediate OPC layers.

Usage

GetOPCInputLayers
-layer *data_layer_name*
-layerOut *return_layer_list*

Description

Returns intermediate OPC layers (OpcInLayers) to a list of output layers defined by the **-layerOut** option.

The PDK security for OpcInLayers must be read-enabled.

Arguments

- **-layer *data_layer_name***

Required keyword and argument specifying the name of the layer containing the data under investigation. The *data_layer_name* can be an original layer or layer set, or a derived polygon layer.

- **-layerOut *return_layer_list***

Required keyword and list argument specifying the names to assign to the layers output by this operation. The *return_layer_list* is a Tcl list of such names. The list must contain the same number of layers as in the OpcInLayers call in the PDK file, however with new layer names.

Examples

PDK:

```
DrawnLayer pc
Security {PvBands 11 ProcessCorrection 11}
OpcInLayers {etch_target opc_target sraf}
```

TVF:

```
LFD::GetOPCInputLayers -layer pc \
    -layerOut {etch_target_out opc_target_out sraf_out}
```

GetOPCLayers

Returns OPC layers and visible layers.

Usage

GetOPCLayers

```
-layer data_layer_name
[-layerOut return_layer_names]
-database db_name
```

Description

Returns OPC and visible layers to a list of output layers defined by the -layerOut option, an RDB file, or an OASIS database.

The read security option for LFD::ProcessCorrection in the PDK must be enabled.

Arguments

- **-layer *data_layer_name***

Required keyword and argument specifying the name of the layer containing the data under investigation. The *data_layer_name* can be an original layer or layer set, or a derived polygon layer.

- **-layerOut *return_layer_names***

Optional keyword and argument specifying a name to assign to the layers output by this operation. If more than one output layer is supplied, *return_layer_names* is a Tcl list of such names, and the length of this list must match what is specified in the PDK.

- **-database *db_name***

Required keyword and argument defining the RDB or OASIS file to which results are written. The format must be one of the following:

- **-database *rdb_file_name***
- **-database OASIS *layer_number* [*layer_data_type*] *OASIS_file_name***

Note

 You must indicate where output are written by specifying **-database** or **-layerOut** or both.

Examples

Sends the OPC layer output derived from within a macro definition to an OASIS file. Outputs the following layer found in the OPC recipe embedded within a macro:

```
mx.opc = LITHO DENSEOPC FILE opc_setup target sraf MAP mx.opc.map
m1.opc = SIZE mx.opc UNDEROVER BY 0.001
```

PDK:

Add to OPC output layers in LFD::[addPDKLayer](#) call:

```
OpcLayers {m1.opc mx.opc}
```

Add to Calibre LFD Macro statement:

```
LFD::Macro \
  -name {mx_recipe} \
  -inputLayers {
    target sraf
    _target_ _sraf_
  } \
  -outputLayers {mx.opc m1.opc} \
  -recipe { ... }
```

TVF:

```
LFD::GetOPCLayers -layer <registered_designlayer> \
  -layerOut {my.mx.opc my.m1.opc}
tvf::RULECHECK m1.opc.raw {copy my.m1.opc}
```

GetRetargetLayers

Returns retarget layers.

Usage

GetRetargetLayers

```
-layer data_layer_name
-layerOut return_layer_names
```

Description

Returns retarget layers to a list of output layers defined by the **-layerOut** option. Retargeting involves modifying dimensions of polygons slightly to overcome dimension-changing effects of manufacturing. This is also known as biasing, however it should not be confused with mask biasing, which is variation in the manufactured mask.

The PDK security for the retarget layer must be read-enabled.

Arguments

- **-layer *data_layer_name***

Required keyword and argument specifying the name of the layer containing the data under investigation. The *data_layer_name* can be an original layer or layer set, or a derived polygon layer.

- **-layerOut *return_layer_names***

Required keyword and argument specifying the name to assign to the layers output by this operation. If more than one output layer is supplied, *return_layer_names* is a Tcl list of such names, and the length of this list must match what is specified in the PDK.

Examples

PDK:

```
DrawnLayer ol
Security {PvBands 11 ProcessCorrection 11}
OpcInLayers {b20 b22}
```

TVF:

```
LFD:::GetRetargetLayers -layer ol -layerOut {b20_out b22_out}
```

GetVisibleLayers

Returns visible layers.

Usage

GetVisibleLayers

-layer *data_layer_name*
-layerOut *return_layer_names*

Description

Returns visible layers to a list of output layers defined by the **-layerOut** option.

The PDK security for the visible layers must be read-enabled.

Arguments

- **-layer** *data_layer_name*

Required keyword and argument specifying the name of the layer containing the data under investigation. The *data_layer_name* can be an original layer or layer set, or a derived polygon layer.

- **-layerOut** *return_layer_names*

Required keyword and argument specifying the name to assign to the layers output by this operation. If more than output layer is supplied, *return_layer_names* must be a Tcl list, and the length of this list must match what is specified in the PDK.

Examples

PDK:

```
DrawnLayer dd
Security {PvBands 10 ProcessCorrection 10}
OpcInLayers {tlu tlo}
```

TVF:

```
LFD::GetVisibleLayers -layer dd -layerOut {tlu_out tlo_out}
```

IncrementalSelect

Performs intelligent clipping of one or more input layers based on your marker layers.

Usage

IncrementalSelect

```
-layer input_layer_name
-layerOut return_layer_name
[-checkRegions regions_layers | -blockingLayer block_layer]
-minWidth width
-halo halo_um
[-supportLayers supportLayers_list
-supportLayersOut supportLayersOut_list]
```

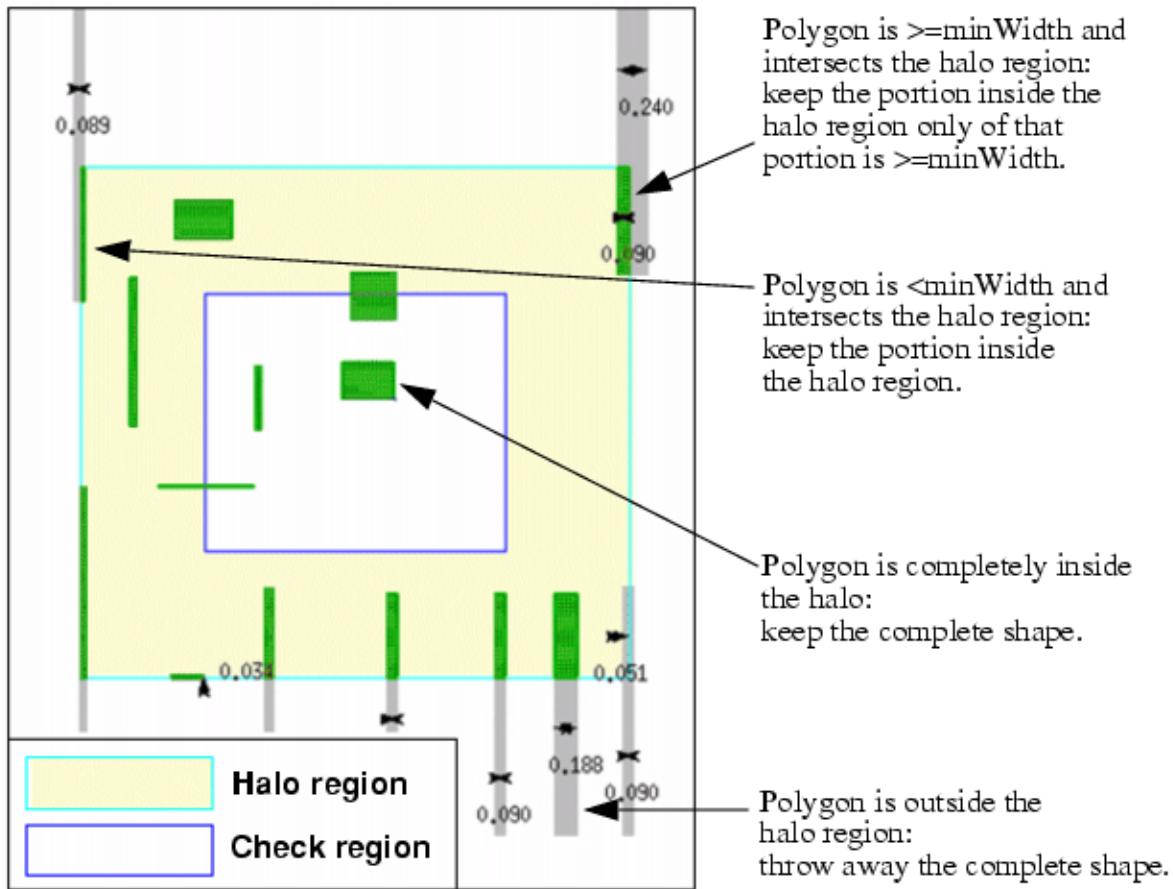
Description

Performs intelligent clipping of one or more input layers based on user-supplied marker layers. This function is used to ensure accurate processing of selected portions of the design by evaluating neighboring features to see whether or not they have any lithographic impact on features within the areas of interest. Those neighboring features that are identified as being lithographically significant are included in the output layers, along with the features within the areas of interest. The intelligent clipping is performed based on the following:

- The -checkRegions layers containing the areas of interest.
- The halo size defining a feature's lithographic impact distance.
- The minimum width of features that are of interest.

One use for this command is to support incremental Calibre LFD in which Calibre LFD is followed by design modifications and modified regions are resubmitted to Calibre LFD for additional processing.

Figure 7-38. IncrementalSelect



Arguments

- **-layer *input_layer_name***

Required keyword and argument specifying the name of the layer containing the design data under investigation. The *input_layer_name* can be an original layer or layer set, or a derived polygon layer.

- **-layerOut *return_layer_name***

Required keyword and argument specifying the name to assign to the primary output layer created by this operation. This return layer contains polygon regions defining the portions of the *input_layer_name* requiring further investigation.

- **-checkRegions *regions_layers***

Optional keyword and argument specifying the name of a layer containing regions that indicate where further processing is required. If more than one check region layer is supplied, *regions_layers* is a Tcl list of such names.

- **-blockingLayer** *block_layer*

Optional keyword and argument specifying the name of a layer containing regions where you do not want Calibre LFD simulation or checking to run.

Note



You must specify either -checkRegions, -blockingLayer, or both.

- **-supportLayers** *supportLayers_list*

Optional keyword and list argument used to specify one or more layers containing supplementary mask data (features that have a lithographic impact on the input layer). These can be original layers or derived polygon layers.

Note



You must also specify -supportLayersOut.

- **-supportLayersOut** *supportLayersOut_list*

Optional keyword and list of layer names that must be specified when -supportLayers is specified. The *supportLayersOut_list* is a list of names to be assigned to the layers returned to the system, which contain intelligently clipped geometries from the supportLayers.

The list of layer names must match the *layers_list* supplied as -supportLayers, both in terms of number and order.

- **-halo** *halo_um*

Required keyword and floating-point number, specified in user units, defining the distance the operation must investigate to identify the lithographic impact on each feature's neighbors. This value is equal to the maximum **halo_um** value used during lithographic simulations.

- **-minWidth** *width*

Required keyword and argument used to control polygon sliver removal of design data. The *width* defines the minimum size for a clipped portion of an original geometry at the external border of the halo region.

- If the original geometry at the external border of the halo region is drawn at a dimension less than the -minWidth value and it intersects the halo region, then the portion of the geometry inside the halo region is kept for future processing.
- If the original geometry at the external border of the halo region is drawn at a dimension greater than or equal to the -minWidth value and it intersects the halo region, then the portion inside the halo region is kept for future processing only if the dimension is greater than or equal to the -minWidth value.

The *width* value is a floating point number specified in user units. It is set 1-2 nm below the specified layer's minimum DRC requirement.

LayoutOptimizer

Defines problem areas based on geometric rules.

Usage

LayoutOptimizer

```
-layer input_layer_name
-macroName macro_name
[-inputLayers input_layers]
-layerOut output_marker_region
[-layerOut2 optional_outputs]
```

Description

LayoutOptimizer, used in conjunction with an LFD::Macro definition, can be used as a geometrically-based optimizer that identifies potentially problematic areas in the input layer.

This command searches through designs for arrays of cells to remove prior to simulation. An error occurs if *macro_name* is not defined with an LFD::Macro command, or if there is a mismatch between the number of layers needed by the macro and the layers specified to this command.

This command is for use with a Calibre LFD kit containing a PDK file.

Arguments

- **-layer *input_layer_name***

Required keyword and argument specifying the name of the layer containing the design data to pass to the macro. The *input_layer_name* can be an original layer or layer set, or a derived polygon layer.

- **-macroName *macro_name***

Required keyword and argument defining the name of the macro as defined in PDK using the LFD::Macro command.

- **-inputLayers *input_layers***

A optional keyword and argument defining the set of input layers in the case that the macro requires more than one input layer.

- **-layerOut *output_marker_region***

Required keyword and argument specifying the name of the layer that is created containing the marker region. This returned layer contains polygon regions defining the portions of the *input_layer_name* requiring further processing.

- **-layerOut2 *optional_outputs***

Optional keyword and argument specifying optional extra layers needed to be output due to the macro definition.

Examples

This example calls a macro in the PDK, and then identifies potential problematic areas. The following is defined in the PDK:

```
LFD::Macro -name LO_layerX \
    -inputLayers {layerX} -outputLayers {layerX_AOI} -recipe { \
        < custom recipe >
    }
```

And the following is defined in your TVF code:

```
LFD::LayoutOptimizer -layer M3i -macroName LO_layerX -layerOut M3_LO
LFD::IncrementalSelect -layer M3i -layerOut M3s -checkRegions M3_LO \
    -halo 1.0 -minWidth 0.045
LFD::LFDregion -layer M3s -region M3_LO -halo 1.0
LFD::RegisterLayer -DesignLayer M3s -ProcessLayer MX -pdk MYPDK
```

LFDregion

Defines regions within the layout, which can then be used to limit checking to those areas.

Usage

LFDregion

```
{-region region_layer_name | -blockingLayer block_layer_name}  
[-layer input_layer_name]  
-halo size
```

Description

Limits Calibre LFD checking in batch mode to specified regions within the layout. These regions apply to all PV-bands created, regardless of how many times you issue the LFD::PVband command.

When used, this command must be placed before the first call to the LFD::PVband command.

LFD::LFDregion cannot be defined in the PDK.

Arguments

- **-region *region_layer_name***

Required keyword and argument specifying a layer that identifies the regions to check. This layer can be either an original layer or a derived layer. It must contain one or more polygons that enclose the portions of the layout to be evaluated with Calibre LFD.

- **-blockingLayer *block_layer_name***

Required keyword and argument specifying the area where you do not want Calibre LFD simulation or checking to run. No error markers are showed in this area.

Note



If you do not define a **-region**, you must define a **-blockingLayer**. The region where Calibre LFD runs is the extent of the layer rather than the blocking layer.

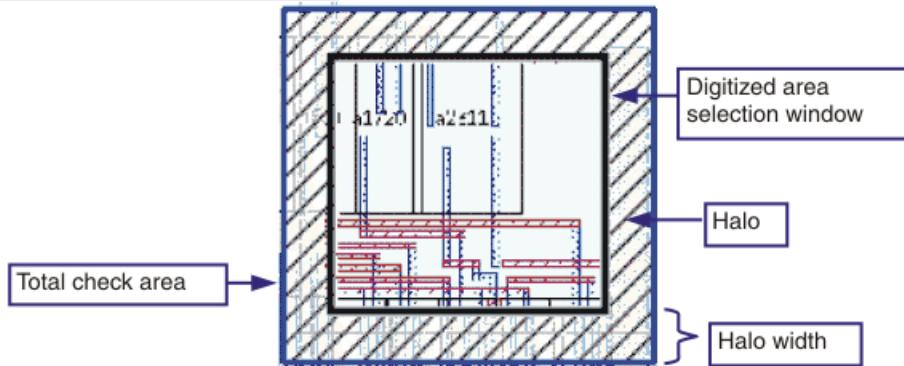
- **-layer *input_layer_name***

Optional keyword and argument associating the Calibre LFD region with a specific target layer. If not specified, the region applies to all layers. When using the -layer option, you can include the LFDregion command multiple times, defining a different region for each target layer. Otherwise, the command can be issued at most one time.

- **-halo *size***

Required keyword and argument defining the size of a halo, specified in microns. The halo is an area outside the area you selected, containing data that is factored into the calculations used to generate the PV-bands. The actual PV-bands extend only to the boundary of the region.

The halo is an area outside the area you selected, containing data that needs to be factored into the rule checks.



LoadPDK

Loads the specified PDK and makes it available to Calibre LFD command calls.

Usage

LoadPDK

```
path_name
[-name pdk_name]
[-version version]
```

Description

Loads the specified PDK and makes it available to Calibre LFD command calls.

Arguments

- *path_name*

Required specified pathname for the file containing the PDK.

- -name *pdk_name*

Optional keyword and argument assigning a name to the PDK being loaded. In most cases, PDK's are referenced using the name supplied in the [createPDK](#) statement within the PDK file. If this keyword is supplied, the name specified here overrides the name in the PDK file and is used to reference the PDK for the duration of the run.

- -version *version*

Optional keyword and argument for use when you have multiple versions of the same PDK available and need to define the version to load. The tool compares the version specified with this keyword to the version specified in the [createPDK](#) statement within the file. If the versions are not the same, the tool generates an error message and aborts. By default, no version checking is performed.

Examples

```
LFD::loadPDK r2.pdk -name pdk1
LFD::RegisterLayer -pdk pdk1 -Designlayer poly -Processlayer pdk_poly
LFD::MinWidthCheck -layer poly -pdkCheckName polyMWC \
    -subwindow 1 -database "lfd_Errors.rdb"
```

Macro

Defines a macro.

Usage

Macro

```
-name macro_name
[-description user_description]
[-security {00 | 01 | 10 | 11}]
{ -recipe svrf_statements | -tvfrecipe tvf_statements }
{ -pdk pdk_name -pdkLayer pdk_process_layer |
  -inputLayers inputLayers_list
  -outputLayers outputLayers_list }
```

Description

Macro is used to define a macro in a Calibre LFD kit. This function defines a macro with the name defined by **-name**. Macro has two modes of operation:

- **Fast mode Calibre LFD** — Recommended to be used in conjunction with [LayoutOptimizer](#) to apply the user-specific optimization recipe.
- **Process Correction (PDK)** — Is used with [addPDKLayer ProcessCorrection](#). To use Macro with **-pdk**, the PDK name is a valid created PDK and the layer is a PDK layer added in a previous LFD::[addPDKSTO](#) command. In this mode of operation, only **-recipe** is supported; **-tvfrecipe** is not supported.

Arguments

- **-name *macro_name***

Required keyword and argument defining the name of the macro.

- **-description *user_description***

Optional argument defining the user description of the macro.

- **-security {00 | 01 | 10 | 11}**

Optional argument defining read and write privileges. The input to security is two flags, the first for read security and the other for write security. This flag is only recognized while using a PDK.

- **-recipe *svrf_statements***

Required keyword and argument defining the SVRF statements that are implemented when the macro is called.

- **-tvfrecipe *tvf_statements***

Required keyword and argument defining the TVF statements that are implemented when the macro is called.

- **-pdk *pdk_name***
Required keyword and argument defining the name of the PDK.
- **-pdkLayer *pdk_process_layer***
Required keyword and argument defining the PDK process layer name.
- **-inputLayers *inputLayers_list***
Required keyword and argument defining the set of layer inputs to the macro.
- **-outputLayers *outputLayers_list***
Required keyword and argument defining the set of layer outputs from the macro.

Examples

This is a general purpose macro:

```
LFD::Macro -name macroName -inputLayers {inLayer1 inLayer2 inLayer3} \
             -outputLayers {outLayer1 outLayer2} -recipe { recipe statements }
```

MLDataGen

Samples and translates features for input to a machine learning training model in the preprocessing and data generation phase of the Calibre LFD deep neural network (DNN) flow.

Usage

```
MLDataGen
  -layer layer_name
    {-minWidth width | -minSpace space}
  -anchorSize size
  -anchorsSeparation separation
  -hotspotLayer layer_name
  -windowSize win_size
  -outputFile output_file_path
  -layerOut output_layer_name
  -metaDataFileOut filename
  [-markerLayer marker]
  {[-growHorizontallyBy value] [-growVerticallyBy value]}
  [-excludeLayer exclude_layer_name]
  [-orientation {yes | no}]
  [-rotateLayer90 {yes | no}]
```

Description

Samples and translates features on the input target layer in a feature extraction step and generates an output (CSV) file with feature information that can be used as input data to the machine learning model training phase of the Calibre LFD DNN flow.

During the data generation, the LFD::MLDataGen command creates the anchors, separates them into hotspot and non-hotspot anchors, and classifies them. Only the unique anchors are included in the data generation that is input to the model training phase of the flow.

The LFD::MLDataGen and LFD::MLOptimizer commands share some of the same parameters for minimum width and space, anchor separation and size, and window size. For accuracy, these parameter values must be consistent across the commands when used in the Calibre LFD DNN flow.

Specify values as non-negative floating-point numbers in user-units unless stated otherwise.

Arguments

- **-layer *layer_name***

Required keyword and argument specifying the input target layer name that you are sampling.

- **-minWidth *width***
Required keyword and value defining the width of polygons in the input layer to be anchored. The value should be defined such that all polygons with a width equal to or slightly larger than the minimum DRC constraint are sampled. This keyword cannot be used with the -minSpace keyword.
- **-minSpace *space***
Required keyword and value defining the spacing between polygons in the input layer to be anchored. The value should be defined such that all polygons with a spacing equal to or slightly larger than the minimum DRC constraint are sampled. This keyword cannot be used with the -minWidth keyword.
- **-anchorSize *size***
Required keyword and value defining the size of the output box representing the anchors. The recommended size is 0.0004 with units in microns.
- **-anchorsSeparation *separation***
Required keyword and value defining the separation distance (spacing) of the anchor points.
- **-hotspotLayer *layer_name***
Required keyword and argument specifying the name of the input hotspot layer.
- **-windowSize *win_size***
Required keyword and value defining the size of the window used for extracting the features around each anchor point.
- **-outputFile *output_file_path***
Required keyword and argument specifying the path of the output CSV file containing the generated data. The information from this file is input to the model training phase of the flow. A standard CSV file format is supported. See “[Preprocessing and Data Generation](#)” on page 587 for information on the CSV file.
- **-layerOut *output_layer_name***
Required keyword and argument specifying the output layer name with the generated anchor points.
- **-metaDataFileOut *filename***
Required keyword specifying to write all metadata related to the data and anchor generation to the specified file. This encoded file is input to the LFD::MLOptimizer and lfd_dnn_train commands to ensure consistent usage of the generated data across all machine learning commands (LFD::MLDataGen, lfd_dnn_train, and LFD::MLOptimizer). The filename parameter may be enclosed in a set of quotation marks (“ ”). If no path is specified, the file is output to the current working directory.
- **-markerLayer *marker***
Optional keyword and argument specifying an input marker layer that defines the regions of interest for data generation.

- **-growHorizontallyBy *value***

Optional keyword and value defining the grow value for the hotspot and exclude layers in the horizontal direction to create the marker layer defining the hotspot region.

This option must be used with the -growVerticallyBy option.

If the grow options are not specified, the input -hotspot_layer and -exclude_layer are considered as the marker layers for those regions.

Specify a value equal to the anchor separation or sampling step size in order to overlap with placed near anchors and to define these anchors as hotspots in the output file.

- **-growVerticallyBy *value***

Optional keyword and value defining the grow value for the hotspot and exclude layers in the vertical direction to create the marker layer defining the hotspot region.

This option must be used with the -growHorizontallyBy option.

If the grow options are not specified, the input -hotspot_layer and -exclude_layer are considered as the marker layers for those regions.

Specify a value equal to the anchor separation or sampling step size in order to overlap with placed near anchors and to define these anchors as hotspots in the output file.

- **-excludeLayer *exclude_layer_name***

Optional keyword and argument defining an input layer with regions that are excluded from the data generation.

- **-orientation {yes | no}**

Optional keyword and argument enabling or disabling the reporting of features of different orientations for each anchor point. The default is to report the orientations represented in the data generation output.

- **-rotateLayer90 {yes | no}**

Optional keyword and argument that enabling or disabling the rotation of the input target layer by 90 degrees. The default is no rotation.

Examples

This example uses the LFD::MLDataGen command with specified parameters to output an anchor layer, an encoded metadata file with data generation information, and a CSV file with feature information. The data generated from this command is input to the **lfd_dnn_train** utility for machine learning model training in the Calibre LFD DNN flow.

```
LFD::MLDataGen -layer M2_E1 \
-layersOut datagen_anchors \
-minSpace 0.100 \
-anchorsSeparation 0.100 \
-anchorSize 0.0004 \
-windowSize 1.4 \
-hotspotLayer hotspots \
-outputFile features.csv \
-metaDataFileOut "model.bin" \
-growHorizontallyBy 0.100 \
-growVerticallyBy 0.100 \
-markerLayer NHS_Marker
```

MLOptimizer

Performs hotspot prediction on a new design using a machine learning trained model in the prediction phase of the Calibre LFD deep neural network (DNN) flow.

Usage

```
MLOptimizer
-layer layer_name
{-metaDataFile filename |
 { {-minWidth width | -minSpace space }
 -anchorsSeparation separation
 -anchorSize size
 -windowSize win_size } }
-layerOut output_layer_name
-model model_path
```

Description

Predicts hotspot candidates on a target layer of a new layout using the machine learning trained model from the training phase of the Calibre LFD DNN flow.

The LFD::MLDataGen and LFD::MLOptimizer commands share some of the same parameters for minimum width and space, anchor separation and size, and window size. For accuracy, these parameter values must be consistent across the commands when used in the Calibre LFD DNN flow.

Specify values as positive floating-point numbers in user units unless stated otherwise.

Arguments

- **-layer *layer_name***

Required keyword and argument specifying the new input target layer you are sampling.

- **-metaDataFile *filename***

Required keyword specifying to read all metadata related to the data and anchor generation from the metadata file output by the LFD::MLDataGen command. Specifying this keyword ensures that the LFD::MLOptimizer command uses parameter values consistent with those used for data generation and training. When the -metaDataFile is specified, the discrete parameters (-minWidth, -minSpace, -anchorSize, -anchorsSeparation, and -windowSize) cannot be used. The encoded information for these parameters is loaded from the metadata file. This is the recommended method for running the Calibre LFD DNN flow. The filename parameter may be enclosed in quotation marks (""). If no path is specified, the file must be in the current working directory.

- **-minWidth *width***

Required keyword and value defining the width of polygons in the input layer to be anchored. The value should be defined such that all polygons with a width equal to or

slightly larger than the minimum DRC constraint are sampled. This keyword cannot be used with the -minSpace keyword or the -metaDataFile keyword.

- **-minSpace *space***

Required keyword and value defining the spacing between polygons in the input layer to be anchored. The value should be defined such that all polygons with a spacing equal to or slightly larger than the minimum DRC constraint are sampled. This keyword cannot be used with the -minWidth keyword or the -metaDataFile keyword.

- **-anchorsSeparation *separation***

Required keyword and value defining the separation distance (spacing) of the anchor points. This keyword cannot be used with the -metaDataFile keyword.

- **-anchorSize *size***

Required keyword and value defining the size of the output box representing the anchors. The recommended size is 0.0004 with units in microns. This keyword cannot be used with the -metaDataFile keyword.

- **-windowSize *win_size***

Required keyword and value defining the size of the window used for extracting the features around each anchor point. This keyword cannot be used with the -metaDataFile keyword.

- **-layerOut *output_layer_name***

Required keyword and argument specifying the name of the output layer with the predicted hotspot sites.

- **-model *model_path***

Required keyword and argument specifying the full path including the filename of the trained model used for the hotspot prediction. The trained model must be a single valid (.pb) file.

Examples

These examples use the LFD::MLOptimizer command with specified inputs that include a machine learning trained model. During the run, litho-simulation and Calibre DRC are performed. This command performs the prediction phase of the machine learning Calibre LFD DNN flow. The output is a hotspot prediction layer along with a Calibre DRC summary report and results database for analysis. The filenames and paths may be enclosed in sets of quotation marks ("").

Example 1

In this example, all meta data related to the data and anchor generation used during the LFD::MLOptimizer command is read from the encoded metadata file. This is the recommended specification to ensure consistent usage of the parameters and generated data across all machine learning commands (LFD::MLDataGen, lfd_dnn_train, and LFD::MLOptimizer).

```
LFD::MLOptimizer -layer M2_E1 \
    -layerOut Space_prediction \
    -model "/home/user/cpp_output_model/saved_frozen_model.pb" \
    -metaDataFile "inputs/model.bin"
```

Example 2

In this example, the parameters for anchor and data generation are specified with the LFD::MLOptimizer command. For accuracy, the values for these discrete parameters must be the same across all machine learning commands (LFD::MLDataGen, lfd_dnn_train, and LFD::MLOptimizer).

```
LFD::MLOptimizer -layer M2_E1 \
    -layerOut Space_prediction \
    -model "/home/user/cpp_output_model/saved_frozen_model.pb" \
    -minSpace 0.100 \
    -anchorsSeparation 0.100 \
    -anchorSize 0.0004 \
    -windowSize 1.4
```

OutputBands

Writes PV-band data to the specified database.

Usage

Syntax 1

OutputBands

```
-layer input_layer_name
-bandType {regular | all | absolute}
-subwindow expr_number
[-errorFilterSize size_filter]
[-overlay {N | S | E | W | NE | SE | SW | NW | C}]
[-dp_mask {1 | 2}]
[-bandHandle band_handle]
{ -database db_name | -layerOut return_layer_name
  | -database db_name -layerOut return_layer_name }
```

Syntax 2

OutputBands

```
-layer input_layer_name
-pdkCheckName check_template
-database db_name
[additional_options]
```

Description

Writes regular or absolute PV-band data to either a derived layer or the specified RDB. This command writes only full PV-bands. To write the inside or outside edge contours to the RDB, use the LFD::[CaptureContour](#) command.

If used with a PDK, this function calls the OutputBands defined in the PDK and runs it for the specified layer, writing check results to the specified database.

Note

 For PDK usage, this command is only available if the security options are set such that writing PV-bands and contours is allowed.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-bandType {regular | absolute | all}**

Required keyword and argument defining the type of PV-band data to write to the database:

- **regular** — Represents the maximum and minimum edge displacement.

- **absolute** — Represents the maximum and minimum deviation from the target layout. The regular PV-bands are always within the absolute PV-bands.
 - **all** — Both regular PV-bands and absolute PV-bands.
- **-subwindow *expr_number***
Required keyword and argument defining the process variation experiment to which this check applies. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the LFD::[PVband](#) command used to generate the PV-band data being checked. Thus, *expr_number* refers to an index to a list of experiments.
 - **-errorFilterSize *size_filter***
Optional keyword and argument used to control which portions of the PV-band are important enough to write to the output database.
 - This operation outputs only the PV-bands around the error markers generated for this layer within a window defined by the *size_filter*.
 - When not specified, this command writes the entire PV-band to the specified database.The *size_filter* must be a positive real number expressed in microns. If *size_filter* is less than 0.01, the tool issues an error message and aborts.
Only the error markers for checks performed prior to this command being issued are used in this filtering.
 - **-overlay {N | S | E | W | NE | SE | SW | NW | C}**
Optional keyword and argument that instructs to check for double-patterning overlay errors, of N, S, E, W, NE, SE, SW, NW, or C, describing the overlay error or shift that is applied to all process subwindows. The “C” value stands for the normal or center PV-band, which refers to a non-shifted PV-band.
You can only specify -overlay if it was used in the LFD::[PVband](#) call (for a double pattern LFD::PVband call) for that layer. If -overlay is specified in LFD::PVband, but not in the LFD::OutputBands command, then all PV-bands listed above are output by that LFD::OutputBands command.
 - **-dp_mask {1 | 2}**
Optional keyword and argument that outputs a single mask for a double-patterning technology, where the argument is set to 1 or 2. You can only use this option if the -doublePatterning option has been instantiated within the LFD::[addPDKLayer](#) or LFD::[PVband](#) statement.
 - **-bandHandle *band_handle***
Optional keyword and argument used to perform Calibre LFD checks on specific PV-band handles. This option is for use with the *band_handle* generated using the [Customizable PV-Bands](#) flow.

- **-database *db_name***

Required keyword and argument defining the RDB to which PV-band data are written. You must indicate where the violations are written by specifying **-database**, **-layerOut**, or both.

- **-layerOut *return_layer_name***

Required keyword and argument defining the name of a derived layer to which the PV-band data are written. This layer exists in memory and can be referenced in subsequent Calibre nmDRC operations. You must indicate where the violations are written by specifying **-database** or **-layerOut** or both. This option can not be used if **-bandType all** is set.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining which PV-bands are to be written to the database.

- *additional_options*

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK OutputBands command. Any additional options specified overrides the options defined within the PDK.

Examples

Example 1

This example writes regular PV-band data to a specified database:

```
LFD:::OutputBands -layer active -bandType all -subwindow 1 \
                    -database $lfdBandDatabase

LFD:::OutputBands -layer active -bandType all -subwindow 2 \
                    -database $lfdBandDatabase
```

Example 2

This example uses the **-overlay** option:

```
LFD:::PVband -layer contact_target \
              -layerRET {lfd_contact_mopc contact_sraf contact_patches} \
              -foreground {clear clear dark} -background {attenuated 0.06} \
              -doublePatterning split \
              -overlay 0.005 \
              -pixel 0.01 \
              -modelDir {./contact/models} -resistFile {contact.mod} \
              -opticalSpanList {{D0 D0 D0 D50 D50 D50} {D0 D0 D0 D75 D75 D75}} \
              -doseSpanList {{0.9800 1.0000 1.0200 0.9800 1.0000 1.0200} \
                            {0.9600 1.0000 1.0400 0.9600 1.0000 1.0400} }

LFD:::OutputBands -layer contact_target \
                  -bandType all -subwindow 1 \
                  -overlay NW \
                  -database $lfdBandDatabase
```

Example 3

This example uses OutputBands within a PDK:

```
LFD::addPDKCheck {  
    -name Active_Band_p1  
    -pdk lfd_demo  
    -type OutputBands  
    -security 11  
    -definition {  
        -bandType regular  
        -subwindow 2  
    }  
}
```

And the following is defined in your TVF code:

```
LFD::OutputBands -pdkCheckName Active_Band_p1 -layer active \  
    -layerOut Active_Band_p1
```

OverlayBand

Creates overlay PV-bands with two input contours.

Usage

OverlayBand

```
-handle handle_name
-layer input_layer_name
-subwindow process_window_number
-minBandEdge inner_contour_layer
-maxBandEdge outer_contour_layer
-shift shift_direction
-origBandHandle band_handle
```

Description

Creates overlay PV-bands with two input contours. Overlay PV-bands represent a shifted version of an original PV-band. For example, if band1 consists of the contours {c1 c2 c3}, a shifted version of this PV-band in the north direction consists of {c1_N c2_N c3_N}, where c1_N, c2_N and c3_N are the shifted versions of the contour in the north direction.

Overlay PV-bands are attached to an original PV-band. If you defined the eight shifted PV-bands for an original PV-band, the checks perform in the same manner they perform when the -overlay option is defined using the LFD::PVband command. In other words, the check is performed on the eight shifted PV-bands and the result is the logical OR of the results of the eight checks.

The overlay PV-band is the maximum and the minimum edges of the PV-band. The PV-band layer and the absolute PV-band layer are generated by Calibre LFD:

- OverlayBandLayer = NOT max_band_edge min_band_edge
- OverlayAbsBand layer = and layer BandLayer

The input “maximum” and “minimum” contours can be either contour handles defined earlier in the Calibre LFD kit, or be original layer names.

Note

 This command is part of the customizable LFD PV-bands interface, which uses the LFD::Contour, LFD::Band, and LFD::OverlayBand commands to provide you flexibility in generating simulation contours and PV-bands, and the ability to use your custom-generated contours with the [Calibre LFD Checks](#). See the [Customizable PV-Bands](#) section for additional information.

Arguments

- **-handle *handle_name***

Required keyword and argument specifying a PV-band handle which must be unique per layer. This argument along with the -layer argument gives the PV-band a unique name that can be used to reference the PV-band in further operations.

- **-layer *input_layer_name***

Required keyword and argument specifying the name of the design layer for this PV-band. The layer passed to this argument is a registered layer. If the layer is not registered, this command registers the layer. This option is not for use in a PDK.

- **-subwindow *process_window_number***

Required argument and integer, defining the severity of the generated PV-band. This option is not for use in a PDK.

- **-minBandEdge *inner_contour_layer***

Required keyword and argument defining the minimum PV-band edge of the overlay PV-band. The argument accepts either a contour handle generated previously during the run or a layer name (original or derived).

- **-maxBandEdge *outer_contour_layer***

Required keyword and argument defining the maximum PV-band edge of the overlay PV-band. The argument accepts either a contour handle generated previously during the run or a layer name (original or derived).

- **-shift *shift_direction***

Required argument defining the shift direction this overlay PV-band represents. Correct values accepted by this option are {N S W E NW NE SW SE}.

- **-origBandHandle *band_handle***

Required argument defining the original, non-shifted version of the PV-band.

Examples

This example generates simulation contours for layer Poly:

- Double patterning split
- Process window:
 - Optical Span List 1 {Poly1_50 Poly1_0 Poly1_50}
 - Dose Span List 1 {0.89 1.00 0.89}
 - Optical Span List 2 {Poly2_50 Poly2_0 Poly1_50}
 - Dose Span List 2 {0.89 1.00 0.89}
- Resist Model 1: poly1.mod

- Etch Model 1: poly1_etch.mod
- Resist Model 2: poly2.mod
- Etch Model 2: poly2_etch.mod
- RETLayers 1: poly_opc_mask1
- RETLayers 2: poly_opc_mask2
- Foreground 1: attenuated 0.06
- Foreground 2: attenuated 0.06
- Overlay: 0.01

In order to generate the PV-band, and the overlay PV-bands for this test case:

```

set opticalList1 {Poly1_50 Poly1_0 Poly1_50}
set doseList1 {0.98 1.00 1.02}
set opticalList2 {Poly2_50 Poly2_0 Poly2_50}
set doseList2 {0.98 1.00 1.02}
set counter 1
array set xshift {N 0 E 0.01 W -0.01 S 0 NE 0.01 NW -0.01 SE 0.01 SW -0.01}
array set yshift {N 0.01 E 0 W 0 S -0.01 NE 0.01 NW 0.01 SE -0.01 SW -0.01}

foreach optical1 $opticalList1 dose1 $doseList1 {
    LFD::Contour -layer poly -handle c$counter\_mask1 \
        -operation simulation -arguments [list -definition [list image \
            mask0 optical \$model1 background dark layer \$layer1 \
            attenuated 0.06 dose \$dose1 size 0 resist_model \$model2 \
            etch_model \$model3]] \
        -inputLayers {poly_opc_mask1}
    -inputModels {$optical1 poly1.mod poly1_etch.mod}
    LFD::Contour -layer poly -handle c$counter\_mask2 \
        -operation simulation -arguments [list -definition [list image \
            mask0 optical \$model1 background dark layer \$layer1 \
            attenuated 0.06 dose \$dose2 size 0 resist_model \$model2 \
            etch_model \$model3]] \
        -inputLayers {poly_opc_mask2}
    -inputModels {$optical2 poly2.mod poly2_etch.mod}
    LFD::Contour -layer poly -handle c$counter -operation or \
        -inputLayers {c$counter\_mask1 c$counter\_mask2}
    foreach shift {N E W S NE NW SE SW} {
        LFD::Contour -layer poly -handle c$counter\_mask1_$shift \
            -operation shift -inputLayers c$counter\_mask1 \
            -arguments {-xshift $xshift($shift) \
                -yshift $yshift($shift) }
    }
    incr counter
}

LFD::Contour -layer poly -handle min_edge -operation and \
    -inputLayers {c1 c2 c3}
LFD::Contour -layer poly -handle max_edge -operation or \
    -inputLayers {c1 c2 c3}
LFD::Band -handle band1 -layer poly -subwindow 1 -minBandEdge min_edge \
    -maxBandEdge max_edge

# Creating overlay PV-bands:
foreach shift {N E W S NE NW SE SW} {
    LFD::Contour -layer poly -handle min_edge_$shift -operation and \
        -inputLayers {c1_$shift c2_$shift c3_$shift}
    LFD::Contour -layer poly -handle max_edge_$shift -operation or \
        -inputLayers {c1_$shift c2_$shift c3_$shift}
    LFD::OverlayBand -handle band1_$shift -layer poly -subwindow 1 \
        -minBandEdge min_edge_$shift -maxBandEdge max_edge_$shift \
        -shift $shift -origBandHandle band1
}

```

PrintableBand

Simulates expected lithographic PV-bands in the absence of an RET recipe.

Usage

PrintableBand

```
-layer input_layer_name
-printableModel {printable_model_path | inline_model}
-doseSpanList dose_list1 [-doseSpanList2 dose_list2]
[-doublePatterning {split | trim}]
[-printableSplitMasks split_masks_list]
[-security {yes | no}]
[-independentWindows yes]
```

Description

Simulate the expected lithographic process variation band (PV-band) in the absence of an RET recipe. This functionally is needed when layout designers move to an advanced process technology before mature RET recipes are available from the foundry.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the target layer you are checking.

- **-printableModel {*printable_model_path* | *inline_model*}**

Required keyword followed by the *printable_model_path* or an *inline_model*, defining the printable model to be use for printable-band generation.

- **-doseSpanList *dose_list1* -doseSpanList2 *dose_list2***

Required keyword, followed by a list of one or more lists defining doses to use for printable-band generation. Doses must be expressed as real numbers which represent percentages.

Each list of doses must be enclosed in a Tcl list, and the entire list of lists must be enclosed in a Tcl list. For example:

```
{ {1.0 0.9} {0.8 1.02} }
```

The optional keyword -doseSpanList2 defines the doses for the second mask used with a double exposure processes. The arguments to -doseSpanList2 must conform to the same constraints as those for **-doseSpanList**. The number of dose lists and the number of dose values in each list must match those for **-doseSpanList**.

- **-doublePatterning {split | trim}**

Optional keyword and argument defining the type of processing to perform when specifying two mask layers and two sets of exposure settings.

The “-doublePatterning split” option refers to pitch-splitting techniques, and the “-doublePatterning trim” option refers to sacrificial printing-assist features, which 32 nm methods often use in favor of pitch-splitting.

- **-printableSplitMasks *split_masks_list***

Optional argument and value used with double patterning. The *split_masks_list* is a list of two masks defining the split masks.

- **-security {yes | no}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-independentWindows yes**

Optional keyword and argument that modifies the default behavior in which PV-bands are calculated. By default, subwindows are assumed to have been constructed as extensions to previously-created subwindows. In other words, the process conditions defined in **-doseSpanList** are assumed to be cumulative. If the first subwindow explicitly defines two process conditions, and the second subwindow defines two different process conditions, then the PV-band for the second subwindow is calculated based on all four process conditions.

Specifying “-independentWindows yes” instructs the command to calculate printable-bands based only on those process conditions defined explicitly for the subwindow.

Examples

In TVF code:

```
LFD::PrintableBand \
    -layer poly \
    -modelDir ./models \
    -printableModel ./models/printableModel \
    -doseSpanList {{1 0.9} {0.8 1.02}}
```

ProcessCorrection

Applies a process correction recipe to the specified design layer.

Usage

ProcessCorrection

```
-layer input_layer_name
-targetLayer retarget_layer_name
-database db_name
```

Description

Applies a process correction recipe to the specified design layer.

This command is only for use with a PDK.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the design layer to be corrected before generating PV-bands.

- **-targetLayer *retarget_layer_name***

Optional keyword and argument defining the name of a referenced layer that is to be the target layer for OPC. If not specified, *retarget_layer_name* is assumed to be the drawn layer.

- **-database *db_name***

Optional keyword and argument defining the RDB to which the contours generated by the check are written.

PVband

Generates PV-band data for the specified layer.

Usage

Syntax 1

PVband

```
-layer input_layer_name
-layerRET {opc_layer | layer_list} [-layerRET2 {opc_layer | layer_list}]
[-ddmFiles {ddmFiles list} [-ddmFiles2 {ddmFiles list}]
-foreground {trans | trans_list} [-foreground2 {trans | trans_list}]
-background trans [-background2 trans]
[-doublePatterning {split | trim} [-overlay overlay_value]]
[-security {no | yes}]
[-pixel pix_size]
-modelDir model_dir
-lithomodelFile model
[-resistFile model]
[-etchFile etch_model]
[-opticalFile {models_list}]
-opticalSpanList models_list1 [-opticalSpanList2 models_list2]
-doseSpanList dose_list1 [-doseSpanList2 dose_list2] ...
[-resistSpanList resist_list1 resist_list2
    [-resistSpanList2 resist_list1 resist_list2]]
[-etchSpanList etch_list1 etch_list2
    [-etchSpanList2 etch_list1 etch_list2]]
[-sizeSpanList size_list1 [-sizeSpanList2 size_list2]]
[-independentWindows yes]
```

Syntax 2

PVband

```
-layer input_layer_name
[-pixel pix_size]
-pdkCheckName check_template
[additional_options]
```

Description

Generates PV-bands for the layer of process variation experiment described by the arguments to this command.

A *PV-band* is a geometry that shows how edge locations respond to process variations. It represents the area within which a feature prints as the process conditions vary. Generating PV-bands involves simulating the printed image at the various process conditions and combining the resulting printed images. PV-bands have widths due to process variation.

PV-band data is *not* written to any database automatically. To write PV-band data to an RDB, you must use the LFD::[OutputBands](#) command.

If used with a PDK, this function calls PVband in the PDK to generate PV-bands for the specified layer.

LFD::PVband Litho Model Support

Using the LFD::PVband function with litho models (litho model mode), requires the inputs to have the necessary information for creating the Calibre OPCverify setup file. This information includes the model definitions.

- **Litho Model Definition** — Litho models can be defined either inline or within a litho model file that resides in the model directory (non-inline). In the case of inline litho models, the model components are replaced with the corresponding symbolic names of the models. Inline litho models must be previously defined by “*_model_load” commands that load each component model. In the case of non-inline litho models, the model components are defined in a litho model file “Lithomodel” that resides in a litho model directory.

Refer to the *Calibre OPCverify User’s and Reference Manual* for a complete description of the litho model and OPCverify setup files.

- **Inline Litho Model Usage** — The inline definition of a litho model is intended only for use in creating encrypted setup files for inclusion in encrypted TVF and PDKs. Do not use this in standard Calibre OPCverify or Calibre nmOPC setup files. The litho model inline definition 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_DEVEL  
OPMENENT=1
```

Refer to “[Encrypting the Calibre LFD Rules](#)” on page 510 for information on how to encrypt the rule files that perform OPC and Calibre LFD.

- **PDK Litho Model Definition** — Using the LFD::PVband function with litho models (litho model mode) in a PDK is similar to the non-PDK command call. Both inline and non-inline litho models are only defined in the -processInfo subsection of the PDK. Model definitions including resist, etch, optical, and DDM are mandatory only if the litho model is inline.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the target layer under investigation. This can be either an original or derived polygon layer.

- **-layerRET** {*opc_layer* | *layer_list*} -layerRET2 {*opc_layer* | *layer_list*}

Required keyword used to specify the names of the RET layers used to generate the PV-band. These can be either original or derived polygon layers.

- Supply a single layer name when generating PV-bands from only the OPC-corrected layer.
- Supply a list of layer names when generating PV-bands from the OPC-corrected layer plus SRAF layers and so on. When you specify a list of layer names, you must also specify a list as the argument for -foreground.

The optional keyword -layerRET2 defines the names of a second set of RET layers for use with double exposure processes. The arguments to -layerRET2 must conform to the same constraints as those for -layerRET.

- **-ddmFiles** {*ddmFiles list*}

Optional argument defining a list of Domain Decomposition Method (DDM) libraries. This list should have the same length as the input list to **-layerRET**. If one of the input layerRET does not have a corresponding DDM file, then its corresponding entry in the ddmFiles list is “NULL”. Input to ddmFiles can be a file name, an inline DDM model, or an environment variable. In the case of using inline litho models, this is an optional list of pairs containing the definitions for all the DDM files pointed to inside the litho model and the symbolic names inside the litho model. This list is not required if the DDM files are located in the litho model directory.

- **-ddmFiles2** {*ddmFiles list*}

Optional argument used, in the case of double patterning or double exposure, to define a list of DDM libraries. This list should have the same length as the input list to **-layerRET**. If one of the input layerRET does not have a corresponding DDM file, its corresponding entry in the ddmFiles list is “NULL”. Input to ddmFiles can be a file name, an inline DDM model, or an environment variable. In the case of using inline litho models, this is an optional list of pairs containing the definitions for all the DDM files pointed to inside the litho model and the symbolic names inside the litho model. This list is not required if the DDM files are located in the litho model directory.

- **-foreground** {*trans* | *trans_list*} -foreground2 {*trans* | *trans_list*}

Required argument defining the optical transmission of the RET layer’s foreground. If you specify a list of RET layer names, you must also specify a list of *trans* values, with the first *trans* value representing the transmission for the first RET layer, the second *trans* value represents the transmission for the second RET layer, and so on. If you supply a list of transmission values, it must be enclosed in a Tcl list.

For each layer, the optical transmission type must be the compliment of the background. For example, you cannot specify clear features if you specify a clear background. [Table 7-9](#) lists the valid type choices.

The transmission can be supplied using any of the following forms: dark, clear, {*attenuated_factor*}, {*real imag*}, which are defined in [Table 7-9](#).

Table 7-9. 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)
{real imag}	<p>Defines the layer's optical transmission to be as specified by the real and imaginary value pair. The attenuated syntax for the transmission is recommended over the REAL,IMAGINARY pair syntax.</p> <p>If you specify background with the {real imag} pair for a specified layer transmission value, the resulting transmission value for the output layer with a REAL,IMAGINARY pair is resolved relative to the background pair. The layer transmission value is resolved according to the following equation:</p> $\text{background (real imag)} + \text{foreground (real imag)} = \text{transmission_value}$ <p>Refer to the Calibre RET Reference Manual for a complete description of foreground, background, and transmission values.</p>
{attenuated factor} {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 <i>factor</i>.</p> <p>When Calibre LITHO tools encounter “attenuated <i>factor</i>”, they translate the factor into a real imaginary pair such that the attenuated background has a transmission value:</p> $\text{RE(layer)} = -\sqrt{\text{percent}/100}$ $\text{IM(layer)} = 0$ <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.

The optional keyword -foreground2 defines the optical transmission values for the second set of RET layers used for double exposure processes. The arguments to -foreground2 must conform to the same constraints as those for -foreground.

- **-background trans** -background2 *trans*

Required keyword describing the optical transmission of the mask background. The transmission can be supplied using any of the following forms: dark, clear, {attenuated factor}, {real imag}, which are defined in [Table 7-9](#).

The optional keyword -background2 defines the optical transmission value for the background of the second mask used for double exposure processes. The argument to -background2 must conform to the same constraints as for -foreground.

- **-doublePatterning {split | trim}**

Optional keyword and argument defining the type of processing to perform when specifying two mask layers and two sets of exposure settings.

By default, when two mask layers and sets of exposure settings are defined, simulations calculate the PV-bands resulting from exposing the wafer surface twice before developing. That is expose twice, then develop once.

When this keyword is specified, simulations calculate the PV-bands resulting from exposing and developing using the first mask, then exposing and developing using the second mask. That is, expose twice and develop twice.

The “-doublePatterning split” option refers to pitch-splitting techniques, and the “-doublePatterning trim” option refers to sacrificial printing-assist features, which 32 nm techniques often use in favor of pitch-splitting.

The use of the keyword -doublePatterning does not effect the use of the -etchfile keyword. When -etchFile is defined, the developing is a two-step process that encompasses separate simulations, one for each the resist process and the etching process.

- **-overlay *overlay_value***

Optional keyword and argument that instructs to check for double-patterning overlay errors, where *overlay_value* is a single real number describing the amount of overlay error in microns that is applied to all process subwindows. The -overlay argument can be added to a LFD::PVband command instance if -doublePatterning is also specified.

- **-security {no | yes}**

Optional argument defining security privileges. If set to “yes”, the setup file is encrypted in the transcript.

- **-pixel *pix_size***

Note

 The -pixel argument is deprecated with the 2016.3 release. You should use the [SimConfig](#) -imagegrid and -setupfile_number options to define the Calibre OPCverify image grid and control the setup file generation per layer specification.

Optional keyword and argument defining the size of the grid used for simulation, specified in microns.

The overall simulation time for image calculations is proportional to the area in this grid. In general, a 10nm grid requires 4 times longer runtime than a 20nm grid. Therefore, it is best to choose the largest possible grid that can support the needed accuracy.

- **-modelDir *model_dir***

Required keyword and argument defining the parent directory which contains the optical and resist models used for simulation. The litho model directory resides in this directory in the case of using non-inline litho models.

- **-lithoModelFile *model***

Required keyword and argument containing the definition for the litho model file (inline) or the name of the litho model directory inside the model_dir directory (non-inline). This argument is required in litho model mode.

- **-resistFile *model***

Optional keyword and argument defining the resist model and its symbolic name inside the litho model to use for simulations. This argument is required in the case of using inline litho models.

The *model* argument must be one of:

- A model name — This identifies a model that resides in the model_dir directory.
- A variable — This must be the name of a variable defined previously in the rule file and set to a string that is the full model description. For a complete description of how to use this option, refer to [Embedding the Resist Models in the Calibre LFD Rule File](#).
- A single real number — This defines the model to be a constant threshold model, with the threshold value as specified.

- **-etchFile *etch_model***

Optional keyword and argument defining an etch model to use for simulating etch effects. This argument is required in the case of using inline litho models.

The *etch_model* argument must be one of:

- A model name — This identifies a model that resides in the model_dir directory.
- A variable — This must be the name of a variable defined previously in the rule file and set to a string that is the full model description.

When -etchFile is defined, the developing (taking an aerial image and calculating the simulated chip) is treated as a two-step process that encompasses separate simulations, one for the resist process and one for the etching process.

- **-opticalFile *models_list***

Optional keyword and list argument containing a list of pairs where each pair is the path or definition of the optical model and its symbolic name used in the litho model file. This argument is used in litho model mode. For example,

```
-opticalFile {o1{P1W_D0}{o2{P1W_D50}}{o3{P1W_D100}}}
```

This keyword is required in the case of using inline litho models.

- **-opticalSpanList *models_list1* -opticalSpanList2 *models_list2***

Required keyword followed by one or more lists of model paths defining the optical models to use for PV-band generation. All optical models in these lists must reside within the model_dir directory.

- Each list must define at least one optical model.
- When only one list is supplied, the LFD::PVband command performs a single process variation experiment.
- When multiple lists are supplied, the LFD::PVband command performs one process variation experiment for each list. The first list (*models_list1*) defines the optical models to use in first experiment, the second list (*models_list2*) defines the optical models to use in second experiment, and so on.

Defining multiple process variations experiments makes it possible to rank errors according to how critical they are. To rank errors accurately, you must supply the model lists in the correct order:

Table 7-10. Model List Positions

Position in List of Lists	Experiment	Criticality(priority)
1	most narrow	most critical / highest priority
2	slightly broader	^
:	broader	v
n	broadest	least critical/ lowest priority

Each list is order-dependent, with the order matching the order in the associated dose list: the first model in *models_list1* is used with the first dose in *dose_list1*, the second model in *models_list1* is used with the second dose in *dose_list1*, and so on.

Each list of optical models must be enclosed in a Tcl list, and the entire list of lists must be enclosed in a Tcl list. For example:

```
{ {P1W_D0 P1W_D50} {P1W_D0 P1W_D0 P1W_D100 P1W_D100} }
```

For lists of optical models defined inside of litho models, the optical models must be positive integers followed by “nm”. For example:

```
{ {0nm 0nm 50nm 50nm} {0nm 0nm 0nm 100nm 100nm 100nm} }
```

The optional keyword -opticalSpanList2 defines the optical models for the second mask used with a double exposure processes. The arguments to -opticalSpanList2 must conform to the same constraints as those for -opticalSpanList. The number of model lists and the number of model paths in each list must match those for **-opticalSpanList**.

- **-doseSpanList** *dose_list1* [-doseSpanList2 *dose_list2*] ...

Required keyword followed by a list of one or more lists defining the doses to use for PV-band generation. Doses must be expressed as real numbers representing percentages.

The lists are order-dependent, with the order matching the order in the associated model list: the first model in **models_list1** is used with the first dose in **dose_list1**, the second model in **models_list1** is used with the second dose in **dose_list1**, and so on.

The number of dose lists and the number of dose values in each list must match model_lists supplied using **-opticalSpanList**.

Each list of doses must be enclosed in a Tcl list, and the entire list of lists must be enclosed in a Tcl list. For example:

```
{ {0.9800 1.0000} {1.0200 1.0500 0.9500 0.9800} }
```

The optional keyword -doseSpanList2 defines the doses for the second mask used with a double exposure processes. The arguments to -doseSpanList2 must conform to the same constraints as those for **-doseSpanList**. The number of dose lists and the number of dose values in each list must match those for **-doseSpanList**.

- **-resistSpanList** *resist_list1* *resist_list2*

Optional keyword and argument used to define resist models in the **PVband** call. The -resistSpanList is of the same length as **-opticalSpanList** and **-doseSpanList**. The contours definition changes when using this option with checks that use simulation contours and not PV-bands. In such case, identify the contour with defocus_model, dose, size, and resistModel.

- **-resistSpanList2** *resist_list1* *resist_list2*

Optional keyword and argument used to define resist models in the LFD::PVband call when -doublePatterning is specified. The resistSpanList is of the same length as **-opticalSpanList** and **-doseSpanList**. The contours definition changes when using this option with checks that use simulation contours and not PV-bands. In such case, identify the contour with defocus_model, dose, size, and resistModel.

- **-etchSpanList** *etch_list1* *etch_list2*

Optional keyword and argument used to define an etch model or models in the LFD::PVband call. You can either use -etchFile or -etchSpanList to define etch models in the LFD::PVband call, but to use -opticalSpanList, you must also specify -resistSpanList. The -etchSpanList is of the same length of **-opticalSpanList** and **-doseSpanList**. The contours definition changes when using this option with checks that work on simulation contours instead of the PV-bands. In this case, identify the contour with defocus_model, dose, size, resistModel, and etchModel.

- **-etchSpanList2** *etch_list1* *etch_list2*

Optional keyword and argument used to define an etch model or models in the LFD::PVband call when -doublePatterning is specified. You can either use -etchFile or -etchSpanList to define etch models in the LFD::PVband call, but to use -opticalSpanList, you must also specify -resistSpanList. The etchSpanList is of the same length of

opticalSpanList and **doseSpanList**. The contours definition changes when using this option with checks that work on simulation contours instead of the PV-bands. In this case, identify the contour with defocus_model, dose, size, resistModel, and etchModel.

- **-sizeSpanList** *size_list1* **-sizeSpanList2** *size_list2*

Optional keyword followed by a list of one or more lists defining the mask bias to apply during PV-band generation. The mask bias must be expressed as real numbers representing the bias size.

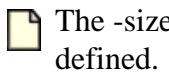
The lists are order-dependent, with the order reflecting the process experiment (subwindow) for which the mask bias is applied: the first size in *size_list1* defines the size of mask bias to apply in the first experiment, the second size in *size_list1* defines the size of mask bias to apply in the second experiment, and so on.

The number of *size_lists* and the number of mask bias sizes in each list must match *model_lists* supplied using **-opticalSpanList**.

Each list of mask bias sizes must be enclosed in a Tcl list, and the entire list of lists must be enclosed in a Tcl list. In litho model mode, the PVband section in the PDK allows an empty mask bias size list ({}) to be specified, and in this case, the mask bias size values from the litho model are used. If PVband mask bias sizes values are specified (non-empty list), these sizes values override the size values specified in the litho model.

The optional keyword **-sizeSpanList2** defines the mask bias sizes for the set of process experiments performed for the second mask used with a double exposure processes. The arguments to **-sizeSpanList2** must conform to the same constraints as those for **-sizeSpanList**. The number of *size_lists* and the number of size values in each list must match those for **-sizeSpanList**.

Note



The **-sizeSpanList2** parameter is required when **-sizeSpanList** and **-layerRET2** are defined.

- **-ddmSpanList** *ddm_list1* **-ddmSpanList2** *ddm_list2*

Optional keyword followed by a list of one or more lists defining the DDM. The DDM list is added as a list of sublists. Each sublist is a DDM list per RET layer, as in the following example:

```
subwindow1 {
    {MET1_D0 MET1_D0 MET1_D0 MET1_D50 MET1_D50 MET1_D50}
    {0.98 1.00 1.02 0.98 1.00 1.02}
    {0.00 0.00 0.00 0.00 0.00 0.00}
    {}
    {}
    {{ddm1 NULL} {ddm1 NULL} {ddm1 NULL} {ddm2 NULL}
     {ddm2 NULL} {ddm2 NULL}}
}
```

- **-independentWindows yes**

Optional keyword and argument that modifies the default behavior in which PV-bands are calculated. By default subwindows are assumed to have been constructed as extensions to previously-created subwindows. In other words, the process conditions defined in **-opticalSpanList**, **-sizeSpanList**, and **-doseSpanList** are assumed to be cumulative, and if the first subwindow explicitly defines two process condition and the second subwindow defines two different process conditions, then the PV-band for the second subwindow is calculated based on all four process conditions.

Specifying “**-independentWindows yes**” instructs the command to calculate PV-bands based only on those process conditions defined explicitly for the subwindow.

- **-pdkCheckName *check_template***

Required keyword and argument specifying the name of the check template defining how this check is performed.

- ***additional_options***

Optional keywords and arguments allowed only when the security settings within the PDK permit you to modify the settings for this command. These can be any of the options for the non-PDK LFD::PVband command. Any additional options specified overrides the options defined within the PDK.

Examples

Example 1

This example is for a typical 90nm contact layer:

- Background — Attenuated phase shift mask with 6% transmission.
- Contact holes — Features are clear.
- Assist feature — Features are clear.
- Chrome patches — Features are dark.

```
LFD:::PVband -layer contact_target \
    -layerRET      {lfd_contact_mopc contact_sraf contact_patches} \
    -foreground     {clear clear dark} \
    -background    {attenuated 0.06} \
    -pixel          0.02 \
    -modelDir      {./contact/models} \
    -resistFile    {contact.mod} \
    -opticalSpanList {{D0 D0 D0 D50 D50} \
                      {D0 D0 D75 D75}} \
    -doseSpanList   {{0.9800 1.0000 1.0200 0.9800 1.0000 1.0200} \
                      {0.9600 1.0000 1.0400 0.9600 1.0000 1.0400}}
```

Example 2

This example specifies pitch splitting and to check for overlay errors of 2 microns:

```
LFD:::PVband -layer active -pixel 0.02 -doublePatterning split -overlay 2
```

Example 3

This example shows an inline litho model definition:

```
set ::env(lithomodel_inline) {{
    version 1
    resist r1
    mask 0 {
        background clear
        mask_layer 0 TRANS atten 0.06
            optical o1 focus 0nm
            optical o2 focus 50nm
            optical o3 focus 100nm
    }
}}
LFD:::PVband \
-layer opc \
-layerRET lfd_poly_mopc \
-resistFile {poly.mod2 r1} \
-opticalFile { o1 {P1W_D0} o2 {P1W_D50} o3 {P1W_D100}} \
-lithomodelFile { $::env(lithomodel_inline) } \
-modelDir "./models" \
-opticalSpanList {{0nm 0nm 50nm 50nm} {0nm 0nm 0nm 100nm 100nm 100nm}} \
-doseSpanList {{1.02 0.96 0.98 1.0} {0.96 1.0 1.02 0.98 1.0 1.02}}
```

Example 4

This example shows a non-inline litho model definition followed by an example of a non-inline litho model directory structure:

```
LFD:::PVband \
-layer opc \
-lithomodelFile " LITHO_MODEL " \
-modelDir "./models" \
-opticalSpanList {{0nm 0nm 50nm 50nm} {0nm 0nm 0nm 100nm 100nm 100nm}} \
-doseSpanList {{1.02 0.96 0.98 1.0} {0.96 1.0 1.02 0.98 1.0 1.02}}
```

Example of a non-inline litho model directory structure:

```
%ls ./models
LITHO_MODEL
%cd LITHO_MODEL
%ls
Lithomodel OP_0 OP_50 OP_100 resist.mod
```

Example 5

This PDK example defines inline litho models and generates PV-bands:

```
-processInfo "{ "
OpticalModels { o1 {PW_0} o2 {PW_50} o3 {PW_100} }
ResistModels { r1 {poly.mod} }
ProcessCorrection {CMACRO macro_1}
DrawnLayer poly
OpcLayers opc
opcInLayers {}
modelDir {./models}
LithoModel {
    Poly_litho_model {
        version 1
        resist r1
        mask 0 {
            background clear
            mask_layer 0 TRANS atten 0.06
            optical o1 focus 0nm
            optical o2 focus 50nm
            optical o3 focus 100nm
        }
    }
}
PvBands {
    subwindow1 {{0nm 0nm 50nm 50nm } {1.02 0.96 0.98 1.0}}
    subwindow2 {{0nm 0nm 0nm 100nm 100nm 100nm } {{0.96 1.0 1.02 0.98 1.0
    1.02}}
}
RetLayers {
    mask0 {
        Layers lfd_mopc
    }
}
"} "
```

Example 6

This PDK example defines non-inline litho models and generates PV-bands:

```
-processInfo "{

ProcessCorrection {CMACRO macro_1}

DrawnLayer poly
OpcLayers opc
opcInLayers {}

modelDir {"./models/"}
Lithomodel {litho_model_poly {LITHO_MODEL} }

PvBands {
    subwindow1 {{0nm 0nm 50nm 50nm } {1.02 0.96 0.98 1.0}}
    subwindow2 {{0nm 0nm 0nm 100nm 100nm 100nm } {{0.96 1.0 1.02 0.98 1.0
1.02}}
}
RetLayers {
    mask0 {
        Layers lfd_mopc
    }
}
}"}
```

ReadECO

Provides layer name for GDS ECO layers.

Usage

ReadECO

-layerNumber *layer_number*
-layerOut *out_layer_name*
[-halo *halo_um*]

Description

Instructs the Calibre LFD software to perform simple layer mapping that allows you to provide layer name for the GDS ECO layers by number. The output is a new SVRF layer.

This read function allows you to read modifications made by place and route tools that are represented as ECO Layer, so that Calibre LFD can run incrementally on only modified regions.

In the incremental flow, the LFD::ReadECO function is followed by an LFD::[IncrementalSelect](#) operation to cut the design by the AOI regions.

Arguments

- **-layerNumber** *layer_number*
Required keyword and argument specifying the *layer_number* of the layer containing ECO regions. It can take either a *layer_number* or a *layer_number.datatype*. It can also take a list of more than one *layer_number* enclosed in braces ({}).
- **-layerOut** *out_layer_name*
Required keyword and argument defining the output layer name. All of the layer numbers specified in **-layerNumber** are mapped to this layer.
- **-halo** *halo_um*
Optional keyword and argument defining the amount by which polygons on the marker layer are sized prior to generating the output layer. The *halo_um* value is a floating point number and is specified in microns.

Examples

```
LFD:::ReadECO -layerNumber 15.22 -layerOut M2_ECO -halo 0.4
```

ReadHIF

Reads a Cadence Hotspot Interface Format (HIF) file and maps all the polygons from the checks for a given layer onto a single output layer.

Usage

ReadHIF

```
-file check_file_name
-layer target_layer_name
-layerNumber layer_number
-layerOut out_layer_name
[-halo halo_um]
[-magnify magnify_ratio]
[-type {CHECK | LITHO}]
```

Description

Reads a Cadence Hotspot Interface Format (HIF) file and maps all the polygons from the checks for a given layer onto a single output layer. The output layer is a new SVRF layer with a layer number.

The function can read CHECK or LITHO files.

This functionality is used in conjunction with place and route in an incremental flow. In the incremental flow, the LFD::ReadHIF function is followed by an LFD::IncrementalSelect operation to cut the design by the AOI regions.

Arguments

- **-file *check_file_name***

Required keyword and argument defining the name of the HIF formatted check file, which can be either LITHO or CHECK.

- **-layer *target_layer_name***

Required keyword and argument defining the name of the layer in the HIF file for which the check areas and errors are read.

- **-layerNumber *layer_number***

Required keyword and number used to specify the new layer number (as specified in the design database) used to place markers on.

- **-layerOut *out_layer_name***

Required keyword and argument defining the output layer name. All polygons in the HIF file for the layer specified in **-layer** are mapped to this layer.

- **-halo *halo_um***

Optional keyword and argument defining the amount by which polygons for the input layer are sized prior to generating the output layer. The *halo_um* value is a floating point number and is specified in microns. The default value is 0.

- **-magnify *magnify_ratio***

Optional keyword and argument instructing the tool to magnify the incoming polygons by the specified ratio. The *magnify_ratio* must be a floating point number. This option can be useful in handling different input and output precisions.

The *magnify_ratio* is determined by dividing the precision used in RET recipes by the user design precision. In most cases, this value is greater than 1.0. For example, the *magnify_ratio* is 4.0 if the precision of the RET recipe is 4000 and the user design precision is 1000. The default value is 1.0.

- **-type {CHECK | LITHO}**

Optional keyword and argument defining the type of HIF file. The file can be a CHECK file or LITHO file.

Examples

```
LFD::ReadHIF -file ./checkfile_lfd.rpt -layer M2 \
    -layerNumber 8 -layerOut M2_eco -halo 1.0 -magnify 4.0
```

ReadiLPC

Reads an iLPC file and maps all the polygons and areas for a given layer onto a single output layer.

Usage

ReadiLPC

```
-file iLPC_file_name
-layer target_layer_name
-layerNumber layer_number
-layerOut out_layer_name
[-halo halo_um]
[-magnify magnify_ratio]
```

Description

Reads an iLPC file and maps all the polygons and areas for a given layer onto a single output layer. The output layer is a new SVRF layer with a layer number. This command is used in conjunction with place and route in an incremental flow. In the incremental flow, the LFD::ReadiLPC function is followed by an LFD::IncrementalSelect operation to cut the design by the area-of-interest regions.

Arguments

- **-file *iLPC_file_name***

Required keyword and argument defining the name of the iLPC-formatted file generated by placing and routing. This option can also take a list of files instead of only one, and in such a case the layer information in the files is collected and used as in the case of single file.

- **-layer *target_layer_name***

Required keyword and argument defining the name of the layer in the iLPC file for which its areas and errors fixed by place and route are read.

- **-layerNumber *layer_number***

Required keyword and number used to specify the new layer number (as specified in the design database) used to place markers on.

- **-layerOut *out_layer_name***

Required keyword and argument defining the output layer name. All polygons in the iLPC file for the layer specified in **-layer** are mapped to this layer.

- **-halo *halo_um***

Optional keyword and argument defining the amount by which polygons for the input layer are sized prior to generating the output layer. The *halo_um* value is a floating point number and is specified in microns. If -halo is not used, the input layer polygons are not sized.

- **-magnify *magnify_ratio***

Optional keyword and argument instructing the tool to magnify incoming polygons by the specified ratio. The *magnify_ratio* can be an integer or a floating point number. The option can be useful in handling different input and output precisions. The default value is 1.

Examples

```
LFD::ReadiLPC \
    -file ./PnR_output.iLPC \
    -layer M2 \
    -layerNumber 10 \
    -layerOut M2_eco \
    -halo 1.0 \
    -magnify 4.5
```

ReadRDB

Reads an RDB file and maps the polygons from the checks for a given layer onto an output layer.

Usage

ReadRDB

```
-checkName check_name
-rdb rdb_file_name
-layerNumber layer_number
-layerOut out_layer_name
[-halo halo_um]
[-magnify magnify_ratio]
```

Description

Reads an RDB file and maps the polygons from all of the checks for a given layer onto a single output layer.

This read function allows you to read modifications made by place and route tools that are represented as RDB files, so that Calibre LFD can run incrementally on only the modified regions. In the incremental flow, the LFD::ReadRDB function is followed by an LFD::IncrementalSelect operation, to cut the design by the AOI regions.

Arguments

- **-checkName *check_name***

Required keyword and argument containing the RDB check name that contains the ECO regions.

- **-rdb *rdb_file_name***

Required keyword and argument defining the name of the RDB file to be read.

- **-layerNumber *layer_number***

Required keyword and number used to specify the new layer number used to place markers on.

- **-layerOut *out_layer_name***

Required keyword and argument defining the output layer name. All polygons for specified check name are mapped to this layer.

- **-halo *halo_um***

Optional keyword and argument defining the amount by which polygons of the specified check name are sized prior to generating the output layer. The *halo_um* value is a floating point number and is specified in microns.

- **-magnify *magnify_ratio***

Optional keyword and argument instructing the tool to magnify the incoming polygons by the specified ratio. The *magnify_ratio* must be a floating point number. The option can be useful in handling different input and output precisions.

The *magnify_ratio* is determined by dividing the precision used in RET recipes by the user design precision. In most cases, this value is greater than 1.0. For example, the *magnify_ratio* is 4.0 if the precision of the RET recipe is 4000 and the user design precision is 1000. The default value is 1.0.

Examples

```
LFD:::ReadRDB -checkName M2_MWC_p1 -rdb lfd_Errors.rdb -layerNumber 8 \
    -layerOut M2_eco_1 -magnify 4.0
```

RegisterBands

Makes PV-band data generated in a previous run available for analysis in the current run.

Usage

RegisterBands

```
-layer input_layer_name
-subwindow expr_number
-maxBandEdge outer_contour_layer
-minBandEdge inner_contour_layer
```

Description

Makes PV-band data generated in a previous run available for analysis in the current run. The input to this command is a pair of contour layers that are generated using the LFD::[CaptureContour](#) command with **-displacement** in some previous run.

This command was designed to be used in place of a LFD::[PVband](#) call, in situations where the PV-band data already exists in the form of contour layers in the design database. Combining LFD::PVband and LFD::RegisterBands within a single rule file (for the same layer) is therefore not permitted.

Combining [Customizable PV-Bands](#) and LFD::RegisterBands within a single rule file (for the same layer) is not allowed.

Recall that all Calibre LFD checks are performed on a specific PV-band, which is uniquely identified by its target layer and process variation (subwindow). This command associates the two existing (input) contour layers with a specific layer/subwindow pair. Once the PV-band data has been registered using this command, any check referencing this layer/subwindow pair is performed on the PV-band that is formed by the two contour layers.

The **outer_contour_layer** and **inner_contour_layer** are Calibre layers. That is, if they exist in a design database, they must be read into Calibre using the LAYER statement prior to registering them as PV-band layers.

Note

 Most Calibre LFD checks treat inner and outer edges of PV-bands differently, so it is important to make the proper edge assignments using the **-maxBandEdge** and the **-minBandEdge** keywords.

This command is not for use within a PDK.

Note

 LFD::[CSI](#) is the only Calibre LFD check that cannot be used with LFD::RegisterBands.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the target layer this previously-generated PV-band data is to be associated with. This can be either an original or derived polygon layer.

- **-subwindow *expr_number***

Required keyword and argument associating this PV-band data with a specific process variation experiment (subwindow). You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **LFD::PVband** command that was originally used to generate the PV-band data. That is, ***expr_number*** refers to an index to a list of experiments.

- **-maxBandEdge *outer_contour_layer***

Required keyword and argument specifying the name of the layer that contains the contour for the outer edge of the PV-band.

- **-minBandEdge *inner_contour_layer***

Required keyword and argument specifying the name of the layer that contains the contour for the inner edge of the PV-band.

Tip

 You can register a contour for use by specifying a single layer as both the **-maxBandEdge** and the **-minBandEdge**.

Examples

```
source layersfile.lyr
LFD::LoadPDK lfd/pdkfile.pdk
LFD::RegisterBands -layer metal1 -subwindow 1 -maxBandEdge outcontlayer \
                    -minBandEdge innercontlayer
LFD::MinSpaceCheck -pdk lfd_pdk -pdkCheckName Metal1_MSC_p1 \
                    -layer metal1 -database rdb/lfd_checks.rdb
```

RegisterContour

Makes contour data generated in a previous run available for analysis in the current run.

Usage

RegisterContour

```
-layer input_layer_name
-subwindow expr_number
-contour contour_layer
-contourCondition contour_condition_list
[-contourNumber contour_order_in_subwindow]
```

Description

Makes contour data generated in a previous run available for analysis in the current run. LFD::RegisterContour is similar to LFD::RegisterBands but rather than registering a PV-band, it registers a contour.

If you want to use LFD::RegisterContour with the LFD::RegisterBands command, you must call LFD::RegisterBands first and then LFD::RegisterContour to construct the PV-band first, and then assign different contours to it. In this case, PV-bands are only generated from LFD::RegisterBands. LFD::RegisterContour have no effect on generated PV-bands, and it simply registers the contours in the package for further use by the checks in the rule file.

If you use LFD::RegisterContour without an LFD::RegisterBands call, PV-bands are generated from the contours registered in this PV-band (subwindow).

Combining [Customizable PV-Bands](#) and LFD::RegisterContour within a single rule file (for the same layer) is not allowed. Combining LFD::PVband and LFD::RegisterContour within a single rule file (for the same layer) is not permitted.

Tip

 You can call LFD::RegisterContour more than once for the same PV-band to register more than one contour in the PV-band.

This command is not for use within a PDK.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the target layer this previously-generated PV-band data is to be associated with. It can be either an original or derived polygon layer.

- **-subwindow *expr_number***

Required keyword and argument associating this PV-band data with a specific process variation experiment (subwindow). You must reference individual process variation

experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the LFD::PVband command that was originally used to generate the PV-band data. That is, *expr_number* refers to an index of experiments.

- **-contour *contour_layer***

Required keyword and argument specifying a layer name that contains the contour to be registered. The *contour_layer* is a layer generated using the LFD::CaptureContour command from a previous run. It is a Calibre layer, thus if it exists in a design database, it must be read into Calibre using the SVRF LAYER statement prior to registering it as a contour layer.

- **-contourCondition *contour_condition_list***

Required keyword and argument defining the contour condition in this previously-generated PV-band.

The *contour_condition_list* must be an ordered list, with either 3, 4, 5, 6, 8, or 10 elements defining an explicit contour condition, and be supplied as follows:

```
{optical1 dose1 size1 [resist1 etch1] \
[optical2 dose2 size2 [resist2 etch2]]}
```

- **-contourNumber *contour_order_in_subwindow***

Optional keyword and argument specifying the order of the contour within this previously generated PV-band (subwindow) list. If this argument is not used, the package generates the contour numbering using the order of this LFD::RegisterContour command among other LFD::RegisterContour commands for a certain PV-band. The first command has *contour_order_in_subwindow*=1, the second command has *contour_order_in_subwindow*=2, and so on.

Examples

In this example, we register both {P1W_D0 1.0000 0} and {P1W_D0 0.9800 0} contours. You can use them with LFD::CSI or any dual layer checks. The PV-bands generated are those from LFD::RegisterBands, but if LFD::RegisterBands is not called, the PV-bands are generated from the two registered contours.

```
LFD:::RegisterBands -layer metal1 -subwindow 1 \
-minBandEdge min_contour_sub1_metal1 \
-maxBandEdge max_contour_sub1_metal1

LFD:::RegisterContour -layer metal1 -subwindow 1 \
-contour loadedcontour1metal1 \
-contourCondition {P1W_D0 1.0000 0}

LFD:::RegisterContour -layer metal1 -subwindow 1 \
-contour loadedcontour2metal1 \
-contourCondition {P1W_D0 0.9800 0}

LFD:::CSI -layer metal1 -layerCondition {P1W_D0 1.0000 0} \
-deviation 0.05 -database $lfdErrorDatabase
```

RegisterLayer

Registers a design layer with the Calibre LFD processing engine.

Usage

RegisterLayer

```
-Designlayer input_layer_name
-Processlayer layer_process_name
-pdk pdk_name
[-inputDesignLayers layer_list]
[-auxLayers layer_list]
[-mtOptions run_options]
```

Description

Associates a design layer with a layer process definition within the PDK.

The layer process definition defines the mask transmission, the OPC recipe, the model location, and so on.

Arguments

- **-Designlayer *input_layer_name***

Required keyword and argument specifying the name of the design layer. This layer must correspond to the DrawnLayer name inside of the PDK's -processInfo subsection.

- **-Processlayer *layer_process_name***

Required keyword and argument defining the layer process definition within the PDK that is to be associated with the design layer. This layer must correspond to the [addPDKLayer-name](#) layer name inside of the PDK.

- **-pdk *pdk_name***

Required keyword and argument specifying the name of the PDK containing the layer process definition.

- **-inputDesignLayers *layer_list***

Optional keyword and list argument used to specify the names of one or more design layers that fit into the placeholders defined in contour and band definitions in the PDK. This option is used by the [Customizable PV-Bands](#) flow.

- **-auxLayers *layer_list***

Optional keyword and list argument used to specify the names of one or more additional layers in addition to the specified design layer. Use of these additional layers varies according to the RET recipe.

- **-mtOptions *run_options***

Optional keyword and argument used to define the MTflex options to be used in the remote run of this layer. The option accepts all Calibre remote options (-turbo, -turbo_litho,

-turbo_all, -remote, -remotefile), and Calibre LFD follows the standard methodology for running in MTflex mode.

Examples

```
LFD::Begin
source layersfile.lyr
LFD::LoadPDK lfd/pdkfile.pdk
LFD::RegisterLayer -pdk lfd_pdk -Designlayer metall \
    -Processlayer pdklyr_metal1
LFD::MinSpaceCheck -pdk lfd_pdk -pdkCheckName Metall_MSC_p1 \
    -layer metall -database rdb/lfd_checks.rdb
LFD::End
```

RemoveErrors

Removes error markers from the output layer of a Calibre LFD check.

Usage

RemoveErrors

```
{-layer layer_name | {-layer1 layer1_name -layer2 layer2_name} }  
-subwindow subwindow_number  
[-subwindow2 subwindow2_number]  
-checkName check_name_list  
-pdkCheck pdk_check_name  
-removeLayer remove_layer_name  
-pdlHandle pattern_handle_name  
[-database removed_errors_database]  
[-removedErrors removed_errors_layer_list]  
[-patternKeys key_list]  
[-interactDistance interactDistance]  
[-maxSecondaryEdges max_anchor_edge_count]  
[-expandCells {yes | no}]  
[-tilesize tile_size]  
[-security yes]  
[-halo halo_value]
```

Description

Removes error markers from the output layer of a Calibre LFD check. You define error markers needing to be removed with either the **-removeLayer** or **-pdlHandle** options.

- If the **-removeLayer** option is used, the input layer indicated with this option is removed from the check output. When LFD::RemoveErrors is used with the **-removeLayer** option, the package simply removes the error markers of the check that interact with the remove layer.
- If the **-pdlHandle** option is used, the input patterns in the pattern matching database file indicated by this option are used to generate error markers that are removed from the check output layer. Generating the error markers from the input patterns defined by **-pdlHandle** is performed using a vertex structure matching method.

Caution

 LFD::RemoveErrors cannot be used with LFD::LineEndCheck when it contains both **-layer1** and **-layer2** input layers (3 layers input total), nor can it be used with any custom check associated with 3 or more input layers.

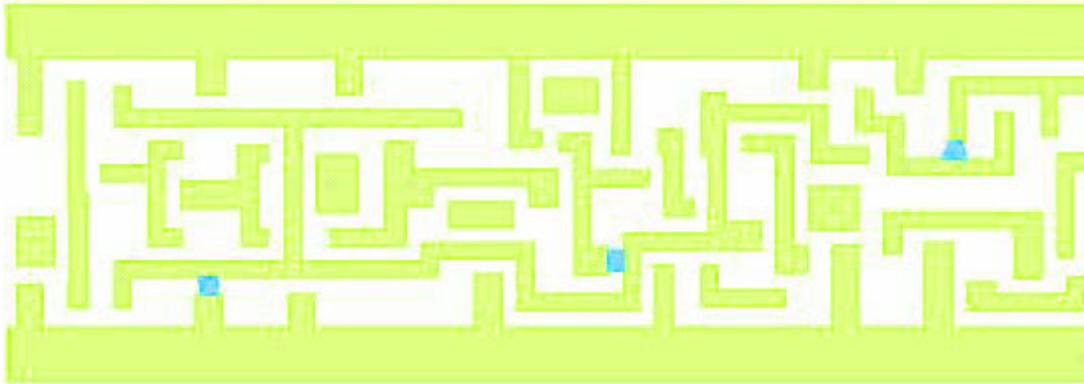
The following steps are followed by the package when the LFD::RemoveErrors command uses the **-pdlHandle** option:

1. Locate the output error markers of the checks defined in **-checkName** option.

2. Size those error markers by a halo value and capture the input layer polygons within the sized markers as shown in [Figure 7-40](#).
3. Match those layer polygons with the patterns defined in the pattern file to find whether any of the check output error markers are removed from the output.

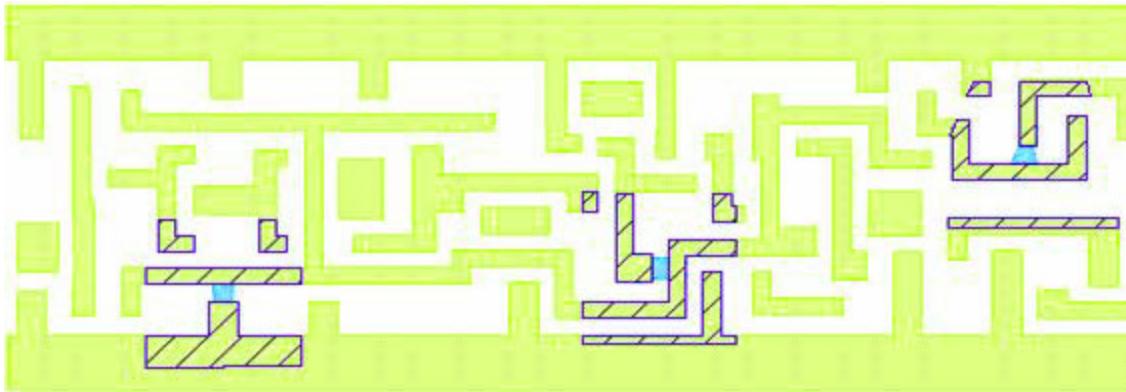
[Figure 7-39](#) shows space check output error markers when LFD::RemoveErrors is not called:

Figure 7-39. Space Check Output Error Markers Without Calling RemoveErrors



[Figure 7-40](#) shows capture of the input layer polygons inside the sized marker to match them with the patterns defined in the pattern file.

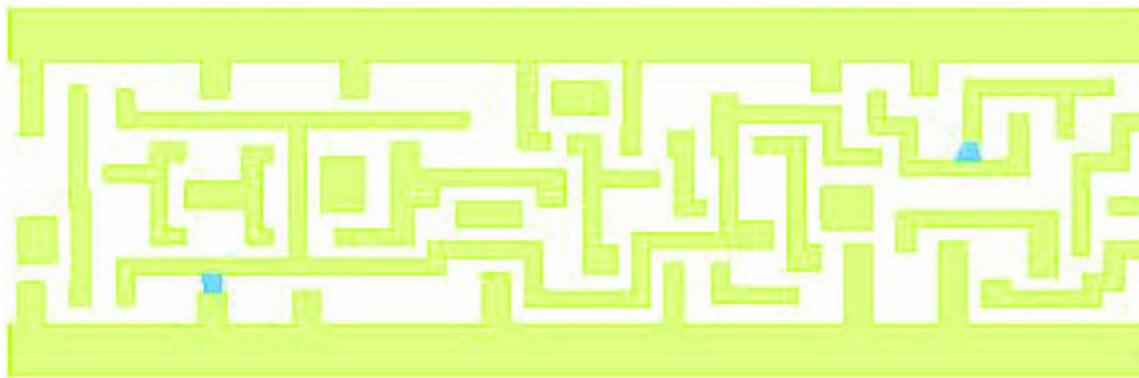
Figure 7-40. Capturing Input Layer Polygons



[Figure 7-41](#) shows a pattern stored in the input pattern file, which is determined and waived from the check output.

Figure 7-41. Stored Pattern

Figure 7-42 shows the output of the space check after waiving the error marker defined in the pattern file.

Figure 7-42. Space Check Output After Waiving Error Marker

Arguments

- **-layer *layer_name***

Required keyword and argument, only used for single layer checks. The layer should already be used in Calibre LFD checks.

- **-layer1 *layer1_name***

Required keyword and argument, only used for dual layer checks. The layer should already be used in Calibre LFD checks.

- **-layer2 *layer2_name***

Required keyword and argument, only used for dual layer checks. The layer should already be used in Calibre LFD checks.

- **-subwindow *subwindow_number***

Required keyword and argument defining the subwindow number used for the check.

- **-subwindow2** *subwindow2_number*

Optional keyword and argument defining the subwindow number used for the check. This keyword can only be used if **-layer2** is used.

- **-checkName** *check_name_list*

Required keyword and argument specifying a list of names of Calibre LFD checks. Each Calibre LFD check name is valid for the specified layer/layer1/layer2.

- **-pdkCheck** *pdk_check_name*

Required keyword and argument specifying the name of a PDK check. In this case, every Calibre LFD check in the rule file that uses this PDK check have their output layer filtered. This option should not be used with the **-checkName** option.

- **-removeLayer** *remove_layer_name*

Required keyword and argument specifying the name of a derived layer containing the error markers that are removed from the output of Calibre LFD checks whose names is given by **-checkName** option or uses a PDK check template whose name is given by **-pdkCheck** option. This option should not be used with the **-pdHandle** option.

Note



Use of either the **-checkName** or the **-pdkCheck** option is required by the LFD::RemoveErrors command, however both options can not be used together.

- **-pdHandle** *pattern_handle_name*

Required keyword and argument defining the input pattern file handle that points to patterns removed from the output of Calibre LFD checks, whose names is given by the **-checkName** option or which uses a PDK check whose name is given by the **-pdkCheck** option. This option should not be used with the **-removeLayer** option.

- **-database** *removed_errors_database*

Optional keyword and argument defining the name of the output removed errors database. This is useful for debugging purpose, such as if you want to view the output error markers removed from each Calibre LFD check after using the LFD::RemoveErrors command.

- **-removedErrors** *removed_errors_layer_list*

Optional keyword and argument defining the output removed layer list. This is useful for debugging purpose, such as if you want to view or operate on the output error markers that are removed from each Calibre LFD check after using the LFD::RemoveErrors command.

- **-patternKeys** *key_list*

Optional keyword and argument specifying the input pattern keys. One or more key names can be specified. Only patterns having keys matching all those in the command line are selected for matching. Selected patterns may have keys in addition to those in the command line. If no keys are specified in the command, then all patterns are selected. Any pattern with fewer keys than in the command is not selected.

- **-interactDistance *interactDistance***

Optional keyword and argument specifying an input parameter to the tiling mechanism. The value specified instructs Calibre LFD to look outward from a tile boundary by the user-specified amount to not miss any patterns that could exist at, or across, a tile boundary. During the matching process, there might be duplicate matches found at the border regions. These duplicates are resolved by the overlap merging process.

If the interaction distance is not specified, the largest pattern diameter in *x* or *y* is used.

Caution

 Your *interactDistance* must be 1 nm larger than the largest pattern diameter in *x* or *y* of any pattern within the used patterns file. Otherwise, you risk a potentially-missed pattern at a tile boundary.

- **-maxSecondaryEdges *max_anchor_edge_count***

Optional keyword and argument defining a control for the anchor edge selector inside of the pattern matching engine. You can specify an integer equal to or greater than 0 to control the number of secondary edges to consider when running the anchoring portion of the code.

One secondary edge is effective at drastically reducing the length of the anchor match list so that most of the entries are exact pattern matches. The length of the anchor match list can be reduced further by using additional secondary edges, but a point of diminishing returns is rapidly reached, and using more than 2 secondary edges is not usually effective (and it increases the pattern matching time). The value is an integer, and the default value is 1.

- **-expandCells {no | yes}**

Optional keyword and argument that can improve the runtime performance of the underlying engine by controlling the *expand_small_cells* statement in the setup file.

- **-tilesize *tile_size***

Optional keyword and argument that is used to control the tile size. This value is an integer, and the default value is 100 u.

- **-security yes**

Optional argument defining security privileges. If defined, the output SVRF is encrypted.

- **-halo *halo_value***

Optional keyword and argument used to define a value the package uses to form the intermediate layer used to create the error markers that are removed. If, for example, LFD::RemoveErrors is used with two Calibre LFD checks, the package generates an intermediate layer by first combining the output error markers of each check then size it and get the input layer polygons inside it. The Calibre LFD package then match the input defined patterns to those polygons and form the output error markers that are removed from the check output. The default value is 0.5.

Examples

Example 1

Here is an example of using the LFD::RemoveErrors command with a Calibre LFD check where neither are tied directly to a PDK definition. The result of which only returns LFD::AreaOverlayCheck hotspots found that do not interact with polygons discovered via LFD::RemoveErrors.

```
LFD:::AreaOverlayCheck -layer1 poly -layer2 contact -subwindow 1
    -overlayError 0.09 -minOverlapArea 0.0009 -overlapAreaType Average
    -database $lfdErrorDatabase -layerOut aoc_1
    -comment {Testing overlay check} -markerLayer markerLayer
    -checkName overlayCheck_1 -priority 1

LFD:::RemoveErrors -layer1 poly -layer2 contact -subwindow 1
    -checkName overlayCheck_1 -removeLayer aoc_1

LFD:::VariabilityIndices -layer contact -windowSize extents -subwindow 1
    -database $lfdErrorDatabase
LFD:::OutputBands -layer contact -bandType regular -subwindow 1
    -layerOut c1 -errorFilterSize 0.1
LFD:::CaptureContour -layer contact -displacement min -reference regular
    -subwindow 1 -errorFilterSize 0.5 -layerOut c3
```

Example 2

Here is an example of the command in the PDK flow. A template for the LFD::RemoveErrors command is defined using LFD::[addPDKCheck](#), and this template is used in the rule deck.

PDK

```
LFD:::addPDKCheck {-name msc_false_errors -pdk PDK45nm -type RemoveErrors \
    -security 11 -definition { \
        -pdlHandle msc_re -halo 0.5 -patternKeys MSC}}
```

Rule Deck

```
LFD::Begin  
  
LFD::loadPDK Pdk1  
  
LFD::RegisterLayer -pdk PDK45nm -Designlayer metal1 \  
-Processlayer pdklyr_M1DG  
  
LFD::RemoveErrors -layer metal1 -pdkCheckName msc_false_errors \  
-checkName {msc1 msc2}  
  
LFD::MinSpaceCheck -layer metal1 -pdkCheckName m1_msc \  
-checkName msc1 -layerOut msc1 -subwindow 1  
  
LFD::MinSpaceCheck -layer metal1 -pdkCheckName m1_msc -checkName msc2\  
-layerOut msc2 -subwindow 2  
  
LFD::End
```

The output layer of both checks are filtered. Patterns defined by the **msc_re** handle are removed from the output layer of both LFD::[MinSpaceCheck](#) commands.

Example 3

Here is an example of enforcing LFD::RemoveErrors within the PDK. In this case, the **-pdkCheck** option is used to filter the output layer of any Calibre LFD check that uses **m1_msc**.

PDK

```
LFD::addPDKCheck {-name msc_false_errors -pdk PDK45nm -type RemoveErrors \  
-security 11 -definition {-pd1Handle msc_false -pdkCheck m1_msc}}  
  
LFD::addPDKCheck {-name m1_msc -pdk PDK45nm -type MinSpaceCheck \  
-security 11 -definition {-subwindow 1 -minDRCspace 0.120 \  
-minLFDspace 0.08}}
```

Rule Deck

```
LFD::Begin  
  
LFD::loadPDK Pdk1  
  
LFD::RegisterLayer -pdk PDK45nm -Designlayer metal1 \  
-Processlayer pdklyr_M1DG  
  
LFD::RemoveErrors -layer metal1 -pdkCheckName msc_false_errors  
  
LFD::MinSpaceCheck -layer metal1 -pdkCheckName m1_msc -checkName msc1\  
-layerOut msc1 -subwindow 1  
  
LFD::MinSpaceCheck -layer metal1 -pdkCheckName m1_msc -checkName msc2\  
-layerOut msc2 -subwindow 2  
  
LFD::End
```

Since both LFD::MinSpaceCheck commands use the PDK check template, **m1_msc**, their output layer is filtered. Patterns defined by the **msc_false** handle are removed from both the output layer of both checks.

Example 4

This example shows demonstrates use of the command in a Non-PDK flow. In this case, the output layer of **mwc_poly** are filtered. Patterns defined by the **mwc_re** handle are removed from the output layer of **mwc_poly**.

```
LFD::Begin  
  
LFD::PVband -layer poly \  
-layerRET lfd_opc_1 \  
-foreground {{attenuated 0.06}} \  
-background clear \  
-pixel 0.02 \  
-modelDir {./setup/models} \  
-resistFile {resist.mod} \  
-opticalSpanList {{P1W_D0 P1W_D50}} \  
-doseSpanList {{1.0000 1.0200}}  
  
LFD::RemoveErrors -layer poly -checkName mwc_poly -pdlHandle mwc_re  
  
LFD::MinWidthCheck -layer poly -subwindow 1 -database $DB \  
-minDRCspace 0.10 -minLFDspace 0.05 \  
-checkName mwc_poly  
  
LFD::End
```

SetSTOMode

Sets the Structure Optimizer (STO) mode.

Usage

SetSTOMode

```
[ -layer layernames ]  
-mode {Simulation | noSimulation | Hybrid}
```

Description

Sets the Structure Optimizer (STO) mode. The Calibre LFD STO functionality includes three modes, as discussed in the reference page for the LFD::StructureOptimizer command. The LFD::SetSTOMode command sets the STO mode for specified layers, or sets it globally for all layers defined in a Calibre LFD kit.

Arguments

- **-layer *layernames***

Optional keyword and argument specifying at least one input layer. This is the layer(s) containing the geometries for which you want to set the mode.

If you call the LFD::SetSTOMode command with the -layer argument, the tool sets the mode for the list of layers mentioned in the command. If you call the command without specifying -layer argument, the tool sets the mode globally.

Note

 If you call the LFD::SetSTOMode command once without specifying the -layer argument (thereby setting the mode globally), and then you call the command an additional time with the -layer argument in the same rule file, the mode of the layer is that set using -layer, regardless of which one you called first.

- **-mode {Simulation | noSimulation | Hybrid}**

Required keyword and argument flag specifying the mode to be set:

- **Simulation** — This is the normal Calibre LFD mode that applies check measurements on PV-bands.
- **noSimulation** — For No-Simulation Mode, Calibre LFD runs checks and gets error markers without any simulation being applied.
- **Hybrid** — For Hybrid Mode, Calibre LFD attaches a property to the output layer of LFD::StructureOptimizer that indicates whether the pattern matching database patterns exactly match the layout patterns.

Examples

```
LFD::Begin
LFD::SetSTOMode -layer metal -mode noSimulation
LFD::StructureOptimizer -layer metal -file $::env(PM_PDL_FILE) \
    -interactDistance $::env(PM_INTERACTION_DISTANCE) \
    -checkName MinSpaceCheck1
LFD::MinSpaceCheck -layer metal -subwindow 1 -minDRCspace 0.10 \
    minLFDspace 0.05 -checkName MinSpaceCheck1 -database $DB \
    -property Space
LFD::END
```

SimConfig

Configures the simulations to be performed as part of Calibre LFD.

Usage

SimConfig

```
[-tilemicrons tile_size]  
[-modelpath pathname]  
[-imagegrid numerator denominator]  
[-setupfile_number setup_file_num]  
[-pixel unit]  
[-layer layer_name]  
[-simulationOptions sim_options]
```

Description

Configures the simulations to be performed as part of Calibre LFD. The simulation settings defined through this command are defined within the PDK, through the [addPDKLayer](#) command. The values supplied using the SimConfig command overrides those settings for all simulations performed during the run.

You can override those settings for specific simulations by using the keywords provided in the [PVband](#) command call inside the TVF control file, if the security setting support those kinds of overrides.

Arguments

- **-tilemicrons *tile_size***

Optional keyword and argument, specified in microns, defining the tile size to use for simulations.

- **-modelpath *pathname***

Optional keyword and argument defining the parent directory containing the optical, resist, and etch models used for simulation.

- **-imagegrid *numerator denominator***

Optional keyword and arguments used to define the Calibre OPCverify setup file image grid as a ratio using the Calibre OPCverify **imagegrid aerial** setup command. The image grid can be specified at most once in a setup file. The use of the -imagegrid option replaces the use of the deprecated -pixel option.

- **-setupfile_number *setup_file_num***

Optional keyword and argument used to define separate setup files for different layers for Calibre OPCverify setup file generation and image simulation. The default setup file number is -1. An error is generated if the -setupfile_number option is used with the deprecated -pixel option.

- `-pixel unit`

Note

 The `-pixel` argument is deprecated with the 2016.3 release. You should use the `-imagegrid` option to define the Calibre OPCverify image grid and the `-setupfile_number` option for controlling setup file generation per layer specification.

Optional keyword and argument used to define the pixel unit size for use during image simulations performed when generating PV-bands.

- `-layer layer_name`

Optional keyword and argument used to define a list of layers that have all the defined simulation configurations applied to only them. If not specified, the command applies to all layers.

- `-simulationOptions sim_options`

Optional keyword and argument used to define extra simulation options (for example, `processing_mode`). Those options should be Calibre OPCverify setup commands.

Examples

Example 1

This example shows the use of the LFD::SimConfig command for a specific simulation.

```
LFD::Begin
LFD::SimConfig -modelpath $env(MODEL_DIR)
LFD::SimConfig -layer "poly contact" -imagegrid "1 2"
LFD::SimConfig -mask_sample_grid rsm
```

Example 2

This example shows the use of the `-imagegrid` option to define the Calibre OPCverify image grid.

```
LFD::SimConfig -layer "poly contact" -imagegrid "1 4"
```

Example 3

This example shows the use of the `-setupfile_number` option to control the Calibre OPCverify setup file generation for different layers. Layers M1 and M2 are both in setup file 1, and layer C1 is defined in setup file 2.

```
LFD::SimConfig -layer M1 -setupfile_number 1
LFD::SimConfig -layer M2 -setupfile_number 1
LFD::SimConfig -layer C1 -setupfile_number 2
```

StructureOptimizer

Reduces layout data set being passed into Calibre LFD simulation, improving overall runtime performance. Pattern matching is used to find matches for patterns containing possible lithographic hotspots.

Usage

```
StructureOptimizer
  -layer layernames
  -macro macro_name
  -file macro_file_path
  -layerOut return_layer_name
  [-interactDistance interactDistance]
  [-patternKey key_list]
  [-property property_list]
  [-tilesize {tile_size | auto}]
  [-pdk pdkname]
  [-pdkSTOName pdkSTOName]
  [-invalidPropertyIndex value]
  [-expandCells value]
  [-progress_meter {on | off | edge}]
  [-security yes]
  [-checkName check_name]
```

Description

The Calibre LFD Structure Optimizer (STO) functionality runs pattern matching in designs to find matches for patterns containing possible lithographic hotspots. This command enables you to reduce the layout data set passed into Calibre LFD simulation, thus improving the overall runtime performance of Calibre LFD. The LFD::StructureOptimizer command is intended for cases in which problematic areas are known. This command provides a rapid method for evaluating a hotspot library in new databases under test, thus significantly reducing the runtime of Calibre LFD checks.

The command detects edge-based configurations within any arbitrary layout inside of Calibre. These configurations are referred to as patterns or structures. Collections of patterns are stored in a Siemens EDA-specific format known as Pattern Matching Database (PMDB). This PMDB file is compiled into an SVRF file that is referred to in LFD::StructureOptimizer command calls. This is a binary format defining geometric patterns using a relative coordinate infrastructure. Once a foundry identifies potential problematic structures that are possible lithographic hotspots, a hotspot library can be developed to capture these potentially problematic areas of interest.

Note

-  For an overview of the commands used to limit the cells checked by Calibre LFD to improve performance, see [Calibre LFD Fast Mode](#). To provide for faster execution, the LFD Fast Mode uses the Calibre Pattern Matching software to identify only those layout areas that require detailed lithographic simulation.
-

The LFD::StructureOptimizer command has three modes of operation, and each mode can be used in both a PDK and non-PDK flow.

- **Simulation Mode** — Calibre LFD applies checks on layers' process variability PV-bands. Contours of the PV-bands come from the Calibre OPCverify tool's simulation using optical, resist, and etch models you define. This is the normal Calibre LFD mode that applies check measurements on PV-bands.
- **No-Simulation Mode** — Calibre LFD runs checks and gets error markers without any simulation being applied. Problematic areas are determined using pattern matching methods. One of those techniques is STO, which obtains good accuracy and performance by identifying structure-corresponding matches. Using this matching technique, the Calibre LFD package bypasses the simulation step and receives the error markers returned by the LFD::StructureOptimizer function call.
- **Hybrid Mode** — Calibre LFD attaches a property called *match_type* to the output layer of LFD::StructureOptimizer that indicates whether the patterns exactly match the layout patterns. The property is set to 0 if the library pattern does not contain any constraints. It is set to 1 if the library pattern does contain constraints, and the match result depends on the constraints to find a match. It is set to 2 if the library pattern contains constraints, and the match results are found without any constraint.

Calibre LFD does simulations and run checks only on the non-exact matched patterns. The output layer of the check is the exact matched patterns combined with the error markers discovered from running the check on the non-exact patterns as the region of simulation.

This command is for use both with and without a PDK.

Arguments

- **-layer *layernames***

Required keyword and argument specifying at least one input layer. This is the layer(s) containing the geometries to be searched. The order of layers in the command is associated with the layers in the patterns. The first layer in the pattern file is searched for on **-layer1**. If more than one layer is specified, then the second layer in the pattern file is searched for on **-layer2**, and so forth.

Note

 The pattern file can contain patterns with different numbers of layers provided that the command keys allow selection of patterns with the number of layers matching the number of layers in the input. Selection of a pattern with a different number of layers than in the command causes an abort.

- **-macro *macro_name***

Required keyword and argument defining the input macro name. This option causes the command to use the pattern matching system for its operations, where the input macro defines the input pattern library and the system configuration.

- **-file *macro_file_path***

Required keyword and argument defining the path to the SVRF file where the input macro is defined.

- **-layerOut *return_layer_name***

Required keyword and argument specifying the name of the output layer containing individual pattern markers indicating matches with the library patterns.

- **-interactDistance *interactDistance***

Optional keyword and argument specifying an input parameter to the tiling mechanism. The value specified instructs Calibre LFD to look outward from a tile boundary by the user-specified amount to not miss any patterns that could exist at or across the boundary. During the matching process, there might be duplicate matches found in the border regions. Duplicates are resolved by the overlap merging process.

If the interaction distance is not specified, the largest pattern diameter in *x* or *y* is used.

Caution

 Your *interactDistance* must be 1nm larger than the largest pattern diameter in *x* or *y* of any pattern within the used pattern file. Otherwise, you risk a potentially-missed pattern at a tile boundary.

- **-patternKey *key_list***

Optional keyword and argument specifying one or more key names. Only patterns having keys matching all those in the command line are selected for matching. Selected patterns may have keys in addition to those in the command line. If no keys are specified in the command, then all patterns are selected. Any pattern with fewer keys than in the command is not selected.

- **-property *property_list***

Optional keyword and argument specifying whether properties are added to the output marker shapes from LFD::StructureOptimizer. Properties first exist within the pattern file for any given pattern. Upon finding a match for a particular pattern with properties, this

command outputs the matching marker and attaches properties to it. All property names are allowed. Properties attached to the output layer are classified into two categories:

- **Calibre LFD Properties** — describe the output error-marker of Calibre LFD checks, such as PVI, Space, MinCD, and Area. If no simulation is run, Calibre LFD properties need to exist in the pattern file. If the patterns come from simulation, Calibre LFD property values are calculated based on simulation results.
- **Matching Properties** — provide the match characteristics of the output markers on locating a match. Only one of these matching properties, “*sto_mode*”, needs to exist in the pattern file as you determine its value. The underlying engine establishes the matching property value as a function of the matched output, taking into account the original pattern definition. The matching properties are *match_source*, *match_type*, and *sto_mode*, as described as follows:
 - *match_source*

This property determine whether the pattern comes from the pattern file. If the pattern comes from the pattern file, *match_source* is set to 1, and otherwise it is set to 0. This property is only useful in Simulation mode. In Simulation mode, if an error marker sits outside of the pattern region, this property is set to 0, indicating that the error-marker is out of the pattern file scope.

In No-Simulation and Hybrid modes, this property always equals 1, as there is no simulation and all the patterns come from the pattern file.

- *match_type*

This property describes the type of matching between the layout pattern and the pattern file pattern. If the layout pattern exactly matches that in the pattern file, this property is set to 0. It is set to 1 if the layout pattern does not exactly match the pattern file patterns. If you have a constrained pattern and the match exactly matches the base pattern, then it is set to 2.

In hybrid mode, *match_type* is meaningless as the output error-marker does not come from the pattern file. In this case, the package assigns a value of -1 to this property.

- *sto_mode*

This property determines whether each pattern in the pattern file is directed towards simulation or no-simulation. The property is used only when running the command in hybrid mode, where if this property is not used, the package runs the simulation only on the region formed by the non-exact matched patterns, which are determined by the *match_type* property. Set this property if you want to direct or force a certain pattern to be used in simulation, regardless of whether it is non-exactly or exactly matched.

For example, if you set *sto_mode* to 1 for a certain pattern in the pattern file, any pattern in the layout that matches that pattern is never used in simulation,

regardless of whether the layout pattern exactly matches the pattern file pattern. If you set *sto_mode* to 2 for a certain pattern in the pattern file, any pattern in the layout that matches that pattern is used in simulation regardless of the matching type of that pattern.

- **-tilesize {*tile_size* | auto}**

Optional keyword and argument that is used to control the tile size. This value is an integer, and the default value is 25 u.

- **-pdk *pdkname***

Optional keyword and argument that is used to get the PDK in which the STO is defined. If -pdk exists, also specify the -pdkSTOName option.

- **-pdkSTOName *pdkSTOName***

Optional keyword and argument that is used to get the STO defined by the foundry in the PDK. If -pdkSTOName exists, also specify the -pdk option.

- **-invalidPropertyIndex *value***

Optional keyword and argument used to enable you to pass in a real value indicating that a property does not exist in the pattern file pattern, to avoid the possibility of collisions.

- **-expandCells *value***

Optional keyword and argument that controls the *expand_small_cells* statement in the setup file.

- **-progress_meter {on | off | edge}**

Optional argument defining whether command progress is displayed in the transcript. You can provide one of the following options as an input to this argument:

- on — Report progress for all operations.
- off — No progress meter output.
- edge — An edge-based progress meter.

- **-security yes**

Optional argument defining security privileges. If defined, the SVRF is encrypted. The default is for the SVRF to not be encrypted.

- **-checkName *check_name***

Optional keyword and argument that is used only in the No Simulation and Hybrid flows to map the output of an LFD::StructureOptimizer command to a certain check.

Examples

These examples show how to use the LFD::StructureOptimizer command in Simulation, No-Simulation, and Hybrid modes, with examples using both PDK and non-PDK flows.

Simulation Mode

You use the Simulation option to apply check measurements on PV-bands.

Note

 The LFD::addPDKSTO command is used in this flow.

Example 1

This example uses the LFD::StructureOptimizer command in Simulation mode, using a PDK flow.

In the PDK:

```
LFD:::createPDK {  
    -name generic65  
    -version 0.2  
    -security {  
        Checks 00  
        Processlayers 11  
        STO 01  
    }  
}  
LFD:::addPDKSTO {  
    -name STO1  
    -pdk generic65  
    -definition {  
        -property {MinCD PVI}  
        -macro msc_sto  
        -file ./macro.svrf  
        -interactDistance 2.0  
    }  
}
```

In the rule file:

```
LFD:::Begin  
LFD:::LoadPDK ./generic65  
LFD:::StructureOptimizer -layer metal1 -pdkSTOName STO1 \  
    -pdk generic65 -layerOut STOoutput  
LFD:::IncrementalSelect -layer metal1 -layerOut IS_Output \  
    -halo 0.5 -checkRegions STOoutput -minWidth 0.09  
LFD:::RegisterLayer -pdk generic65 -Designlayer IS_Output \  
    -Processlayer pdklyr_metal1  
LFD:::LFDregion -layer IS_Output -region STOoutput -halo 0.5  
// Running lfd checks  
LFD:::MinSpaceCheck -layer IS_Output -pdkCheckName METAL1_MSC\  
    -database $lfdErrorDatabase  
LFD:::End
```

Example 2

This example uses the LFD::StructureOptimizer command in Simulation mode, using a non-PDK flow.

In the rule file:

```
LFD::Begin
LFD::StructureOptimizer -layer metal1 -macro sto_patterns1 \
    -file ./macro1.svrf -interactDistance 2.0 -layerOut STOoutput
LFD::IncrementalSelect -layer metal1 -layerOut IS_Output \
    -checkRegions STOoutput -minWidth 0.09 -halo 0.5
LFD::RegisterLayer -Designlayer IS_Output
LFD::LFDregion -layer IS_Output -region STOoutput -halo 0.5
LFD::PVband -layer IS_Output \
    -layerRET lfd_opc_1 \
    -foreground {{attenuated 0.06}} \
    -background clear \
    -pixel 0.02 \
    -modelDir {./setup/models} \
    -resistFile {resist.mod} \
    -opticalSpanList {{P1W_D0 P1W_D50}} \
    -doseSpanList {{1.0000 1.0200}}
// Running lfd checks
LFD::MinSpaceCheck -layer IS_Output -subwindow 1 -database $DB \
    -minDRCspace 0.10 -minLFDspace 0.05 \
    -checkName MSC_METAL1
LFD::End
```

No-Simulation Mode

In No-Simulation mode, Calibre LFD runs checks and get error markers without simulation being applied. The tool bypasses the simulation step, and obtains the error markers returned by the LFD::StructureOptimizer command.

Setting the STO mode of a layer to No-Simulation is accomplished by adding a mapping between the LFD::StructureOptimizer command and the check that bypasses the simulation. To run in No-Simulation (or Hybrid mode) using a PDK flow, you need to set the STO mode using the LFD::addPDKLayer -STOmode flag. The role of this argument is to set the STO mode of the layer defined in the Calibre LFD kit. The valid STO modes are Simulation, No-Simulation, or Hybrid, which are indicated through the -STOmode values of 1, 2, and 3 respectively. For example:

```
LFD::addPDKLayer {
    -name pdklyr_METAL1
    -pdk 45nm
    -STOmode 2 // No-Simulation mode
    ...
```

This code sets the STO mode of pdklyr_METAL1 layer to No-Simulation. Consequently, in the Calibre LFD run no simulations are performed on this layer and no PV-bands or contours are generated. Any check that operates on that layer bypasses the simulation and get its error markers from the LFD::StructureOptimizer command, which is mapped to the check.

Setting the simulation mode is accomplished by setting the LFD::addPDKLayer -simEngine flag inside the PDK. For example:

```
LFD::addPDKLayer {  
    -name pdklyr_METAL1  
    -pdk 45nm  
    -STOmode 2  
    -simEngine D2C // Simulation mode is D2C  
    ...}
```

The -simEngine option can be set to either D2C or PVband. D2C means that Calibre LFD calls the LFD::Drawn2Contour command to do simulations for this layer. PVband means that Calibre LFD calls the LFD::PVband command to do simulations for this layer.

Simulations are per layer so each layer has only one -STOmode and only one -simEngine.

Mapping between an LFD::StructureOptimizer call and a certain check is accomplished inside the PDK, using the LFD::addPDKSTO -pdkCheckName sub-option. For example:

```
LFD::addPDKSTO {  
    -name m1_msc_STO  
    -pdk 45nm  
    -pdkLayer pdklyr_METAL1  
    -definition {  
        -property {}  
        -macro m1_msc  
        -file ./m1_msc_macro.svrf  
        -pdkCheckName METAL1_MSC  
    }  
}  
  
LFD::addPDKCheck {  
    -name METAL1_MSC  
    -pdk 45nm  
    -type MinSpaceCheck  
    -security 11  
    -definition {  
        -subwindow 1  
        -property Space  
        -minDRCspace 0.100  
        -minLFDspace 0.064  
    }  
}
```

The PDK mapping shown in this example states that the check METAL1_MSC is used only with LFD::StructureOptimizer command in either No-Simulation or Hybrid mode. In the rule file, you only need to call the check without needing to call LFD::StructureOptimizer. This is demonstrated in the following example.

Example 3

This example uses the LFD::StructureOptimizer command in No-Simulation mode, using a PDK flow.

In the PDK:

```
LFD::addPDKLayer {
    -name pdklyr_metal1
    -pdk 45nm
    -STOmode 2
    -simEngine PVband
    -processInfo {
        Security {
            OpticalModels 00
        ...
    }

    LFD::addPDKSTO {
        -name m1_msc_STO
        -pdk my_45nm
        -pdkLayer pdklyr_metal1
        -definition {
            -property {}
            -macro m1_msc
            -file m1_msc.svrf
            -pdkCheckName METAL1_MSC
        }
    }

    LFD::addPDKCheck {
        -name METAL1_MSC
        -pdk my_45nm
        -type MinSpaceCheck
        -security 11
        -definition {
            -subwindow 1
            -property Space
            -minDRCspace 0.100
            -minLFDspace 0.064
        }
    }
}
```

In the rule file:

```
LFD::Begin
LFD::LoadPDK ./fa_45nm
LFD::RegisterLayer -pdk fa_45nm -Designlayer metal1 \
    -Processlayer pdklyr_METAL1
LFD::MinSpaceCheck -layer metal1 -pdkCheckName METAL1_MSC \
    -database $lfdErrorDatabase
LFD::End
```

From the rule file code, Calibre LFD knows to load the PDK, which sets the STO mode of the layer pdklyr_metal1 to No-Simulation by setting the LFD::addPDKLayer -STOmode flag to 2. The mapping between the STO m1_msc_STO and the check METAL1_MSC is accomplished with the LFD::addPDKSTO -pdkCheckName METAL1_MSC call. You then call the check in the rule file, without first calling the LFD::StructureOptimizer command, as it is called automatically. This example check bypasses the simulation and have error markers returned from the mapped LFD::StructureOptimizer command.

Note

- Use the LFD::RegisterLayer command in No-Simulation and Hybrid modes while using a PDK flow to register the drawn layer in the rule file prior to its usage (as illustrated in the previous example).
-

Example 4

This example uses the LFD::StructureOptimizer command in No-Simulation mode, using a non-PDK flow.

To set the STOmode, you need to use the LFD::SetSTOMode command. It sets the STO mode of the layers, or set it globally for all layers defined in the Calibre LFD kit.

After setting the mode, you invoke the LFD::StructureOptimizer command to obtain matched patterns.

Remember that you must perform an explicit “source” to incorporate the Calibre LFD pattern file created using LFD::StructureOutput.

The following rule file fragment demonstrates No-Simulation mode, using a non-PDK flow:

```
LFD::Begin
LFD::SetSTOMode -layer metal -mode noSimulation
LFD::StructureOptimizer -layer metal -macro m1_msc \
    -file ./macro.svrf -interactDistance $::env(PM_INTERACTION_DISTANCE) \
    -checkName MinSpaceCheck1
LFD::MinSpaceCheck -layer metal -subwindow 1 -minDRCspace 0.10 \
    -minLFDspace 0.05 -checkName MinSpaceCheck1 -database $DB \
    -property Space
LFD::END
```

The LFD::SetSTOMode call sets the metal layer to No-Simulation mode, which means that the Calibre LFD package for the metal layer bypasses the simulation step. The LFD::StructureOptimizer command in turn gets the violation layer and then maps itself to a certain check through its -checkName option holding the name of the check (MinSpaceCheck1).

Hybrid Mode

The Hybrid mode uses a property (match_type) attached to the output layer of LFD::StructureOptimizer. The match_type is set to 0 if the library pattern does not contain any constraints. It is set to 1 if the library pattern does contain constraints, and the match result depends on the constraints to find a match. It is set to 2 if the library pattern does contain constraints, and the match result are found without any constraint at all.

The Calibre LFD tool does simulations and run the check (which is mapped to that STO) only on the non-exact matched patterns. Finally, the output layer of the check is the exactly matching patterns combined with the error markers that come from running the check on the non-exact patterns.

To run in Hybrid mode for a certain layer, first set the STO mode of this layer to Hybrid. Second, perform mapping between an LFD::StructureOptimizer command and the check that takes the output layer of the STO.

Setting Hybrid mode parameters is accomplished using the LFD::addPDKLayer -STOInfo option. Calibre LFD applies simulations only on the non-exact matched patterns coming from STO. This is accomplished by calling LFD::IncrementalSelect followed by the LFD::LFDregion command, where both the -region option to LFD::LFDregion and the -checkRegions option to LFD::IncrementalSelect are given the name of the layer holding the non-exact matched patterns. Other options to these two commands, -halo and -minWidth, vary the simulation results. Calibre LFD assigns these options default values, however you cannot overwrite the -minWidth option which has a default value of 0.01 um. You can overwrite the -halo option in both commands using the -STOInfo option in the LFD::addPDKLayer command, inside the PDK. An example follows:

```
LFD::addPDKLayer {  
    -name pdklyr_METAL1  
    -pdk 45nm  
    -STOmode 3  
    -STOInfo {  
        ISHalo 0.52  
        lfdRegionHalo 0.53  
        STORegionsSizing 0.3  
    }  
    ...  
}
```

Example 5

This example uses the LFD::StructureOptimizer command in hybrid mode using a PDK flow.

In the PDK:

```
LFD::addPDKLayer {
    -name pdklyr_metal1
    -pdk 45nm
    -STOmode 3
    -simEngine PVband
    -processInfo {
        Security {
            OpticalModels 00
        ...
    }
}
LFD::addPDKSTO {
    -name m1_msc_STO
    -pdk 45nm
    -pdkLayer pdklyr_metal1
    -definition {
        -property {}
        -macro m1_msc
        -file ./m1_msc_macro.svrf
        -pdkCheckName METAL1_MSC
    }
}
LFD::addPDKCheck {
    -name METAL1_MSC
    -pdk 45nm
    -type MinSpaceCheck
    -security 11
    -definition {
        -subwindow 1
        -property Space
        -minDRCspace 0.100
        -minLFDspace 0.064
    }
}
```

In the rule file:

```
LFD::Begin
LFD::LoadPDK ./45nm
LFD::RegisterLayer -pdk 45nm -Designlayer metal1 \
    -Processlayer pdklyr_METAL1
LFD::MinSpaceCheck -layer metal1 -pdkCheckName METAL1_MSC \
    -database $lfdErrorDatabase
LFD::End
```

In the rule file, Calibre LFD loads the PDK. The STO mode of the layer pdklyr_metal1 is set to Hybrid by setting the LFD::addPDKLayer -STOmode flag to 3.

Note

-  Exact matched patterns in the output layer of STO have the property *match_type* set to 0.
Non-exact matching patterns are set to 1.
-

Example 6

This example uses the LFD::StructureOptimizer command in Hybrid mode, using a non-PDK flow.

```
LFD::Begin
LFD::SetSTOMode -layer metal1 -mode Hybrid
LFD::PVband -layer metal1 \
    -layerRET lfd_metal1_mopc \
    -foreground {{attenuated 0.06}} \
    -background clear \
    -pixel 0.02 \
    -modelDir {./setup/metal1/models} \
    -resistFile {metal1.mod} \
    -opticalSpanList {{P1W_D0 P1W_D50}} \
    -doseSpanList {{1.0000 1.0200}}
LFD::StructureOptimizer -layer metal1 -macro msc_m1 \
    -file ./msc_m1_macro.svrf -interactDistance 2.0 -checkName MSC_METAL1
LFD::MinSpaceCheck -layer metal1 -subwindow 1 -minDRCspace 0.10 \
    -minLFDspace 0.05 -checkName MSC_METAL1 -database $DB
LFD::END
```

The LFD::SetSTOMode command sets the mode of the metal layer to Hybrid. This means that the Calibre LFD package for the metal layer does simulations and runs the check only on non-exact matched patterns within the simulation region.

StructureOutput

Takes hotspot and target layers and creates a pattern description file.

Usage

StructureOutput

```
-layer target_layer_name
{-hotspotLayer hotspot_layer} | {-checkName LFD_check_name}
-outputFile output_file_name
[-patternRadius pattern_radius]
[-maskLayer pattern_mask_layer]
[-patternKeys key_list]
[-patternProperties property_list]
[-maxSearch max_search]
[-maxLength max_length]
[-tilesize {tile_size | auto}]
[-patternPrefix pattern_prefix
[-indexProperty iproperty_name] [-indexStartNum start_num]]]
```

Description

Takes hotspot and target layers and creates a pattern matching database (PMDB) file. The pattern is anchored based on target shapes surrounding the hotspot shape. Duplicate patterns are ignored, including patterns that have different orientations. A pattern file can come out empty, if the Calibre LFD check results in no hotspots.

Allows for an optional usage where a pattern mask layer is defined. This is simply a route to create patterns via SVRF, where the shape of the pattern is already defined. In this case, the mask defines the pattern footprint using polygons on the mask layer. Each polygon defines one pattern. Target shapes under the mask make up the pattern shapes. If specified, the hotspot shapes under the mask layer become the pattern marker layer. If no hotspot shapes are found under the mask, the mask defines the marker extent. Duplicate patterns are removed.

Note

 The LFD::StructureOutput command does not run in MTflex mode. It only runs in ST and MT modes.

Arguments

- **-layer *target_layer_name***

Required keyword and argument defining the name of the layer used to create the patterns.

- **{-hotspotLayer *hotspot_layer*} | {-checkName *LFD_check_name*}**

Takes a hotspots layer and creates the output patterns which are the layer polygons around these hotspots. You determine the hotspot layer by using either the **-hotspotLayer** or the **-checkName** option. The **hotspot_layer** for the **-hotspotLayer** option is the layer that holds

the hotspot polygons. The **LFD_check_name** for the **-checkName** option is the check name where the hotspots are the output hotspot layer from the Calibre LFD check whose argument value matches the name given to this option.

- **-outputFile *output_file_name***

Required keyword and argument defining the name of the output Pattern Matching Database (PMDB) file.

- **-patternRadius *pattern_radius***

Optional keyword and argument defining the amount to grow the hotspot seed to make the pattern from the pattern center. If not mentioned, the package assigns *pattern_radius_um* a default value of 0.2 um. The **-patternRadius** option cannot be used with the **-maskLayer** option.

- **-maskLayer *pattern_mask_layer***

Optional keyword and argument defining the mask layer name. The **-patternRadius** and **-maxSearch** options cannot be used with the **-maskLayer** option.

- **-patternKeys *key_list***

Optional keyword and argument defining a list of keys to be added to all output patterns.

- **-patternProperties *property_list***

Optional keyword and argument defining a list of [Calibre LFD Check Properties](#) with values to be added to all output patterns.

- **-maxSearch *max_search***

Optional keyword and argument defining the distance to search from a given hotspot to a nearby pattern seed location.

If **-maxSearch** is not set, the package assigns it a default value of 0.2 um. The **-maxSearch** option cannot be used with the **-maskLayer** option.

- **-maxLength *max_length***

Optional keyword and argument used to force the creation of multiple seed lines for a hotspot, useful in the case of long hotspots shapes. Hotspots that have a width or height larger than *max_length* are broken into marker segments at *max_length* intervals. The application attempts to find an anchor position for each marker segment close to the pattern center. If not set, the tool assigns a default *max_length* of 0.2 um.

- **-tilesize {*tile_size* | auto}**

Optional keyword and argument that is used to control the tile size. This value is an integer, and the default value is 100.

- **-patternPrefix *pattern_prefix***

Optional keyword and argument defining the name prefix to be added to all output pattern names. The pattern name is the prefix followed by an integer beginning with 1 for the first pattern output. The number is incremented by 1 for each subsequent pattern.

- **-indexProperty *iproperty_name***

Optional keyword and argument used to begin the number portion of the pattern name at an integer higher than 1. This is useful in case you want to merge different pattern library files, where you could use this option to have non-overlapping pattern names before merging. This option can only be used with the **-patternPrefix** option.

- **-indexStartNum *start_num***

Optional keyword and argument used to copy the number chosen for the pattern name into a pattern property with the given name. This option can only be used with **-patternPrefix**.

Examples

In this example, LFD::StructureOutput creates the output patterns by selecting the polygons of the “poly” layer around the hotspots determined by the input layer “poly_widthViolation”. The patterns are then saved to the output file “poly_width_viol.pmdb”, which are subsequently used for LFD::StructureOptimizer purposes.

```
LFD::StructureOutput \
    -layer poly \
    -hotspotLayer poly_width_violation \
    -outputFile poly_width_viol.pmdb
```

Here, LFD::StructureOutput creates the output patterns by selecting the polygons of the “metal1” layer around the mask layer. It generates the output error marker of each pattern using the input hotspot layer. In this case, the hotspot layer is the output error markers of the Calibre LFD check “msc_metal1”. Then the command saves these patterns to the output file “msc_metal1.pmdb”, which could be used next for pattern matching purposes. The keys “key1” and “key2” are added to all the patterns in the pattern file.

The two properties “A” and “B” are added to all patterns in the output pattern file. -indexStartNum is used to control the number portion of the pattern name. -indexProperty is used to attach a property to each pattern holding that value. The first pattern in the pattern file is named “metal1_msc_50” instead of “metal1_msc_1”, with property “pattern_count” equal to 50.

```
LFD::StructureOutput \
    -layer metal1 \
    -checkName msc_metal1 \
    -maskLayer mask_m1 \
    -outputFile msc_metal1.pmdb \
    -patternPrefix metal1_msc_ \
    -patternKeys {key1 key2} \
    -patternProperties {A 0.2 B 0.15} \
    -indexProperty pattern_count \
    -indexStartNum 50
```

TopCellOptimizer

Aids checking geometries at top level of the design hierarchy.

Usage

TopCellOptimizer

```
-layer input_layer_name
-layerOut return_layer_name
[-layerOut2 return_layer_name2]
-halo size
[-minWidth value]
-cell cell_name
```

Description

Aids checking of geometries at the top level of the design hierarchy by limiting Calibre LFD execution to layout features located at the top cell level. Typically, designers at the place and route stage in the design flow cannot control the IP design block and standard cell geometries.

However, at this point, what designers have control over is how the router places connecting shapes. Therefore, using the TopCellOptimizer command allows them to run Calibre LFD on what the router has already laid down.

The command returns a derived polygon layer identifying areas requiring further investigation. Using this command reduces the amount of area that is actually processed, which improves performance. This command is useful for running Calibre LFD full chip where only routing layers need to be checked.

If you specify -layerOut2, then all shapes that exist under the halo region are output. If you specify both -layerOut2 and -minWidth, then sliver removal applies to layerOut2.

This command is not for use in a PDK.

Note

 The expected command flow is:

TopCellOptimizer > IncrementalSelect > LFD::LFDregion > LFD::RegisterLayer

Arguments

- **-layer *input_layer_name***

Required keyword and argument specifying the name of the layer containing the design data under investigation. The *input_layer_name* can be an original layer or layer set, or a derived polygon layer.

- **-layerOut *return_layer_name***

Required keyword and argument specifying the name to assign to the primary output layer created by this operation. This return layer contains polygon regions defining the portions of the *input_layer_name* of interest for further processing.

- **-layerOut2 *return_layer_name2***

Optional keyword and argument specifying the name to assign to the secondary output layer created by this operation. The layer contains original design data of interest for further processing. That is, the layer contains the portions of the *input_layer_name* that are inside of the polygon regions as defined by the **-layerOut *return_layer_name***.

- **-halo *size***

Required keyword and floating-point number, specified in user units, and greater or equal to 0.0. The *size* value refers to the amount to size the selected shapes by, to generate a larger area of interest

- **-minWidth *value***

Optional keyword and argument used to turn on polygon sliver removal of design data. The *value* argument is a floating point number specified in user units. This is the trigger to turn on polygon sliver removal. The same sliver removal rules from IncrementalSelect apply here. This option is only used when -layerOut2 is also specified.

- **-cell *cell_name***

Required keyword and argument in which *cell_name* is a valid existing cell name in the database. Only one cell name can be specified per call. All metal that is found at this level of the cell within the cell boundaries are to be returned to the caller. Any geometries found outside or below this cell in the hierarchy are to be removed.

VariabilityIndices

Generates variability indexes (scores) for the specified layer.

Usage

VariabilityIndices

```
-layer input_layer_name
-windowSize {x | xy | extents | custom}
-database db_name
[-property
{PVI | DVI}]
[-subwindow expr_number]
```

Description

Calculates DVI and PVI scores and writes them to the database specified through **-database**.

- The DVI — How likely it is that the variations in printing negatively impacts yield.

$$DVI = \sum \frac{AREA(LFDerrors)}{AREA(window)}$$

- The PVI score — The degree to which printing is impacted by changes in the process.

$$PVI = \sum \frac{AREA(pvband(layer))}{AREA(layer)}$$

The PVI score is calculated differently based on if the -subwindow option is used:

- With -subwindow *or* without -subwindow and only one subwindow defined in the **PVband** command:

$PVI = \text{Area (Absolute PV-band of that given subwindow)} / \text{Area (drawn Layer)}$

- Without -subwindow *and* more than one subwindow defined in the LFD::PVband command:

$PVI = 0.7 * \text{Area (Absolute PV-band of subwindow #1)}$

$+ 0.5 * \text{Area (Absolute PV-band of subwindow #2)} / \text{Area (drawn Layer)}$

Note

 If the drawn layer has more than two subwindows with associated checks, the first two subwindows that have checks associated with them are used for PVI calculations.

Arguments

- **-layer *input_layer_name***

Required keyword and argument defining the name of the layer you are checking. This is the layer for which PV-bands are generated.

- **-windowSize {*x* | *xy* | *extents* | *custom*}**

The size of the windows into which the cell is to be partitioned before scoring. When generating scores, the tool scores each window separately.

- ***x*** (a single value) — Defines the same value for both width and height.
- ***xy*** (two values) — The first defines the width, and the second defines the height. When specifying two values, you must enclose them in braces ({}).
- ***extents*** — Instructs the operation to calculate one PVI and one DVI reflecting the score for the entire region for which the PV-bands are generated (no partitioning performed).
- ***custom*** — Instructs the operation to calculate PVI and DVI scores for each of the regions defined through the LFDregion command.

- **-database *db_name***

Required keyword and argument defining the scores database to which results of the scores are written.

- **-property {PVI | DVI}**

Optional keyword to instruct the check to calculate either PVI only or DVI only. PVI is the ratio of the PV-band area associated to the reference layout. DVI is the ratio of the error marker area to the reference layout. By default both properties are calculated.

- **-subwindow *expr_number***

Optional keyword and argument defining the process variation experiment for which scores are calculated. You must reference individual process variation experiments by their positions in the **-opticalSpanList** and **-doseSpanList** arguments to the **PVband** command used to generate the PV-band data being checked. Thus, ***expr_number*** refers to an index to a list of experiments.

Examples

```
// Calculate separate scores for each 3 by 3 capture window  
  
LFD::VariabilityIndices {-layer active -windowSize 3 \  
-database $lfdErrorDatabase}  
  
// Calculate a single score for the entire layer  
  
LFD::VariabilityIndices -layer metal1 -windowSize extents \  
-database $lfdErrorDatabase -subwindow 4
```

WriteHIF

Writes a Litho hotspot file that adheres to the Cadence Hotspot Interface Format (HIF).

Usage

WriteHIF

```
-HIF HIF_file_name
-rdb LFD_RDB_file_name
-errmap errormap_file_name
[-polygons {emit | skip | extents}]
[-max_severity value]
```

Description

Writes a Litho hotspot file that adheres to the Cadence Hotspot Interface Format (HIF). This file passes hotspot data from Calibre LFD to the Cadence tools.

Arguments

- **-HIF *HIF_file_name***

Required argument specifying the name to use for the HIF file generated by the utility. If a file with this name already exists, the function aborts without generating the HIF file.

- **-rdb *LFD_RDB_file_name***

Required argument specifying the name of the RDB file name containing the Calibre LFD hotspots to be evaluated using the Cadence routing tools.

- **-errmap *errormap_file_name***

Required argument specifying the name of a separate ASCII file containing the 4-tuples detailing the correspondence between a Calibre LFD hotspot check name to Cadence HIF parameters. Refer to the description of the [errmap File Format](#) for the exact syntax.

- **-polygons {emit | skip | extents}**

Optional keyword and value specifying how polygon Calibre LFD hotspot error markers are translated to HIF with the following actions:

- emit — Write all Calibre LFD hotspot polygons as HIF polygons. This is the default operation.
- skip — Skip all Calibre LFD hotspot polygons.
- extents — Convert Calibre LFD hotspot polygons to bounding box extents.

- **-max_severity *value***

Optional keyword and argument specifying the level of process variation that is of interest. Recall that the [PVband](#) command that generates the actual PV-band data allows you to generate multiple sets of PV-bands, each with a different level of process variation. The smallest set of focus and dose settings, reflecting the least process variation, is always defined first, and is referred to as “subwindow 1”. The next set of focus and dose settings,

represent greater process variation, is defined second and referred to as “subwindow 2” and so on.

The `-max_severity value` is the subwindow, or position in the list of process variation experiments reflecting the level of process variation you want to correct for. Any Calibre LFD errors generated based on process conditions outside the specified subwindow are not translated into HIF data and not written to the Litho hotspot file. Thus, `-max_severity` is interpreted as follows:

- 1 — Write all errors found in PV-bands generated for subwindow 1.
- 2 — Write all errors found in PV-bands generated for subwindow 2. Because the PV-bands for subwindow 1 always lies complete inside the PV-bands for subwindow 2, this set includes the `-max_severity 1` errors.
- 3 — Write all errors found in PV-bands generated for subwindow 3. This set includes the `-max_severity 1` and `-max_severity 2` errors.

Examples

```
LFD::WriteHIF -HIF myout.hif -rdb hotsp.rdb \
-errmap explore.errmap -polygons emit -max_severity 1
```

WriteICC

Writes a Litho hotspot file that adheres to the Synopsys IC Compiler format (ICC).

Usage

WriteICC

- ICC *ICC_file_name***
- rdb *LFD_results_file***
- errmap *errormap_file_name***

Description

Writes a hotspot file that adheres to the Synopsys IC Compiler (ICC) format. This command takes the hotspot data generated by Calibre LFD and found in the RDB file and then writes it out in ICC format.

Arguments

- **-ICC *ICC_file_name***

Required argument specifying the name to use for the ICC file generated by the utility. If a file with this name already exists, the function aborts without generating the ICC file.

- rdb *LFD_results_file***

Required argument specifying the name of the RDB file name containing the Calibre LFD hotspots to be evaluated using the Synopsys ICC tool.

- errmap *errormap_file_name***

Required argument specifying the name of a separate ASCII file containing the 4-tuples detailing the correspondence between a Calibre LFD hotspot check name to Synopsys ICC parameters. The errmap file contains information not found in the RDB file that is needed for writing the ICC file. Refer to the description of the [errmap File Format](#) for the exact syntax.

Examples

```
LFD:::WriteICC -ICC myout.icc -rdb hotsp.rdb -errmap explore.errmap
```

PDK Commands

A process-specific Calibre LFD kit is the set of rule files and models required for applying Calibre LFD practices to a specific design. A Calibre LFD kit can take the form of either a single file, called a Process Design Kit (PDK), or a set of individual files and directories. PDKs offer the advantages of portability, security, and ease of use.

For PDK usage, PV-bands can be obtained using the [LoadPDK](#) and [RegisterLayer](#) commands.

Note

 For additional commands, see the [Calibre LFD Checks](#) and [Calibre LFD General-Purpose Commands](#) sections.

Table 7-11. Calibre LFD Commands for use with PDKs

Function	Definition
addPDKCheck	Creates a check template and assigns it a name by which it can be referenced.
addPDKLayer	Defines a layer process to use when investigating a design.
addPDKSTO	Defines a Structure Optimizer (STO) template and assigns it a name by which it can be referenced.
createPDK	Registers the PDK, setting up the appropriate namespace within Calibre LFD.

addPDKCheck

Creates a check template and assigns it a name by which it can be referenced.

Usage

```
addPDKCheck "{"
    -name checkname
    -pdk pdkName
    -type checkType
    [-security {00 | 01 | 10 | 11}]
    [-pdkLayer pdk_process_layer]
    -definition "{" arg1 value1 [argN valueN...] "}"
    "}"
```

Description

Creates a check template and assigns it a name by which it can be referenced.

Arguments

The set of arguments used with this command must be enclosed in braces, “{” ... “}”.

- **-name *checkname***

Required keyword used to supply a name for the check being defined.

- **-pdk *pdkName***

Required keyword and argument defining the PDK with which this check is associated.

- **-type *checkType***

Required keyword, **-type**, followed by the type of check to perform. Must be one of:

AddCustomCheck	AreaCheck
AreaOverlayCheck	CaptureContour
CSG	CSI
InterLayerSpaceCheck	LineEndCheck
MaxAreaVariabilityCheck	MaxCDVariabilityCheck
MinAreaOverlapCheck	MinOverlayCheck
MinSpaceCheck	MinWidthCheck
NonPrintingCheck	OutputBands

- **-security {00 | 01 | 10 | 11}**

A optional keyword and argument defining the security level that applies to this check.
Refer to [Table 7-13](#) for an explanation of these settings.

- **-pdkLayer *pdk_process_layer***
Optional keyword and argument defining the input process layers to STO which should be operated on. It does not operate on any other process layer.
- **-definition “{” *arg1 value1* [*argN valueN...*] “}”**
Required keyword followed by the check arguments, supplied as keyword and value pairs. These check arguments must match the arguments defined for the command in the [Calibre LFD Syntax Descriptions](#).

addPDKLayer

Defines a layer process to use when investigating a design.

Usage

```
addPDKLayer {"  
    -name layer_process_name  
    -pdk pdkName  
    [-STOmode {1 | 2 | 3}]  
    [-STOInfo {"  
        [ISHalo is_halo]  
        [lfdRegionHalo lfd_region_halo]  
        [STORegionsSizing sto_regions_sizing]"}]  
    [-simEngine {D2C | PVband | PrintableBand}]  
    -processInfo {"  
        [Security {" ProcessCorrection rw PvBands rw "}]  
        OpticalModels opt_model_list  
        ResistModels {" res_model_name1 {" res_model1 "} "}  
        [EtchModels {" etch_model_name1 {" etch_model1 "} "}]  
        [ddmFiles {" ddmFileName {" ddmFile "} "}]  
        ProcessCorrection {"CMACRO macroName "}  
        DrawnLayer layer  
        OpcLayers opc_layers_list  
        [TargetLayer {" layer "}]  
        [Targetforeground {"transmission"}]  
        [VisibleLayers visible_layers_list]  
        [Visibleforeground {"transmission"}]  
        opcInLayers opc_in_layers_list  
        [tags2boxes tags_layers_list]  
        [IslandLayers island_layers_list]  
        [AuxLayers aux_layers_list]  
        [modelDir {" path "}]  
        [doublePatterning {yes | trim | split}]  
        [IndependentWindows yes]  
        RetLayers {"mask0 {"  
            layers layer1_list  
            transmission trans_list  
            background bkgd  
            [ddmFiles ddmFilesNamesList]  
            "  
            [mask1...]  
        }"  
        PvBands subwindow_list  
        [Drawn2Contour layer]  
        [PrintableModel {" name printableModelPath | inlineModel "}]
```

```
[PrintableBand subwindow_list]
[PrintableSplitMasks {"split_masks_list"}]
{}
```

Description

Defines a layer process to use when investigating a design using Calibre LFD. The layer process describes the following:

- The RET corrections that are applied to the mask.
- The optical and resist models represent the lithographic process used at the fabrication facility.
- The process variation experiments to be investigated.

Arguments

The set of arguments used with this command must be enclosed in braces, {" ... "}.

- **-name *layer_process_name***

Required keyword used to supply a name for this layer process definition.

- **-pdk *pdkName***

Required keyword and argument defining the PDK with which this layer process is associated.

- **-STOmode {1 | 2 | 3}**

Optional keyword and argument flag specifying the mode is to be set to the StructureOptimizer (STO) mode of the layer defined in the Calibre LFD kit:

- 1 — LFD::StructureOptimizer runs in Simulation mode for this layer.
- 2 — LFD::StructureOptimizer runs in No-Simulation mode for this layer.
- 3 — LFD::StructureOptimizer runs in Hybrid mode for this layer.

- **-STOInfo {}**

Optional keyword and arguments specifying the Hybrid mode STO parameters defined in the Calibre LFD kit:

```
-STOInfo {
    ISHalo is_halo
    lfdRegionHalo lfd_region_halo
    STORegionsSizing sto_regions_sizing
}
```

Three parameters can be defined:

- ISHalo *is_halo*

Optional keyword and argument specifying a value to be passed to the -halo option of the LFD::[IncrementalSelect](#) command. If this option is not set, Calibre LFD assigns it a value of 0.5 um.

- lfdRegionHalo *lfd_region_halo*

Optional keyword and argument specifying a value to be passed to the -halo option of the LFD::[LFDregion](#) command. If this option is not set, Calibre LFD assigns it a default value of 0.5 um.

Note

 The value for ISHalo is greater than that for lfdRegionHalo.

- STORegionsSizing *sto_regions_sizing*

Optional keyword and argument used to size the region of the simulation non-exact region before passing it to the LFD::[IncrementalSelect](#) and LFD::[LFDregion](#) commands. If you want to apply simulation to the region around the non-exact matched patterns, you need to size the simulation region by that value. Also running simulations on the area around non-exact matched patterns allows you to view complete PV-bands around the patterns. If this option is not set, Calibre LFD assigns it a default value of 0.2 um.

Note

 As the STORegionsSizing value increases, the simulation runtime increases as well.

- **-simEngine {D2C | PVband}**

Optional keyword and argument specifying the simulation engine to be used:

- D2C — Calibre LFD calls LFD::[Drawn2Contour](#) to do simulations for this layer.
- PVband — Calibre LFD calls LFD::[PVband](#) to do simulations for this layer.

- **-processInfo “{” ... “}”**

Required keyword introducing the process information for this type of layer. The process information includes the input layers and the simulations to be performed. The process information block must be enclosed in braces.

- **Security “{” *data_type rw* {*data_type rw*}... “}”**

Optional keyword and security settings specific to this process definition. These settings override the default security settings defined through the security settings in the [createPDK](#) statement.

You can define read/write security for any of these *data_types*:

- ProcessCorrection — SVRF or TVF rule file to use for applying any needed process correction to the mask layers before generating the PV-bands.

- PvBands — PV-band data generated by Calibre LFD.

Allowed values for any of the security_levels are 00, 01, 10, and 11. The first digit (or bit) defines the read security and the second digit defines the write security. In general, the settings are interpreted as follows:

Table 7-12. Security Settings within a Layer Process Definition

Setting	Description
read = 0	When Calibre LFD is run, no information related to this type of data is written to the transcript or output to any resulting files.
read = 1	When Calibre LFD is run, all relevant information related to this type of data is written to the transcript or output to any resulting files, including RDB files.
write = 0	Settings defined in this PDK to control the generation of this type of data cannot be modified by you through the TVF rule file used to run Calibre LFD.
write = 1	Settings defined in this PDK to control the generation of this type of data may be modified by you through the TVF rule file used to run Calibre LFD.

- **OpticalModels *opt_model_list***

Required keyword followed by a list of name/model pairs defining the optical models to use for PV-band generation. All optical models must either be supplied inline or reside within the model_dir directory.

Each optical model, whether supplied by name or inline, must be enclosed in braces ({}) and the entire list of name/model pairs must be enclosed in an outer pair of braces.

The expanded syntax is as follows:

OpticalModels “{” *opt_model_name1* “{” *opt_model1* “}” [*opt_model_nameN* “{” *opt_modelN* “}”...“}”]

- **ResistModels “{”*res_model_name1* “{”*res_model1* “}” “}”**

Required keyword defining the resist model to use for PV-band generation. The first argument is a name by which the model is referenced. The second argument is the resist model, and must either be supplied inline or reside within the model_dir directory.

- **EtchModels “{”*etch_model_name1* “{”*etch_model1* “}” “}”**

Optional keyword defining the etch model to use for PV-band generation. The first argument is a name by which the model is referenced. The second argument is the etch model, and must either be supplied inline or reside within the model_dir directory.

- **ddmFiles “{”*ddmFileName* “{”*ddmFile*“}” “}”**

Optional argument defining a list of domain decomposition method (DDM) libraries. Input to ddmFiles can be a file name, an inline DDM model, or an environment variable.

- **ProcessCorrection “{” CMACRO *macroName* “}”**

Required keyword and value specifying the RET recipe block to use for applying any needed process correction to the mask layers before generating the PV-bands. This block can take either of the following forms:

- A macro defined elsewhere in the PDK. The argument to the ProcessCorrection keyword is always the macro name plus any arguments required by the macro.
- Simple SVRF commands. When using this method, you do not benefit from the namespacing provided by using a macro.

The entire RET recipe block must be enclosed in an outer pair of braces ({}). Contact your Siemens representative for support when writing the RET recipe block.

- **DrawLayer *layer***

Required keyword and value specifying the name by which the drawn layer is originally referenced inside of the ProcessCorrection block of code. This layer is very close, if not exact, to what the designer implemented prior to any retargeting or pre-opc processing.

- **OpcLayers *opc_layers_list***

Required keyword and list defining the names of the output layers (corrected using OPC) generated by the RET recipe in the ProcessCorrection block.

The expanded syntax for this keyword is as follows:

opcLayers “{” *layer* [*layerN...*] “}”

Note

 For any of the keywords used to specify zero or more layers, the entire list must be enclosed in braces ({}).

- **TargetLayer “{” *layer* “}”**

Optional keyword and argument to specify the name of a layer referenced in the ProcessCorrection block that is to be the target layer for OPC. If not specified, the target layer for OPC is assumed to be the DrawnLayer. This keyword is only used when an optimizer is defined.

- **Targetforeground “{” *transmission* “}”**

Optional keyword and argument used to define a foreground transmission for the TargetLayer. This keyword is used when the TargetLayer is not included in the list of OpcInLayers. The transmission can be supplied using any of the following forms: dark, clear, {attenuated factor}, {real image}, which are explained in [Table 7-9](#). This keyword is only used when an optimizer is defined.

- **VisibleLayers *visible_layers_list***

Optional keyword and list used to specify the names of one or more optically visible layers referenced in the ProcessCorrection block. This list includes layers such as scattering bar or fill layers. This keyword is only used when an optimizer is defined.

The expanded syntax for this keyword is as follows:

`VisibleLayers {"layer [layerN...] "}"`

- `Visibleforeground {"transmission "}"`

Optional keyword and argument used to define a foreground transmission for the `VisibleLayers`. This keyword is used when the `VisibleLayers` are not included in the list of `RetLayers`. The transmission can be supplied using any of the following forms: dark, clear, `{attenuated factor}`, `{real image}`, which are explained in [Table 7-9](#). This keyword is only used when an optimizer is defined.

- **`opcInLayers` *opc_in_layers_list***

Required keyword and list defining all layers referenced in the `ProcessCorrection` block that are to be passed to LITHO OPC operation in addition to the `DrawnLayer`. That is, the list does not include the name of the `DrawnLayer`. However, it may include the names of layers that are also specified as the `TargetLayer`, `VisibleLayers`, `AuxLayers`, or as `tags2boxes` layers.

The expanded syntax for this keyword is as follows:

`opcInLayers {"layer [layerN...] "}"`

- `tags2boxes` *tags_layers_list*

Optional keyword and list used to specify the names of one or more layers referenced in the `ProcessCorrection` block. These are layers that are designed to hold marker geometries generated by the LITHO tools as a part of the process correction step.

The expanded syntax for this keyword is as follows:

`tags2boxes {"layer [layerN...] "}"`

- `IslandLayers` *aux_layers_list*

Optional keyword and list used to specify the names of one or more user-supplied island layers of special interest or requiring special treatment.

The expanded syntax for this keyword is as follows:

`IslandLayers {"layer [layerN...] "}"`

- `AuxLayers` *aux_layers_list*

Optional keyword and list used to specify the names of one or more additional layers referenced in the `ProcessCorrection` block. Use of these layers varies according to the RET recipe.

The expanded syntax for this keyword is as follows:

`AuxLayers {"layer [layerN...] "}"`

- `modelDir` {"*path* "}"

Optional keyword and argument used to supply the pathname for the model directory, when models are not supplied inline.

- doublePatterning {yes | trim | split}

Optional keyword and argument defining the type of processing to perform when specifying two mask layers and two sets of exposure settings. When this keyword is specified, you must also specify a second mask with the **RetLayers** “{” “}” keyword.

By default, when two mask layers and sets of exposure settings are defined, simulations calculate the PV-bands resulting from exposing the wafer surface twice before developing. That is, expose twice, then develop once.

When this keyword is specified, simulations calculate the PV-bands resulting from exposing and developing using the first mask, then exposing and developing using the second mask.

The “doublePatterning split” option refers to pitch-splitting techniques, and the “doublePatterning trim” option refers to sacrificial printing-assist features, which 32 nm techniques often use in favor of pitch-splitting.

The use of the keyword doublePatterning does not effect the use of EtchModels. When EtchModels are defined, the developing is a two-step process that encompasses separate simulations, one for the resist process and one for the etching process.

- IndependentWindows yes

Optional keyword and argument that modifies the default behavior in which PV-bands are calculated. By default subwindows are assumed to have been constructed as extensions to previously-created subwindows. In other words, the process conditions defined in “{“*focuslist1*“}” “{“*doselist1*“}” “{“*sizelist1*“}” are assumed to be cumulative, and if the first subwindow explicitly defines two process condition and the second subwindow defines two different process conditions, then the PV-band for the second subwindow is calculated based on all four process conditions.

Specifying “IndependentWindows yes” instructs the command to calculate PV-bands based only on those process conditions defined explicitly for the subwindow.

- **RetLayers** “{” “}”

Required keyword used to introduce the mask data for the process layer being defined. This keyword is followed by a minimum of one set of mask data. Supply one mask description when only one mask is required to print the process layer, two mask descriptions when two masks are required to print the process layer, and so on. The complete syntax is as shown below:

```
RetLayers "{" "{mask0 {"  
    layers layer_list  
    transmission trans_list  
    ddmFiles ddm_files_list  
    background bkgd "}" "}"  
    [ "{" mask1 {"mask_description"} ..." }" ]  
}"
```

The secondary keywords that introduce the mask data are as follows:

- mask0 — For the first mask.

- mask1 — For the second mask.

Each mask description must include three sets of arguments:

- **layers *layer_list***

Required keyword and argument used to specify the names by which the individual mask layers used to generate the PV-band. These can be either original or derived polygon layers.

- Supply a single layer name when generating PV-bands from only the OPC-corrected layer.
- Supply a list of layer names when generating PV-bands from the OPC-corrected layer plus SRAF layers and so on. When you specify a list of layer names, you must also specify a list as the argument for -foreground.

- **transmission *trans_list***

Required argument, transmission, followed by the list of optical transmission values for the mask's foreground.

The number of optical transmission values must match the number of layers in *layer_list*, and they must be supplied in the same order with the first *trans* value representing the transmission for the first layer, the second *trans* value representing the transmission for the second layer, and so on. Each transmission value must be enclosed in braces ({}).

When more than one transmission value is specified, the entire list must be enclosed in braces.

The transmission can be supplied using any of the following forms: dark, clear, {attenuated factor}, {real image}, which are explained in [Table 7-9](#).

- **background *bkgd***

Required argument, background, followed by the optical transmission.

The background can be supplied using any of the following forms: dark, clear, {attenuated factor}, {real image}. In addition, for each layer the background must be the compliment of the foreground transmission value.

- **ddmFiles *ddmFilesNamesList***

Optional argument defining a list of domain decomposition method (DDM) libraries. Input to ddmFiles can be a file name, an inline DDM model, or an environment variable.

- **PvBands *subwindow_list***

Required keyword and list defining the process experiments for which PV-bands are to be generated. The process experiments must be supplied as a list of at least one subwindow and

experiment pair. The entire list must be enclosed in a Tcl list, and each list within the *subwindow_list* must be enclosed in a Tcl list.

The expanded syntax for *subwindow_list* is as follows:

```
PvBands {"  
    subwindow1 {"  
        {"focuslist1"}  
        {"doselist1"}  
        {"sizelist1"}  
        [{"resistlist1"} {"etchlist1"}]  
    }  
    subwindow2 {"  
        {"focuslist2"}  
        {"doselist2"}  
        {"sizelist2"}  
        [{"resistlist2"} {"etchlist2"}]  
    } ...  
}
```

To use the [Customizable PV-Bands](#) interface, you can define contours, PV-bands and overlay PV-bands for each subwindow, by making [Contour](#), [Band](#), and [OverlayBand](#) calls respectively. Focus, dose, size, etch and resist models are defined by the customized PV-bands. In this case, the expanded syntax for *subwindow_list* is as follows:

```
PvBands {"  
    subwindow1 {"  
        Contour {"Contour_command_options"}...  
        Band {"Band_command_options"}...  
        OverlayBand {"OverlayBand_command_options"}...  
    }  
    subwindow2 {"  
        Contour {"Contour_command_options"}...  
        Band {"Band_command_options"}...  
        OverlayBand {"OverlayBand_command_options"}...  
    } ...  
}
```

The subwindows are referenced as follows:

- subwindow1 — For the first experiment.

- subwindow2 — For the second experiment.
- and so on.

Each process experiment must take the form of list of three lists:

- the first list must be the focuslist.
- the second list must be the doselist.
- the third list must be the sizelist.

The contents of the lists are as follows:

- focuslist

A list of optical model names defining the models to use for PV-band generation. Optical model names must match the names of optical models defined through the **OpticalModels** keyword. Optical model names may be repeated as needed to define the conditions to be tested.

The list is order-dependent, with the order matching the order in the associated doselist *and* sizelist.

The list of optical model names must be enclosed in braces ({}).

- doselist

A list of values defining the doses to use for PV-band generation. Doses must be expressed as real numbers representing percentages.

The list is order-dependent, with the order matching the order in the associated focuslist *and* sizelist.

The list of doses must be enclosed in braces ({}).

- sizelist

A list of values defining the mask bias to apply during PV-band generation. The mask bias must be expressed as real numbers representing the bias size.

The list is order-dependent, with the order matching the order in the associated focuslist *and* doselist.

The list of mask bias sizes must be enclosed in braces ({}).

- PrintableModel “{” name *printableModelPath* | *inlineModel* “}”]
Optional keyword and list containing the printable model name and model definition. The model definition can be a model path or an inline model.
- PrintableBand *subwindow_list*
Optional keyword and list defining the process experiments for which PV-bands are to be generated. The process experiments must be supplied as a list of at least one subwindow/

experiment pair. The entire list must be enclosed in a Tcl list, and each list within the *subwindow_list* must be enclosed in a Tcl list, as in the **PvBands** definition above.

If **PrintableBand** is defined, the PV-bands are generated using the **PrintableModel**, as if **LFD::PrintableBand** has been called.

- **PrintableSplitMasks** “{” *split_masks_list* “}”]

Optional keyword and Tcl list used to specify the names of one or more split masks.

- **-d2cInfo** “{” *subwindows* “}”

Optional keyword introducing the D2C information. The **-d2cInfo** block *subwindows* information must be enclosed in braces. If **-d2cInfo** is defined for a certain layer, the PV-bands for this layer is not called by default. You must call either **LFD::PVband** or **LFD::Drawn2Contour**.

```
subwindow1 " { minBandFile " { { " minBandFile_content " } } "
maxBandFile " { { " maxBandFile_content " } } "
[minBandSetup " { setup_file_content " } ]
[maxBandSetup " { setup_file_content " } ]
[maxBandTag max_band_tag]
[minBandTag min_band_tag] " } "

subwindowN " { minBandFile " { { " minBandFile_content " } } "
maxBandFile " { { " maxBandFile_content " } } "
[minBandSetup " { setup_file_content " } ]
[maxBandSetup " { setup_file_content " } ]
[maxBandTag max_band_tag]
[minBandTag min_band_tag] " } "
```

Note

 If **minBandTag** and **maxBandTag** are set, they need to be defined in their respective **minBandSetup** and **maxBandSetup** content lists.

addPDKSTO

Defines a Structure Optimizer (STO) template and assigns it a name by which it can be referenced.

Usage

```
addPDKSTO "{"
  -name STOname
  -pdk pdkName
  [-security {00 | 01 | 10 | 11}]
  [-pdkLayer pdk_process_layer]
  -definition {"arg1 value1 [argN valueN...]"}
  [-pdkCheckName pdk_check_name]
}"
```

Description

Defines a StructureOptimizer (STO) template and assigns it a name by which it can be referenced.

Arguments

The set of arguments used with this command must be enclosed in braces, “{” ... “}”.

- **-name *STOname***
Required keyword and argument used to supply a name for the STO being defined.
- **-pdk *pdkName***
Required keyword and argument defining the PDK with which this STO is associated.
- **-security {00 | 01 | 10 | 11}**
Optional keyword and argument defining read/write privileges. The input to security is two flags, the first for the read security and the other for write security. Refer to [Table 7-13](#) for an explanation of these settings.
- **-pdkLayer *pdk_process_layer***
Optional keyword and argument defining the input process layers to STO which it should operate on. It does not operate on any other process layer.
- **-definition {"*arg1 value1* [*argN valueN...*]“”}**
Required keyword followed by the LFD::[StructureOptimizer](#) arguments, supplied as keyword/value pairs. These STO arguments must match the arguments defined for the LFD::StructureOptimizer command in its syntax description.
- **-pdkCheckName *pdk_check_name***
Optional keyword and argument to **-definition** used only in case of using LFD::StructureOptimizer in No-Simulation or Hybrid mode in a PDK flow to do the mapping between an STO and a specific check.

Examples

```
LFD::addPDKSTO {
    -name STO_1
    -pdk pdk_2
    -pdkLayer pdklyr_metal1
    -definition {
        -property {PVI MinCD}
        -macro msc_sto
        -interactDistance 2.0
    }
}
```

createPDK

Registers the PDK, setting up the appropriate namespace within Calibre LFD.

Usage

```
createPDK {"  
    -name pdkName  
    [-description description]  
    [-halfNode yes]  
    [-version pdkVersion]  
    [-security {" Checks rw_checks Processlayers rw_layers "}]  
}
```

Description

Registers the PDK, setting up the appropriate namespace within Calibre LFD.

Arguments

The set of arguments used with this command must be enclosed in braces, “{” ... “}”.

- **-name pdkName**

Required keyword/argument pair defining the name of the PDK to create.

- **-description {description}**

Optional keyword/argument pair defining a text description of the PDK. When the description contains spaces it must be enclosed in braces ({}).

- **-halfNode yes**

Optional keyword and argument used when running Calibre LFD checks on the half-node, a foundry-process technology of reducing the size of a chip without redesigning the circuits to fit the smaller area. If the -halfNode argument is set to “yes”, a half-node warning message is displayed in the output transcript. The -halfNode argument accepts only “yes” as an input, and reports an error otherwise.

- **-version pdkVersion**

Optional keyword/argument pair specifying a version number for this PDK. Version numbers are recommended.

- **-security {" Checks rw_checks Processlayers rw_layers "}**

Optional keyword, followed by an array containing one or more security settings, which define the default global security levels (read and write access control) for the information stored within the PDK. The complete set of security settings must be enclosed in braces ({}).

Table 7-13. Security Settings

Setting	Behavior for Checks	Behavior for ProcessLayers
00	<ul style="list-style-type: none"> Check details are not written to the transcript. Only the -pdk versions of the CaptureContour and OutputBands commands are allowed. Check arguments defined in the PDK cannot be overridden in the TVF file. 	<ul style="list-style-type: none"> Layer data is not written to the transcript Process correction arguments cannot be overridden.
01	<ul style="list-style-type: none"> Check details are not written to the transcript. Only the -pdk versions of the CaptureContour and OutputBands commands are allowed. You can override check arguments defined in the PDK. 	<ul style="list-style-type: none"> Layer data is not written to the transcript Process correction arguments can be overridden.
10	<ul style="list-style-type: none"> Check details are written to the transcript Non-pdk versions of CaptureContour and OutputBands are permitted. Check arguments defined in the PDK cannot be overridden in the TVF file. 	<ul style="list-style-type: none"> Layer data is written to the transcript. Process correction arguments cannot be overridden.
11	<ul style="list-style-type: none"> Check details are written to the transcript. Non-pdk versions of CaptureContour and OutputBands are permitted. You can override check arguments defined in the PDK. 	<ul style="list-style-type: none"> Layer data is written to the transcript. Process correction arguments can be overridden,

o Checks *rw_checks*

Optional keyword/argument pair defining the default read and write access controls for checks. This setting applies to all checks in the PDK. Override this setting on a check-by-check basis through the Security keyword in each [addPDKCheck](#) command.

o Processlayers *rw_layers*

Optional keyword/argument pair defining the default read and write access controls for process layers. This setting applies to all process layers in the PDK. Override this setting on a layer-by-layer basis through the Security keyword in each [addPDKLayer](#) command.

System-Generated Names, Priorities, and Scores

Calibre LFD uses system-generated names for checks and calculates priorities and scores.

The tables that follow define the naming schemes for system generated items in terms of four user-defined strings:

- <layer> is the layer name as specified by the -layer input parameter.
- <layer1> is the name of the target layer being evaluated using a two-layer check, as specified by the -layer1 input parameter.
- <layer2> is the name of the layer with respect to which **-layer1** is checked using a two-layer check, as specified by the -layer2 input parameter.
- <subwindow> is process variation experiment, referenced by the order specified in the -doseSpanList and -opticalSpanList.

See the following tables:

- [Table 7-14, System Generated Check Names](#)
- [Table 7-15, Default Priorities](#)
- [Table 7-16, LFD Scoring](#)

Table 7-14. System-Generated Check Names

Check	Name and Example
min space between PV-bands	lfd_<layer1>_MSC_p<subwindow> Example: lfd_active_MSC_p1
area check of PV-bands	lfd_<layer1>_AC_p<subwindow> Example: lfd_contact_AC_p1
predicted overlay of PV-bands for two layers	lfd_<layer1>_<layer2>_AOC_p<subwindow> Example: lfd_metal_via_AOC_p1
min space check between PV-bands on -layer1 and PV-bands on -layer2	lfd_<layer1>_<layer2>_ILSC_p<subwindow> Example: lfd_contact_poly_ILSC_p1
overlay check between	lfd_<layer1>_<layer2>_MOC_p<subwindow> Example: lfd_poly_contact_MOC_p1
min width of PV-bands	lfd_<layer1>_MWC_p<subwindow> Example: lfd_active_MWC_p1

Table 7-14. System-Generated Check Names (cont.)

Check	Name and Example
overlap of 2 PV-bands	lfd_<layer1>_Over_<layer2>_MAOC_p<subwindow> Example: lfd_active_Over_poly_MAOC_p1
variability area of PV-band over target -layer1	lfd_<layer1>_Over_<layer2>_MAVC_p<subwindow> Example: lfd_active_Over_poly_MOVC_p1
critical dimension (CD) variability of PV-band	lfd_<layer1>_MCDVC_p<subwindow> Example: lfd_active_MCDVC_p1
PV-band and target layer do not interact	lfd_<layer1>_NPC_p<subwindow> Example: lfd_active_NPC_p1
check for line-end deviation from target	lfd_<layer1>_LEC_p<subwindow> Example: lfd_poly_LEC_p1
user-defined check	lfd_<layer1>_CC_p<subwindow> Example: lfd_active_CC_p1
PVI®score	lfd_pvi_<layer>_p<subwindow> Example: lfd_pvi_active_p1
DVI score	lfd_dvi_<layer>_p<subwindow> Example: lfd_dvi_active_p2

Table 7-15. Default Priorities

Check	Rank	Expression	Example
min space (MSC)	1	1000 - 100*<subwindow> + (11 - rank)	if subwindow=2, priority = 810
min width (MWC)	1	1000 - 100*<subwindow> + (11 - rank)	if subwindow=2, priority = 810
non printing (NPC)	1	1000 - 100*<subwindow> + (11 - rank)	if subwindow=2, priority = 810
line-end check (LEC)	1	1000 - 100*<subwindow> + (11 - rank)	if subwindow=2, priority = 810
custom checks (CC)	1	1000 - 100*<subwindow> + (11 - rank)	if subwindow=2, priority = 810
overlap (MAOC)	2	1000 - 100*<subwindow> + (11 - rank)	if subwindow=2, priority = 809

Table 7-15. Default Priorities (cont.)

Check	Rank	Expression	Example
min overlay (MOC)	2	$1000 - 100 * <\text{subwindow}> + (11 - \text{rank})$	if subwindow=2, priority = 809
interlayer space (ILSC)	2	$1000 - 100 * <\text{subwindow}> + (11 - \text{rank})$	if subwindow=2, priority = 809
overlay (AOC)	3	$1000 - 100 * <\text{subwindow}> + (11 - \text{rank})$	if subwindow=2, priority = 808
variability area (MAVC)	3	$1000 - 100 * <\text{subwindow}> + (11 - \text{rank})$	if subwindow=2, priority = 808
CD variability (MCDVC)	2	$1000 - 100 * <\text{subwindow}> + (11 - \text{rank})$	if subwindow=2, priority = 809
area check (AC)	3	$1000 - 100 * <\text{subwindow}> + (11 - \text{rank})$	if subwindow=2, priority = 808

Table 7-16. LFD Scoring

Check	Score (PVI value)
min space (MSC)	Area of PV-band in the error marker, normalized by the area of the error marker.
min width (MWC)	Area of PV-band in the error marker, normalized by the area of the error marker.
overlap (MAOC)	Area of PV-band in the error marker, normalized by the area of the error marker.
overlay (AOC)	Predicted overlay area.
line-end deviation (LEC)	Area of PV-band in the error marker, normalized by the area of the error marker.
variability area (MAVC)	Area of PV-band overlapping the target layer, normalized by the area of the overlapping target layers.
CD variability (MCDVC)	Size of the error (total area) in square microns.
area check (AC)	Size of the error (total area) in square microns.

Table 7-16. LFD Scoring (cont.)

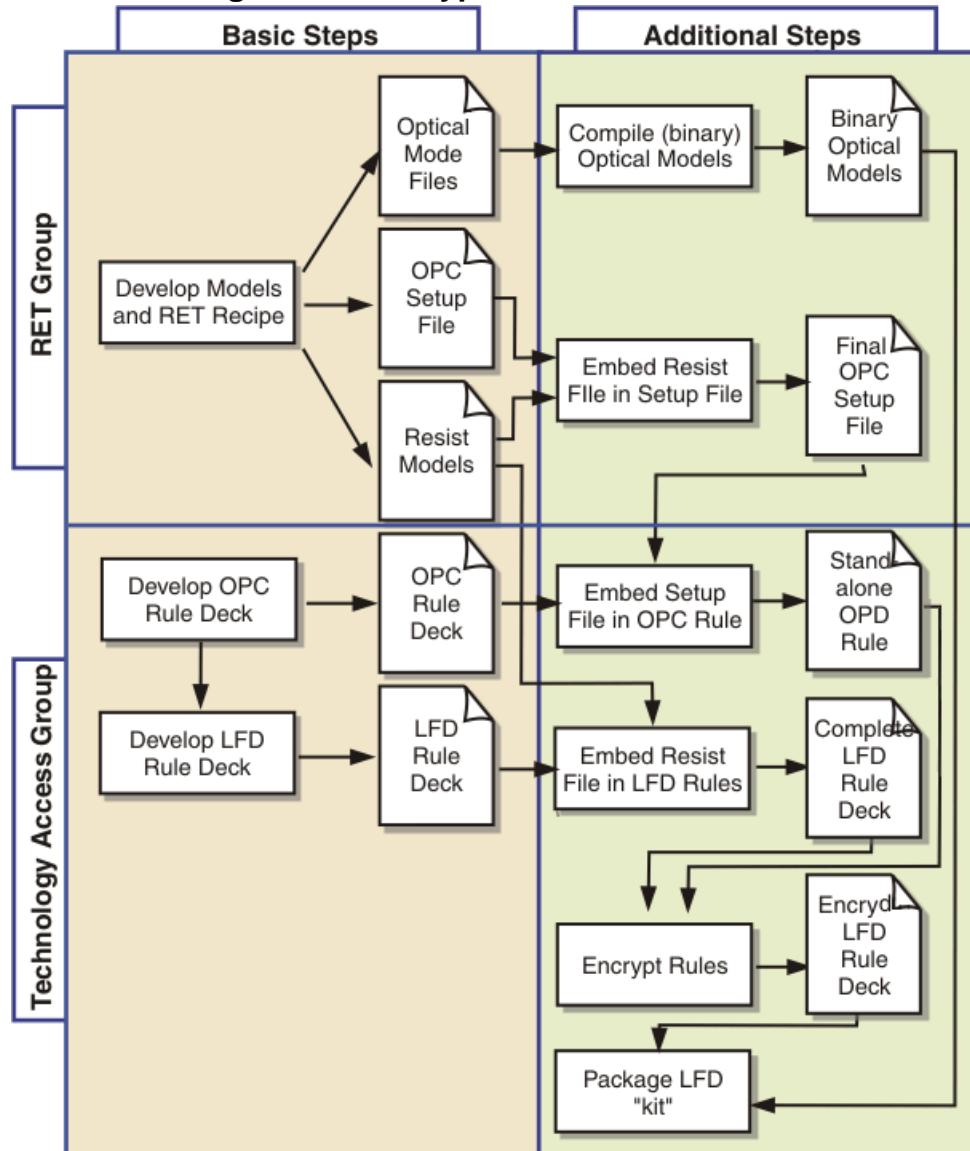
Check	Score (PVI value)
min overlay (MOC)	Area of PV-band in the error marker, normalized by the area of the error marker.
interlayer space (ILSC)	Area of PV-band in the error marker, normalized by the area of the error marker.
non printing (NPC)	N/A.
custom check (CC)	Area of PV-band in the error marker, normalized by the area of the error marker.
design variability (DVI)	Area of PV-band errors from all LFD checks performed, normalized by the window area.
process variability (PVI)	Area of PV-band normalized by the area of target layer (both within the window to which the score applies).

Chapter 8

Encrypted Calibre LFD Flow

Calibre LFD provides several mechanisms for protecting foundry IP. A basic flow for compiling and encrypting your process data for use by designers is described.

Figure 8-1. Encrypted Calibre LFD Flow



For the Litho Team	500
For the CAD Group.....	503
Packaging the Calibre LFD Kit	512

For the Litho Team

Encrypting IP for distribution to customers requires two additional steps for the litho team.

Caution

-  Because the Calibre LFD rule file typically incorporates the RET recipes for multiple layers, all layer names and variable names used within the LITHO setup files or rule files must be unique. This is even more important when encrypting IP, because you cannot modify encrypted variable names.
-

Creating Binary Optical Models	500
Embedding the Resist Model in the Setup File	501

Creating Binary Optical Models

Binary optical models are fully functioning model files that protect your IP. When you ship these files as part of your Calibre LFD kit, the designer can run optical simulations but has no visibility to the optical parameters that describe your illumination system. These special optical models work with any command that references an optical model.

Procedure

1. Add the following line to your optical parameters file:

```
skipModelHeader 1
```

2. Run opticsgen as usual.

The command generates an optical model directory as usual. However, the directory contains only binary model files that are free of header text or other optical information.

Examples

The following code shows an optical model parameters file for creating a binary optical model.

```
version 5
opticalsystem 0.248 0.68 0.85 0.5
illumtype ANNULAR
defocuslevels 4 -0.07 0.10
hoodpix 1.280
approxorder 10
kerngrid 0.010
engine TCCcalc
skipModelHeader 1
```

Related Topics

[Encryption Debugging](#)

Embedding the Resist Model in the Setup File

Resist models and setup files are required for generating PV-bands. To encrypt the resist model, you embed it in the LITHO setup file. Embedding resist models within the LITHO setup file provides maximum IP protection because you can then embed the setup file in the rule file, and encrypt the entire rule file.

Prerequisites

- This procedure assumes that both the resist model and the setup file have been fully tested.

Procedure

1. Open the setup file in the ASCII text editor of choice.
2. Edit the argument for the resistpolyfile keyword, replacing the name of the resist model file with the literal string “inline”.
3. Add the following lines of code directly above the first tagging command:

```
inline_resist {  
    model_params  
}
```

4. Open your resist model file, then select and copy the contents.
5. In the setup file, delete “model params” and paste the resist file data in its place.
6. Save the edited setup file.

For a complete discussion of inline resist models, refer to the [*Calibre Rule-Based OPC User’s and Reference Manual*](#).

Examples

The following code shows a sample setup file with an embedded resist model.

```
# ----- Simulation models -----
modelpath .:models:~/calibrewb_workspace/models
opticalmodel mcoptimopt
resistpolyfile inline
# ----- OPC algorithm -----
iterations 4
tilemicrons 100
stepsize 0.002
gridsize 0.001
aspect 0.0
cornerSiteStyle CONSERVATIVE
lineEndAdjDist 0.19
convexAdjDist 0.19
concaveAdjDist 0.19
# ----- Fragmentation -----
minfeature 0.20
minedgelength 0.05
maxedgelength 100
cornedge 0.05 .05 0.05
concavecorn 0.05 .05 0.05
interfeature -interdistance 0.8 -ripplelen 0.18 -num 5 -ripplestyle 1
seriftype 0
minjog 0.17
lineEndLength 0.45
# ----- Layer info -----
background clear
# Layers
layer 237 L1 17 0 opc dark
#-----Inline Resist Model -----
inline_resist {
    type VT-5
    version 3
    referenceThreshold 0.2
    hoodpix 1.28
    kerngrid 0.01
    CKERNEL D1 "gauss 0.2"
    CKERNEL D2 "gauss 1.7"
    BPAR SLOPE D1 D2
    btermCount 3
    BTERM 0.073 SLOPE 1
    BTERM 0.09 D1 1
    BTERM -0.027 D2 1
    maxBias 0.080
}
#--- Arbitrary commands can follow this line. Don't delete this line! ---
concavedepth 0.02
```

Related Topics

[Encryption Debugging](#)

For the CAD Group

Encrypting IP for distribution to customers requires three additional steps for the CAD group.

Embedding the Setup File in the Rule File	503
Embedding the Resist Models in the Calibre LFD Rule File.....	505
Encrypting the Calibre LFD Rules	510
Encryption Debugging.....	512

Embedding the Setup File in the Rule File

Setup files are required for generating PV-bands. To encrypt the setup file, you embed it in the rule file then encrypt the rule file.

Prerequisites

- This procedure assumes you have a fully debugged rule file for performing OPC, and this rule file references one or more standalone setup files.

Procedure

1. Open the rule file in a text editor.
2. For each setup file required:
 - a. Add the LITHO FILE command.

```
LITHO FILE name [  
    inline_file  
]
```
 - b. Open the setup file, then select and copy the contents.
 - c. In the rule file, replace “name” with the name by which you reference the setup file. Each inline file must have a unique name.
 - d. Delete “inline_file” and paste the setup file data in its place.
3. For each LITHO command, replace the setup file name with the name of the associated inline setup file.
4. Save the edited rule file.

For a complete discussion of embedded setup files, refer to the *Standard Verification Rule Format (SVRF) Manual*.

Examples

The following code shows a rule file with an embedded setup file.

Encrypted Calibre LFD Flow

Embedding the Setup File in the Rule File

```
LAYOUT SYSTEM GDSII
LAYOUT PRIMARY "PUB26"
LAYOUT PATH "../GDS/pub26.gds"

PRECISION 1000
RESOLUTION 5
LAYOUT ERROR ON INPUT NO

FLAG SKEW YES
FLAG ACUTE YES
FLAG OFFGRID YES

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE result.gds GDSII PSEUDO
DRC SUMMARY REPORT result.rep
DRC MAXIMUM VERTEX 199

LAYER POLY 2

LITHO FILE poly_setup [
# ----- Simulation models -----
modelpath .:models:~/calibrewb_workspace/models
opticalmodel mcoptimopt
resistpolyfile inline
# ----- OPC algorithm -----
iterations 4
tilemicrons 100
stepsize 0.002
gridsize 0.001
aspect 0.0
cornerSiteStyle CONSERVATIVE
lineEndAdjDist 0.19
convexAdjDist 0.19
concaveAdjDist 0.19
# ----- Fragmentation -----
minfeature 0.20
minedgelength 0.05
maxedgelength 100
cornedge 0.05 .05 0.05
concavecorn 0.05 .05 0.05
interfeature -interdistance 0.8 -ripplelen 0.18 -num 5 -ripplestyle 1
seriftype 0
minjog 0.17
lineEndLength 0.45
# ----- Layer info -----
background clear
# Layers
layer 237 L1 17 0 opc dark
#-----Inline Resist Model -----
inline_resist {
type VT-5
version 3
referenceThreshold 0.2
hoodpix 1.28
kerngrid 0.01
CKERNEL D1 "gauss 0.2"
CKERNEL D2 "gauss 1.7"
BPAR SLOPE D1 D2
```

```
btermCount 3
BTERM 0.073 SLOPE 1
BTERM 0.09 D1 1
BTERM -0.027 D2 1
maxBias 0.080
}
---- Arbitrary commands can follow this line. Don't delete this line! ----
concavedepth 0.02
]

POLY_OPc {
    LITHO OPCPRO FILE poly_setup POLY
}
POLY {COPY POLY}
DRC CHECK MAP POLY 2
DRC CHECK MAP POLY_OPc 3
```

Related Topics

[Encryption Debugging](#)

Embedding the Resist Models in the Calibre LFD Rule File

You can embed the resist model within the LITHO setup file. Embedding resist models within the LITHO setup file provides maximum IP protection because you can then embed the setup file in the rule file and encrypt the rule file.

In the example below, we use a Tcl variable to include the resist file in the rule file, rather than referencing it.

Procedure

1. In the TVF rule file for running Calibre LFD, create a string variable for each resist file.
2. Do the following for each resist file:
 - a. Open the resist file and copy its contents.
 - b. In the TVF rule file, create a Tcl variable to contain the resist file.

```
set first_resist_model {{}}  
}}
```

- c. Paste the contents of the resist model into the Tcl set variable.

```

set first_resist model {{
    type VT-5
    version 3
    referenceThreshold 0.2
    hoodpix 1.28
    kerngrid 0.01
    CKERNEL D1 "gauss 0.2"
    CKERNEL D2 "gauss 1.7"
    BPAR SLOPE D1 D2
    btermCount 3
    BTERM 0.073 SLOPE 1
    BTERM 0.09 D1 1
    BTERM -0.027 D2 1
    maxBias 0.080
}}

```

Remember to use double braces ({{ }}) around the resist file to ensure that the Tcl interpreter treats the model as a one element list (a single string) and does not perform any variable substitution or evaluation.

3. Edit the -resistFile portion of each LFD::PVband command, replacing the name of the resist file with the variable that now stores the resist file:

```

LFD::PVband "-layer contact_target \
    -layerRET {lfd_contact_mopc contact_sraf contact_patches} \
    -foreground {clear clear dark} \
    -background {attenuated 0.06} \
    -pixel 0.02 \
    -modelDir {./contact/models} \
    -resistFile {$first_resist_model} \
    -opticalSpanList {{D0 D0 D0 D50 D50 D50} \
        {D0 D0 D0 D75 D75 D75}} \
    -doseSpanList {{0.9800 1.0000 1.0200 0.9800 1.0000 1.0200} \
        {0.9600 1.0000 1.0400 0.9600 1.0000 1.0400}}"

```

4. Save the edited Calibre LFD rule file.

Examples

The following code shows a Calibre LFD rule file with an embedded resist model.

```

#! tvf
namespace import tvf::*;

=====
# Load Calibre LFD package
namespace import CalibreDFM_LFD::*;

=====
# Specify Calibre LFD related output databases

set lfdErrorDatabase      "./rdb/lfd_dense.rdb"
set lfdBandDatabase        "./rdb/lfdBand_dense.rdb"
set lfdTopClassification   "./rdb/lfdTopClass_dense.rdb"
set lfdPrecision           1000

=====
# Start the actual recipe
VERBATIM "
// Input Section.
LAYOUT SYSTEM GDSII
LAYOUT PATH "./gds/test_design.gds\""
LAYOUT PRIMARY \"Top\"
FLAG SKEW YES
LAYOUT ERROR ON INPUT YES
PRECISION $lfdPrecision
RESOLUTION 1
// Map existing layers
LAYER poly_target 13

// Output section.
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "\"$lfdTopClassification\"" ASCII
DRC SUMMARY REPORT "/dev/null"
DRC MAXIMUM VERTEX 199
DRC KEEP EMPTY YES

//
=====

// Do or copy OPC.
//

=====
INCLUDE ./drc/RET.svrf

//
=====

// Embed resist model
//

=====
set first_resist_model {{
    type VT-5
    version 3
    referenceThreshold 0.2
    hoodpix 1.28
    kerngrid 0.01
    CKERNEL D1 "gauss 0.2"
    CKERNEL D2 "gauss 1.7"
    BPAR SLOPE D1 D2
}

```

Encrypted Calibre LFD Flow

Embedding the Resist Models in the Calibre LFD Rule File

```
btermCount 3
BTERM 0.073 SLOPE 1
BTERM 0.09 D1 1
BTERM -0.027 D2 1
maxBias 0.080
}

//
=====
// Use LFDregion to support DRC check areas in Calibre Interactive
//
//
=====

LFD::LFDregion "-region interactive -halo .56"

//=====
// Create BV-bands
//=====
"
LFD::PVband "-layer poly_target \
    -layerRET lfd_poly_mopc \
    -foreground {{attenuated 0.06}} \
    -background clear \
    -pixel 0.02 \
    -modelDir {./poly/models} \
    -resistFile {$first_resist model} \
    -opticalSpanList {{P1W_D0 P1W_D0 P1W_D0 P1W_D50 P1W_D50 P1W_D50} \
        {P1W_D0 P1W_D0 P1W_D0 P1W_D0 P1W_D0 P1W_D100 \
            P1W_D100 P1W_D100 P1W_D100 P1W_D100} \
        -doseSpanList {{0.9800 1.0000 1.0200 0.9800 1.0000 1.0200} \
            {0.9600 0.9800 1.0000 1.0200 1.0400 0.9600 0.9800 1.0000 1.0200 \
                1.0400}} \
    VERBATIM "
//
=====
// Calculate the checks themselves
//
=====
"
LFD::MinSpaceCheck -layer poly_target -subwindow 1 -minDRCspace 0.10 \
    -minLFDspace 0.06 -database $lfdErrorDatabase
LFD::MinWidthCheck -layer poly_target -subwindow 1 -minDRCwidth 0.15 \
    -minLFDwidth 0.05 -database $lfdErrorDatabase
LFD::MinSpaceCheck -layer poly_target -subwindow 2 -minDRCspace 0.10 \
    -minLFDspace 0.06 -database $lfdErrorDatabase
LFD::MinWidthCheck -layer poly_target -subwindow 2 -minDRCwidth 0.15 \
    -minLFDwidth 0.05 -database $lfdErrorDatabase

VERBATIM "
//
=====
// Create variability indices.
//
=====
"
```

```
LFD::VariabilityIndices "-layer poly_target -windowSize 3 \  
-database $lfdErrorDatabase"  
  
VERBATIM "  
//  
=====  
// Output PV-bands  
//  
=====  
"  
  
LFD::OutputBands -layer poly_target -bandType all -subwindow 1 \  
-database $lfdBandDatabase  
LFD::OutputBands -layer poly_target -bandType all -subwindow 2 \  
-database $lfdBandDatabase
```

Related Topics

[Encryption Debugging](#)

Encrypting the Calibre LFD Rules

After embedding your setup files and resist files within the rule files that perform OPC and Calibre LFD, you are ready to encrypt these files and hide your IP.

This discussion assumes the following:

- Your Calibre LFD rule file has been fully tested and debugged.
- The Calibre **caltvfencrypt** software is available. Refer to the *Calibre TVFencrypt User's Manual* and the *Calibre Administrator's Guide* for more information.

You encrypt Calibre LFD rules differently, depending on whether you are using a PDK. If you are not using a PDK, you also have the option of just encrypting the rule check commands and PV-band generation commands, rather than the complete rule file.

Encrypting the Calibre LFD Rule File Without a PDK.....	510
Encrypting LFD Commands Without a PDK.....	510
Encrypting Calibre LFD Rules with a PDK	511

Encrypting the Calibre LFD Rule File Without a PDK

You can encrypt the complete Calibre LFD rule file.

Procedure

1. In the Calibre LFD rule file, add the option -encrypted yes to LFD::[Begin](#).
2. Encrypt the rule file using the following command:

```
caltvfencrypt -id <unique_id> <input_file> <encrypted_output_file>
```

Results

You now have an encrypted rule file. Use the file *encrypted_output_file* from Step 2 as the rule file when you run Calibre LFD.

Related Topics

[Encryption Debugging](#)

Encrypting LFD Commands Without a PDK

If you are not using a PDK, you also have the option of just encrypting the rule check commands and PV-band generation commands, rather than the complete rule file.

Procedure

1. Move all PV-band generation commands and check commands from the original Calibre LFD rule file and into a separate file. It is assumed that all OPC rule files are INCLUDED in the TVF file.

In this file, you should take the following into consideration:

- Add `-security yes` to all LFD::PVband calls to protect your Calibre LFD rules.
- Namespaces that were imported into the original rule file are not imported to this separate file after decryption, so you need to add the namespace prefix to all your commands. For example, LFD::PVband and tvf::VERBATIM.
- Tcl variables defined in the original rule file cannot be used in the referenced PV-bands and checks file after encryption. You need to redefine the variables you use in your new referenced file.

2. Encrypt the new file with the LFD commands using the following command:

```
caltvfcrypt -id <unique_id> <input_file> <encrypted_output_file>
```

3. Source the encrypted version of the file in the main rule file, after LFD::Begin and before LFD::End.

Related Topics

[Encryption Debugging](#)

Encrypting Calibre LFD Rules with a PDK

You can encrypt the Calibre LFD rules when you are using a PDK. The method used is different than without a PDK.

Procedure

1. Ensure that security option settings are set properly in the PDK, as they control read and write access to the data within the PDK. Review the security options of the following PDK commands:
 - LFD::createPDK
 - LFD::addPDKLayer
 - LFD::addPDKCheck
 - LFD::Macro
2. Any TVF block that generates SVRF is converted in an LFD::Macro with the appropriate security settings.

3. Do not set any TCL variables in the PDK to use them in your control TVF because they are not accessible.
4. Encrypt the PDK using the following command:

```
caltvfencrypt -id <unique_id> <pdk_file> <encrypted_pdk_file>
```

Related Topics

[Encryption Debugging](#)

Encryption Debugging

During Calibre LFD kit development, it is recommended that you use the variable `CALIBRE_ENCRYPTED_TVF_NO` 296578. This simplifies debugging as it runs the byte-compiled package rather than the encrypted package. With the byte-compiled package, full error messages are enabled containing details useful to Calibre LFD kit developers. In addition, the kit developer can use the Calibre -E option to echo SVRF code related to running Calibre LFD to produced files.

Note

 When the Calibre LFD kit is developed and encrypted, the variable `CALIBRE_ENCRYPTED_TVF_NO` 296578 does not work, the Calibre -E option is disabled, and the error messages are less useful.

Packaging the Calibre LFD Kit

You provide customers with the encrypted files and the binary optical models when encrypting the Calibre LFD rules.

Specifically, you provide the following:

- The encrypted rule file:

Without a PDK, provide one of the following:

- The encrypted Calibre LFD rule file. See “[Encrypting the Calibre LFD Rule File Without a PDK](#)” on page 510.
- The encrypted file with LFD commands. See “[Encrypting LFD Commands Without a PDK](#)” on page 510

With a PDK, provide the following:

- The encrypted PDK.

- The binary optical models.

Because Calibre LFD requires several optical models, and each one is stored as a separate directory, we recommend that you use file compression software, such as TAR to package all your optical models as a single file.

Related Topics

[Encryption Debugging](#)

Appendix A

Running Calibre LFD with Calibre Interactive

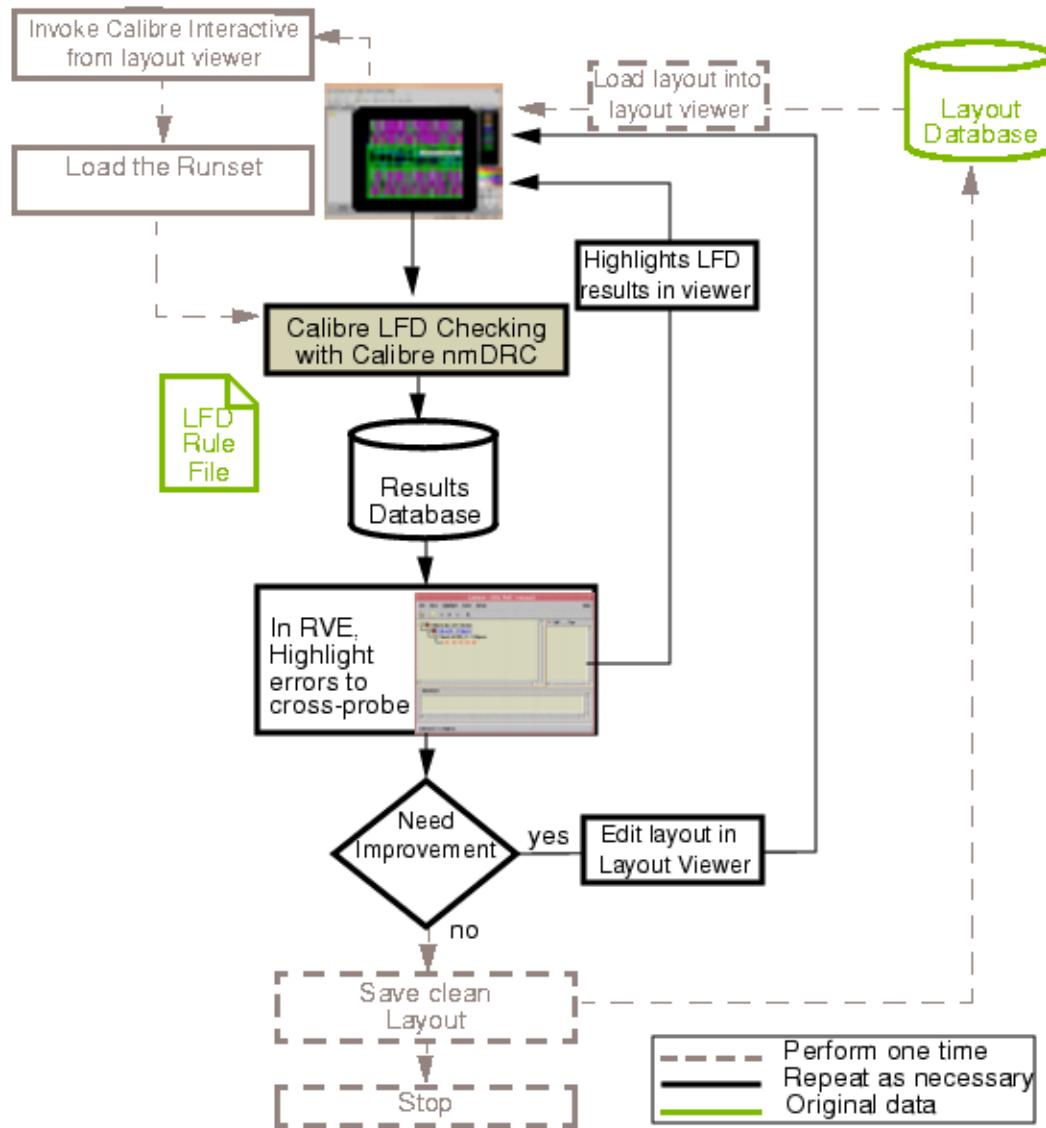
Using Calibre Interactive, you can launch a Calibre run from your layout viewer/editor and use the current layout as input. This lets you check a correction before you commit to it.

Basic Calibre LFD Flow with Calibre Interactive	515
Creating and Using a Calibre Interactive Runset for Calibre LFD	517

Basic Calibre LFD Flow with Calibre Interactive

The basic flow for using Calibre Interactive with Calibre LFD includes connecting to a supported design tool and viewing results using Calibre RVE.

Figure A-1. Using Calibre Interactive with Calibre LFD



You can simplify using Calibre Interactive to run Calibre LFD by creating a runset with your Calibre Interactive settings. See “[Creating and Using a Calibre Interactive Runset for Calibre LFD](#)” on page 517.

Related Topics

[Calibre Interactive User’s Manual](#)

Creating and Using a Calibre Interactive Runset for Calibre LFD

You can use a runset to save your Calibre Interactive settings for the Calibre LFD run. You can create your own runset or start with an existing runset.

Prerequisites

- “[Calibre LFD Requirements](#)” on page 20
- A Calibre Interactive license.
- (Optional) A design open in a supported layout viewer.

Procedure

1. Invoke Calibre Interactive DFM as described in “[Invoking Calibre LFD From Calibre Interactive](#)” on page 22.
Be sure to set a run type of “Litho Friendly Design (LFD)” on the Inputs pane.
2. Click the buttons on the left panel to make settings as required for the Rules, Inputs, Outputs, and Run Control panes.
3. (Optional) Select **Setup > DFM Options** to set additional options.
4. Select **File > Save Runset** to save the all the settings to a runset.

Results

You now have a runset you can use to run Calibre LFD with the same settings. Select **File > Load Runset** to load the runset in Calibre Interactive

Related Topics

- [About Calibre Interactive Runsets \[Calibre Interactive User's Manual\]](#)
[Running Calibre Interactive in Batch Mode \[Calibre Interactive User's Manual\]](#)
[Using Calibre Interactive to Perform DFM \[Calibre Interactive User's Manual\]](#)

Appendix B

Sample Calibre LFD Rule File and PDK

An example rule file and PDK for Calibre LFD are provided.

Sample Rule File	519
Sample PDK	526

Sample Rule File

The rule file is written using TVF. The Tcl Verification Format (TVF) is a programmable expansion to the Standard Verification Rule Format (SVRF) language for Calibre.

For a complete discussion of TVF, refer to “[Tcl Verification Format](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

Sample Calibre LFD Rule File and PDK Sample Rule File

```
#! tvf
namespace import tvf::*;

# This is a Calibre LFD demo that showcases much of the Calibre
# LFD functionality.

#####
# Load Calibre LFD-generic library
#####
package require CalibreDFM_LFD

#####
# Import extra utilities. This is required for the Calibre LFD
# demo only! It does illustrate how a user may create their own
# procedures local to their environment. For documentation on
# what each procedure is doing, please view the utilities file
# itself or the demo manual.
#
# The procedures exposed via the utilities file are NOT supported
# by Siemens Digital Industries Software with the exception of the
# current demo
# application.
#####
source ./bin/CalibreDFM_LFD_DEMO_UTILS.tcl

#####
# Layer Select (1=run, 0=don't run)
#####
set RUN_ACTIVE 1
set RUN_POLY 1
set RUN_CONTACT 1
set RUN_METAL1 1

#####
# Specify Calibre LFD input files
#####
set LayersFile "./lfd/lfd_demo.lyr"
set PdkFile "./lfd/lfd_demo.pdk"
set LayoutFile "./gds/lfd_demo.gds"

#####
# Specify Calibre LFD output files
#####
set LayoutDB "\$RDB_DIR/lfd_output.oas"
set ChecksDB "\$RDB_DIR/lfd_checks.rdb"
set BandsDB "\$RDB_DIR/lfd_pvbands.rdb"
set IndexDB "\$RDB_DIR/lfd_indices.rdb"
set SummaryDB "\$RDB_DIR/lfd_results.sum"

#####
# Settings and Configurations
#####
set LayoutSystem GDSII
set LayoutPrimary *
set LayoutMagnify 10
set Precision 10000
set Resolution 1
set LayoutDBType "OASIS"
```

```
set LayoutDBOptions "USER MERGED"
set SummaryDBType ""
set SummaryDBOptions ""

#####
# Start the actual recipe
#####

=====
# (Optional) Encrypted Run
=====
if { [ info exists ::env(CALIBRE_ENCRYPTED_TVF_YES) ] } {
set PdkFile "./lfd/lfd_demo.pdke"
}

=====
# Predefined Runtime Options
=====
tvf::LAYOUT PATH \"$LayoutFile\"
tvf::LAYOUT SYSTEM $LayoutSystem
tvf::LAYOUT PRIMARY \"$LayoutPrimary\"
tvf::LAYOUT MAGNIFY $LayoutMagnify
tvf::LAYOUT PROCESS BOX RECORD YES
tvf::LAYOUT ERROR ON INPUT NO
tvf::PRECISION $Precision
tvf::RESOLUTION $Resolution
tvf::DRC RESULTS DATABASE \"$LayoutDB\\" $LayoutDBType $LayoutDBOptions
tvf::DRC SUMMARY REPORT \"$SummaryDB\\" $SummaryDBType $SummaryDBOptions
tvf::DRC MAXIMUM RESULTS all
tvf::DRC MAXIMUM VERTEX 199
tvf::DRC KEEP EMPTY YES
tvf::FLAG SKEW YES
tvf::FLAG ACUTE NO
tvf::FLAG OFFGRID YES

=====
# Start Calibre LFD
=====
LFD::Begin

=====
# Calibre LFD Global Configurations
=====
LFD::SimConfig -modelpath [ getenv MODEL_DIR ] -pixel 0.020

=====
# Layer Definitions
=====
source $LayersFile

=====
# Load the PDK
=====
LFD::LoadPDK $PdkFile

=====
# Run Active
=====
```

Sample Calibre LFD Rule File and PDK Sample Rule File

```
if { $RUN_ACTIVE } {

# Register Layer and Generate PV-bands
# -----
LFD:::RegisterLayer \
    -pdk lfd_demo \
    -Designlayer active \
    -Processlayer pdklyr_active

# Start Checks
# -----
LFD:::MinSpaceCheck -pdkCheckName Active_MSC_p1 -layer active \
    -database $ChecksDB
LFD:::MinSpaceCheck -pdkCheckName Active_MSC_p2 -layer active \
    -database $ChecksDB
LFD:::MinWidthCheck -pdkCheckName Active_MWC_p1 -layer active \
    -database $ChecksDB
LFD:::MinWidthCheck -pdkCheckName Active_MWC_p2 -layer active \
    -database $ChecksDB

# Generate Output
# -----
LFD:::OutputBands -pdkCheckName Active_Band_p1 -layer active \
    -layerOut Active_Band_p1
LFD:::OutputBands -pdkCheckName Active_Band_p2 -layer active \
    -layerOut Active_Band_p2
LFD:::CaptureContour -pdkCheckName Active_CNTR_nom -layer active \
    -layerOut Active_CNTR_nom
LFD:::VariabilityIndices -layer active -windowSize extents \
    -database $IndexDB
LFD:::Demo::RuleCheck active OASIS 11 0 $LayoutDB
LFD:::Demo::RuleCheck Active_Band_p1 ASCII $BandsDB
LFD:::Demo::RuleCheck Active_Band_p2 ASCII $BandsDB
LFD:::Demo::RuleCheck Active_CNTR_nom ASCII $BandsDB
}

=====
# Run Poly
=====
if { $RUN_POLY } {

# Register Layer and Generate PV-bands
# -----
LFD:::RegisterLayer \
    -pdk lfd_demo \
    -Designlayer poly \
    -Processlayer pdklyr_poly

# Start Checks
# -----
LFD:::MinSpaceCheck -pdkCheckName Poly_MSC_p1 -layer poly \
    -database $ChecksDB
LFD:::MinSpaceCheck -pdkCheckName Poly_MSC_p2 -layer poly \
    -database $ChecksDB
LFD:::MinWidthCheck -pdkCheckName Poly_MWC_p1 -layer poly \
    -database $ChecksDB
LFD:::MinWidthCheck -pdkCheckName Poly_MWC_p2 -layer poly \
    -database $ChecksDB}
```

```

LFD:::MaxCDVariabilityCheck -pdkCheckName Poly_MCVC_p1 \
    -layer poly -database $ChecksDB
LFD:::MaxCDVariabilityCheck -pdkCheckName Poly_MCVC_p2 \
    -layer poly -database $ChecksDB

# Generate Output
# -----
LFD:::OutputBands -pdkCheckName Poly_Band_p1 -layer poly \
    -layerOut Poly_Band_p1
LFD:::OutputBands -pdkCheckName Poly_Band_p2 -layer poly \
    -layerOut Poly_Band_p2
LFD:::CaptureContour -pdkCheckName Poly_CNTR_nom -layer poly \
    -layerOut Poly_CNTR_nom
LFD:::VariabilityIndices -layer poly -windowSize extents -database $IndexDB
LFD:::Demo:::RuleCheck poly OASIS 13 0 $LayoutDB
LFD:::Demo:::RuleCheck Poly_Band_p1 ASCII $BandsDB
LFD:::Demo:::RuleCheck Poly_Band_p2 ASCII $BandsDB
LFD:::Demo:::RuleCheck Poly_CNTR_nom ASCII $BandsDB
}

=====
# Run Contact
=====
if { $RUN_CONTACT } {

# Register Layer and Generate PV-bands
# -----
LFD:::RegisterLayer \
    -pdk lfd_demo \
    -Designlayer contact \
    -Processlayer pdklyr_contact

# Start Checks
# -----
LFD:::MinSpaceCheck -pdkCheckName Contact_MSC_p1 -layer contact \
    -database $ChecksDB
LFD:::MinSpaceCheck -pdkCheckName Contact_MSC_p2 -layer contact \
    -database $ChecksDB
LFD:::AreaCheck -pdkCheckName Contact_AC_p1 -layer contact \
    -database $ChecksDB
LFD:::AreaCheck -pdkCheckName Contact_AC_p2 -layer contact \
    -database $ChecksDB

# Generate Output
# -----
LFD:::OutputBands -pdkCheckName Contact_Band_p1 -layer contact \
    -layerOut Contact_Band_p1
LFD:::OutputBands -pdkCheckName Contact_Band_p2 -layer contact \
    -layerOut Contact_Band_p2
LFD:::CaptureContour -pdkCheckName Contact_CNTR_nom -layer contact \
    -layerOut Contact_CNTR_nom
LFD:::VariabilityIndices -layer contact -windowSize extents \
    -database $IndexDB
LFD:::Demo:::RuleCheck contact OASIS 15 0 $LayoutDB
LFD:::Demo:::RuleCheck Contact_Band_p1 ASCII $BandsDB
LFD:::Demo:::RuleCheck Contact_Band_p2 ASCII $BandsDB
LFD:::Demo:::RuleCheck Contact_CNTR_nom ASCII $BandsDB

```

Sample Calibre LFD Rule File and PDK Sample Rule File

```
}

#=====
# Run Metal1
#=====
if { $RUN_METAL1 } {

    # Register Layer and Generate PV-bands
    # -----
    LFD::RegisterLayer \
        -pdk lfd_demo \
        -Designlayer metal1 \
        -Processlayer pdklyr_metal1

    # Start Checks
    # -----
    LFD::MinSpaceCheck -pdkCheckName Metal1_MSC_p1 -layer metal1 \
        -database $ChecksDB
    LFD::MinSpaceCheck -pdkCheckName Metal1_MSC_p2 -layer metal1 \
        -database $ChecksDB
    LFD::MinWidthCheck -pdkCheckName Metal1_MWC_p1 -layer metal1 \
        -database $ChecksDB
    LFD::MinWidthCheck -pdkCheckName Metal1_MWC_p2 -layer metal1 \
        -database $ChecksDB
    LFD::LineEndCheck -pdkCheckName Metal1_LEC_p1 -layer metal1 \
        -database $ChecksDB
    LFD::LineEndCheck -pdkCheckName Metal1_LEC_p2 -layer metal1 \
        -database $ChecksDB

    # Generate Output
    # -----
    LFD::OutputBands -pdkCheckName Metal1_Band_p1 -layer metal1 \
        -layerOut Metal1_Band_p1
    LFD::OutputBands -pdkCheckName Metal1_Band_p2 -layer metal1 \
        -layerOut Metal1_Band_p2
    LFD::CaptureContour -pdkCheckName Metal1_CNTR_nom -layer metal1 \
        -layerOut Metal1_CNTR_nom
    LFD::VariabilityIndices -layer metal1 -windowSize extents \
        -database $IndexDB
    LFD::Demo::RuleCheck metal1 OASIS 16 0 $LayoutDB
    LFD::Demo::RuleCheck Metal1_Band_p1 ASCII $BandsDB
    LFD::Demo::RuleCheck Metal1_Band_p2 ASCII $BandsDB
    LFD::Demo::RuleCheck Metal1_CNTR_nom ASCII $BandsDB
}

#=====
# Calibre LFD Checks: Multiple Layers
#=====
if { $RUN_METAL1 && $RUN_CONTACT } {

    LFD::MinAreaOverlapCheck \
        -pdkCheckName Contact_Metal1_MAOC_p1 \
        -layer1 contact \
        -layer2 metal1 \
        -database $ChecksDB
}

if { $RUN_POLY && $RUN_ACTIVE } {
```

```

LFD:::MaxAreaVariabilityCheck \
    -pdkCheckName Poly_Active_MAVC_p1 \
    -layer1 poly \
    -layer2 active \
    -database $ChecksDB
}

if { $RUN_POLY && $RUN_CONTACT } {

LFD:::AreaOverlayCheck \
    -pdkCheckName Poly_Contact_AOC_p1 \
    -layer1 poly \
    -layer2 contact \
    -database $ChecksDB

LFD:::InterLayerSpaceCheck \
    -pdkCheckName Poly_Contact_ILSC_p1 \
    -layer1 poly \
    -layer2 contact \
    -database $ChecksDB

LFD:::MinOverlayCheck \
    -pdkCheckName Poly_Contact_MOC_p1 \
    -layer1 contact \
    -layer2 poly \
    -database $ChecksDB
}

if { $RUN_POLY && $RUN_ACTIVE && $RUN_CONTACT } {

# AddCustomCheck: Gate to Contact Space Check (GCSC)
tvf:::SETLAYER Gate_CNTR_nom = Poly_CNTR_nom and Active_CNTR_nom
tvf:::SETLAYER Active_Contact_CNTR_nom = Contact_CNTR_nom and
Active_CNTR_nom
tvf:::SETLAYER gate_cont_marker_0 = EXTERNAL Gate_CNTR_nom
    Active_Contact_CNTR_nom < 0.040 REGION
tvf:::SETLAYER gate_cont_marker_1 = SIZE gate_cont_marker_0 BY 0.010
tvf:::SETLAYER gate_cont_marker_2 = EXTENTS gate_cont_marker_1
tvf:::SETLAYER gate_cont_marker_3 = OR poly contact
tvf:::SETLAYER gate_cont_marker_4 = NOT gate_cont_marker_2
    gate_cont_marker_3
tvf:::SETLAYER gate_cont_marker_5 = NOT COIN EDGE gate_cont_marker_4
    gate_cont_marker_3
tvf:::SETLAYER gate_cont_marker_6 = EXPand EDGE gate_cont_marker_5
    INSIDE BY 0.010
tvf:::SETLAYER gate_cont_marker = NOT gate_cont_marker_4 gate_cont_marker_6
tvf:::SETLAYER gate_cont_property = DFM PROPERTY gate_cont_marker
    Gate_CNTR_nom Active_Contact_CNTR_nom INTERSECTING
    PVI=(AREA(Gate_CNTR_nom)
    + AREA(Active_Contact_CNTR_nom)) / AREA(gate_cont_marker)\] >= 0

LFD:::AddCustomCheck \
    -layer active \
    -customCheck gate_cont_property \
    -subwindow 1 \
    -checkName Gate_Contact_GCSC_p1 \
    -priority 800 \
}

```

```
-comment {Increase separation} \
-database $ChecksDB
}

=====
# Finish Calibre LFD
=====
LFD::End

#####
# End the actual recipe
#####

#####
```

Sample PDK

A typical PDK for Calibre LFD is provided.

```

LFD::createPDK {-name generic65 -description {Internal PDK for testing
purposes} -security {Checks 11 ProcessLayers 11}}
LFD::addPDKLayer {-name diffusion -pdk generic65 -processInfo {
    DrawnLayer activeI
    IslandLayers {}
    OpcLayers active_mopc
    OpticalModels {
        OD_0 {OD_D0}
        OD_50 {OD_D50}
        OD_100 {OD_D100}
    }
    ProcessCorrection { CMACRO active_mopc_macro}
    PvBands {
        subwindow1 {{OD_0 OD_0 OD_0 OD_50 OD_50 OD_50}
            {0.9800 1.0000 1.0200 0.9800 1.0000 1.0200} {0 0 0 0 0 0}}
        subwindow2
            {{OD_0 OD_0 OD_0 OD_0 OD_0 OD_100 OD_100 OD_100 OD_100 OD_100}
            {0.95 0.9800 1.0000 1.0200 1.05 0.95 0.9800 1.0000 1.0200 1.05}
            {0 0 0 0 0 0 0 0 0 0}}
    }
    ResistModels {
        active_mod {
            type VT-5
            version 3
            sampleSpacing 0.02
            referenceThreshold 0.25
            searchRange 0.3
            bound IMAX 0.19223 0.798292
            bound SLOPE 0.898645 4.40989
            bound ISLOPE 0.17 0.17
            bound IMIN 0.00314129 0.147129
            bound FACTOR -1.0743 4.18486
            minThreshold 0.05
            maxThreshold 1.0
            minEigenval 0.002
            hoodpix 1.8
            kerngrid 0.01
            ttermCount 1
            TTERM 0.25
        }
    }
    RetLayers{
        mask0 {
            background clear
            layers active_mopc
            transmission {{attenuated 0.06}}
        }
    }
    Security{
        PvBands 11
        ProcessCorrection 11
    }
    TargetLayer activeTarget
    VisibleLayers {}
    opcInLayers activeTarget
    tags2boxes {}
    -optimizerInfo {
        FrameCellOptimizer {
    
```

Sample Calibre LFD Rule File and PDK

Sample PDK

```
halo 1.0
minwidth 0.088
}
Optimizer {
halo 1.0
minwidth 0.088
interactDistance 0.160
MinCD 0.100
maxCD 0.100
var1D 0.015
var2D 0.025
optical OD_0
grid 0.001
tilesize $env(TILE)
fpopcFile activeF.mod
setupFile {

# ----- Simulation models -----
modelpath ./active/models
opticalmodel P1W_D0
resistpolyfile active.mod
# ----- OPC algorithm -----
iterations 10
tilemicrons 100.000000
stepsize 0.001
gridsize 0.001
siteinfo AERIAL -numx 40 -center 15
cornerSiteStyle SITES_ON_EDGE
# ----- Fragmentation -----
minfeature 0.130000
minedgelength 0.10
maxedgelength 1.00
cornedge 0.100
concavecorn 0.100
interfeature -interdistance 0.64000 -ripplelen 0.100000 -num 0 -shield 1 -
ripplestyle 1
seriftype 0
minjog 0.050000
lineEndLength 0.230000
# ----- Layer info -----
background clear
# Layers
layer 3 active 17 0 opc attenuated 0.06 0 1
#---- Arbitrary Commands Can Follow This Line. Don't delete this line! --
---
# ----- Simulation models -----
# ----- OPC algorithm -----
# ----- Fragmentation -----
# ----- Layer info -----
# Layers
#---- Arbitrary Commands Can Follow This Line. Don't delete this line! --
---
sse OPC_MIN_INTERNAL 0.07
sse OPC_MIN_EXTERNAL 0.07
}
}
}
}

# INSERT MACROS BELOW THIS LINE -----
# for Active
```

```

tvf::VERBATIM {
activeTarget = COPY activeI
LITHO FILE active_file /*

# ----- Simulation models -----
modelpath $env(MODEL_DIR)
opticalmodel $env(MODEL_OD)
resistpolyfile inline

# ----- OPC algorithm -----
iterations 10
tilemicrons 100.000000
stepsize 0.001
gridsize 0.001
siteinfo AERIAL -numx 40 -center 15
cornerSiteStyle SITES_ON_EDGE

# ----- Fragmentation -----
minfeature 0.130000
minedgelength 0.10
maxedgelength 1.00
cornedge 0.100
concavecorn 0.100
interfeature -interdistance 0.64000 -ripplelen 0.100000 -num 0 -shield 1 \
    -ripplestyle 1
seriftype 0
minjog 0.050000
lineEndLength 0.230000

# ----- Layer info -----
background clear
# Layers
layer 3 active 17 0 opc attenuated 0.06 0 1
#--- Arbitrary Commands Can Follow This Line. Don't delete this line! ---
# ----- Simulation models -----
# ----- OPC algorithm -----
# ----- Fragmentation -----
# ----- Layer info -----
# Layers
#--- Arbitrary Commands Can Follow This Line. Don't delete this line! ---
sse OPC_MIN_INTERNAL 0.07
sse OPC_MIN_EXTERNAL 0.07
inline_resist {
    type VT-5
    version 3
    sampleSpacing 0.02
    referenceThreshold 0.25
    searchRange 0.3
    bound IMAX 0.19223 0.798292
    bound SLOPE 0.898645 4.40989
    bound ISLOPE 0.17 0.17
    bound IMIN 0.00314129 0.147129
    bound FACTOR -1.0743 4.18486
    minThreshold 0.05
    maxThreshold 1.0
    minEigenval 0.002
    hoodpix 1.8
    kerngrid 0.01

    ttermCount 1
    TTERM 0.25
}

```

Sample Calibre LFD Rule File and PDK
Sample PDK

```
sse LITHO_ALLOW_MAP_BY_ORDER 1
*/]
active_mopc = LITHO OPC FILE active_file _activeTarget MAP active
}
}
```

Appendix C Guidelines for Calibre LFD

There are some specifications often required by the foundry, constraints on the RET recipe decks, as well as general guidelines.

Specifications Required From the Foundry for Calibre LFD.....	531
Constraints on RET Recipe Decks Used for Calibre LFD.....	531
Basic Calibre LFD Guidelines	532

Specifications Required From the Foundry for Calibre LFD

The foundry may have the certain specifications.

- Calibre LFD layer list (Name, GDS layer/datatype).
- Mask transmission values for each Calibre LFD layer.
- Nominal focus optical model file for each Calibre LFD layer.
 - If required by optical model file, illumination SRC map file.
- At least one resist model file for each Calibre LFD layer.
- Dose and focus conditions for each process window analysis, for each layer.
- Rule specification for each to perform (space, width, overlap, area variation, ...).
- Statement defining which parts of the Calibre LFD run must be encrypted.

Constraints on RET Recipe Decks Used for Calibre LFD

There are requirements regarding the layer names, models, and SVRF statements used in RET recipe decks for Calibre LFD.

The RET recipe deck *for each layer* must have:

- The same layer names as the Calibre LFD layer list.
- OPC performed at the same precision for all Calibre LFD layers.
- One resist model file.

- One optical file at nominal focus.
- Optical file based on Process Window Modeling.

RET recipe decks *cannot* have:

- Duplicate layer names.
- Conflicting mapping of GDS layer/datatype definitions.
- Duplicate variable definitions.
- Precision/grid definitions.
- SVRF specification statements.
- Magnify commands.
- Custom RDB files created through DRC CHECK MAP commands.

Basic Calibre LFD Guidelines

There are some basic guidelines for the layout, models, and rules when performing Calibre LFD.

- **Layout** — The layout is sufficiently large to address the areas of influence imposed by the process.
- **Process Models** — Should reliably identify the maximum and minimum pattern responses within the process variations. In the absence of a single model that can explain all process variations, they should limit prediction to specific and well defined effects. In this way a composite PV-band can be used to identify those regions that present maximum variability across many process effects and retain the information of the largest contributor to the pattern variation.
- **Process-Based Design Rules** — These rules define the design violations and help to identify those regions that are most sensitive. Their results are used to flag regions of maximum variability and extract a quantitative metric to manufacturability.
- **PV-Band Calculation** — Uses the process models in conjunction with the process-based design rules.
- **Layout Ranking** — Before proceeding with electrical simulations, this layout ranking metric serves as an additional constraint. Even when the process-based design rules do not return any errors it is possible to look at a continuous metric that provides additional opportunities for improvement.
- **Correction** — It resides in the design environment and requires a different interpretation of the results provided by the process-based design rules. While typical design-rule violations can be fixed by topological (for example, compacting features) or morphological changes (for example, clipping corners), these rules require, in general, a

topological change. More details about the correction mechanisms are subject to a future study.

Appendix D

Defining Highlight Colors to Match Histograms

Calibre RVE is a graphical debug program that provides you with the ability to organize results into histograms, then highlight the results associated with individual histogram bins. The colors used for histograms within Calibre RVE can be matched or defined to colors used for highlights and colormaps in the layout viewer.

You can do the following from a histogram in Calibre RVE:

- Highlight places in the layout where the errors associated with histogram bins occurred.
- Generate a color map showing which portion of a layout is associated with each histogram bin, and overlay that map on the layout.

See these topics for information on creating histograms in Calibre RVE:

- “[Viewing Design Variability Index \(DVI\) Data](#)” on page 123
- “[Creating Histograms and Colormaps](#)” in the *Calibre RVE User’s Manual*

Matching Histogram Colors to Highlight Colors [536](#)

Defining Colors for Calibre DESIGNrev and Calibre WORKbench..... [536](#)

Matching Histogram Colors to Highlight Colors

It is helpful if the colors used for histograms within Calibre RVE match the colors used for highlights and colormaps in the layout viewer.

Calibre DESIGNrev, Calibre WORKbench, and Some Other Viewers.....	536
Cadence Virtuoso.....	536

Calibre DESIGNrev, Calibre WORKbench, and Some Other Viewers

No special action is needed to match histogram colors to the colors for highlights and colormaps. Calibre RVE handles the color matching automatically.

Cadence Virtuoso

Cadence Virtuoso has a setting to allow color matching and this setting is enabled by default.

See “[Matching Calibre RVE Colors to Cadence Virtuoso Highlight Colors](#)” in the *Calibre RVE User’s Manual*.

Defining Colors for Calibre DESIGNrev and Calibre WORKbench

You can define the layout viewer colors for histograms and highlights with a custom layer properties file.

Procedure

1. If you have a custom layer properties file you use with your layout, proceed to step 3.
2. In Calibre DESIGNrev, choose **Layer > Save Layer Properties** and create a custom layer properties file.
3. Open your custom layer properties file in the text editor of your choice.
4. Copy the following ten lines of text and paste them at the bottom of the layer properties file.

```
4158 navyblue speckle rve 1 4
4159 slateblue speckle rve 1 4
4160 turquoise speckle rve 1 4
4161 lightgray speckle rve 1 4
4162 wheat speckle rve 1 4
4163 goldenrod speckle rve 1 4
4164 yellow speckle rve 1 4
4165 sandybrown speckle rve 1 4
4166 coral speckle rve 1 4
4167 red speckle rve 1 4
```

5. Save the layer properties file.
6. In Calibre DESIGNrev, choose **Layer > Load Layer Properties** and load the modified layer properties. The new layers do not appear on layer palette until you highlight errors from Calibre RVE.

Note

 The colormap example above is applied for layers 4158-4167 only. This infers that your histogram divisions are set to 10. If set to 12, then colors 11 and 12 have to be defined in the same manner as the first 10 layers.

7. To access these layer properties every time you open a Calibre viewer, set this custom layer properties as your default with **Layer > Save as Default**.

Examples

For Cadence Virtuoso:

Consult your Siemens representative and request the following file:

```
colormap.display.drf
```


Appendix E

Calibre LFD Error Messages

There are Calibre LFD-specific error messages that you may encounter when running Calibre LFD. Some error messages apply to all Calibre LFD checks, while others are check and command specific.

Common Errors to All Checks	539
Check and Command Specific Errors	539

Common Errors to All Checks

Error messages that are common to all checks are often related to invalid name or argument specification.

[Table E-1](#) lists error messages that apply to all Calibre LFD checks. .

Table E-1. Error Messages Common to All Calibre LFD Check

Message
Error: Inconsistent <CheckName> call. There should be an even number of arguments, not <argument list's length> in: < argument list >.
Error: Invalid argument <argument> Invalid <CheckName> call with parameters <argument list>
Error: <argument> required argument missing. Invalid <CheckName> call with parameters <argument list>
Error: Invalid -layerOut argument. layerOut must be unique name Invalid <checkName> call with parameters <argument list >
Error: -database or -layerOut: required argument missing.
Error: Argument -priority required to be an integer.
CheckName conflict for <checkName>. Remove check or use -checkName option

Check and Command Specific Errors

Error messages that are specific to certain commands and checks are often related to invalid name or arguments that are used for the particular check or command.

Table E-2 lists error messages that are specific to individual checks or commands. **Table E-2** is organized according to the following checks and commands:

addPDKCheck	FrameCellOptimizer	MinSpaceCheck
addPDKLayer	InterLayerSpaceCheck	MinWidthCheck
AreaCheck	LFDregion	NonPrintingCheck
AreaOverlayCheck	LineEndCheck	OutputBands
CaptureContour	LoadPDK	PVband
createPDK	MaxCDVariabilityCheck	RegisterLayer
CSG	MinAreaOverlapCheck	VariabilityIndices
CSI	MinOverlayCheck	

Table E-2. Check-Specific Error Messages

Message
addPDKCheck
<argument>: required argument missing
Error: Invalid addPDKCheck call with parameters <arguments>
<argument>: Invalid argument to addPDKCheck
Error: Invalid addPDKCheck call with parameters <arguments>
addPDKLayer
<argument>: Required argument missing for addPDKLayer
Error: pdk: Invalid optimizerInfo argument, should have an even number of arguments
Error: pdk: Invalid FrameCellOptimizer argument, should have an even number of arguments
Error: FrameCellOptimizer halo: Required argument missing for addPDKLayer
Error: pdk: Invalid Optimizer argument, should have an even number of arguments
Optimizer \$argu: required argument missing for addPDKLayer
Optimizer \$argu: Invalid argument
Error: Invalid processInfo argument, should have an even number of arguments
<argument>: invalid member for processInfo List
<argument>: required argument missing in -processInfo argument
Error: RetLayers should have even number of members
<argument>: invalid argument
Error:mask<0/1> should have an even numbers of parameters

Table E-2. Check-Specific Error Messages (cont.)

Message
<argument>: required argument missing
mask1(<argument>): required argument missing
Error: PvBand list should have an even number of parameters
Error: Arguments to IndependentWindows can only be {yes}
Error: use either IslandLayers or AuxLayers
Invalid addPDKLayer call
AreaCheck
-minLFDarea or -maxLFDarea: required argument missing
either -minLFDarea or -maxLFDarea should be specified
-minLFDarea must be an integer or real number not <minLFDarea value>
-maxLFDarea must be an integer or real number not <maxLFDarea value>
AreaOverlayCheck
Error: Arguments to -overlapAreaType of AreaOverlayCheck can only be Average MinArea MaxArea
Invalid AreaOverlayCheck with parameters <argument list>
CaptureContour
This -dose2 option requires -optical2
Invalid CaptureContour call with parameters <argument list>
Arguments to -displacement can only be “min max”
Invalid CaptureContour call with parameters <argument list>
Arguments to -reference can only be “regular absolute”
Argument to CaptureContour can only be -focus -dose OR -displacement -reference -subwindow
Invalid CaptureContour call with parameters <argument list>
Error: Argument -errorFilterSize required to be a real number
Invalid CaptureContour call with parameters <argument list>
Error: Argument -errorFilterSize should be greater than zero
Invalid CaptureContour call with parameters <argument list>
CaptureContour -optical <optical model> -dose <dose> -size <size> dose not exist. Review Pvband specification <layer>
Error: Cannot produce contours due to foundry restriction

Table E-2. Check-Specific Error Messages (cont.)

Message
CreatePDK
Error: invalid argument <argument> in createPDK
-name: required argument missing
CSG
Error: -poly and -seedLayer have the same value
Invalid CSG call with parameters <argument list>
Error: -active and -seedLayer have the same value
Invalid CSG call with parameters <argument list>
Arguments to -polyCondition can only be: optical dose size [optical2 dose2 size2]
Invalid CSG call with parameters <argument list>
Arguments to -activeCondition can only be: optical dose size [optical2 dose2 size2]
Invalid CSG call with parameters <argument list>
Argument to -shift should be {<xshift> or <yshift>}
Invalid CSG call with parameters <argument list>
CSG -optical <optical> -dose <dose> -size <size> does not exist. Review Pvband Specification for the layer <polyLayer>
CSG -optical <optical> -dose <dose> -size <size> does not exist. Review Pvband Specification for the layer <activeLayer>
Error: CSG for layer \$mA(-poly), \$mA(-active) cannot be calculated. There are no PV-bands for layer \$mA(-poly)
CSI
Error: Argument to -layerCondition of CSI can only be: optical dose size [optical2 dose2 size2]
Invalid CSI call with parameters <argument list>
Error: CSI -optical <optical> -dose <dose> -size <size> does not exist. Review Pvband specification for -layer <layer>
FrameCellOptimizer
ERROR: LFD::FrameCellOptimizer -IF '-invert' is specified, the associated value must be "yes"
InterLayerSpaceCheck
Error: -minDRCspace must be greater than -minLFDspace
Invalid InterLayerSpaceCheck with parameters <argument list>

Table E-2. Check-Specific Error Messages (cont.)

Message
Error: -indexFilter should be larger than zero Invalid InterLayerSpaceCheck with parameters <argument list>
IfdRegion
-halo must be an integer or a real number not <halo value> Error: Invalid IfdRegion call with parameters <argument list>
-halo must be greater than zero Error: Invalid IfdRegion call with parameters <argument list >
-region or -cell: required argument missing Error: Invalid IfdRegion call with parameters <argument list>
Only one IfdRegion call is allowed without -layer option Error: Invalid IfdRegion call with parameters <argument list>
LineEndCheck
Arguments to -displacement can only be {min max} Invalid LineEndCheck call with parameters <argument list>
Arguments to -property can only be {yes no} Invalid LineEndCheck call with parameters <argument list>
LoadPDK
ERROR: duplicate pdk_name
MaxCDVariabilityCheck
Error: Arguments to -mode can only be {absolute or ratio} Invalid MaxCDVariabilityCheck call with parameters <argument list>
-maxDRCwidth must be greater than -minDRCwidth Invalid MaxCDVariabilityCheck call with parameters <argument list>
Arguments to -property can only be {non max max_ratio target_cd all} Invalid MaxCDVariabilityCheck call with parameters <argument list>
MaxCDVariabilityCheck argument -minLineEnd must be larger than zero Invalid MaxCDVariabilityCheck call with parameters <argument list>
MinAreaOverlapCheck
Arguments to -areaType can only be: “relative absolute” Invalid MinAreaOverlapCheck call with parameters <argument list>
MinOverlayCheck

Table E-2. Check-Specific Error Messages (cont.)

Message
Error: -minDRCencl must be larger than -minLFDencl
-indexFilter must be greater than zero
Invalid MinOverlayCheck call with parameters <argument list>
MinSpaceCheck
Error: -minDRCspace must be greater than -minLFDspace
Error: MinSpaceCheck argument -property has the value <property val>
Error: MinSpaceCheck argument -property can only have the value “Space”
-indexFilter must be greater than zero
Error: Invalid MinWidthCheck call with parameters <arguments>
-minMarkerArea must be greater than zero
Error: Invalid MinWidthCheck call with parameters <arguments>
-extendMarker must be greater than zero.
Error: Invalid MinWidthCheck call with parameters <arguments>
Arguments to -bandType can only be {absolute regular}
Error: Invalid MinWidthCheck call with parameters <arguments>
-minLineEnd must be larger than zero.
Error: Invalid MinWidthCheck call with parameters <arguments>
Argument to -security can only be {yes}.
Error: Invalid MinWidthCheck call with parameters <arguments>
MinWidthCheck
Error: -minDRCwidth must be greater than -minLFDwidth
Error: MinWidthCheck argument -property has the value <input value>
Error: MinWidthCheck argument -property can only have the value “MinCD”
Error: Invalid MinWidthCheck call with parameters <arguments>
-indexFilter must be greater than zero
Error: Invalid MinWidthCheck call with parameters <arguments>
-minMarkerArea must be greater than zero
Error: Invalid MinWidthCheck call with parameters <arguments>
-extendMarker must be greater than zero
Error: Invalid MinWidthCheck call with parameters <arguments>

Table E-2. Check-Specific Error Messages (cont.)

Message
Arguments to -bandType can only be {absolute regular} Error: Invalid MinWidthCheck call with parameters <arguments>
-minLineEnd must be larger than zero Error: Invalid MinWidthCheck call with parameters <arguments>
Argument to -security can only be {yes} Error: Invalid MinWidthCheck call with parameters <arguments>
NonPrintingCheck
Error: Argument -maxExtent required to be an integer Invalid NonPrintingCheck call with parameters <argument list>
Error: Argument -maxExtent required to be less than 25 Invalid NonPrintingCheck call with parameters <argument list>
Error: Argument -markerSize required to be larger than zero Invalid NonPrintingCheck call with parameters <argument list>
OutputBands
-layerOut is not compatible with -bandType all. use regular absolute instead Invalid OutputBands call with parameters <argument list>
Error: Argument -errorFilterSize required to be a real number Invalid OutputBands call with parameters <argument list>
Error: Argument -errorFilterSize should be greater than zero Invalid OutputBands call with parameters <argument list>
Error: Cannot produce Bands due to foundry restriction
Pvband
Only one "PVband -layer \$layer" Call is allowed
Error: Arguments to -independentWindows can only be {yes} Invalid PVband Call with parameters <argument list>
-doseSpanList should have the same number of elements as: -opticalSpanList -doseSpanList2 should have the same number of elements as -opticalSpanList2
-sizeSpanList should have the same number of elements as -opticalSpanList -sizeSpanList2 should have the same number of elements as -opticalSpanList2

Table E-2. Check-Specific Error Messages (cont.)

Message
Error: for double exposure PV-band -layerRET2, -opticalSpanList2, -doseSpanList2 are required arguments
Invalid PV-band call with parameters <argument list>
There is no PV-bands defined for \$mA(-layer) in the pdk \$mA(-pdk)
A window defined by opticalSpanList must contain at least two process conditions
Error: Model <model> must be uniquely defined, please correct PVband for layer <layer>
Error: <model type> Model <modelName> does not exist
Error: Invalid Pvband call
You have to call LFD::ProcessCorrection"
RegisterLayer
Error: <pdkName> is not defined
Error: <Processlayer> definition is not in pdk <pdk>
VariabilityIndices
Error: Variability Indices for <Layer Name> cannot be calculated
There are no Pvbands for layer <Layer Name>
Error: -windowSize custom requires lfdRegion
Error: subwindow <subwindow> does not exist. Review Pvband specification for -layer <layer>
Error: Variability Indices or -errorFilterSize for <layer> cannot be calculated. There are no checks associated to layer <layer>

Appendix F

Using Calibre LFD in the Cadence Environment

Siemens Digital Industries Software supports an exchange of data between the Calibre Litho-Friendly Design (Calibre LFD) software and Cadence routing tools such as Chip Optimizer (Catena) and NanoRoute.

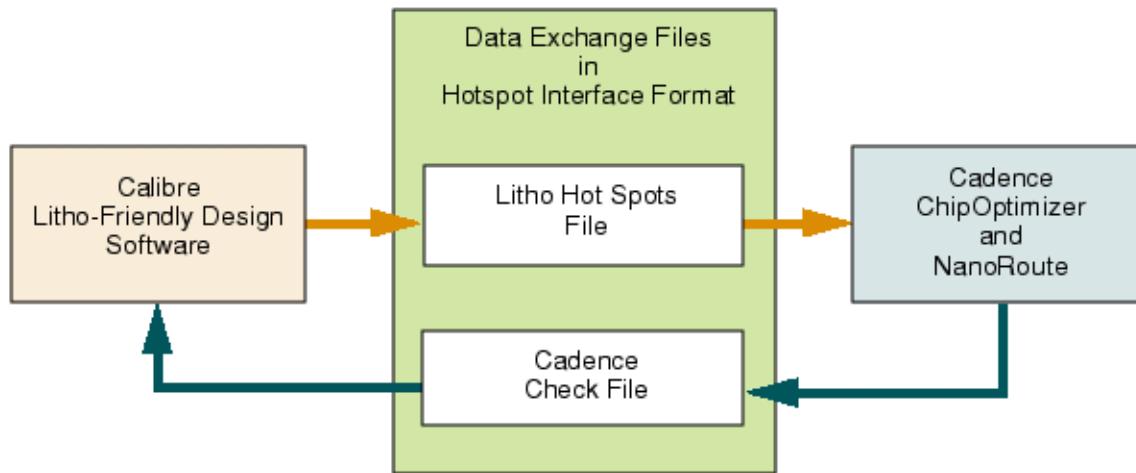
Data Flow Between Calibre LFD and the Cadence Environment	547
Running Calibre LFD After Cadence Routing.....	548
Passing Calibre LFD Results to Cadence Tools	549
Calibre LFD errmap and Command Reference for Cadence Tools.....	551

Data Flow Between Calibre LFD and the Cadence Environment

The data flow between Calibre LFD and Cadence tools is done through a set of data exchange files. These files identify the areas of the design that need hotspot correction and limit LFD to the modified areas of the design.

The flow of data between these tools is illustrated in [Figure F-1](#).

Figure F-1. Running Calibre LFD in the Cadence Environment



Within the Siemens EDA environment, this flow involves two additional tasks:

- Limiting Calibre LFD checking to areas modified by one of the Cadence tools.
- Generating the data exchange file in Hotspot Interface Format (HIF), identifying those areas requiring correction using the Cadence tools.

This appendix describes both these tasks. Refer to Cadence tool documentation for a description of the steps required on the Cadence side of the environment.

Refer to the LFD::[ReadHIF](#) command description for information on reading a HIF-formatted check file, and refer to the LFD::[WriteHIF](#) command description for information on writing a Litho hotspot file that adheres to the HIF standard.

This appendix also describes the stand-alone [rdb2hif Command](#) that performs the LFD::[WriteHIF](#) functionality from the command line.

Running Calibre LFD After Cadence Routing

You can run Calibre LFD on a design modified with any of the Cadence routing tools.

Prerequisites

- The layout database containing the modified routing.
- The HIF-formatted Check File identifying the areas that were modified.
- The TVF rule file needed to run the Calibre LFD checks on this design.

Procedure

1. Add the LFD::[ReadHIF](#) command to the TVF rule file:
 - a. Place the command after the LFD::[Begin](#), and before any other commands.
 - b. For the *check_file_name*, supply either an absolute path or a relative path based on the current working directory.
 - c. Add the LFD::ReadHIF command for each layer to be investigated. Make sure the *out_layer_name* in **-layerOut** matches the name of the layer to be used later for LFD::[IncrementalSelect](#) and each has a unique **-layerNumber**.
 - d. Set the **-halo** size to a value large enough to capture all data required that might have a lithographic impact on the modified features. Typically, this is equivalent to the optical diameter used during lithographic simulations.
 - e. Set the **-minWidth width_filter** as needed to filter out inconsequential modifications. Any polygons with a width less than this value are removed on the return layer. This value is set 1-2 nm below the specified layer's minimum DRC requirement.

- f. If the layout is magnified in the TVF rule deck using LAYOUT MAGNIFY SVRF command, add the -magnify parameter. Set *magnification* equal to the magnification specified in the SVRF command.
2. For each layer to be investigated, check that the layer names match up properly. The *out_layer* (last layer name in each -outLayers list) must match the following:
 - a. If you are using a PDK — the **-Designlayer** *input_layer_name* in the RegisterLayer statement.
 - b. If you are not using a PDK — the **-layer** *layer_name* in the LFD::PVband command.
3. Run Calibre LFD as usual.

Passing Calibre LFD Results to Cadence Tools

Pass Calibre LFD results to Cadence tools by generating a data exchange file (or Litho Hot Spots file) in Hotspot Interface Format (HIF). This file identifies those areas requiring correction using the Cadence tools.

In most cases it is most efficient to generate the Litho Hot Spots file as part of the Calibre LFD run. However, it is possible to generate this file from an existing RDB file.

Prerequisites

- The RDB generated by the Calibre LFD run.
- An errmap file that maps the check names in the RDB to error type, layer, and severity data that can be interpreted by the LFD::WriteHIF command. See “[errmap File Format](#)” on page 552.

Procedure

1. Create the basic TVF rule file with all the required standard code for TVF and for SVRF. This should define the inputs (such as LAYOUT SYSTEM and LAYOUT PATH) and outputs (such as DRC RESULTS DATABASE and DRC SUMMARY REPORT)
2. Copy the *namespace import* and *Load LFD package* code from the sample Calibre LFD file provided in the [Sample Rule File](#) section of this manual. Paste this code after the opening line (#! tvf) of the new TVF file.
3. Add the LFD::Begin and LFD::End statements to define the Calibre LFD block within the rule file.
4. Add the LFD::WriteHIF command after the LFD::Begin statement.
 - a. For the **HIF_file_name**, supply either an absolute pathname or a relative pathname based on the current working directory. Make sure this file does not already exist.

-
- b. Set the **-fdb** RDB file name to the name of the RDB file that has been generated by Calibre LFD. If the Calibre LFD rule file is set up to create multiple RDBs and you want data from more than one RDB to be passed to the Cadence tools, you must write a separate WriteHIF command for each one.

Note

 You can optionally flatten an RDB file with respect to its layout by using the **ldb_flattener** utility. See “[ldb_flattener](#)” on page 154.

-
-
-
- c. For the *errormap_file_name*, specify either an absolute pathname or a relative pathname based on the current working directory.
- d. Set the **-max_severity** value as needed to control the level of process variation that is of interest.
5. Run calibre -lfd as you usually do, passing in the name of the new TVF file as the rule file.

Calibre LFD errmap and Command Reference for Cadence Tools

The Calibre LFD errmap and rdb2hif command are used to translate hotspot check information to a format used by Cadence tools.

errmap File Format	552
rdb2hif Command	553

errmap File Format

Input for: WriteHIF and WriteICC

The errmap file is an ASCII file that describes the correspondence between a Calibre LFD hotspot check name and Cadence HIF parameters. This file format is also used to describe the correspondence to the Synopsys IC Compiler (ICC) hotspot file.

Format

An errmap file contains one line for each check in the Calibre LFD RDB. Each line contains four strings:

RDB_CheckName error_type layer severity

Parameters

- ***RDB_CheckName***
The name of a check, as specified in the Calibre LFD RDB.
- ***error_type***
The type of error represented by the check. Must be one of: SPACING or WIDTH.
- ***layer***
The name of the original design layer containing the feature that caused the error.
- ***severity***
The process-variation experiment, or subwindow, in which this error was found. Refer to the LFD::[WriteHIF](#) -max_severity value for an explanation of how subwindows relate to severity.

Examples

```
Metall1_MSC_P1_check SPACING M1 1
Metall1_MSC_P2_check SPACING M1 2
Metall2_MWC_P1_check WIDTH M2 1
Metall2_MWC_P2_check WIDTH M2 2
```

rdb2hif Command

The stand-alone RDB2HIF command writes a Litho hotspot file that adheres to the Cadence Hotspot Interface Format (HIF). This file passes hotspot data from Calibre LFD to the Cadence tools. The command performs the LFD::WriteHIF functionality from the command line.

Usage

```
rdb2hif
  -HIF HIF_file_name
  -rdb LFD_RDB_file_name
  -errmap errmap_file_name
  [-polygons {emit | skip | extents}]
  [-max_severity value]
```

Arguments

- **-HIF *HIF_file_name***

Required argument specifying the name to use for the HIF file generated by the utility. If a file with this name already exists, the utility aborts without generating the HIF file.

- **-rdb *LFD_RDB_file_name***

Required name of the RDB file name containing the Calibre LFD hotspots to be evaluated using the Cadence routing tools.

- **-errmap *errmap_file_name***

Required name of a separate ASCII file containing the 4-tuples detailing the correspondence between a Calibre LFD hotspot check name to Cadence HIF parameters. Refer to the description of the [errmap File Format](#) for exact syntax.

- **-polygons {emit | skip | extents}**

Optional keyword and value specifying how polygon Calibre LFD hotspot error markers are translated to HIF with the following actions:

- emit — Write all Calibre LFD hotspot polygons as HIF polygons. This is the default operation.
- skip — Skip all Calibre LFD hotspot polygons.
- extents — Convert Calibre LFD hotspot polygons to bounding box extents.

- **-max_severity *value***

Optional keyword and argument specifying the level of process variation that is of interest. Recall that the [PVband](#) command that generates the actual PV-band data allows you to generate multiple sets of PV-bands, each with a different level of process variation. The smallest set of focus and dose settings, reflecting the least process variation, is always defined first, and is referred to as “subwindow 1”. The next set of focus and dose settings, represent greater process variation, is defined second and referred to as “subwindow 2” and so on.

The `-max_severity` *value* is the subwindow, or position in the list of process variation experiments reflecting the level of process variation you want to correct for. Any Calibre LFD errors generated based on process conditions outside the specified subwindow are not translated into HIF data and not written to the Litho hotspot file. Thus, `-max_severity` is interpreted as follows:

- **1** — Write all errors found in PV-bands generated for subwindow 1.
- **2** — Write all errors found in PV-bands generated for subwindow 2. Because the PV-bands for subwindow 1 always lie complete inside the PV-bands for subwindow 2, this set includes the `-max_severity 1` errors.
- **3** — Write all errors found in PV-bands generated for subwindow 3. This set includes the `-max_severity 1` and `-max_severity 2` errors.

Examples

```
% rdb2hif -HIF myout.hif -rdb hotsp.rdb -errmap explore.errmap  
-polygons emit -max_severity 1
```

Appendix G

Using Calibre LFD in the Synopsys Environment

Siemens Digital Industries Software supports an integration between Calibre Litho-Friendly Design (Calibre LFD) software and the Synopsys IC Compiler (ICC) routing tool. The integration supports automatic detection and fixing of lithographic hotspots.

Calibre LFD detects potential hotspots and provides details of these hotspots to Synopsys IC Compiler so that ICC can fix these problem areas.

This Calibre LFD and Synopsys IC Compiler integration enables lithographic hotspot corrections to be performed with the Synopsys ICC tool after identifying lithographic hotspots with Calibre LFD. See “[Running Lithographic Hotspot Corrections in the Synopsys Environment](#)” on page 556.

Supporting the conversion process is the Synopsys RDB2RGF tool which provides IC Compiler Zroute support to translate Calibre’s Results Database (RDB) format into Synopsys’ Route Guidance Format (RGF). The Synopsys RDB2RGF tool is currently released as a beta version, and the tool supports RGF version 1.0. Contact Synopsys for access to the RDB2RGF tool.

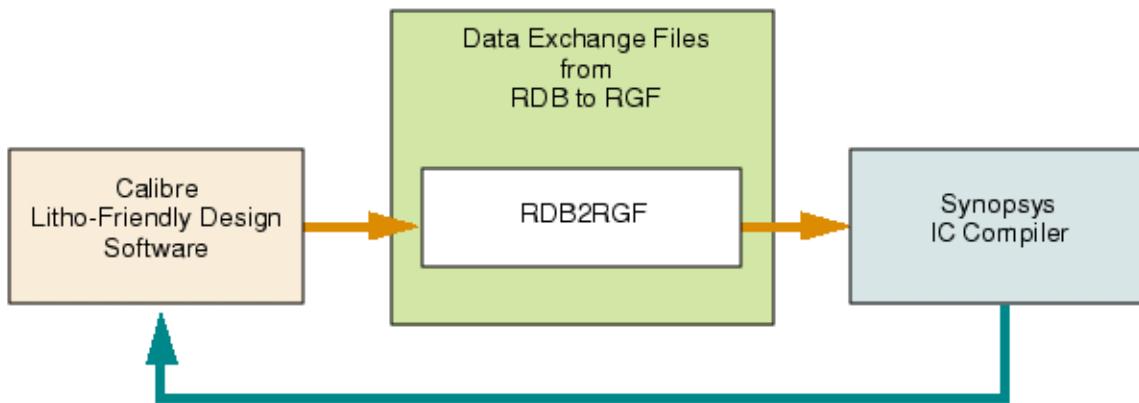
Data Flow Between Calibre LFD and the Synopsys Environment	555
Running Lithographic Hotspot Corrections in the Synopsys Environment	556

Data Flow Between Calibre LFD and the Synopsys Environment

Calibre’s RDB format files contain hotspot information for each lithographic weak point, including location, type, and properties. Synopsys’ RGF format files contain router fix guidance information, such as hotspot locations and metal avoidance areas.

The flow of data between these three tools is illustrated in [Figure G-1](#).

Figure G-1. Running Calibre LFD in the Synopsys Environment



Running Lithographic Hotspot Corrections in the Synopsys Environment

You can run Calibre LFD with the Synopsys routing tools.

Prerequisites

- The layout database containing the routing.
- The TVF rule deck and corresponding Calibre LFD kit required to run Calibre LFD on this design.

Procedure

1. Invoke the Synopsys IC Compiler tool with the gate-level routing netlist, and generate a layout database file for the Calibre LFD tool.
2. Invoke the Calibre LFD tool with the layout database file.
3. Use Calibre LFD to perform lithographic checks, outputting lithographic hotspots in RDB format. The RDB file contains hotspot information for each lithographic weak point, including location, type, and properties.

Note

 You can optionally flatten an RDB file with respect to its layout by using the **ldb_flattener** utility. See “[ldb_flattener](#)” on page 154

4. Run the Synopsys RDB2RGF tool, which translates the RDB file hotspot information into router fix guidance information in RGF. This fix guidance information provides hotspot location and metal avoidance areas. The RDB2RGF tool provides IC Compiler Zroute support to translate RDB format files into RGF.
5. Invoke the Synopsys IC Compiler using Zroute technology with the RGF file as input to fix lithographic hotspots. To fix lithographic weak points, Zroute uses the guiding

information to create metal avoidance areas that are avoided during the reroute, to fix hotspots.

6. Use IC Compiler to generate a revised layout database file containing fixed hotspots.

Related Topics

[WriteICC](#)

Appendix H Drawn-To-Contour Flow (D2C)

The Drawn-to-Contour (D2C) flow can be used as a replacement for the LFD::PVband function. The main goal of D2C is to use the LFD::Drawn2Contour function to generate PV-bands when the full OPC recipe and models are not available.

D2C Functionality	559
dtcoptimize Command	560

D2C Functionality

D2C replaces the OPC computation and subsequent contour calculation with an edge bias function based on an empirically-calibrated, directional, two-dimensional function. Because the edge bias function does not provide adequate control over the corner locations, a spline-based smoothing process is applied. The outcome is a piecewise-linear curve similar to those obtained by full lithographic simulations.

Related Topics

- [PVband](#)
- [Drawn2Contour](#)
- [dtcoptimize](#)

dtcoptimize Command

D2C uses the dtcoptimize Calibre WORKbench Tcl command to optimize model and fragment parameters as well as D2C image contour parameters.

dtcoptimize [561](#)

dtcoptimize

Calibre WORKbench Tcl command to optimize model and fragment parameters as well as Drawn-to-Contour (D2C) image contour parameters.

Usage

```
dtcoptimize
  -reference_layer reference_layer_name
  -contour_layer contour_layer_name
  -layout_name layout_name
  [-visible_layer visible_layer_name]
  [-handle handle_name]
  [-tilesize width_double]
  [-outdir directory]
  -setupfile setupfile_name
  [-modelfile_out modelfile_out_name]
  [-modelfile_in modelfile_in_name]
  [-log logfile_name]
  [-verbosity level_int]
  [-nostdout]
  [-bestValueOut outputfile_name]
  [-restartLogFile path_to_previous_logfile]
  [-remotestring remote_host_specification | -remotefile remote_hostfile_name]
  [-foptions sections_unsigned_int points_unsigned_int norm_type_unsigned_int]
  -coarse
    [explore count_unsigned_int]
    [runtime hours_double]
    [culltolerance distance_unsigned_int]
    [sam_cluster_size_minimum min_unsigned_int]
    [sam_cluster_threshold min_double]
    [sample_spacing microns_double]
    [square_core kernels_per_side_unsigned_int]
    [square_max_count max_num_kernels_unsigned_int]
    [[-] parameter_name min_double max_double]
    [[-] parameter_name constant_double...]
  ]
  -fine
    [explore count_unsigned_int]
    [runtime hours_double]
    modelfile_in modelfile_name
    [[-] parameter_name min_double max_double]
    [[-] parameter_name constant_double...]
  ]
```

Description

The dtcoptimize Calibre WORKbench Tcl command performs optimization of models and fragment parameters in the “coarse” mode. It performs optimization of D2C image contour parameters in the “fine” mode. The end result of the command is a calibrated model that may be applied to new layouts to accurately simulate layout mask contours given litho-correct layers as input.

This command performs optimization in two stages, coarse and fine. The command may be invoked to perform only coarse optimization, only fine optimization, or both. Either one or both of **-coarse** and **-fine** arguments must be specified. When both **-coarse** and **-fine** arguments are set, the stages are run sequentially, first coarse optimization and then fine optimization.

Coarse Mode

This mode reads in two (and optionally a third, “visible” layer) layers and produces an SDK model file with coefficients computed by a linear regression based on the calibration layout.

The basic steps are these: a sparse-mode density simulation on a user-supplied “calibration” layout to produce a SAM-formatted density/EPE file. The SAM data is filtered to reject data points greater than the user-specified cull tolerance. A second sparse-mode simulation is then launched using the calibrated SDK model and the user-supplied calibration layout. The output of this simulation is the D2C result on the calibration data. Finally, a measurement pass is performed which computes the average displacement at all fragment measurement sites between the generated optimized simulated contour and the user-provided calibration contour. This result forms a figure of merit used in determining how well the constructed model fits the golden calibration contour. The process iterates until the explore or runtime limit is met. During this process five parameters are simultaneously optimized. The required input parameters are these:

- hoodpix
- kerngrid
- largeFrag
- smallFrag
- minSpace

The first two parameters control the size and resolution of the model; the last three parameters control contour fragmentation. When the inter-feature distance is less than *minSpace* then size *smallFrag* fragments are used to create a contour, otherwise size *largeFrag* fragments are employed.

This step is the most time-consuming and therefore only a small number of explorations are encouraged. This time also depends in the size of the calibration layout. For layouts that are approximately 40 x 40 microns, 100 explorations are sufficient, and they are achieved in about 1 CPU-day.

Ranges or constant values for each of these parameters must be supplied; there are no default values. Input parameter values and ranges are checked for compliance against Calibre LFD OPC internal limits. At the end of the optimization run, the SDK model with the smallest average displacement value is written out. The parameter values which produced this result are inserted, in comment form, into the coarse-optimized model file.

Fine mode

This mode reads in two (and optionally a third, “visible” layer) layers and calculates optimized fitting parameters. An output model file, which records the optimized result, is always written out.

Note

 If the `-visible_layer` option is set for coarse optimization, it must subsequently be set for fine optimization to guarantee a consistent result.

The steps follow: The spline interpolation between is re-constituted from the existing coarse optimization as input. This is done by reading in the `modelfile_in` argument if only **-fine** optimization is to be run, or by reading in the just-generated model from the preceding **-coarse** stage when both optimizations are requested. A sparse-mode density simulation performed on this model re-creates the coarse-optimized layer result. Next the optimization parameters are substituted into the curve command and the resulting contour is exclusive-ORed against the user-supplied golden benchmark contour layer. There are eighteen required optimizable parameters:

- `default_cpdist`,
- `convex_cpdist`, `convex_maxdist`, `convex_endpt_offset`,
- `concave_cpdist`, `concave_maxdist`, `concave_endpt_offset`,
- `line_end_cpdist`, `line_end_maxdist`, `line_end_endpt_offset`, `line_end_width`,
`line_end_length`,
- `space_end_cpdist`, `space_end_maxdist`, `space_end_endpt_offset`, `space_end_width`,
`space_end_length`,
- `jog_length`

Ranges or constant values for each of these parameters must be supplied; there are no default values. The area of the XOR-created layer, normalized by the area of the golden contour, is used as the figure of merit to judge result quality. The process iterates until the explore or runtime limit is met. The curve parameter values which gave the smallest normalized XOR area are then inserted, in comment form, into the SDK model file (optionally-named by the user) constructed by this fine optimization stage.

Arguments

- **-reference_layer *reference_layer_name***

This is the drawn input layer that is manipulated by the optimizer to produce a result which matches the **-contour_layer** as closely as possible.

- **-contour_layer *contour_layer_name***

This is the golden contour layer which represents the goal for D2C optimization.

- **-layout_name *layout_name***

The pathname of the GDS or OASIS layout file which contains the circuit design with the reference and contour layers.

- **-visible_layer *visible_layer_name***

This layer is used in the initial creation of the SAM file during the sparse OPC density simulation. In the sparse mode simulation, this layer is marked as “visible” with the same Boolean flags as the target reference layer. Optional parameter, but must be consistent; if set for **-coarse**, it must also be set for **-fine**.

- **-handle *handle_name***

If a handle is set, it is assumed that you have already loaded a layout in Calibre WORKbench from disk, and the handle *handle_name* has been previously attached. If the handle option is not used, it is assumed that the dtcoptimize command must first locate and load the layout from disk. When no handle is set, the dtcoptimize command uses the handle name “layout0” as its internal name.

- **-tilesize *width_double***

Sets the size of tiles the layout are broken into when running. For example, the sparse-mode OPC engine. A positive, real value in microns is expected. If not set, the default tile size of 50 microns is used.

- **-outdir *directory***

If it does not already exist, creates the directory, *directory* to store the logfile, bestValueOut, and modelfile(s) (“DTC.coarse.mod”, “DTC.fine.mod”). If -outdir is not set, the directory “DtcOpt_dir” is created in the current directory.

- **-setupfile *setupfile_name***

The user must supply a setup file. Statements in the setup file may be expressed in terms of the following variables: _hoodpix, _kerngrid, _smallFrag, _largeFrag _minSpace, _maxDistance, _square, _tilesize, and _gridsize. Layer statements in the setup file are ignored, but are used to indicate statements which are placed before and after software-generated layer statements. offsetModel statements are automatically appended to the setup file, and you do not need to provide them.

- **-modelfile_out *modelfile_out_name***

Name of the constructed output model. If supplied, when the last requested optimization stage completes, an output model file, named “<outdir>/*modelfile_out_name*” is

constructed. The default name of the -coarse result is “DTC.coarse.mod”, and the default name of the -fine result is “DTC.fine.mod”. If only **-coarse** optimization is being run, the best values of the coarse parameters appears in the model file as comments. If the last stage was **-fine** optimization then, in addition, the model file is updated to reflect the 18 optimized parameters to the spline-based interpolation.

- **modelfile_in** *modelfile_in_name*

If only **-fine** stage optimization is being performed, this argument is mandatory, and otherwise it is forbidden. This is the full pathname of the input model file, presumably the result of a previous **-coarse** stage optimization.

- **-log** *logfile_name*

Name of a logfile to record the results of each coarse or fine iteration. The *logfile_name* appears inside the **-outdir** *directory*. If not set, the default name of the logfile is “dtcoptimize”.

- **-verbosity** *level_int*

Controls the amount of information printed to the **-log** logfile. Verbosity levels greater than or equal to 3 causes intermediate results from the coarse and fine optimization stages to be logged.

- **-nostdout**

Normally iteration results are written both to the logfile and to stdout. With this option, iteration results are only written to the logfile.

- **-bestValueOut** *outputfile_name*

If set, writes a separate file name *outputfile_name*, stored in the **-outdir** *directory* to hold the value of the best result measured by the figure of merit.

- **-restartLogFile** *path_to_previous_logfile*

If set, open and read in results of the previous logfile. The dtcoptimize command counts the number of iterations already performed and continues from that point. Thus, if explore is set to 150, and 100 iterations are found in the supplied previous log, this invocation of the dtcoptimize command runs 50 more iterations. If the **-fine** stage is being restarted, it is not necessary to provide a previous coarse model, as dtcoptimize can re-create it from the supplied logfile. However, specifying the coarse modelfile via **modelfile_in** decreases the runtime.

- **-remotestring** *remote_host_specification* | **-remotefile** *remote_hostfile_name*

If either of these mutually exclusive options is set, the dtcoptimize command runs in distributed mode. This enables parallel execution on the machines specified.

Syntax is: “remote host <hostname> <number-of-CPUs-to-use> launchname <name_of_primary_machine> dir <path_to_visible_working_base_directory> mgc_home <path_to_visible_mgc_home> exec <path_to_visible_calibrewb_binary>”.

The host <hostname> <number-of-CPUs-to-use> and directory <path_to_visible_working_base_directory> and exec <path_to_visible_calibrewb_binary> arguments are required.

- **-f**options *sections Unsigned_int points Unsigned_int norm_type Unsigned_int*

The dtcoptimize command uses the FRONT global optimizer. These options tune the optimizer. Legal integer values for <sections_unsigned_int> are 25, 5, and 3. Legal integer values for <points_unsigned_int> are 2, 3, and 5. <points> must be always less than <sections>. Legal integer values for <norm_type_unsigned_int> are 1, 2, and 3.

- **[-coarse] [-fine]**

Selects the optimization stage. The coarse stage is employed to optimize the 5 SDK model parameters; the fine stage is employed to optimize 18 parameters used for the spline interpolation. See “[Coarse Mode](#)” on page 562 and “[Fine mode](#)” on page 563 for complete descriptions.

Either one or both of -coarse and -fine must be specified. If both arguments are set, then the dtcoptimize command runs the optimizations sequentially, first the coarse stage and then the fine stage.

- **explore** *count Unsigned_int*

If -explore is set, limits the number of optimization iterations to this positive integer. The default value is 20000. This argument may be set separately for the **-coarse** and **-fine** optimization stages.

- **runtime** *hours Double*

If -runtime *hours_double* is set, optimization terminates after -runtime hours or -explore iterations, whichever occurs first. This argument may be set separately for the **-coarse** and **-fine** stages of optimization.

- **culltolerance** *distance Unsigned_int*

In **-coarse** mode, rejects sites where the measured displacement exceeds the set value. Tolerance must be a positive integer, expressed in nanometers. The default value is 40 nanometers. This parameter has no meaning in **-fine** mode.

- **sam_cluster_size_minimum** *min Unsigned_int*

In **-coarse** mode, decides whether a cluster is written out. Default is 1, which indicates that all clusters found are written out. A larger integer indicates that only aggregates of this number of samples or larger are written.

- **sam_cluster_threshold** *min Double*

In **-coarse** mode, sets the cluster's radius. A larger threshold results in larger clusters (and a small number of clusters) and in a higher compression of the SAM data. The default value is 0.05, with a step size of 0.01.

- `sample_spacing microns_double`

In **-coarse** mode, sets the basic spacing between sites for the final measurement phase between the generated coarse contour and the golden reference contour. Using a constant value insures that the cost function makes a consistent comparison between coarse iterations. The default value is 0.02 microns.

- `square_core kernels_per_side_unsigned_int`

In **-coarse** mode, sets the number of single-pixel kernels per side to create to cover the central model area. The default value is 40 single-pixel kernels per side. This implies the generation of 800 pixels ($40 \times 40 = 1600$ pixels, divided by 2 due to x-axis symmetry).

- `square_max_count max_num_kernels_unsigned_int`

In **-coarse** mode, sets the maximum number of kernels to use to cover the optical area (hoodpix). The default is 4096, which is presently the largest value allowed. Smaller values result in faster runtimes, but they may give less accurate results.

- `[-] parameter_name min_double max_double [-] parameter_name constant_double`

The optimizer presently accepts 5 specifically named variables and accompanying parametric ranges in coarse mode and 18 named variables in fine mode. In coarse mode hoodpix, kerngrid, minSpace, largeFrag and smallFrag must be specified. In fine mode default_cpdist, convex_cpdist, convex_maxdist, convex_endpt_offset, concave_cpdist, concave_maxdist, concave_endpt_offset, line_end_cpdist, line_end_maxdist, line_end_endpt_offset, line_end_width, line_end_length, space_end_cpdist, space_end_maxdist, space_end_endpt_offset, space_end_width, space_end_length, and jog_length must be specified. These variables have no defaults. The syntax for each specification is `<parameter_name> <min_double> <max_double>`. All values are specified in microns. If a particular parameter is to be held constant, then just its single constant value may be set. The leading “-” to the named variable is optional.

Examples

```
dtcoptimize \
-reference_layer referenceHole \
-contour_layer band_max \
-negative_layer negative \
-layout_name ./gds/RX.oas \
-log ob_M1_sc.log \
-outdir ob_M1_dta_sc \
-setupfile setup.in \
-nostdout \
-tilesize 100.0 \
-verbosity 12 \
-coarse \
explore 10 \
cullTolerance 30 \
sample_spacing 0.020 \
square_core 20 \
square_max_count 264 \
hoodpix 0.6 0.9 \
kerngrid 0.01 0.02 \
minSpace 0.05 0.08 \
largeFrag 0.21 0.60 \
smallFrag 0.05 0.10 \
-fine \
explore 10 \
default_cpdist 0.01 0.04 \
convex_cpdist 0.01 0.06 \
convex_maxdist 0.01 0.025 \
convex_endpt_offset -0.01 0 \
concave_cpdist 0.01 0.06 \
concave_maxdist 0.01 0.03 \
concave_endpt_offset -0.02 0 \
line_end_length 0.02 0.04 \
line_end_width 0.05 0.06 \
line_end_cpdist 0.01 0.06 \
line_end_maxdist 0.01 0.025 \
line_end_endpt_offset -0.01 0 \
space_end_length 0.02 0.04 \
space_end_width 0.05 0.06 \
space_end_cpdist 0.01 0.06 \
space_end_maxdist 0.01 0.025 \
space_end_endpt_offset -0.02 0 \
jog_length 0.005 0.010
```

Appendix I

Optimizing Printability

You can use the printableoptimize Calibre WORKbench Tcl command to optimize a printable model to be used with the LFD::PrintableCheck and LFD::PrintableBand operations. For a newer node, the printability model is compared against a target PV-band while for a mature process it is compared against dose-variation PV-bands.

Printability Commands **570**

Printability Commands

The LFD::PrintableCheck and LFD::PrintableBand operations are used with the printableoptimize Calibre WORKbench Tcl batch command.

The LFD::PrintableCheck command categorizes error markers defined in previous Calibre LFD checks into two main categories: *layout-problem* or *OPC-problem*. Error markers output by this check indicate that the error is due to the drawn layout. Error markers that pass through this printability checking indicate that this shape is printable if the foundry modifies the RET recipes.

The LFD::PrintableBand command simulates the expected lithographic PV-band in the absence of an RET recipe. This functionally is needed when layout designers move to an advanced-process technology before mature RET recipes are available from the foundry.

See “[printableoptimize](#)” on page 571 for the Calibre WORKbench Tcl batch command.

printableoptimize..... [571](#)

printableoptimize

A Calibre WORKbench Tcl command that performs optimization of printable models to be used with the LFD::PrintableCheck and LFD::PrintableBand operations.

Note

 If you need to cancel execution of the printableoptimize command, Control-C does not work. Instead, terminate the parent Calibre WORKbench process using the ‘kill -9 *calibrewb_process_id*’ shell command.

Usage

printableoptimize

-Input

```
layout_system layout_system_type
layout_path layout_path
layout_primary layout_primary
layout_precision layout_precision
reference_layer reference_layer_name
target_band contour_layer_name
dose dose1 dose2
optical_modelpath optical_model_path
optical_modelname optical_model_name
```

-Output

```
printable_model_path printable_model_path
printable_model_name printable_model_name
```

-Constant_parameters

```
frame_size frame_size
frame_extra frame_extra
mask_tone {dark | clear}
mask_background transmission_fraction phase_degrees
mask_foreground transmission_fraction phase_degrees
```

-Variable_parameters

```
explore count_unsigned_int
[mtf_threshold min_real max_real]
[intensity_threshold min_real max_real]
[edge_constraint_offset min_real max_real]
[edge_contrast min_real max_real]
[max_iterations min_real max_real]
[convex_corner_radius min_real max_real]
[concave_corner_radius min_real max_real]
```

Arguments

- **-Input**

Required argument that marks the start of the input parameters for printability model optimization.

layout_system *layout_system_type*

Required keyword and argument defining the type of the input layout file. Refer to the [Layout System](#) statement in the *Standard Verification Rule Format (SVRF) Manual* for further information.

layout_path *layout_path*

Required keyword and argument defining the path of the input layout file containing the target PV-band that is used in the calibration process.

layout_primary *layout_primary*

Required keyword and number used to specify the top-level cell in the layout file. Refer to the SVRF **Layout Primary** statement for further information.

layout_precision *layout_precision*

Required keyword and argument defining the precision of the input layout file. Refer to the SVRF **Layout Precision** statement for further information.

reference_layer *reference_layer_name*

Required keyword and argument defining the input drawn layer to be manipulated by the optimizer to produce PV-bands that match the **target_band** as closely as possible.

target_band *contour_layer_name*

Required keyword and argument defining the reference PV-band which represents the goal for the printability-model optimization.

dose *dose1* *dose2*

Required keyword and argument defining the dose variations at which the intermediate printability models generates the PV-bands. These values are mapped to two concurrent printability calls, which differ only by the **dose** parameter.

The **dose1** and **dose2** arguments define the minimum and maximum dose variations, and they are typically set to approximately 0.95 and 1.05 (+/- 5% respectively).

optical_modelpath *optical_model_path*

Required keyword and argument defining the path to a valid Calibre optical model.

optical_modelname *optical_model_name*

Required keyword and argument defining the name of the input optical model whose location is determined by the **optical_modelpath** keyword.

- **-Output**

Required argument that marks the start of the output parameters defining the name and the path of the output printable model.

printable_model_path *printable_model_path*

Required keyword and argument defining the path to the output calibrated printable model.

printable_model_name *printable_model_name*

Required keyword and argument defining the name of the output calibrated printable model whose location is determined by the **printable_model_path** option.

- **-Constant_parameters**

Required argument that marks the start of the constant parameters that are not used in the optimization process.

frame_size *frame_size*

Required keyword and argument defining the area in which internally printability solves the feasibility problem. It is a level of numerical approximation. Lower numbers improve performance, however they may cause inaccuracies. The frame size is in microns.

frame_extra *frame_extra*

Required keyword and argument defining the context area around the **frame_size**. Smaller numbers improve performance, but at convergence and accuracy's expense. The **frame_extra** is in microns.

mask_tone {**dark** | **clear**}

Required keyword and argument defining the *background_is_clear* parameter in the printableoptimize command. For dark tone: *background_is_clear* 0; For clear tone: *background_is_clear* 1.

mask_background transmission_fraction phase_degrees**mask_foreground transmission_fraction phase_degrees**

Required keywords and arguments used to calculate the *min_mask_transmission* and *max_mask_transmission* parameters of the printableoptimize command.

See “[Example 2](#)” on page 575.

- **-Variable_parameters**

Required argument that marks the start of the variable parameters that are optimized.

explore count_unsigned_int

Required keyword and argument defining the number of full explorations to find the optimal parameters. Typical values are between 10 and 1000.

mtf_threshold min_real max_real

Optional keyword and argument defining the threshold to use with the MTF. K-vector components whose MTF is below this value are *not* included in the optimization.

The *min_real* and the *max_real* numbers form the range within which the optimizer finds an optimum value for **mtf_threshold**.

Example: `mtf_threshold 0.0001 0.1`

intensity_threshold *min_real max_real*

Optional keyword and argument defining the threshold at which the process is centered. The *min_real* and the *max_real* numbers form the range within which the optimizer finds an optimum value for intensity_threshold.

Example: intensity_threshold 0.2 0.4

edge_constraint_offset *min_real max_real*

Optional keyword and argument defining how much tolerance is on each side. Is comparable to the width of the PV-band in general. The *min_real* and the *max_real* numbers form the range within which the optimizer finds an optimum value for edge_constraint_offset.

Example: edge_constraint_offset 0.002 0.005

edge_contrast *min_real max_real*

Optional keyword and argument defining the change of intensity per change in length. The *min_real* and the *max_real* numbers form the range within which the optimizer finds an optimum value for edge_contrast.

Example: edge_contrast 0.001 0.01

max_iterations *min_real max_real*

Optional keyword and argument defining the maximum number of iterations while solving the LP problem. The *min_real* and the *max_real* numbers form the range within which the optimizer finds an optimum value for max_iterations.

Example: max_iterations 10 20

convex_corner_radius *min_real max_real*

Optional keyword and argument defining the softening constraints around convex corners. The *min_real* and the *max_real* numbers form the range within which the optimizer finds an optimum value for convex_corner_radius

Example: convex_corner_radius 0.04 0.07

concave_corner_radius *min_real max_real*

Optional keyword and argument defining the softening constraints around concave corners. The *min_real* and the *max_real* numbers form the range within which the optimizer finds an optimum value for concave_corner_radius.

Example: concave_corner_radius 0.04 0.07

If any of the variable parameters is not mentioned, the package does not optimize it. In such cases, the printableoptimize command assigns a constant default value. If you want to set a variable parameter to any value other than the default value, you have to specify the variable

parameter and assign both *min_real* and *max_real* the desired value. This is shown in the following example:

```
printableoptimize \
...
...
-Variable_parameters \
    explore 50 \
    intensity_threshold 0.2 0.4 \
    concave_corner_radius 0.021 0.021
```

In this case the *intensity_threshold* parameter is optimized within the range of 0.2 to 0.4, while the *concave_corner_radius* parameter is not optimized. The value 0.021 is assigned to the *concave_corner_radius* parameter instead of the default value.

Examples

Example 1

```
printableoptimize \
-Input \
    layout_system OASIS \
    layout_path ./input/LAYOUT/ref_layout.oas \
    layout_primary ICV_75 \
    layout_precision 4000 \
    reference_layer target 7 \
    target_band target_band 9 \
    dose 1.0 0.90 \
    optical_modelpath ./models \
    optical_modelname M1_D0 \
-Output \
    printable_model_path "./output" \
    printable_model_name output_model.dat \
-Constant_parameters \
    frame_size 0.5 \
    frame_extra 0.3 \
    mask_tone dark \
    mask_background 0 0 \
    mask_foreground 1 0 \
-Variable_parameters \
    explore 50 \
    intensity_threshold 0.2 0.4 \
    edge_constraint_offset 0.002 0.0023
```

Example 2

Here are some example settings for the *-Constant_parameters* arguments:

- Binary dark field:

```
mask_tone dark
mask_background 0 0
mask_foreground 1 0
```

- Binary clear field:

```
mask_tone clear
mask_background 1 0
mask_foreground 0 0
```

- Attenuated 6% dark field:

```
mask_tone dark
mask_background 0.06 180
mask_foreground 1 0
```

- Attenuated 6% clear field:

```
mask_tone clear
mask_background 1 0
mask_foreground 0.06 180
```

Related Topics

[PrintableCheck](#)

[PrintableBand](#)

Appendix J

Using Pattern Matching Libraries

You can use Calibre LFD to generate a pattern library for use with the LFD::StructureOptimizer and LFD::RemoveErrors commands.

Patterns can be captured from a layout design using the LFD::StructureOutput command.

The batch mode utility, [pdl_lib_mgr](#), supports and facilitates customized pattern library development flows. The pdl_lib_mgr utility also contains the prompt console which give you access to pattern matching API functions.

The appendix also provides samples of some of the more common calibration flows the library development functionality is modeled around, however it does not describe advanced pattern library calibration techniques and methods.

Pattern Library Generation Flow	577
Pattern Capture and Output	578
Command Line Invocations for Pattern Matching Flow.....	580
How to Use LFD::RemoveErrors.....	582

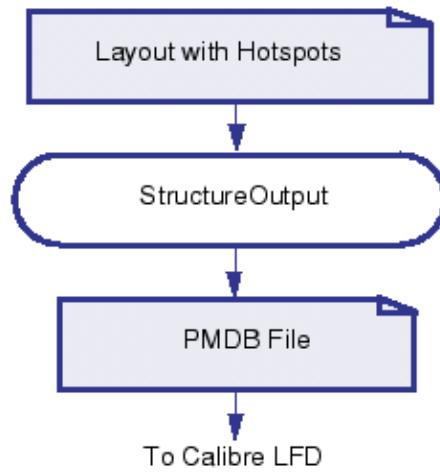
Pattern Library Generation Flow

The ability to create a pattern library is a prerequisite for invoking Calibre LFD functionality that uses pattern matching techniques for topology identification. Within the Calibre LFD framework, you can create a rule file that imports a layout design and processes the incoming information to generate a pattern library file.

Once the patterns have been captured, the tool writes out a Pattern Matching Database (PMDB) file. A PMDB file is the primary pattern library database. It must undergo the conversion process to extract the file format expected by Calibre LFD. It can be directly fed into the Pattern Matching GUI, the [pdl_lib_mgr](#) script.

The following flow demonstrates how to create a pattern library.

Figure J-1. Pattern Library Creation



Pattern Capture and Output

You can use the LFD::StructureOutput command to capture patterns directly from a GDS layout.

The prerequisite for running this operation is to have a layout with hotspot markers, or pattern extent markers in conjunction with the pattern layer(s) intended to be captured. This command must be written within a Calibre LFD deck.

The Calibre LFD deck uses LFD::StructureOutput to capture patterns from the incoming database:

```
#!tvf
namespace import CalibreDFM_LFD
<SVRF_Header_Info>

LAYER patLyr 1
LAYER hsLyr 2

LFD::Begin
LFD::StructureOutput \
    -layer patLyr \
    -hotspotLayer hsLyr \
    -patternRadius 0.3 \
    -outputFile "out.pmdb"
LFD::End

<eof>
```

The example Calibre LFD file above illustrates that this command can take two layers as input, and it writes out two outputs. For every polygon encountered on the hotspot layer (**hsLyr**), a pattern is stored within the pattern library. Each pattern consists of one or more pattern layer

geometries clipped using an internally calculated position relative to the pattern layer itself where the final diameter of the pattern is approximately twice the pattern radius specified. The hotspot marker is then included in the pattern definition itself as the pattern marker.

Note

 During the capture process, any hotspot markers whose topological context is precisely the same is compressed into a single pattern. This process accounts for all eight possible orientations for a given pattern to capture.

Once the patterns have been captured, the LFD::[StructureOutput](#) command writes out PMDB files.

Related Topics

[RemoveErrors](#)

[StructureOptimizer](#)

Command Line Invocations for Pattern Matching Flow

Two example command line invocations for the pattern match flow are provided along with a command reference for the pdl_lib_mgr batch command.

For running Calibre LFD and capturing a pattern library:

```
calibre -lfd [-hier] [-turbo [n]] [-turbo_litho [N]] ... rules
```

For merging two or more PMDB files:

```
pdl_lib_mgr merge input filename1 input filename2 output filename
```

pdl_lib_mgr **581**

pdl_lib_mgr

Batch command that operates on Pattern Matching Database (PMDB) files. The **merge** option can be used to merge two or more PMDB files, and the **prompt** option can be used to give you access to pattern matching API functions. The need for merging typically occurs when you are either running multiple databases to generate patterns, or when a preexisting pattern library file needs to be appended to another.

Usage

```
pdl_lib_mgr
{merge inputfilename [inputfilename] ... outputfilename} |
{prompt [tcl_script]}
```

Arguments

- **merge**

Required keyword used to combine multiple PMDB libraries into a single PMDB library. Duplicate patterns (all orientations are considered) are removed during the merging process. Output can optionally be sent to compile for automatic Macro file creation.

- **inputfilename**

Required keyword and argument specifying the input PMDB file. This argument can be specified more than once at the command line in series. All PMDB files used as input is merged.

- **outputfilename**

Required keyword and argument specifying the output PMDB filename that contains all of the unique patterns from each input library.

- **prompt** [*tcl_script*]

Required keyword and optional file that provides access to the pattern matching Tcl-based API batch interface, which allows you to edit and traverse pattern libraries.

- *tcl_script*

Optional argument specifying a Tcl script file.

Note

 The pattern matching API interface is discussed in the [Calibre Pattern Matching Reference Manual](#).

Examples

The following example takes *pmfile1.pmdb* and *pmfile2.pmdb* as input, and outputs the merged *new.pmdb* file.

```
$ pdl_lib_mgr merge input pmfile1.pmdb input pmfile2.pmdb output new.pmdb
```

This next example invokes the **pdl_lib_mgr prompt** console, processes some patterns, and exits.

```
$ pdl_lib_mgr prompt
// Calibre PDL Lib Manager v2014.1 ...
%
# Load a pattern library and save a list of library pattern names
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
%
% set patterns [pmatch::get_pattern_names -lib $x_lib]
p1_exact p4_bcm p5_bcm
%
# Output the number of patterns in the pattern library
% set num_patterns [llength $patterns]
3
% puts "There are $num_patterns patterns in this library"
There are 3 patterns in this library
%
% exit
$
```

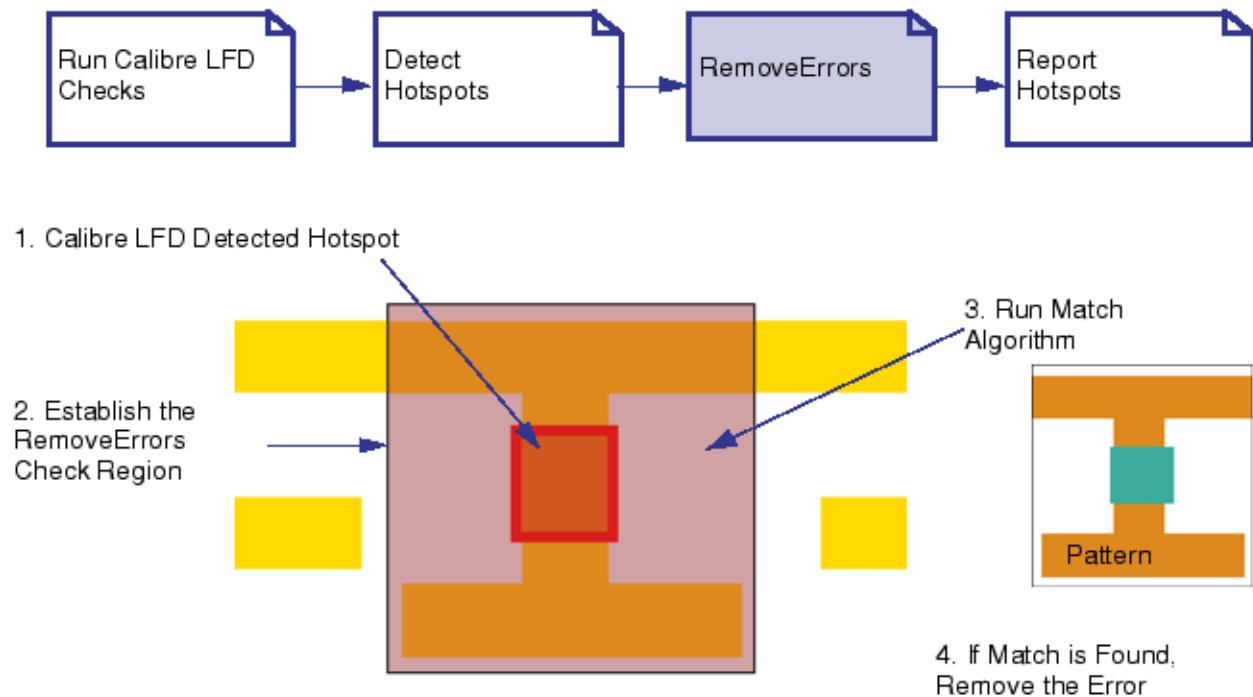
Alternatively, you can run Tcl script files as follows:

```
$ pdl_lib_mgr prompt script_name.tcl
// Calibre PDL Lib Manager v2014.1 ...
$
```

How to Use LFD::RemoveErrors

The LFD::RemoveErrors operation allows you to remove error markers that have been reported by any Calibre LFD check. A pattern library is used while scanning the results of Calibre LFD to detect any Calibre LFD errors whose context matches that of any pattern stored in the library. If any matches occur, then the underlying error markers are removed from the Calibre LFD check report.

Figure J-2. RemoveErrors Flow



To incorporate this methodology in your Calibre LFD kit, you must include the pattern database first by specifying an explicit source instruction, followed by adding the LFD::[RemoveErrors](#) function itself. In invoking the latter, you must know the name used when generating the pattern library. In addition, you must use **-checkName** with the given check to remove errors from. The LFD::[RemoveErrors](#) results is checked against the results from the associated check only.

The following is an example of a Calibre LFD deck including LFD::[RemoveErrors](#).

```
#!tvf
package require Calibre LFD
...
LFD::Begin
LFD::RegisterLayer \
    -layer metal
source "my.pmdb"
LFD::RemoveErrors \
    -layer metal \
    -checkName msc_1
    -pdlHandle metal_msc_1
LFD::MinSpaceCheck \
    -layer metal \
    -checkName msc_1
    -database "results.rdb"
LFD::End
<eof>
```

LFD::[RemoveErrors](#) results can only be mapped to the associated check you specify. Thus, there needs to be a one-to-one correspondence of Calibre LFD check to LFD::[RemoveErrors](#) function calls within the Calibre LFD deck, when specifying more than one check that requires LFD::[RemoveErrors](#). For more information and syntax regarding the LFD::[RemoveErrors](#) command, see the command reference chapter.

Appendix K

Calibre LFD Deep Neural Network (DNN) Flow

You can use the Calibre LFD deep neural network (DNN) flow to prepare a machine learning model that is trained with available hotspot data for a technology and layer to predict hotspot candidates in new designs with acceptable accuracy.

The flow uses Calibre LFD fast mode combined with machine learning algorithms and DNNs. It enables you to input a raw untrained model along with regions of interest and generate the necessary data to run a machine learning DNN training utility to create a trained model that can be used in the hotspot prediction phase of the flow.

Consult your Siemens representative for detailed information on the model and LFD kit content used in the Calibre LFD DNN flow.

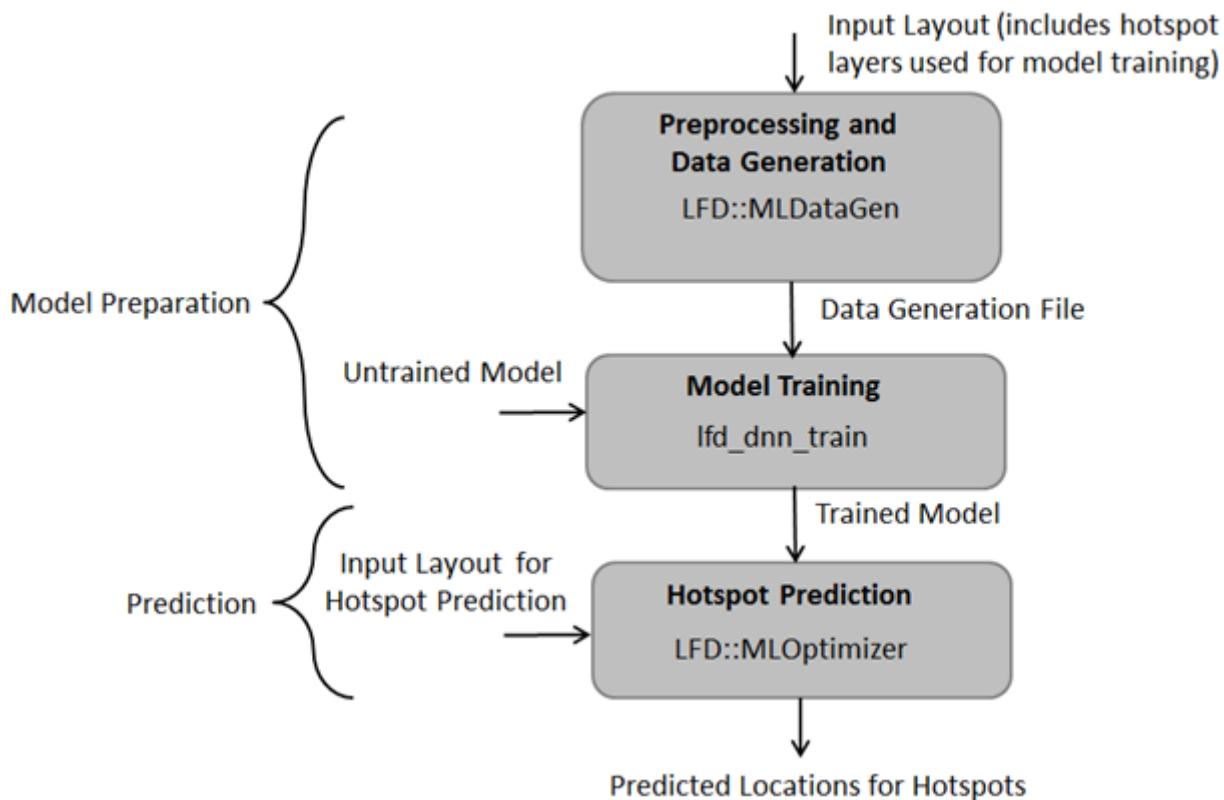
Calibre LFD DNN Machine Learning Model Preparation and Usage	586
Preprocessing and Data Generation	587
Model Training	589
Prediction	597
Trials to Enhance Accuracy	598

Calibre LFD DNN Machine Learning Model Preparation and Usage

You use machine learning to run the Calibre LFD DNN flow in three phases: preprocessing and data generation, model training, and hotspot prediction.

The flow begins with the preprocessing of the initial input layer that includes the hotspots used for model training. The regions of interest on this input layer are identified and anchors are set based on user-specifications for critical width or space, anchor separation and size, and window size. This phase of the flow uses the LFD::[MLDataGen](#) command to generate the data used to create an untrained model for the training phase of the flow. Once the untrained model is available, it is input to the `lfd_dnn_train` utility, which uses machine learning and DNNs to output a trained model that can be frozen and saved. In the last phase of the flow, hotspot prediction, the trained model is input to the LFD::[MLOptimizer](#) command, which ends the flow by generating hotspot candidate locations for a new input design layer.

Figure K-1. Machine Learning Model Preparation and Usage



Consult your Siemens representative for information on creating an untrained model and an LFD kit for the Calibre LFD DNN flow.

Preprocessing and Data Generation	587
Model Training	589

Prediction	597
Trials to Enhance Accuracy	598

Preprocessing and Data Generation

The functionality of the LFD::MLDataGen command is used for preprocessing and data generation in the first phase of the Calibre LFD DNN flow.

The preprocessing and data generation phase is used for model preparation in the Calibre LFD DNN flow. The goal of this phase is to use the available hotspot information in your input design to generate data for preparing a machine learning model for the DNN training phase of the flow.

Some parameters are common across the LFD::[MLDataGen](#) and LFD::[MLOptimizer](#) commands. The same values must be used for these parameters in order to ensure accurate prediction of hotspots in a new layout of interest.

The inputs required for the preprocessing and data generation phase of the Calibre LFD DNN flow are as follows:

- **Input layout** — The input layout includes the target layer (for example, metal 3), the critical hotspot layer, and the non-hotspot layer. OASIS or GDSII database format is supported. The input layout must be DRC clean.
- **Window Size** — The initial window size is estimated before the model preparation (preprocessing and data generation phase) by capturing the patterns of the input design with a starting size (for example, 10 pitches) and then matching these patterns on the same design to see how many locations match the hotspot patterns.
- **Anchor Settings** — The anchor placement includes all critical hotspots and selected regions of non-hotspots in unique cells. The anchor settings are based on the minimum width and space of the drawn layer. The LFD::MLDataGen command creates the anchors, separates them into hotspot and non-hotspot anchors, and classifies them. Only the unique anchors are included in the data generation that is input to the model training phase of the flow.
- **Rule File** — The Tcl Verification Format (TVF) rule file used for running Calibre LFD fast mode on the input layout. This rule file calls Calibre LFD, which includes the Calibre LFD rule block with the LFD::MLDataGen command statement.

The outputs from the preprocessing and data generation phase of the Calibre LFD DNN flow are as follows:

- **CSV file** — The output file containing the generated data with feature information from the input layout. This file is used for the model training phase of the Calibre LFD DNN flow. Standard CSV format is supported. The CSV file must be post-processed, before it can be input to the [lfd_dnn_train](#) utility and used for model training. This can be done

with a Linux sort command. For example, the csh “sort” command with text-processing commands is used to format the CSV file:

```
sort -r -u -k3 -t, datagen_out.csv > \
data_center_3600_UNIQ.csv
mkdir -p UNIQ
awk -F, '{tmp=$(NF); $(NF)="" ; OFS=FS; print tmp,$0}' \
data_center_3600_UNIQ.csv | sort -t, -k4 -u | awk -F, '{tmp=$1; \
$1="" ; OFS=FS; print substr($0 tmp,2)}' > UNIQ/\
data_center_3600_UNIQ.csv
```

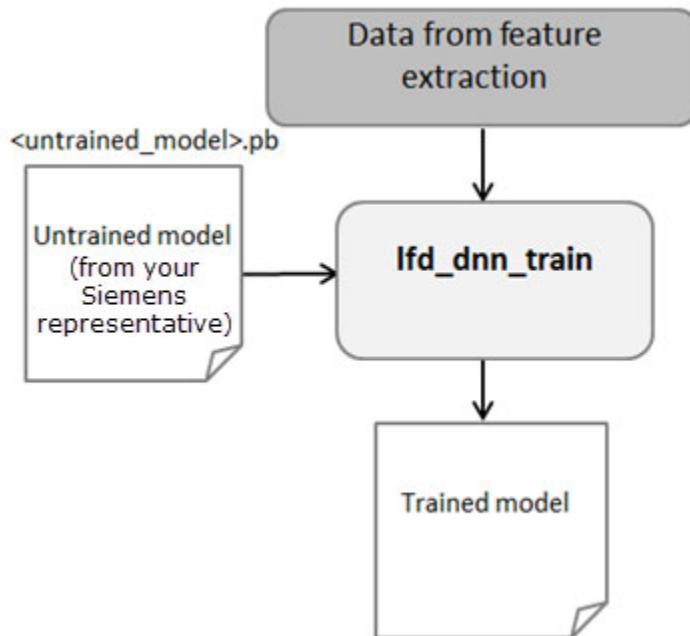
- **Results Database** — The results database from the Calibre DRC run. This database contains layer results that can be used for analysis.
- **DRC Summary Report** — The summary report from the Calibre DRC run.

Model Training

A Calibre LFD machine learning DNN training utility is used to perform model training in the second phase of the LFD DNN flow.

The model training utility, [lfd_dnn_train](#), uses the input design information in the CSV file from the previous preprocessing and data generation phase of the flow along with a raw (untrained) model to produce a trained model for the prediction phase of the Calibre LFD DNN flow. The training utility produces a trained model with an acceptable level of accuracy for predicting hotspots in a new design. A training transcript file that shows the iteration cost and accuracy of the training results can also be output.

Figure K-2. Calibre LFD DNN Model Training



Consult your Siemens representative for detailed information on the model and LFD kit content used in the Calibre LFD DNN flow.

lfd_dnn_train	590
Training Transcript Information	594

lfd_dnn_train

A Calibre LFD executable utility that uses machine learning to train a raw (untrained) model with the generated CSV file output from the preprocessing and data generation phase of the Calibre LFD DNN flow.

Usage

```
lfd_dnn_train
  -model_pb_file=pb_file_path
  -data_dir=data_dir_path
  -model_dir=model_dir_path
  [-batch_size=value]
  [-epochs=value]
  [-batch_balance=value]
  [-train_percentage=value]
  [-seed=value]
  [-freeze_only=yes]
  [-incremental=yes]
  [-validate_only=yes]
  [-meta_data_file=file_path]
  [-save_checkpoints=value]
  [-load_checkpoint=checkpoint_name]
  [-turbo= int]
```

Description

Generates a machine learning trained model that can be used for predicting hotspots on a design target layer. The inputs for running this utility are an untrained model and the CSV file output from the preprocessing and data generation phase of the Calibre LFD DNN flow. During processing, the data is passed through a deep neural network architecture with a set of ordinated steps and training algorithms driven by the hotspot classification. You control the output of the lfd_dnn_train utility by specifying the sampling parameters and iterations that result in an acceptable level of accuracy for the generated trained model.

Arguments

- **-model_pb_file=*pb_file_path***

Required keyword and argument specifying an input model that is a binary file containing the untrained network architecture output from a machine learning flow (*saved_model.pb*).

Consult your Siemens representative for information on creating an untrained model and LFD kit for the Calibre LFD DNN flow.

- **-data_dir=*data_dir_path***

Required keyword and argument specifying an input directory path that contains the files from the data generation feature extraction. Files that do not have a *.csv* file extension are

ignored. A standard CSV file format is supported. See “[Preprocessing and Data Generation](#)” on page 587 for information on the CSV file.

This keyword is not used with -freeze_only.

- **-model_dir=***model_dir_path*

Required keyword and argument specifying the directory path for the trained output model. This directory path is also used for an existing trained model that you want to “freeze” with the -freeze_only option.

- **-batch_size=***value*

Optional keyword and value specifying the number of training samples for each batch in training. When this option is specified, the training data is split into batches and passed through the network in groups of -batch_size (not greater than the total data read in). Specify *value* as a positive integer. The default is 100.

This keyword must not be used with -freeze_only or -validate_only.

- **-epochs=***value*

Optional keyword and value specifying the number of training iterations. When all data is passed (forward and backward) through the network, it constitutes one epoch or iteration. Specify *value* as a positive integer. The default is 50.

This keyword must not be used with -freeze_only or -validate_only.

- **-batch_balance=***value*

Optional keyword and value specifying the percentage (0 through 100) of hotspots that should be included per batch. Specify *value* as 0, a positive integer, or a constraint. The default is 0. For example,

- batch_balance=0 — No hotspot percentage is fixed, and the batch selection process is completely random (default).
- batch_balance >0 <100 — All batches contain a fixed percentage of hotspots within the constraint. This means the hotspots are oversampled, and each hotspot is used more than once during an iteration depending on the original distribution in the data.

This keyword must not be used with -freeze_only or -validate_only.

- **-train_percentage=***value*

Optional keyword and value specifying the percentage (0 through 100) of training data that should be used for training. Specify *value* as 0 or a positive integer less than or equal to 100. The default is 100.

For example, a value of 90 means that 90 percent of the data is used for training, and 10 percent is used for validation in each iteration. This can be useful for comparing the training accuracy with the accuracy of a blind subset of the data not included in training. The percentage of hotspots to non-hotspots in the original data files is preserved in the validation set.

The -train_percentage keyword must not be used with -freeze_only or -validate_only.

- `-seed=value`

Optional keyword and value specifying a seed for random initialization of network parameters and random batch selection during training. Specify *value* as a positive integer.

The `-seed` keyword must not be used with `-freeze_only` or `-validate_only`.

- `-freeze_only=yes`

Optional keyword and argument specifying whether to freeze an unfrozen trained model in the `-model_dir` directory path. Specify “yes” to freeze and save the model in the `-model_dir` directory path with the generated name *saved_frozen_model.pb*.

For model checkpoints, if `-load_checkpoint` is not specified with `-freeze_only`, the `<model_dir>/variables` path must contain the *variables.data-00000-of-00001* and the *variables.index* files, as this is the default checkpoint to be loaded. See the `-load_checkpoint` keyword for more information.

The `-freeze_only` keyword must not be used with `-save_checkpoints`, `-incremental`, or `-validate_only`.

- `-incremental=yes`

Optional keyword and argument enabling incremental training on a previously trained model for an additional number of epochs. The training options or data samples do not have to match the previous model’s training configuration. A valid model path must be provided for the `-model_dir` directory.

Note



The previous model is overwritten with the incremental training model.

For model checkpoints, if `-load_checkpoint` is not specified with `-incremental`, the `model_dir/variables` path must contain the *variables.data-00000-of-00001* and the *variables.index* files, as this is the default checkpoint to be loaded. See the `-save_checkpoints` and `-load_checkpoint` keywords for more information.

The `-incremental` keyword must not be used with `-freeze_only` or `-validate_only`.

- `-validate_only=yes`

Optional keyword and argument enabling the `lfd_dnn_train` functionality to run in validation mode only. This mode tests a trained model from the `-model_dir` directory on the input files in `-data_dir` directory and then prints the prediction results (hits and misses for hotspots and non-hotspots) for all the data in the files.

For model checkpoints, if `-load_checkpoint` is not specified with `-validate_only`, the `<model_dir>/variables` path must contain the *variables.data-00000-of-00001* and the *variables.index* files, as this is the default checkpoint to be loaded. See the `-load_checkpoint` keyword for more information.

The `-validate_only` keyword must not be used with other optional keywords where specified.

- `-meta_data_file=file_path`

Optional keyword and argument enabling the lfd_dnn_train utility to load information about the CSV data generation files, for example, the number of columns to skip, the number of feature vectors to read, and the number of labels. This information is parsed from the encoded metadata file output by the LFD::MLDataGen command from the data generation phase of the flow. The metadata file contains the parameters used to generate the data and ensures consistent usage across all of the machine learning commands (LFD::MLDataGen, lfd_dnn_train, and LFD::MLOptimizer). If the metadata file is specified without a path, it must be in the current working directory.

- `-save_checkpoints=value`

Optional keyword and argument specifying to save the model every N iterations, where N is a positive integer specified by *value*. The model is saved with the `<checkpoint_name>` prefix “*model.ckpt-<iteration #>*”. The lfd_dnn_train flow saves the model in the last iteration by default prefixed with the word “variables” even when the `-save_checkpoints` keyword is not specified, or the number of epochs is not divisible by N . For incremental training purposes, the last iteration is saved twice when `-save_checkpoints` is specified. All checkpoints are saved under the `<model_dir>/variables` path, and each checkpoint saves two files:

```
<checkpoint_name>.data-00000-of-00001
<checkpoint_name>.index
```

If `-save_checkpoints=N` is used with `-incremental=yes`, then the behavior is as follows:

If `-load_checkpoint=model.ckpt-<iteration X>`, the first model to be saved is *model.ckpt-<X+N>*, not *model.ckpt-<N>*.

If `-load_checkpoint=variables`, the first model to be saved is *model.ckpt-<N>*, and the model with the “variables” prefix is replaced on the last iteration. See the `-incremental` keyword description for more information.

The `-save_checkpoints` keyword must not be used with `-freeze_only` and `-validate_only`.

- `-load_checkpoint=checkpoint_name`

Optional keyword and argument specifying to load a valid checkpoint model. To be valid, the checkpoint model must exist under the `<model_dir>/variables` path and contain two files with the `<checkpoint_name>` prefix as saved by `-save_checkpoints`. For example,

```
<model_dir>/variables/model.ckpt-100.data-00000-of-00001
<model_dir>/variables/model.ckpt-100.index
```

The `-load_checkpoint` option is enabled only if either `-incremental` or `-validate_only` or `-freeze_only` is set to “yes”.

- `-turbo=int`

Optional keyword and argument specifying to restrict the number of CPUs to the specified number of threads. For example, `-turbo=8` runs on eight CPUs. If `-turbo` is not specified, all CPUs are used as needed.

Examples

This example uses the `lfd_dnn_train` utility with the specified input files (untrained model file (`saved_model.pb`), data generation files (`.csv`), and metadata file (`model.bin`)), to generate the trained model. The trained model can then be used by the `LFD::MLOptimizer` command for the hotspot prediction phase of the Calibre LFD DNN flow. You can specify to output the training transcript with the “`| tee`” command. The filenames and paths may be enclosed in sets of quotation marks (“ ”).

```
$MGC_HOME/bin/lfd_dnn_train \
  -model_pb_file="/home/user/saved_model.pb" \
  -data_dir="/home/user/datagen_files_path/" \
  -model_dir="./cpp_output_model" \
  -epochs=10 \
  -batch_size=20 \
  -seed=1 \
  -batch_balance=10 \
  -train_percentage=100 \
  -turbo=8 \
  -meta_data_file="model.bin"
```

Training Transcript Information

The transcript from the model training phase of the Calibre LFD DNN flow contains the training results and run information.

The training transcript from the `lfd_dnn_train` utility shows the time, cost, and a train accuracy percentage for each iteration or epoch along with other information for the run. The training transcript can also be used to review trends in values. A decreasing cost value indicates that the model is converging to a solution that fits the input data. If the cost value continues to decrease and does not reach a saturation value, more iterations can be added to reach a saturation value at the minimum cost. The percentages for “Total Accuracy” for the validation and the training data are shown after the last epoch along with the following classification statistics.

- **Recall (HS Accuracy)** — A measurement of the accuracy of the hotspot labeling. This metric is determined by the number of hotspot hits divided by the total number of hotspots. The score is reported as a percentage with higher scores interpreted as better values.
- **Precision** — A measurement of the accuracy of the hotspot labeling that accounts for both hotspot hits and non-hotspot misses. This metric is determined by the number of hotspot hits divided by the sum of the hotspot hits and non-hotspot misses. The score is reported as a percentage with higher scores interpreted as better values.
- **Extras Percentage** — A measurement of the accuracy of the non-hotspot labeling that accounts for both hotspot hits and non-hotspot misses. This metric is determined by the number of non-hotspot misses divided by the sum of the hotspot hits and non-hotspot misses. The score is reported as a percentage with higher scores interpreted as better values.

- **F1 Score** — A measurement of the accuracy of the hotspot and non-hotspot classification test. This metric is a weighted average of the precision and the recall values. The score is reported as a percentage.

The training transcript also reports the accuracy results and total counts for the hotspot and non-hotspot categories.

- **Category 0** — This is the non-hotspot accuracy; if the number of misses is high, you may need to include more non-hotspots in the training input or *decrease* the balance value (the percentage of hotspots included per training batch).
- **Category 1** — This is the hotspot accuracy; if the number of misses is high, the prediction accuracy decreases, and you may need to *increase* the balance value (the percentage of hotspots included per training batch).

The following example shows a transcript from a sample training run with a training percentage value of 90 percent, which means that 90 percent of the data is used for training, and 10 percent is used for validation in each iteration.

Example K-1. Training Transcript

```
// - LFD
Running training with compressed features
model_pb file /user/Mgc/home/src/lfd/dnn/pb_models/saved_model_3600.pb
Running on 48 physical + 48 virtual cores
Reading file data center_3600_UNIQ0.csv
Total data read: 7924
Feature size: 3600
Finished reading data in: 1 s
Batch size: 20
Balance: 0.1
Debug misses: 0
Train percentage: 90%
Epochs: 10
Mode: classification
Epochs    Time(s)      Cost      Training Accuracy          Validation Accuracy
-----  -----  -----
1        42       7.39945   Category 0: 89.0028, Category 1: 9.34343, Combined: 81.0288
2        29       2.86722   Category 0: 98.3146, Category 1: 1.13636, Combined: 88.587
3        28       2.44673   Category 0: 99.6208, Category 1: 0, Combined: 89.6486
4        24       2.19725   Category 0: 99.6348, Category 1: 0, Combined: 89.6613
5        26       1.89758   Category 0: 99.8034, Category 1: 0.126263, Combined: 89.8256
6        16       1.67345   Category 0: 99.9298, Category 1: 0, Combined: 89.9267
7        14       1.45622   Category 0: 99.986, Category 1: 0, Combined: 89.9772
8        13       1.28688   Category 0: 99.9017, Category 1: 0.126263, Combined: 89.9141
9        16       1.12732   Category 0: 99.8876, Category 1: 0, Combined: 89.8888
10       14       0.986237  Category 0: 99.9017, Category 1: 0, Combined: 89.9014
```

The Total CPU time and Real time for the run are printed at the bottom of the transcript.

```
Validation data statistics :
* Total Accuracy      = ( HS Hits + NHS Hits ) / ( Total HS + Total NHS )
= ( 0 + 791 )/( 2 + 792 ) = 99.6222 %

* Recall (HS Accuracy) = HS Hits / Total HS
= 0 / 2 = 0 %

* Precision           = HS Hits / ( HS Hits + NHS Misses )
= 0 / ( 0 + 1 ) = 0 %

* Extras Percentage   = NHS Misses / ( HS Hits + NHS Misses )
= 1 / ( 0 + 1 ) = 100 %

* F1 Score             = 2 * ( Precision * Recall ) / ( Precision + Recall )
= 2 * ( 0 * 0 ) / ( 0 + 0 ) = -nan %

#####
# Total Category [NHS] count = 792 :
#   hits = 791
#   misses = 1
#
# Total Category [HS] count = 2 :
#   hits = 0
#   misses = 2
#
Training data statistics :
* Total Accuracy      = ( HS Hits + NHS Hits ) / ( Total HS + Total NHS )
= ( 0 + 7116 )/( 10 + 7120 ) = 99.8036 %

* Recall (HS Accuracy) = HS Hits / Total HS
= 0 / 10 = 0 %

* Precision           = HS Hits / ( HS Hits + NHS Misses )
= 0 / ( 0 + 4 ) = 0 %

* Extras Percentage   = NHS Misses / ( HS Hits + NHS Misses )
= 4 / ( 0 + 4 ) = 100 %

* F1 Score             = 2 * ( Precision * Recall ) / ( Precision + Recall )
= 2 * ( 0 * 0 ) / ( 0 + 0 ) = -nan %

#####
# Total Category [NHS] count = 7120 :
#   hits = 7116
#   misses = 4
#
# Total Category [HS] count = 10 :
#   hits = 0
#   misses = 10
#
Total CPU time = 18741 s, Real time = 227 s
```

Prediction

The LFD::MLOptimizer command uses the trained output model to perform the hotspot prediction in the third and final phase of the Calibre LFD DNN flow.

Once the model training phase of the Calibre LFD DNN flow is complete, you can use the trained output model and the functionality of the LFD::[MLOptimizer](#) command to generate anchor points and then predict which of the anchor point locations are candidates for hotspots in a new layout.

Some options are common across the LFD::[MLDataGen](#) and LFD::MLOptimizer commands. The same values must be used for these options in order to ensure accurate prediction on a new layout.

The output trained model from model training and the LFD::MLOptimizer command used for the hotspot prediction phase are included in the LFD kit. Consult your Siemens representative for detailed information on the model and LFD kit content used in the Calibre LFD DNN flow.

The inputs required for the hotspot prediction phase of the Calibre LFD DNN flow are as follows:

- **Input layout (new)** — The input layout can be of OASIS or GDSII database format. The input layout must be DRC clean
- **Window Size** — The window size estimated by capturing the hotspots of the initial input design with a starting size (for example, 10 pitches) and then matching these patterns on the same design to see how many locations match the hotspot patterns.
- **Anchor Settings** — The anchor settings based on the minimum width and space of the drawn layer.
- **Model Path** — A full path including the filename of the trained model file output from the training phase of the Calibre LFD DNN flow. The trained model must be a single valid (.pb) file.
- **Rule File** — The Tcl Verification Format (TVF) rule file used for running Calibre LFD fast mode on the input layout. This rule file calls Calibre LFD, which includes the Calibre LFD rule block with the LFD::MLOptimizer command statement.

The outputs from the hotspot prediction phase of the Calibre LFD DNN flow are as follows:

- **Hotspot Prediction Layer** — OASIS or GDSII output database with the hotspot candidate anchor locations.
- **Results Database** — The results database from the Calibre DRC run. This database contains layer results that can be used for analysis.
- **DRC Summary Report** — The summary report from the Calibre DRC run.

Trials to Enhance Accuracy

Certain settings and actions can be used to improve the accuracy of the hotspot predictions in Calibre LFD DNN flow.

- **Window Size Selection** — Before starting the process of preparing the machine learning model for a specific layer, you should select a suitable window size for data generation, training, and prediction.

Using a random window size can result in model training on small patterns. This may not include enough context to differentiate between hotspots and non-hotspots, which can lead to excessive prediction of hotspot candidates.

To avoid this situation, capture the hotspots with a starting window size (for example, 10 pitches) and then match these patterns on the same design to see how many locations match the hotspot patterns.

If the matched area is comparable to the ideal hotspot area, then this window size can be an acceptable starting point. If the matched area is much larger than the ideal hotspot area, this means that extra features are included in it, and the pattern size is not large enough to define the hotspot. In this case, a larger context area (increased window size) should be used for the pattern in order to represent the hotspot.

The LFD::MLDataGen and LFD::MLOptimizer commands share some of the same specifications for minimum width and space, anchor separation and size, and window size. For accuracy, these values must be consistent across all commands used in the Calibre LFD DNN flow.

- **Step Size and Hotspot Sizing** — Accuracy increases when the sampling step size (anchor separation) decreases; however, this can result in increased run times for the flow. You can perform trials to select the optimum step size that results in acceptable accuracy and runtime.

A relation exists between the sampling step size value and the size or grow value for the hotspot in the preprocessing and data generation phase of the flow. The size or grow value should be between 0.5 and 1.0 of the sampling step size. However, if 0.5 is selected to be the sizing value for the hotspot, some hotspots located at line-ends can be missed, because they may not be covered by any sample point (anchor).

The LFD::MLDataGen and LFD::MLOptimizer commands share some of the same specifications for minimum width and space, anchor separation and size, and window size. For accuracy, these values must be consistent across all commands used in the Calibre LFD DNN flow.

- **Non-Hotspot Locations** — Adding more non-hotspot locations in the preprocessing and data generation phase can improve the accuracy of the predicted hotspot locations. The selection of non-hotspot areas should include representative samples from unique cells. This is accomplished by custom markers in unique cells of interest. The

LFD::[MLDataGen](#) command is used for defining the regions of interest in the preprocessing and data generation phase of the Calibre LFD DNN flow.

- **Iteration and Batch Balance** — During the training phase of the flow, you can adjust the -epochs value of the [lfd_dnn_train](#) utility to increase the number of iterations through the training network within your runtime tolerance. The -batch_balance option for the training utility can also be adjusted to increase the number of hotspots that are included in each sample batch with a similar consideration to the training phase runtime in the Calibre LFD DNN flow.

Index

— Symbols —

.csg File, 162

[] , 30

{ } , 30

| , 30

— A —

Absolute contours, 100

Absolute PV-Bands, 50

AC, 195

AddCustomCheck, 183

AddCustomOPCVCheck, 186

Anchor, 307

AreaCheck, 194

AreaOverlayCheck, 202

Attenuated masks, 418

— B —

Band, 320

Bands Database, 102, 103, 118

Begin, 322

Bold words, 29

Braces

 required around transmission values, 487

 required in spanLists, 93

Bridging, 61

— C —

Cadence

 Chip Optimizer, 547

 HIF, 429, 474, 548

 NanoRoute, 547

Calibre Contour Simplification for Gates, 157

Calibre DESIGNrev, 58, 60, 119, 537

Calibre LFD

 goals, 53

 how it works, 34

Calibre LFD block, 552

Calibre LFD flow, 51, 159

Calibre Litho-Friendly Design, 20

Calibre nmLVS flow, 159

Calibre RVE, 58, 99, 116, 121, 535

Calibre WORKbench, 560, 570

CaptureContour, 323

Check and Command Specific Errors, 539

Check Database, 102

Check Databases, 117

Check Names, 495

Checks

 built in, 94

 custom, 62, 95

ClassifyConfig, 328

Command line arguments, 74

Commands

 AddCustomCheck, 183

 AddCustomOPCVCheck, 186

 Anchor, 307

 AnchorRule, 314

 AreaCheck, 194

 AreaOverlayCheck, 202

 Band, 320

 Begin, 322

 CaptureContour, 323

 ClassifyConfig, 328

 Contour, 334

 CSG, 354

 CSI, 364

 Drawn2Contour, 367

 End, 371

 EndCapCheck, 210

 FrameCellOptimizer, 372

 GenerateHints, 374

 GetMacroLayers, 380

 GetOPCInputLayers, 382

 GetOPCLayers, 383

 GetRetargetLayers, 385

 GetVisibleLayers, 386

 InterLayeSpaceCheck, 217

 LayoutOptimizer, 390

 LFDregion, 392

-
- LineEndCheck, 227
 - LoadPDK, 394
 - Macro, 395
 - MaxAreaVariabilityCheck, 235
 - MaxCDVariabilityCheck, 242
 - MinAreaOverlapCheck, 250
 - MinOverlayCheck, 259
 - MinSpaceCheck, 269
 - MinWidthCheck, 280
 - MLDataGen, 397
 - MLOptimizer, 401
 - NonPrintingCheck, 289
 - OutputBands, 404
 - OverlayBand, 408
 - PrintableBand, 412
 - PrintableCheck, 294
 - ProcessCorrection, 414
 - PVband, 415
 - ReadECO, 428
 - ReadHIF, 429
 - ReadiLPC, 431
 - ReadRDB, 433
 - RegisterBands, 435
 - RegisterContour, 437
 - RegisterLayer, 439
 - RemoveErrors, 441
 - SetSTOMode, 449
 - SimConfig, 451
 - StructureOptimizer, 453
 - StructureOutput, 466
 - TopCellOptimizer, 469
 - VariabilityIndices, 471
 - ViaWidthCheck, 298
 - WriteHIF, 474
 - WriteICC, 476
 - Common Errors to All Checks, 539
 - Contacts
 - checking, 281
 - non-resolving, 60
 - Contour, 334
 - Contour simplification, 163
 - Contours, 100
 - Control file, 105, 108
 - Courier font, 29
 - CSG, 157, 158, 354
 - CSI, 364
 - Custom Checks, 62
 - registering, 183
 - writing, 95
 - Custom Metrics, 64
 - Customizable PV-Bands, 81, 320, 334, 409, 488
 - Band, 320
 - Contour, 334
 - OverlayBand, 408
 - D —
 - D2C, 367, 559
 - Deep Neural Network (DNN) Flow, 585
 - Derived Layers
 - saving contours as, 100
 - Design Flow, and LFD, 115
 - Design improvements, 150
 - Design Variability Index, 39, 63, 123
 - Detecting Printing Assist Features, 289
 - Detecting Side-Lobes, 289
 - Deviation from Drawn, 51
 - DFM Analyze, 98
 - doseSpanList, 92
 - Double pipes, 30
 - Drawn2Contour, 367, 562
 - dtcoptimize, 561
 - DVI, 39, 63, 64, 123
 - calculating, 471
 - equation for, 63, 471
 - understanding, 124
 - viewing, 123
 - E —
 - Echoing, 74
 - Embedding resist models in setup files, 501
 - Encrypting optical models, 500
 - End, 371
 - EndCapCheck, 210
 - Error Messages, 539
 - addPDKLayer, 540
 - AreaCheck, 541
 - AreaOverlayCheck, 541
 - CaptureContour, 541

-
- CreatePDK, 542
 - CSG, 542
 - CSI, 542
 - FrameCellOptimizer, 542
 - InterLayerSpaceCheck, 542
 - lfdRegion, 543
 - LineEndCheck, 543
 - LoadPDK, 543
 - MaxCDVariabilityCheck, 543
 - MinAreaOverlapCheck, 543
 - MinOverlayCheck, 544
 - MinSpaceCheck, 544
 - MinWidthCheck, 544
 - NonPrintingCheck, 545
 - OutputBands, 545
 - Pvband, 545
 - RegisterLayer, 546
 - VariabilityIndices, 546
 - Expressions
 - for calculating scores, 98
 - in metric calculations, 65
 - Extents, 97
- F —**
- Failures, 94
 - Flows
 - Calibre LFD, 51
 - Calibre LFD in design, 115
 - creating encrypted LFD kits, 500
 - creating process-specific LFD kits, 54
 - LFD, 41
 - Focus/dose settings, 91
 - Formulas
 - DVI, 63
 - PVI, 64
 - Foundry IP, 499
 - FrameCellOptimizer, 372
- G —**
- GDS, writing PV-bands to, 105
 - Generating PV-Bands, 415
 - Geometric optimizer, 390
 - GetMacroLayers, 380
 - GetOPCInputLayers, 382
 - GetOPCLayers, 383
 - GetRetargetLayers, 385
- H —**
- GetVisibleLayers, 386
 - Halos, 153
 - Heavy font, 29
 - Hints for correcting problems, 150
- I —**
- Improving designs, 150
 - Improving performance, 372, 469
 - Indexing Database, 102
 - Indexing Databases, 116
 - Inner contours, 100
 - Inner edge layers, 62
 - InterLayerSpaceCheck, 217
 - IP, protecting, 499
 - Italic font, 29
- L —**
- Ladjusted, 161
 - Layer names, 72, 500
 - Layers
 - derived by LFD, 62
 - region, 79
 - required for LFD, 90
 - LayoutOptimizer, 390
 - LFD
 - definition, 19
 - in the design flow, 115
 - scores, 497
 - LFD Block, 89
 - LFD Blocks, 75, 322, 371
 - LFD check registry, 183
 - LFD Fast Mode, 77, 395, 453
 - LFD flow, 41
 - lfd_dnn_train, 590
 - LFDregion, 392
 - LineEndCheck, 227
 - Lists, order dependency, 422, 489
 - LoadPDK, 394
- M —**
- Macro, 395
 - MAOC, 251
 - Markers, 38
 - MAVC, 236
 - MaxAreaVariabilityCheck, 235

-
- MaxCDVariabilityCheck, 242
 MBE,*see* Model based extraction, 354
 MBH Rule and Map Files, 132
 Metrics
 custom, 64, 98
 designing, 63
 standard, 63
 MinAreaOverlapCheck, 250
 Minimum keyword, 30
 MinOverlayCheck, 259
 MinSpaceCheck, 269
 MinWidthCheck, 280
 Model based extraction, 354, 364
 Model List impact on error ranking, 421
 Model paths, 420
 Model-Based Hints (MBH), 127
 Modifying layouts, 150
 MSC, 236, 251, 270, 281
 Multiple mask definition, 418
 Multi-ThreadedProcessing, 25
 MWC, 281
- N —**
- Namespaces, 105
 Non-Printability Region, 50
 NonPrintingCheck, 289
 Non-resolving contacts, 61
- O —**
- OASIS, writing PV-bands to, 105
 OPC, 21, 36, 52, 81
 Optical models
 binary, 500
 creating, 57
 Optical parameter hiding, 500
 Optical Transmission, 417, 487
 opticalSpanList, 92
 Optimizer, 390
 Outer contour, 100
 Outer edge layers, 62
 OutputBands, 404
 OverlayBand, 408
- P —**
- Parentheses, 30
 Pattern Matching, 441, 453, 466, 577
- using pattern matching libraries, 577
 using pdl_lib_mgr, 581
 using RemoveErrors, 441, 582
 using StructureOptimizer, 453
 using StructureOutput, 466
 Pattern matching API, 77, 581
 PDK
 definition, 70
 running LFD with, 105
 security settings in, 482
 pdl_lib_mgr, 577, 581
 Performance improvement, 372, 469
 Performance window, 35
 Phase-shifting transmissions, 418
 Pinching, 38, 61
 Pipes, 30
 Predicting Overlay Area, 202
 Prerequisites for LFD, 70
 Printability Region, 48
 PrintableBand, 412
 PrintableCheck, 294
 Printing Assist Features, 289
 Printing SRAFs, 61
 Process Definition Kit, 70
 Process Variability Index, 38, 64
 Process Variation, 35
 experiments, 55, 421
 experiments, multiple, 91
 Process window, 35
 ProcessCorrection, 414
 Process-Specific Calibre LFD Kits, 54
 Protecting IP, 499
 Pullbacks, 228
 PVband Command, 415
 PV-Band layers, manipulating, 96
 PV-Bands, 47
 absolute, 50
 contours, 100
 definition, 36
 illustration, 48
 regular, 50
 viewing, 58, 124
 writing to GDS database, 105
 writing to OASIS database, 105
 PVI, 38

equations for, 64, 97, 471
formula for, 471
generating, 471
understanding, 124
viewing, 123

— Q —

Quotation marks, 30

— R —

Ranking Errors, 66, 421
RC extraction using LFD, 163
ReadECO, 428
ReadHIF, 429
ReadiLPC, 431
ReadRDB, 433
Regions, 392
RegisterBands, 435
RegisterContour, 437
Registering Custom Checks, 183
RegisterLayer, 439
Regular PV-Bands, 50
RemoveErrors, 441
Resist models, 501
RET, 36, 42, 44, 51, 81, 412, 531
Rule Deck for Running CSG, 167
Rule deck sample, 519
Runsets CI, 515

— S —

Sample rule deck, 519
Scores
 DV values, 497
 partitioning designs for, 97
 understanding, 124
 viewing, 123
Scoring Designs, 63
Scoring Processes, 63
Security Settings in PDK, 482
SetSTOMode, 449
Side-Lobe Detection, 289
SimConfig, 451
Simplified contours, 163
Slanted words, 29
SPICE model formulas with MBE, 162
Square parentheses, 30

Standard Metrics, 63
Statistics, 123
STO, 577
StructureOptimizer, 453, 577
StructureOutput, 466
Subwindows, 421
Synopsys
 IC Compiler, 476, 555
 RDB2RGF, 555
 RGF, 555
 Zroute, 555
Syntax Conventions, 29
System-Generated Names, 495

— T —

Tcl Verification Format, 26, 72
Timing analysis using LFD, 159
Timing Improvement, 37
Timing Simulations, 157
TopCellOptimizer, 469
-turbo, 25
-turbo_all, 25
-turbo_litho, 25
TVF, 72
 echo_svrf, 74
 get_tvf_arg, 74
-tvfarg, 74

— U —

Underlined words, 29

— V —

Variability
 index, 124
 predicting, 52
 region, 48
Variability index, 97
VariabilityIndices, 471
Variable names, 500
Variables
 and namespace issues, 74
VERBATIM, 105
Vias, 281
ViaWidthCheck, 298
Viewing
 checking, 117

indexing, [116](#)
PV-bands, [118](#), [124](#)
scores, [123](#)

— **W** —

Wadjusted, [161](#)
WriteHIF, [474](#)
WriteICC, [476](#)

— **X** —

xRC tools and LFD, [163](#)

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.

